Proceedings of the
Fourth International Working Conference on
Dynamic Modelling and Information Systems
Introduction

In 1994, the working conference on dynamic modelling and information systems will be held for the fourth time (DYNMOD IV). Three successful dynamic modelling conferences have been held in 1990 (the Netherlands), 1991 (USA), and 1992 (the Netherlands). In 1992 we decided to organize this conference biannually to encourage the presentation of new research results.

Dynamic modelling is an approach for analyzing dynamic aspects of organizations and their information systems. The approach recognizes that activities happen concurrently in organizations, and that methods are needed to model dynamics and concurrency within organizations and information systems. The objective of the International Working Conferences on Dynamic Modelling and Information Systems is to bring together interested information systems users and developers, researchers and educators for cross-fertilization and exchange of information and ideas. The expectation is that continued development of dynamic models of information systems and organizations will lead to better organizational problem solving and the solution of problems in information systems development and operation.

When looking at the papers that were selected for inclusion in the proceedings, a broad spectrum of approaches to dynamic modelling can be recognized: object oriented modelling, Petri Nets, simulation and animation, rule-based modelling, agent based systems, and role based systems. When soliciting papers for this conference, we especially asked for papers about approaches and tools, including tests of the methods and instruments in real-life cases.

We would like to thank the international program committee and the organizing committee of this conference for the massive amount of work they have done.

Delft, September 1994

Alexander Verbraeck
Henk G. Sol
Pieter W.G. Bots
CONFERENCE CHAIRS

General Chair: Henk G. Sol
Program Chairs: Alexander Verbraeck
Pieter W.G. Bots

PROGRAM COMMITTEE AND REFEREING

Valeria de Antonellis
Politecnico di Milano, Italy

Pieter W.G. Bots
Delft University, The Netherlands

Robert Crosslin
Technology Economics, Inc., USA

Jan L.G. Dietz
University of Limburg, The Netherlands

Eric Dubois
Facultés Universitaires de Namur, Belgium

John B. Evans
University of Hong Kong, Hong Kong

Alan Fowler
Ernst & Young Mgmt Consultants, UK

Ronald Lee
Erasmus University, The Netherlands

Mike McCracken
Georgia Institute of Technology, USA

Jeff Rothenberg
The RAND Corporation, USA

Vesa Savolainen
University of Jyväskylä, Finland

Henk G. Sol
Delft University, The Netherlands

Aphrodite Tsalgatidou
University of Athens, Greece

Annikka Tuukkanen
Helsinki University of Technology, Finland

Alexander Verbraeck
Delft University, The Netherlands

James R. Warren
University of South Australia, Australia

Folkert W. Wierda
Origin Darmstadt, Germany

ORGANIZING COMMITTEE

Stephan O. den Hengst
Roeland J.R. van der Spek
Alexander Verbraeck
# Table of Contents

**MOBILE: A Modular Workflow Model and Architecture**  
Stephan Jablonski. Digital Equipment GmbH, Karlsruhe, Germany  

**Information Systems Modelling using LOOPN++, an Object Petri Net Scheme**  
Chris Keen, Charles Lakos. University of Tasmania, Hobart, Tasmania  

**A Logic to Specify Real-Time Object Behavior**  
Klemens Böhm. GMD-IPSI, Darmstadt, Germany  
Amilcar Sernadas. INESC, Lisboa, Portugal  

**TEMPO: a Support for the Modeling of Objects with Dynamic Behavior**  
Noureddine Belkhatir, Walcélio L. Melo. LGI, Grenoble, France  

**Dynamic Performance Evaluation of Office Information Systems**  
Vesa Savolainen. University of Jyväskylä, Finland  

**T-ORM: Evolving Objects and Roles**  
Elisa Peressi, Angelo Montanari. Università di Udine, Italy  
Barbara Pernici. Politecnico di Milano, Italy  

**Dynamic Modelling and Simulation of Information Systems using Multi-Agent Systems**  
Christophe Roche, F. Deloule, J.C. Profizi. Université de Savoie, Le Bourget du Lac, France  

**Automated Design of Electronic Trade Procedures using Documentary Petri Nets**  
Ronald M. Lee, Roger W.H. Bons, Clive D. Wrigley, René W. Wagenaar. Erasmus University, Rotterdam, The Netherlands  

**Newspaper Business Redesign: Organization Models and Economic Consequences**  
Annikka Tuukkanen. Helsinki University of Technology, Espoo, Finland  

**Dynamic Modelling of a Manufacturing Firm's Information System with OLYMPIOS-Design**  
Ramzi Guetari, Frédéric Piard. Université de Savoie, Annecy, France  

**Expressing Production Control Principles in the DEMO Communication Model**  
Nardo B.J. van der Rijst, Jan L.G. Dietz. Delft University of Technology, The Netherlands  

**Simulating Individual and Team Expertise in a Dynamic Decision Making Environment**  
Michael D. Coover, David W. Dorsey. University of South Florida, Tampa, USA
Agent-Oriented Requirements Engineering. A Case Study using the ALBERT Language ......................................................... 205
Eric Dubois, Philippe Du Bois, Frédéric Dubru, Michaël Petit. University of Namur, Belgium

A Hard Core for Soft Problems. A Business Engineering Case Study within the Amsterdam Municipal Police Force ......................................................... 239
Jeroen W. van Meel, Pieter W.G. Bots, Henk G. Sol. Delft University of Technology, The Netherlands

Trusted Linkages and System Chaos: The Case for Dynamic Modelling of Organisations ................................................................. 271
Doug Seeley. InterDynamics, Inglewood, Australia
James R. Warren, University of South Australia, The Levels, Australia

Modelling Organisational Coordination ......................................................... 303
Gert-Jan de Vreede, Daniel T.T. van Eijck. Delft University of Technology, The Netherlands

Dynamic Process Modelling Through Multi-Level RBNs ......................................................... 327
Aphrodite Tsalgatidou, Dimitris Gouscos, Constantin Halatsis. University of Athens, Greece
Proceedings of the
Fourth International Working Conference on
Dynamic Modelling and Information Systems
A. Verbraeck, H.G. Sol, P.W.G. Bots (Eds.)

September 28-30, 1994
Noordwijkerhout, the Netherlands
MOBILE: A Modular Workflow Model and Architecture

Stefan Jablonski
Digital Equipment GmbH, AIT Karlsruhe
Vincenz-Priessnitz-Strasse 1, D-76131 Karlsruhe
Tel.: [49] (721) 6902-25 Fax: [49] (721) 696816
e-mail: jablonski@kampus.enet.dec.com

Abstract. Workflows are characterized by multiple aspects: the functional aspect describes what has to be executed; the behavioral aspect tells when to execute a workflow; the organizational aspect determines agents that have to perform a workflow; the informational aspect defines data flow between workflows. In this paper we present a workflow model which follows this separation into various aspects. The major principle of the workflow model is modularity what is the prerequisite for reusability, extensibility, robustness, and ease of use. Finally, we briefly introduce an architecture of a Workflow Management System which implements the various aspects of the workflow model.

Keywords. Workflow Management, Workflow Model, Architecture of a Workflow Management System.

1. Introduction

The design of Cooperative Information Systems is a complex task that needs methodological support. It combines methods and techniques from organization theory, behavioral theory, decision theory, and database theory. These techniques are necessary in order to obtain a complete and formal conceptual model for an Information System. Such a model is necessary in order to support human understanding of a system, process optimization, and automated execution support.

Modeling information systems mostly has focused on the data related features of Information Systems. Data are analyzed in order to obtain a conceptual database scheme; sometimes also dataflow is captured [Shlaer and Mellor 1988] [Yourdan 1989]. Modeling system dynamics, i.e. the specification of processes and their control flow, has been neglected fairly or is too much theory driven and therefore not practical for real world problem modeling (e.g. Petri-Net approaches). In this paper we introduce an approach for process modeling that is tailored to model business processes in office environments. Since we are specifically looking into the area of Cooperative Information Systems, processes are supposed to support collaborative work settings.

Cooperative Information Systems are characterized by the following main challenges which will be discussed in the following:

- integration of different process categories
- response to continuous change requests
- incorporation of new features (e.g. new control constructs)
In Cooperative Information Systems not only well-specified processes are relevant which are executed according to strictly defined rules but also loosely defined activities have to be performed which merely follow vague guidelines. The challenge is to cope with these two process categories simultaneously because they exist concurrently in office environments. For example, the travel claim reimbursement process is a very strictly defined process: people have to act precisely according to exactly defined rules [Jablonski et al. 1991]. Nevertheless, one step within that process might deal with a negotiation that has to clarify whether certain receipts will be accepted. This latter kind of process can be characterized as conversational teamwork [Kirsche et al. 1994]. A user-friendly system solution should support both categories of processes: the users do not have to switch execution environments in order to execute them.

Current application systems are characterized by rapidly changing business processes. Continuously, business processes have to be redesigned in order to cope with changing market requirements. Change management must be supported by a model which accurately describes a business process and which uniquely presents each feature of a business process. Having fulfilled these conditions process maintenance is supported optimally.

A third characteristic and challenge for Cooperative Information Systems stems from the fact that models of application systems are continuously getting more complex. Therefore, new modeling constructs are requested that simplify models and make them more handy and readable. For example, new control flow constructs (macros) are demanded which describe control flow in a very compact manner.

The dynamics of Cooperative Information Systems which is reflected by the three issues discussed above requires a modular process model. Meyer [Meyer 1988] states that modularity (of software) is the prerequisite for the so-called external quality factors correctness, robustness, extendibility, reusability, compatibility, efficiency, portability, verifiability, integrity, and ease of use. We see easily that robustness, extendibility, and reusability are requirements directly derived from the preceding discussion of Cooperative Information Systems. It goes without mentioning that factors like verifiability, integrity, and ease of use are mandatory for all modeling approaches.

This paper introduces an approach to process modeling which is based on a modular process model; it is called MOBILE2. This model separates elements of an application system into aspects/perspectives like the functional, the behavioral, the organizational, and the informational. Thus, modularity of a model for an application system is obtained which alleviates change management and also allows to customize and extend aspects individually such that they meet the requirements of certain problem areas. Especially the extendibility of our approach allows to model an application system completely since application-specific aspects can be added whenever they are needed.

The main advantage of our approach is that the principle of modularity is sustained consistently to guarantee the advantages of modular software mentioned above. We have not found an approach to process modeling for Workflow Management that supports this feature with the same consistency (cf. Section 8).

The work presented in this paper is part of a comprehensive project where the link between Business Process (Re-)Engineering and Workflow Management (WFM) is investigated. WFM systems (WFMSs) are considered to provide the execution infrastructure for business process reengineering; they identify a new platform for process execution in distributed environments. This paper deals with artefacts which are relevant on the level of business process modeling. It

---

$^{2}$ MOBILE is the name of the new version of the ABS workflow model which has been described in former papers.
is not intended to talk pure logic, although many artefacts introduced later look very logic oriented (cf. Section 4); nonetheless, profound logic is needed for the final implementation of our approach (e.g. [Klein 1991]). Therefore the criteria our approach must be measured against are not logic driven (e.g. completeness) but modeling issues like ease of use, reusability, tailorability, coherence.

Before we are going to introduce our model with an example (Section 2) we have to clarify the term process which we have used extensively up to this point. Because process is a widely spread term everybody interprets it differently. We adopt for the discussions in this paper the definition from [Humphrey and Feller 1992]. There, processes are defined as sets of partially ordered steps intended to reach a goal. Processes are composed of process elements. The most fine-grained, atomic process element is a process step. According to [Curtis et al. 1992] processes are characterized by the following aspects/perspectives:

<table>
<thead>
<tr>
<th>Perspective/Aspect</th>
<th>Characterization</th>
</tr>
</thead>
<tbody>
<tr>
<td>functional</td>
<td>What processes are performed?</td>
</tr>
<tr>
<td>behavioral</td>
<td>When are processes performed?</td>
</tr>
<tr>
<td>organizational</td>
<td>Who performs processes?</td>
</tr>
<tr>
<td>informational</td>
<td>What information is produced and consumed by processes?</td>
</tr>
</tbody>
</table>

Sections 3, 4, 5 and 6 are discussing the aspects introduced above in the context of the MOBILE workflow model. Section 7 presents the architecture of the MOBILE Workflow Management System which deploys the model introduced in this paper. Section 8 discusses related work. Section 9 concludes this paper.

2. Introductory Example

The perspectives of a process introduced in Section 1 define how people involved in a process cooperate. The functional aspect declares what has to be done, the behavioral aspect says when it has to be done, the organizational aspect denotes who has to do it, and the informational aspect describes which information is exchanged between cooperating agents.

In order to give an idea of how the aspects introduced above act in combination, we discuss the example depicted in Figure 1: The process 'process_travel_claim' has to be executed. It is refined by the process steps 'submit_travel_claim' and 'approve_travel_claim' and the process element 'reimburse_client'. Process steps are implemented by applications, i.e. executable pieces of code like programs; process elements are implemented by process elements or process steps. Therefore, the process steps 'submit_travel_claim' and 'approve_travel_claim' are implemented by the program system 'Travel Claim System'. In order not to complicate the figure, we have not shown further refinements of process element 'reimburse_client'. Processes, process elements and process steps constitute the functional aspect.

The course of execution is defined by control flow constructs between process elements/steps. For instance, in the example of Figure 1 all three components of process 'process_travel_claim' have to be executed sequentially. More control flow constructs will be discussed in connection with the behavioral aspect of processes in Section 4.

The organizational aspect of a process is described by agents who are attached to processes, process elements and process steps. These agents are supposed to execute the particular piece
they are associated with. In the example of Figure 1 a manager has to perform the process step ‘approve_travel_claim’ while a financial clerk must execute the process element ‘reimburse_client’.

The fourth aspect introduced in Section 1, the informational aspect, is defined by local data of a process, process element, or process step and by data flow connectors between a process and its components or between the subcomponents of a process (element). In Figure 1, the identification of the person posting a travel claim (Client_Id) and an identification of the trip (Trip_Id) that person wants to be reimbursed for is input for the process ‘process_travel_claim’. ‘Trip_Id’ is passed to all three subcomponents of the process; ‘Client_Id’ is only transferred to the process step ‘submit_travel_claim’. In process step ‘submit_travel_claim’, a travel claim will be defined whose identification (TC_Id) will be passed to the remaining steps of the process through the local data variable ‘TC_Id’. Note that in Figure 1 the informational aspect is simplified in order to convey the principle idea clearly.

Before the aspects introduced with the example are discussed broadly we want to give an overview on how the execution of a process is organized. When a process is initiated, its input parameters are transferred to the subcomponents which have to be executed first according to the precedence structure defined among them. When one or more process elements/steps are determined to be executed next, the corresponding agents are resolved. They are notified about work to do, i.e. the process elements/steps are executed. Process elements are performed by determining its components that have to be executed firstly, process steps are executed by calling the applications associated with them.

3. The Functional Perspective

3.1 Workflows

If processes, process elements, and process steps are distinguished they will most probably be modeled differently. Thus, reusability is hardly to support. It is just not possible to reuse a process within another process without modifying its description; concretely, it has to be converted into a process element. To make reusability possible, we map either processes, process elements, and process steps to workflows. Thus, only one concept is needed to describe all three process-related elements. Each artefact can now be reused in an arbitrary context. Consequently, a workflow is a recursive structure, i.e. a workflow may consist of further workflows.
Figure 1 shows the workflow 'process_travel_claim'; it consists of three so-called subworkflows, 'submit_travel_claim', 'approve_travel_claim', and 'reimburse_client'; 'process_travel_claim' is the superworkflow of the latter three subworkflows. Although not shown in the figure, the subworkflow 'reimburse_client' is composed of further subworkflows like 'return_approved_travel_claim_to_client' and 'transfer_money'. An outermost workflow, i.e. a workflow that does not have any superworkflows, is called top-level workflow. Workflows containing other workflows are called composite workflows. Workflows not composed of other workflows are called elementary.

Applications are referenced by elementary workflows. They implement the functionality of an elementary workflow. Applications consist of programs, transactional steps of TP-monitors, command procedures, or any other executable piece of code together with a so-called wrapper which makes these pieces of code accessible for the WFMS. Thus, from a modeling point of view, applications are regarded as a very specialized form of workflow, i.e. among other things they show the same interface as workflows and therefore can be dealt with equally.

Two main groups of applications can be distinguished: legacy applications and non-legacy applications. Legacy applications are characterized by the fact that they cannot be modified. They are inherited from past system states and have to be treated as they are. Non-legacy applications are easier to cope with. They can be tailored to the specific needs of WFMS if they have to be incorporated into a workflow.

In this paper we want to discuss both workflows and applications under the functional aspect. However, in other publications we separate out the technological aspect which covers the integration of applications into workflows [Bussler and Jablonski 1994a] [Bussler and Jablonski 1994b].

Like procedures in programming languages, workflows are black boxes; only the signature determined by the type of a workflow and the types of its in- and out-parameters are externalized. This feature supports reusability of workflows which is an absolute requirement for modern software development. The objective is to populate public libraries of workflows such that maximum synthesis can be reached for workflow design [Tsichritzis et al. 1988].

Two main classes of workflows can be distinguished:

Prescriptive workflows. Eligible (sub-)workflow instances (and also the precedence structure between them) are known a priori.

Descriptive workflows. Instances of participating (sub-)workflows are not known beforehand but are determined during processing. However, their types are known a priori.

Conventional office procedures (e.g. 'process_travel_claim') are examples for prescriptive workflows: each step of execution is very well known in advance. Design tasks are examples for descriptive workflows; although workflow types needed to describe a certain problem are known, the concrete way of processing them, i.e. the adequate workflow instances, is not known beforehand.

Generally, a workflow type is defined as follows:

WORKFLOW TYPE workflow-name (formal-workflow-parameters)
  workflow-parameter-conditions
  workflow-body

formal-workflow-parameters are the input and output parameters of the workflow type; workflow-parameter-conditions describe conditions that have to be fulfilled by the actual

---

3Non-terminal symbols are written with small letters; terminal symbols are written with capital letters. In order to increase the readability of the grammar of the workflow language, we omit colons, semi-colons, and periods.
workflows. For instance, they describe that some of the parameters are mandatory, some other parameters are optional. workflow-body represents the kernel of a workflow. According to the four aspects we discuss in this paper workflow-body is classified into four sections:

\[
\text{workflow-body} ::= \text{functional-aspect} \\
\text{behavioral-aspect} \\
\text{organizational-aspect} \\
\text{informational-aspect}
\]

The various parts of workflow-body will be refined stepwise in the following sections.

### 3.2 Prescriptive Workflows

Prescriptive workflows are specified by declaring the subworkflow and application instances that participate in the workflow. This is done in functional-aspect:

\[
\text{functional-aspect} ::= \{\text{SUBWORKFLOWS} \\
\text{workflows} \} \\
\text{functional-aspect} ::= \{\text{APPLICATIONS} \\
\text{applications} \}
\]

<workflow-type> defines the type of subworkflows which are specified as <workflow-variable>; obviously multiple subworkflow instances of the same type can be declared for a workflow. Each subworkflow instance is restricted by constraints (CONSTRAINTS). Typical constraints are time limits for subworkflows that define how long the execution of a subworkflow may take at most, or constraints that allow certain subworkflows not to start at a specific time of the day.

Although not very relevant for this paper we want to show how applications are defined for elementary workflows:

\[
\text{technological-aspect} ::= \{\text{APPLICATIONS} \\
\text{applications} \}
\]

In this case, technological-aspect must also be a subsection of workflow-body.

For the example of Figure 1 the definition of the workflow type ‘process_travel_claim’ with respect to the functional aspect is shown in the following:

\[
\text{WORKFLOW TYPE process_travel_claim ()} \\
\text{SUBWORKFLOWS} \\
\text{submit_travel_claim: submit_travel_claim;} \\
\text{approve_travel_claim: approve_travel_claim;} \\
\text{reimburse_client: reimburse_client;}
\]

It is interesting to notice that process_travel_claim is a new workflow type that has to be specified; submit_travel_claim, approve_travel_claim, and reimburse_client are already existing workflow types that are referenced in the new workflow specification. In some commercial WFMSs unfortunately it is possible to define a new workflow type within the definition of another new workflow type. From the perspective of software engineering this is not advisable.
3.3 Descriptive Workflows

Some sort of processes (e.g. design) are characterized by knowing what to do, but not knowing exactly when and how to do it in detail. Descriptive workflows are supposed to cope with this problem. Two examples clarify the need for descriptive workflows: Negotiations can be modeled by the workflow types 'suggesting', 'asking', 'answering', 'agreeing', 'disagreeing' [Kirsche 1993]. For a brainstorming session, the workflow types 'articulate_idea' and 'articulate_assumption' are essential. Although these workflow types might be sufficient to set up the framework for negotiations and brainstorming sessions, respectively, it is not known whether and how many instances will be needed eventually in order to model a concrete scenario. In well going negotiations there is no need for workflows of type 'disagreeing'; in cumbersome negotiations, unfortunately, the workflow type 'agreeing' is not used at all (cf. Section 4.3). Descriptive workflows are therefore characterized by not knowing the workflow instances needed eventually, but being able to perceive the types of instances needed principally (cf. example 'negotiation' above).

Because the modeling approach discussed above is rather vague and fuzzy one might doubt it in principal and might argue to neglect it completely. However, there are two important arguments in favor of our approach: consistency and tracking. If the set of workflow types is given principally, consistency is sustained better because workflow types which should not be used at all can be excluded. Also tracking of a workflow execution is easier when the workflow types that can be applied are known beforehand.

Intensional modeling [Beech 1988] promises to be a suitable approach for descriptive workflows. In contrast, prescriptive workflows are modeled extensionally: all extensions, i.e. instances, are defined a priori. Intensional modeling provides an intensional framework for workflow execution: workflow types that might be needed for solving a specific problem are declared together with a statement about the goal and purpose of a workflow. Besides, mandatory control flow restrictions are also specified (cf. Section 4.1.3). In [Tsichritzis et al. 1988] the logical foundation for descriptive workflows are given. Figure 2 demonstrates how descriptive workflows are specified. The five subworkflows of workflow 'negotiate' are only specified on the type level. Multiple instances can be created during its execution. The control constructs defined on the type level must be obeyed by all instances.

![Figure 2: Specification of a Descriptive Workflow](image)

To specify descriptive workflows (and applications) functional-aspect (and technological-aspect) has to be extended by the following sections:

---

4The intensional framework for descriptive workflow modeling is defined by two 'rules': the instantiation 'rule' says that instances can be derived from declared workflow types; the control flow 'rule' says that instances have to be put in the precedence structure defined by the control flow constructs.
For each subworkflow (application) a description of its need and purpose is attached (description). It supports the process of finding the right workflow instances to perform a certain function. In [Käfer 1991] and [Tschirizs et al. 1988] approaches to specify intensional artefacts like goals (described via mandatory features) are shown. In the case of descriptive workflows (applications), constraints must be applied to all instances that are created.

As an example we specify the workflow type 'negotiate' described in Figure 9 (Section 4.3):

```
WORKFLOW TYPE negotiate()

SUBWORKFLOWS
suggest;
commit;
agree;
ask;
answer;
```

4. The Behavioral Perspective

The interdependencies and interrelationships between subworkflows, i.e. its behavioral aspect, is characterized through control flow (synonymously: flow control). Flow control determines when workflows are performed. Flow control is specified in terms of execution rules determining the sequence to perform workflows. Note that control flow is only specified for subworkflows of a common superworkflow. However, we use the term workflow instead of subworkflow in the following discussion whenever its interpretation is unique.

Flow control determines when resources executing workflows are cooperating and collaborating. There is a wide range for cooperation and collaboration respectively. Prescriptive and descriptive control types can be distinguished. Tightly fixed forms of control prescriptively define the way workflows are executed, i.e. how cooperation and collaboration takes place; loosely described forms of control merely establish an ordering framework for workflows. This framework opens many degrees of freedom for workflow processing. The actual execution of workflows must comply with this framework.

In Section 4.1, we introduce meaning, graphic and textual representation of flow control statements. In Section 4.2 the semantics of flow control statements is discussed. Section 4.3 presents a comprehensive example.

4.1 Control Flow Types

Flow control is specified in behavioral-aspect of a workflow definition. The concrete syntax is given below:
behavioral-aspect ::= \[\text{CONTROL FLOW } \{\text{control-expression}\}\]
control-expression ::= control-predicate \{control-term, control-expression\}
| control-term
control-predicate ::= control-construct \{control-name((\{parameter\}))\}
control-term ::= \{control-expression\}
| workflow-variable
| \{(\{workflow-variable\})\}
control-construct ::= -> | α | "||" | << | >> | => | while_do
| repeat_until | p | ...

Two major features can be derived from the above syntax: firstly, multiple courses of processing may be specified by defining multiple control flows \{(control-expression)\}. These control flows are performed independently from each other. Multiple control expressions are combined by a logical conjunction.

Secondly, each control construct (which will be introduced in the remainder of this subsection) is associated with a control function \{control-name\} which defines constraints for the control construct. For instance, a typical constraint enforces that a control construct is executed within a time limit. In case of sequential execution of workflows B and C this means that workflow C starts execution within the specified time limit after B has terminated; otherwise execution is regarded as having failed.

We distinguish between prescriptive and descriptive flow control. It is not the case that prescriptive (descriptive) flow control is only applicable for prescriptive (descriptive) workflows (cf. Section 3) and vice versa. However, both types of workflows are target for both types of flow control.

4.1.1 Prescriptive Flow Control
There are three basic forms of prescriptive flow control, serial execution, alternative execution, and parallel execution. They will be explained in this subsection.

Serial Execution
Serial Execution is a very restrictive form of ordering subworkflows. Exclusively one course of processing is possible, namely strictly sequential execution.

![Figure 3: Types of Prescriptive Workflows](image)

Because the ‘bubbles and arcs’ notation is very common and popular for the description of workflows we are presenting sample workflows in terms of this notation. Figure 3a shows an example of a workflow A which enforces strictly sequential processing of its subworkflows B, C; the execution sequence [BC] is mandatory. We use the notation ‘[...]’ to denote the history of execution; this notation is similar to the notation introduced in [Bernstein et al. 1987] used for describing execution histories for transactions in database systems.
Below, the script version of the example in Figure 3a is presented:

**WORKFLOW TYPE A**

**CONTROL FLOW**

\[ \rightarrow ((B, C)) \]

For convenience, the following language transformation is valid (B, C, D are workflows), because sequencing is associative:

\[ \rightarrow ((B, \rightarrow (C, D))) = \rightarrow (((B, C), D)) = \rightarrow (B, C, D) \]

We also allow to generalize the notation. Instead of B and C, multiple workflows can be specified in brackets to express multiple sequential relationships, i.e. multiple independent control flows. The following transformation is supported:

\[ \rightarrow ((B_1, \ldots, B_b), (C_1, \ldots, C_c)) \]

\[ \rightarrow ((B_1, \ldots, B_b), C_1) \wedge \ldots \wedge \rightarrow ((B_1, \ldots, B_b), C_c) \]

\[ \rightarrow (B_1, C_1) \wedge \ldots \wedge \rightarrow (B_b, C_1) \wedge \ldots \rightarrow (B_1, C_c) \wedge \ldots \rightarrow (B_b, C_c) \]

**Alternative Execution**

The order of processing is less strictly determined for workflows in which alternative execution branches can be specified; alternating courses are mutually exclusive; they have to be joined eventually. Conditions attached to alternative branches must be logically disjoint like in 'if-then-else' constructs of programming languages.

Figure 3b depicts an example for workflows with alternative courses of execution. Either [B] or [C] are executed depending on whether condition \( \text{cond}() \) is evaluated to true or to false. \( \text{cond}() \) is a problem specific condition (\( \alpha \equiv \text{alternative} \)).

**WORKFLOW TYPE A**

\[ \ldots \]

**CONTROL FLOW**

\[ \alpha \text{ cond}() (B, C) \]

Looping, i.e. iterative processing of (sequences of) workflows, is enabled through the introduction of alternative execution. Of course, a construct like the \textit{goto} construct known from programming languages is necessary as well. However, we do not intend to introduce this construct at the user interface but use it internally only. For the following two popular forms of loops self-explanatory control constructs are introduced:

\[ \text{while}_\text{do} \text{ cond}() (B, \Delta) \]

\[ \text{repeat}_\text{until} \text{ cond}() (B, \Delta) \]

\( \Delta \) represents the \textit{empty workflow}. We can also support an extended notation of the control construct for alternative execution to simplify the specification of complex control flows:

\[ \alpha \text{ cond}() ((B_1, \ldots, B_b), (C_1, \ldots, C_c)) \]

\[ \alpha \text{ cond}() ((B_1, \ldots, B_b), C_1) \wedge \ldots \wedge \alpha \text{ cond}() ((B_1, \ldots, B_b), C_c) \]

\[ \alpha \text{ cond}() (B_1, C_1) \wedge \ldots \wedge \alpha \text{ cond}() (B_b, C_1) \wedge \ldots \wedge \alpha \text{ cond}() (B_1, C_c) \wedge \ldots \wedge \alpha \text{ cond}() (B_b, C_c) \]

**Parallel Execution**

As a third form of prescriptive execution control parallel execution (\( \parallel \)) is introduced. In this case workflows can be activated concurrently.

Figure 3c shows the following example: B and C as the only subworkflows of composite workflow A have to be executed in parallel [B](C).

**WORKFLOW TYPE A**

\[ \ldots \]

**CONTROL FLOW**

\[ \parallel (B, C) \]
Because parallel execution is associative, the following transformation is valid (B, C, D are workflows):
\[ \langle (B, \langle (C, D) \rangle \rangle = \langle (\langle B, C \rangle, D) \rangle = \langle (B, C, D) \rangle \]

A generalized notation for parallel execution is defined in the following:
\[ \langle \langle B_1, \ldots, B_b \rangle, \langle C_1, \ldots, C_c \rangle \rangle \leftrightarrow \langle \langle B_1, \ldots, B_b \rangle, \langle C_1, \ldots, C_c \rangle \rangle \]
\[ \langle \langle B_1, C_1 \rangle \wedge \ldots \wedge \langle \langle B_b, C_1 \rangle \wedge \ldots \wedge \langle \langle B_1, C_c \rangle \wedge \ldots \wedge \langle \langle B_b, C_c \rangle \rangle \]

4.1.2 Descriptive Flow Control

For the specification of descriptive flow control two different types of conditions are introduced, *temporal* and *existence conditions*. As elaborated in [Klein 1991] and approved in [Attie et al. 1993] these two types of dependencies are adequate and necessary to describe almost all usual protocols for the synchronization of computing agents. In our case these agents are materialized as workflows.

In contrast to prescriptive flow control where each specification results in a unique and concrete template for processing, descriptive flow control merely characterizes equivalence classes of processing. Temporal conditions denote a temporal relationship between workflows; existence conditions describe mandatory existential dependencies.

**Temporal Condition ‘Deadline’**

A first type of temporal flow control has the form

\[
\text{WORKFLOW $\text{TYPE A}$} \\
\ldots \\
\text{CONTROL FLOW} \\
\langle \langle B, C \rangle \rangle
\]

It is called *deadline* (\(<<\)) since the execution of workflow \(B\) is limited by the occurrence of workflow \(C\) with respect to time. The above statement expresses that \(B\) has to be executed before \(C\), if it is processed at all. It is forbidden that \(B\) and \(C\) are executed in parallel. Therefore, the following rules describe permissible executions:

- If \(C\) hasn't started yet, \(B\) can be executed.
- If \(B\) hasn't started yet or has already terminated, \(C\) can be executed.
- \(B\) need not to be executed at all.
- \(C\) need not to be executed at all.

Thus, the following sequences of execution are valid: \([\] , \([B]\) , \([C]\) , \([BC]\) . The empty sequence \([\] \) is allowed since the deadline rule does not demand to execute participating workflows at all.

Recall that the modeling construct *deadline* is used to express dependencies between workflows. It must not be mixed up with the *temporal constraint* bearing the same name mostly. This latter construct denotes one specific point in time which delimits the occurrence of events in general, execution of workflows in our special case. It has to be regarded as an artefact which belongs to execution policies (cf. Section 5). A workflow which is not executed within such a temporal deadline is regarded as having failed. This is totally different to our language construct which does not indicate whether workflow execution failed: here, some workflows are just not executed because of specific relationships to other workflows. Those workflows are called to be disabled. Being disabled is different from having failed (cf. Section 4.2).

Each specification of descriptive control therefore produces a class of equivalent and valid processing scenarios. In Figure 4 either the graphical notation for deadline and also equivalent sequences of execution which fulfill the deadline semantics are shown. These alternative execution sequences form a so-called *set of valid execution sequences (SVES)* of the deadline condition. Each valid sequence only shows a piece of the semantics expressed by the deadline
condition. All sequences together show the complete semantics of this temporal condition. Figure 4 demonstrates that if only sequences, alternatives, and parallelism are available the declaration of workflows would drastically get cumbersome. This observation also applies to the forthcoming discussions on delay and existence conditions.

The generalization of the deadline construct is defined next:

\[<<(\{B_1 \ldots B_p\}, \{C_1 \ldots C_q\})>>\] \[<<(\{B_1 \ldots B_p\}, C_1) \wedge \ldots \wedge <<(\{B_1 \ldots B_p\}, C_q)\] \[<<(B_1, C_1) \wedge \ldots \wedge <<(B_p, C_1) \wedge \ldots \wedge <<(B_p, C_q)\] A \(B_i\) can execute before an arbitrary \(C_j\) starts execution.

A sample example from the academic area explains the use of the deadline control flow construct. When somebody wants to do a doctorate (s)he has to apply for admittance. (S)he can withdraw his application \(B\) only if it was not rejected from the examining board so far \(C_1\) or (s)he has already begun his/her oral examination \(C_2\):

\[<<(B, C_1 C_2)>>\] \[<<(B, C_1) \wedge <<(B, C_2)\] The meaning of the execution sequences \([\cdot], [C_1], [C_2]\) is obvious. We also see that the sequences \([C_1B]\) and \([C_2B]\) must be prevented. Thus, an adequate control function (cf. Section 4.1) must additionally be defined that \(B\) is not executed after either \(C_1\) or \(C_2\) was already executed.

**Temporal Condition ‘Delay’**

Delay (\(\gg\)) is a second form of temporal flow control:

**Workflow Type A**

\[
\begin{align*}
\text{CONTROL FLOW} & \gg (C, B) \\
\end{align*}
\]

Workflow \(C\) must be delayed until workflow \(B\) has finished or will never be executed. This means that

- \(C\) can be processed only if
  - either \(B\) was executed already and has finished
  - or \(B\) decides never to execute at all.
- \(B\) need not to be executed at all.
- \(C\) need not to be executed at all.

In order to know whether workflow \(B\) will be processed a specific variable has to be inquired (see Section 4.2 for details). The following sequences of execution are valid: \([\cdot], [C], [B], [BC]\).

In Figure 5 either the notation for delays and the SVES for the delay condition are shown.
Although the SVESs for deadline and delay conditions look equal, their semantics, i.e. the reasons for coming into existence, are significantly different (cf. Section 4.2). In case 'delay' B can be executed as long as an indicating condition holds. This condition is dependent on application oriented matters but not directly on workflow C. In case 'deadline' the situation is quite different: the occurrence of C determines and delimits abruptly the chance of B to be executed.

As an application of the delay condition the start procedure of an airplane is discussed (workflow A). Workflow C stands for taxiing and take off; workflow B models the boarding procedure. If boarding has terminated, take off can take place ([BC]). If boarding cannot take place at all (for example because it is a non-intended intermediate stop), take off can occur without having to wait for the end of boarding [C]; in this case boarding was disabled before. In case the flight has to be canceled because of bad whether conditions, take off cannot be done, although boarding might already have finished ([B]). If the whole flight is canceled before boarding started yet, neither B nor C is executed (|).

Next the generalized notation of the delay control flow construct is derived.

\[ (B_1 \ldots B_b)(C_1 \ldots C_c) \Rightarrow (B_1 \ldots B_b)(C_1) \land \ldots \land (B_1 \ldots B_b)(C_c) \]

All \( C_i \) are delayed by a \( B_i \). Only if each \( B_i \) has executed or has promised not to execute at all, a \( C_j \) can be performed. 

Existence Condition

Existence conditions \((=>)\) are formulated as follows:

**WORKFLOW TYPE A**

```
  CONTROL FLOW
  =>(B; C)
```

The execution of \( C \) is enforced when \( B \) was executed. Thus if \( C \) is disabled and cannot be executed any more, \( B \) must be disabled, too. If \( B \) will not be executed, the execution of \( C \) is optional. [BC], [CB], [ClIB], [C], and | are permissible sequences of execution. Figure 6 depicts the graphic notation for existence conditions as well as the SVES.

**Figure 6: Condition Type ‘Existence’**

As an example, workflow A represents a scenario dealing with the demonstration of a prototype. Workflow C comprises everything that has to be done in order to set up the demonstration. Workflow B stands for all matters necessary to announce a demo. Principally it does not matter whether an announcement takes place firstly or a demo is set up firstly ([BC], [CB], [ClIB]). But if it becomes obvious that the demo cannot be set up, no announcement should be made (|). Even if no announcement is made a demo might be prepared ([C]). The example shows how the outcome of an existence condition is dependent on the ability to execute a workflow C. Only the knowledge that C can be executed allows to initiate execution of workflow B. This knowledge has to be reflected in the system.

Next, the generalization of the existence control flow construct is shown:
A \( B_i \) can only execute if all \( C_j \) can be executed.

**Examples:**
The six primitives introduced above can also be nested by replacing a workflow placeholder with another control construct. In the example

\[
\Rightarrow ((B_1 \ldots B_b), \ (C_1 \ldots C_c)) \Rightarrow
\]

\[
\Rightarrow ((B_1 \ldots B_b), \ C_1) \wedge \ldots \wedge \Rightarrow ((B_1 \ldots B_b), \ C_c) \Rightarrow
\]

\[
\Rightarrow (B_1, C_1) \wedge \ldots \wedge \Rightarrow (B_b, C_1) \wedge \ldots \wedge \Rightarrow (B_b, C_c)
\]

\( B_i \) can only execute if all \( C_j \) can be executed.

B is substituted with a deadline dependency between D and E, and C is substituted with parallel execution of F and G:

\[
\Rightarrow (<< (D; E)); \ || (F; G))
\]

The execution semantics is as follows: after the deadline construct is performed, F and G can execute in parallel. Note that either workflow D, E, F, and G are all subworkflows on the same level.

In the example

\[
\Rightarrow (B; C)
\]

B and C as the placeholders of the delay primitive are substituted with \( \Rightarrow (D, E) \) and \( \Rightarrow (F; G) \), respectively:

\[
\Rightarrow (\Rightarrow (D; E); \Rightarrow (F; G))
\]

Before D (and afterwards E) can execute, the existence relationship between F and G must be performed, i.e. either G, F and G, or none of the two workflows have been performed.

The set of prescriptive and descriptive control flow constructs introduced so far make up a decent basis for the definition of additional problem-specific control flow constructs (*macros*; cf. Section 4.1.4).

### 4.1.3 Control Flow Specification for Descriptive Workflows

For descriptive workflows instead of workflow instances, workflow types are referenced in control constructs. Instances of these types can be generated in order to implement the functionality of a workflow. When control is defined between the workflow types \( t_m \) and \( t_n \) as

\[
\Rightarrow (t_m, t_n)
\]

instances of these types have to obey the control order specified on the type level. For example instances \( t_m,i \) of workflow type \( t_m \) and \( t_n,i \) of workflow type \( t_n \) have to obey the control flow specification

\[
\Rightarrow (t_m,i, t_n,i)
\]

As another example we model a fragment of the sample workflow depicted in Figure 9. Generally, an existence dependency is defined between the subworkflows 'ask' and 'answer' of superworkflow 'negotiate':

\[
\text{WORKFLOW TYPE} \ negotiate
\]

\[
\text{CONTROL FLOW}
\]

\[
\Rightarrow (\text{ask, answer});
\]

Whenever questions are asked or answers are given they have to obey the existence dependencies which is specified above.

Control specification on the type level together with descriptive control types allow to specify unstructured forms of interactions (e.g. negotiation). [Kirsche et al. 1994] conveys the global idea of how this can be accomplished.
4.1.4 Macros

After having provided a basic set of primitives for control flow specification, most kinds of problem specific control types can be formulated. We want to present some examples. As a first example the so-called skip macro is introduced. It means that a workflow has to be skipped when a certain condition holds.

\[ \alpha \text{cond}() (B, \Delta) \]

defines the semantics of the ‘skip’ macro.

The control flow construct for parallel execution motivates another interesting language construct which can be regarded as a special case of parallel execution. Let’s assume that a workflow B has to be executed multiple times, however not necessarily sequentially but concurrently. The former way of execution can be achieved with a loop; for the latter way of execution a new construct must be introduced. An appropriate notation would look like:

\[ \Pi (B, B, B, \ldots) \]

B has to be executed as often as a problem specific condition prescribes; but this hardly can be expressed by using the dot notation. A language construct for this form of execution is called repetition (p). A more appropriate notation for repetition is

\[ p \text{cond}() (B, B) \]

It depends on the implementation of flow control whether an upper and/or lower bound for repetition has to be specified. It is also implementation dependent when instances of workflow B can be generated. Restrictive types of implementation would only allow to generate instances at the very beginning of the execution of a repetition; dynamic implementations would allow to generate new instances while other are already in execution. \text{cond}() generally denotes a condition function, controlling how often B is going to be instantiated. Besides other things, \text{cond}() can be related to

- the number of instances that can be created,
- the time frame which delimits the creation of instances.

An example justifies the meaning of the replication construct: For an assembly different parts have to be ordered. It is hard to determine a priori how many different parts are needed because either each assembly is different and the stocks vary as well. For each part to be ordered the order_part() workflow must be called. If this would be done within a loop, all parts are ordered sequentially which unnecessarily extends latencies. This is a waste of time and also might bother the personnel since they block each other from ordering parts. It is obvious how easily this situation can be modeled with the repetition construct in a natural way and how execution time can be saved.

More application-specific control flow constructs can be built with the basic set of control flow constructs introduced above. For example, ‘n out of m’ which means that n workflows out of m workflows have to be executed, is another interesting control flow construct. ‘Execute n workflows in an arbitrary order but sequentially’ is also a very powerful control flow construct. Note, that the main advantage of macros is the simplification of a model of an application system, since application-specific, compact control flow constructs can be used which shall reduce the modeling effort drastically.

4.2 Semantics of Control Flow Types

4.2.1 State Transition Diagrams for Workflows

In order to describe the semantics of the execution model for workflows we use state transition diagrams. The following states for workflow execution have to be introduced (A is a workflow the states listed below are associated with):
executed
Workflow A is executed. Note that execution happens atomically, has no timely extension and cannot fail. (This is only needed for the following simplified discussion of the model.)
disabled
Workflow A is not permitted to be executed.
blocked
Workflow A is currently not executable.
enabled
Workflow A is ready to be executed.

We assume that each workflow is in one of the four states enabled, disabled, blocked, executed (cf. Figure 7). By issuing the operation enable() a state transition from blocked to enabled happens. disable() sets the state of a workflow from either enabled or blocked to disabled. execute() transforms the state of a workflow from enabled to executed. blocked() sets a workflow from state enabled back to state blocked. The above mentioned operations are the only permissible ones and only applicable in the situations described below. Setting the state of a workflow to executed causes an instantiation of the workflow. disabled and executed are final states.

![Figure 7: State Transition Diagram for Workflows (basic version)](image)

Notice that the above introduced states are sufficient for showing the basic semantics of the model; for explaining implementation issues - the timely extension of workflow execution has to be taken into consideration - we have to refine the state executed as well as the operation execute(). The latter operation must be split up into operations start() and finish() at least; it is recommended to add operations like pause() and resume() in order to achieve a more handy system. The introduction of these new operations also requires the introduction of new states: started, finished, paused. Figure 8 depicts the extended state transition diagram for workflows. When the state finished is introduced executed is no longer a final state but finished replaces it.

![Figure 8: State Transition Diagram for Workflows (extended version)](image)

Problem specifically more operations can be added. Also, error recovery has to be reflected in a more complete version of a state transition diagram for workflows.
4.2.2 Semantics Specification

To illustrate the semantics of the control constructs introduced in Section 4.1 we are going to show the eligible state transitions which can occur during processing the control constructs. Each semantics diagram consists of three parts: In the middle part workflows and their states / state transitions are depicted. At the top of a diagram user actions are shown while at the bottom part system actions are described. If no user action is involved, system actions occur instantaneously. Otherwise, they are triggered by the user action(s) initiated in the same time slot. Time proceeds from left to right. In order to keep the diagrams simple and readable, alternative execution sequences are depicted in separate diagrams.

In the following, we assume a compound workflow $A$ that consists of subworkflows $B$ and $C$. $B$ and $C$ are going to be executed. In order to simplify the description we use the simplified state diagrams for workflows (cf. Figure 7). When all subworkflows of a workflow are either disabled or executed, i.e. they are in a final state, processing of a workflow terminates.

**Serial Execution:** $\rightarrow (B, C)$

Initially workflow $B$ is enabled. After having started $B$, $C$ becomes enabled automatically. After $C$ is processed, the workflow terminates.

**Alternative Execution:** $\alpha \text{cond}(\cdot) (B, C)$

Firstly, $B$ and $C$ are blocked. Depending on condition $\text{cond}(\cdot)$ either $B$ or $C$ is enabled by the system automatically and will be executed eventually. The other workflow will be disabled automatically.

**Parallel Execution:** $\parallel (B, C)$

Either workflow $B$ and workflow $C$ must execute; however, they can execute independently.
Both workflows B and C can be skipped by disabling them. When workflow C is executed at first, workflow B cannot be executed any more; it will be disabled automatically by the system.
**Delay:**

\[ \rightarrow(C, B) \]

The only but from a pragmatic point of view absolutely significant difference to the deadline construct is, that the delayed workflow cannot be executed without knowing that the delaying workflow has either been disabled or executed. Therefore, first B must be disabled or must have been executed before C can possibly execute.

**Existence:**

\[ \rightarrow(B, C) \]

In case of existence conditions, a new feature of our model has to be introduced. The existence condition says that C has to execute if B is executed. That means: B can only be executed if
either C was executed already or will be executed eventually. The existence condition does not imply any time order on the execution of both workflows. In order to capture this semantics our modeling means have to be extended. We introduce execute'(.) in order to express, that either a workflow has executed or is promised to execute eventually. Therefore, the case 'C executes before B' is justified in the SVES for existence (Figure 6). In this scenario, C has promised to execute eventually after B has finished. If B executes first, then C must be executed eventually. We do not prescribe how this can be achieved; however, an implementation must guarantee it.

4.3 Example

This subsection is to demonstrate a comprehensive example that consists of either prescriptive and descriptive elements. The overall purpose of the example is to set up a meeting (Figure 9); the whole process is initiated by a manager. First a preparing step has to be performed: a secretary is collecting dates about vacant meeting rooms. After that the potential participants of the meeting have to negotiate about date and location. After the participants have agreed upon date and location this room will be reserved by a secretary and the participants will be invited finally. Meeting data (MD) are exchanged between the workflows. They contain information about meeting place, date, and also about the potential participants.

This example shows that either prescriptive workflows - searching for a room and finally reserving it - and pretty loosely structured, descriptive workflows are needed simultaneously in the same scenario. The specification of the step 'negotiate' demonstrates how workflow type declarations are used in order to build the framework for a negotiation. For instance, if somebody is making a commitment, vague suggestion must not follow any more; if a question is asked, an answer must follow. Note, that these statements are valid for pairs of instances (e.g. suggest/commit) that belong to the same topic. Of course, multiple suggest/commit pairs can be created which then are called to talk about different topics. These pairs are independent from each other.

Notice that the "bubbles" used in the node 'negotiate' are representing workflow types and not instances. An potentially arbitrary number of instances can be generated in order to solve the problem of agreeing on date and place for the meeting.

```
(prepue ~ (suggest .. ,)
.......
~ --
M ....
mer avoids
)

Figure 9: Example: Setting up a Meeting
```

This example nicely demonstrates the advantages of our approach. Well-structured and negotiation-oriented types of processes can be modeled in the same framework. Usually two different systems - a WFMS and a Conferencing System - would be required to implement an application scenario as described in Figure 9. Taking into account that both types of processes share the same data and contribute to the same goal, namely to set up a meeting, the integration of both types of systems within an 'extended' WFMS as proposed in this paper is
advantageous. A side effect of this integration is that a user only has to deal with one interface, the interface of the ‘extended’ WFMS, which supports user acceptance.

5. The Organizational Perspective

So far, we have elaborated how a workflow is functionally structured and how workflow execution is controlled. We will investigate organizational issues in this section. Particularly, we tackle the problem of who has to execute a workflow. In this paper only some fundamental issues are discussed; [Bussler 1992], [Bussler and Jablonski 1994a] and [Bussler and Jablonski 1994b] detail organizational issues in the realm of process engineering in a very comprehensive manner.

The following major organizational concepts are introduced: organizations, notification and synchronization, and organizational policies. Altogether they facilitate the association of actors to tasks, i.e. workflows that need to be executed.

5.1 The Organization

The organization is the basic concept for the enactment of the organizational aspect. In principle, it is absolutely workflow-independent and therefore exists also when no WFMS is deployed. It describes every entity type and every entity instance within an enterprise. The organization is built up by the organizational structure and the organizational population.

Because we are not able to anticipate all organizational structures which might be relevant for enterprises, we support the definition of arbitrary organizational structures. For that we propose to use general organizational objects and organizational relationships to describe them.

Two kinds of organizational objects are distinguished, agent types and non-agent types. Typical examples of agent types are humans (e.g. employees); however, also mechanical machines or server processes are regarded as agent types. Non-agent types are characterized by their grouping effect. Departments, divisions, roles, and task-force are examples of non-agent types.

Organizational relationships interrelate organizational objects. Examples of organizational relationships are ‘is_member_of’, ‘plays_role’, ‘is_manager_of’, ‘is_assigned_to’.

Either organizational objects and organizational relationships are characterized by specific properties. Among other things, in the context of workflow execution the capabilities of managers might be of interest: it is important to know whether a particular manager is allowed to approve local or international travels or both. An interesting property of an organizational relationship is the timeframe when it is valid. For example, the assignment to a task-force is only valid as long as the company’s profit is below a certain level.

Finally, we have to fill the organizational structure by defining instances, which are altogether called the organizational population.

The following language constructs are used to specify both the organizational structure and the organizational population.

```
ORGANIZATION organization-name
  [ORGANIZATIONAL OBJECT TYPES {object-type-name}]''
  [ORGANIZATIONAL RELATIONSHIPS {relationship-type-name}]''
  [ORGANIZATIONAL OBJECT INSTANCES
   {object-type-name: {object-instance-name}}]''
  [ORGANIZATIONAL RELATIONSHIP INSTANCES
   {relationship-type-name: {relationship-instance-name}}]''
```
Note, the specification of an organization is not part of a workflow specification. In the following example we define the organizational object type *manager* and declare a list of people who are instances of that object type.

**ORGANIZATION** SampleOrganization

**ORGANIZATIONAL OBJECT TYPES** manager

**ORGANIZATIONAL OBJECT INSTANCES**

manager: Lynn, Bryan, Joseph

To retrieve agents from the organization, so-called *agent-selections* are defined. An agent-selection is a function operating on the organization; it returns active elements of the organization who can perform a certain task. For example, `managers()` is an agent-selection that retrieves all agents that are eligible to play the role of a manager; `group(g)` selects all agents that belong to group g. Composed agent-selections consists of multiple agent-selections that are connected by set operators. For instance,

```
managers() ∩ group(g)
```

selects all managers who work for group g.

The following simplified language construct facilitates the definition of agent-selections:

**AGENT SELECTION** agent-selection-name (formal-agent-selection-parameters)

**RETURNS** (agent-selection-implementation)

It would be beyond the scope of this paper to define exactly *agent-selection-implementation*. We just want to mention that this expression can consist of multiple agent-selections combined by set operators and of program code that retrieves directly agents from the organization repository.

### 5.2 Notification and Synchronization

In order to inform agents about work to do *notification* has to be provided for [Bussler and Jablonski 1994a] [Bussler and Jablonski 1994b]. Notifications contain indications about what to do, why to do it, and how to do it, in order to make the execution context clear to the resource. Notifications are organized in so-called *work-to-do lists*. Each agent is associated with one or multiple work-to-do lists. (S)he might maintain one work-to-do list for each role (s)he is able to play. Whenever an entry appears in a work-to-do list the associated action should be performed. Mostly this means to execute an application or to authorize, i.e. to initialize, a composite workflow.

For different situations, i.e. in the context of different workflows, specialized notifications can be used. For example, it is recommended to use synchronous notification for time critical work. A (simplified) language construct to specify different forms of notification for an agent is given below:

```
NOTIFICATION FOR agent-name
  (notification-name description)+
```

Without going into details, *notification-name* describes a specific form of notification (e.g. e-mail), *description* defines when this form of notification should be applied.

We will see that normally more than one agent is notified about work to do. However, not all work items are supposed to be executed by any agent that knows about them, i.e. by anybody who has received a corresponding entry in her/his work-to-do list. Thus, a specific *synchronization* mechanism must take care that these work items are synchronized, i.e. that the correct number of agents execute a particular piece of work [Bussler and Jablonski 1994a] [Bussler and Jablonski 1994b].

Two examples will shed some light into the issue of synchronization. In the first scenario, multiple members from an accounting department are informed about a particular task (i.e. a workflow execution due). However, only one financial clerk is supposed to perform the work.
When that clerk has decided to do the work, the corresponding work item has to be erased from the work-lists of the other clerks that were notified as well.

5.3 Organizational Policies

We are now able to define organizational policies. An organizational policy relates agent-selection(s), notification(s), and synchronization(s) to a workflow. In this way, agents are determined who are eligible and responsible to execute a workflow. Due to its complexity we omit the complete syntax for organizational policies. We merely want to say that within an organizational policy agent-selections, notification, and synchronization are conditionally assigned to workflows. As an example we partially show how agents of a sales department are determined who are allowed to sign a project contract:

if small_project (budget) then manager(sales_dept) else VP(sales_dept)

If the project is a small one, the manager of the sales department is eligible to sign the contract; otherwise, the corresponding vice president (VP) has to sign it. This example also shows how parameters from the actual context of workflow execution (budget) are used in organizational policies. Besides actual context information also historical information is important for organizational policies. This allows to express that 'a workflow x has to executed by the same agent that executed workflow y ten days ago' for example.

Note, that so far everything we have defined in the context of organizational policies happened externally to the definition of a workflow. This is important out of several reasons [Bussier and Jablonski 1994a]. Better reusability of either workflow definitions and organizational policy definitions, and security are two important issues. Within the specification of a workflow the declaration of organizational policies is allowed. However, organizational policies are independently specified. The Policy Server (cf. Section 7.2) can take this additional information optionally to assign the right organizational policies. The workflow definition can thus be extended by the following language construct:

organizational-aspect ::= [ORGANIZATIONAL POLICIES
                           (policy-name({parameter}'))']

Finally, we want to give some examples of organizational policies, specifically we discuss some versions of agent assignments to workflows. The first one is very primitive and directly assigns agents to a workflow. The particular section of the organizational policies is

get_user(Linda) ∨ get_user(Bryan)

Assigning agents (Linda and Bryan) to a workflow denotes that the workflow should be performed by them. Although feasible in principal, this solution is pretty inflexible. When people (agents) change their status, all assignments to workflows have to be revisited whether they are still valid or not. For example, after an agent left a company, all assignments which directly reference that agent have to be updated. To check assignments for validity is a very time-consuming and cumbersome task and also might cause severe integrity violations.

To overcome the drawbacks stemming from the static assignment of agents, roles are introduced. Roles are attached to workflows expressing that workflows can be executed by agents who are able to play the roles attached. In a specific scenario a workflow has to be executed by a role secretary. All agents currently able to play this role are permitted to perform the workflow. Role assignment is very often resistant to changes in an organization. For example, in case a person qualified as secretary leaves the company, workflow descriptions referring to the secretary role do not have to be changed, because other persons are also able to play the role.
Nevertheless, roles are not sufficient to cope with all situations of agent assignments to tasks. Only organizational policies which also comprise conditional assignments and relate to either historical or actual execution information are powerful enough to express the most important application scenarios.

6. The Informational Perspective

The informational aspect of processes deals with data production and data consumption by workflows, i.e. with data flow between workflows. We also refer to [Jablonski 1992] for a comprehensive discussion of data flow in distributed systems.

In the area of WFM two classes of data are distinguished: control data and production data. While production data exist without WFM, control data only come into existence through the deployment of a WFMS. Production data comprise all data that are essential for an application area. For example, travel claim forms or CAD drawings are production data. Control data are the minimal set of data which have to be exchanged between workflows in order to indicate against what (production) data set a workflow (i.e. the applications) should execute. Very often control data are pointers to production data; the latter have to be used in the applications called within workflows. For example, between two elementary workflows control data - pointers to CAD drawings - are exchanged indicating that the applications called in these elementary workflows have to work on these CAD drawings.

We introduce the following language construct to declare the informational aspect of a workflow:

\[
\text{informational-aspect} \quad ::= \quad \text{[LOCAL VARIABLES \{data-type: \{data-variable\}\}]} \\
\text{[DATA FLOW}}
\quad \text{[[PRE-CONVERSIONS}}
\quad \quad \text{\{conversion-name\{parameter\}\}]} \\
\quad \quad \text{\{workflow-variable\{parameter\}\}]
\quad \text{[POST-CONVERSIONS}}
\quad \quad \text{\{conversion-name\{parameter\}\}]} \]

For each workflow local variables are defined (LOCAL VARIABLES). Like the actual calling parameters of the workflow they are available for all functions and subworkflows referenced inside the workflow. The data flow is defined in an extra section DATA FLOW which will be explained by some examples in the following.

Data flow occurs between a superworkflow and its subworkflows, and between subworkflows. Data flow between a superworkflow and a subworkflow takes place when subworkflows use the local data of the surrounding superworkflow as actual parameters. Depending on whether these data are input or output data for the subworkflow, data are transmitted from the superworkflow to the subworkflow or from the subworkflow back to the superworkflow. The following example demonstrates some typical situations:

```
WORKFLOW TYPE workflow_type_1 (IN integer: x, OUT integer y)

... SUBWORKFLOWS
    workflow_type_2: workflow2
    workflow_type_3: workflow3

LOCAL DATA
    integer: z
    string: ss

DATA FLOW
    workflow2 (x, ss)
    workflow3 (ss, z)
```
MOBILE: A Modular Workflow Model and Architecture

WORKFLOW TYPE workflow_type_2 (IN integer: x, OUT string: s)
...
WORKFLOW TYPE workflow_type_3 (IN string: s, OUT integer: x)
...

We see that data flow occurs from workflow_type_1 to workflow2 (through x) and from workflow3 to workflow_type_1 (through z). Also data flow between workflow2 and workflow3 via s.

We have not shown so far the meaning and purpose of conversion-name() which is the header of a conversion function. We need conversion functions to adjust data such that they can be exchanged and used by different workflows. For instance, two workflows want to exchange budget data. The first workflow counts the budget in $, the second workflow uses DM as currency. A conversion function which can be executed before and after a workflow will convert the budget such that it is computable by both workflows.

In Figure 1 we see how data are exchanged either between a super- and a subworkflow and between subworkflows. 'Trip_Id' is transmitted between the surrounding workflow and all its subworkflows. 'TC_Id' is passed from the subworkflow 'submit_travel_claim' to the subworkflows 'approve_travel_claim' and 'reimburse_client'.

One of the most interesting effects of data management in the realm of WFM is its impact on control flow. To illustrate this observation, we consider two workflows workflow1 and workflow2 which belong to independent (top level) workflows. Assuming both workflows have to access the same data element d exclusively, a interdependence between the two workflows is created. The exclusive data access enforces serialized execution of workflow1 and workflow2.

7. Implementation

This section introduces the architecture of the MOBILE Workflow Management System. We distinguish between the build-time architecture (Section 7.1) and the run-time architecture (Section 7.2). We refer to [Bussler and Jablonski 1994b] and [Schuster et al. 1994] when details about the architecture are to be studied. Currently, a skeleton of the overall MOBILE WFMS is implemented, i.e. the components are implemented as black boxes with well-defined interfaces and without functionality. Some of the components like the Control Server and the Policy Server are already implemented. We subsequently are going to implement the remaining components.

7.1 Build-time Architecture

The build-time architecture consists of two major components, the MOBILE Build-time Work Area and the MOBILE Repository. The MOBILE Build-time Work Area itself is composed of three subcomponents, Workflow Definition, Organization Definition, Data and Function Definition. In Workflow Definition workflows are designed either on a graphic and on a script level, applications are defined, i.e. wrappers are written for programs, and macros for control flow constructs are specified. We have already discussed in Section 5.3 that the definition of an organization should be separated from the definition of workflows. Besides, in some enterprises we can presume that there is an organizational database already available. The module Organization Definition then must maintain a link between this preexisting database and the MOBILE Repository. In an ideal environment, also data and functions would be organized in a global dictionary. The elements of the dictionary would be available enterprise-wide and therefore could also be used by the WFMS. Because this ideal infrastructure normally
Stephan Jablonski

is not provided yet, we at least use a separated subcomponent to define data and functions in order to prepare the link to such a global dictionary.

![Figure 10: MOBILE Build-time Architecture](image)

The MOBILE Repository can be regarded as a huge class library which stores all workflow relevant information. Having such a repository available the specification of a new workflow more resembles a configuration step than a programming step. Figure 11 shows how the specification of a new workflow type can take place. The use of a repository supports reusability of MOBILE artefacts significantly.

![Figure 11: Specification of MOBILE Workflows using the MOBILE Class Libraries](image)

7.2 Run-time Architecture

The MOBILE Run-time Architecture consists of two major blocks, the MOBILE Kernel and the MOBILE Shell which altogether define the MOBILE Execution Engine. The Controller as the only component of the MOBILE Kernel interprets the workflow specifications stored in the MOBILE Repository which represents the communication medium between MOBILE Build-time Architecture and MOBILE Run-time Architecture. The MOBILE Kernel is responsible to drive the execution of workflows. It evaluates workflow descriptions in order to
find out which workflows to execute next. The MOBILE Kernel is surrounded by the MOBILE Shell. It consists of multiple servers which provide the functionality that is needed to execute the aspects of the MOBILE workflow model presented in this paper.

The Policy Server resolves roles and policies and determines agents for workflow execution. The Application Server links to application programs which have to be executed in the context of elementary workflows. The Data Server provides the connection to external data management systems in order to link between control data and production data. The Notification Server handles the dialogue of the Kernel with the agents' work-to-do lists: agents are notified about work-to-do - the Kernel is notified about agents' intentions to do work. If synchronization between agents is required the Synchronization Server is called.

The execution of workflows starts with the determination of the subworkflows that have to be executed first. After these subworkflows are found, interface data are transmitted to them. Then, the Policy Server is called in order to find out who should execute the workflow. The Notification Server informs agents about upcoming tasks; the Synchronization Server coordinates agents. After a subworkflow is finished, the Controller determines the subworkflows that have to be executed next or terminates the execution of the superworkflow.

Instead of subworkflows applications are called in elementary workflows.

---

8. Related Work

In this section we cite some related work which covers a broad spectrum of different approaches. Of course, the comparison of these approaches to the approach introduced in this paper is sometimes pretty 'uneven', since not all approaches emphasize the aspects fundamental to our work. We do not intend to compare our approach to approaches that only tackle process modeling, but we want to compare to approaches which deal with process modeling and process execution as well.

The DISDES approach is presented in [Reim 1992]. The model for an Organizational and Information System consists of Workflows, Processes, Positions, Persons, Users, Organizational Groups, among other things. These elements are aiming at the functional and the organizational aspects of process modeling. Although the approach is rather promising, a
bottleneck will be its limitation to a predefined set of objects available to reconstruct a problem space. In contrast, our approach allows to introduce and define arbitrary user-defined objects (e.g., for expressing organizational issues or for defining behavior). For instance, the 'reports to' relationship seems to be the only one to relate objects of Organizational Groups in DISDES. If a particular application area requires another relationship between organizational objects, DISDES cannot support it.

Almost the same observation made for DISDES applies to ActMan [Jablonski et al. 1991]. ActMan nicely copes with the integration of existing applications into workflows; also the informational aspect of process modeling is considered. But both are tackled in the same static way as in DISDES. Besides no organizational aspects are considered. Also the behavioral aspect is very limited to prescriptive courses of processing.

The AMIGO workflow model is described as a model for Group Communication processes [Danielsen and Pankoke-Babatz 1988]. It represents a simple but powerful model for workflow management. AMIGO is lacking a structured way for workflow definition; nesting and reuse of workflows is not an issue. All control aspects have to be modeled by condition/action pairs which makes the definition of behavior pretty cumbersome even in simple cases (e.g., when serial execution has to be expressed). Also the logical correctness of the specification might be difficult to proof. From an organizational point of view, only roles are known. They have to be defined separately and independently for each workflow which might violate security [Bussler 1992]. The interoperability of workflows designed independently is also not possible since roles might be used differently in those workflows.

Another interesting approach to process modeling, CIMOSA, can be found in [Kosanke 1992]. There the relationship between Domain Processes, Business Processes, and Enterprise Workflow is shown nicely. To model Enterprise Workflows - they are equal to workflows introduced in this paper - procedural rules are used which describe the control flow between workflows. So called functional entities represent resources, who can executed pieces of work. CIMOSA does not allow to specify descriptive types of control flow. Specification of policies is only possible in a very limited manner.

A number of other approaches to WFM can be found in literature [Ghoneimy et al. 1991] [McCarthy and Sarin 1993]. However, many approaches which are called WFMS merely deal with the behavioral aspect of processes [Breitbart et al. 1993] [Georgakopoulos et al. 1993] [Shet and Ruskiewicz 1993]. Besides, many approaches to extended transaction management are also put into the category of WFM [Attie et al. 1993] [Guenthoer 1993] [Klein 1991]. We prefer to sustain a distinction between these approaches and WFMSs. We see very close relationships among these fields, but want to clearly separate them. We also agree, that there is mutual leverage among the fields as can be seen in [Jablonski 1993].

9. Conclusion and Outlook

We detailed the functional, the behavioral, the organizational and the informational aspect of the MOBILE workflow model. At the end, we also presented briefly the architecture of the MOBILE Workflow Management System which is based on the MOBILE workflow model. We want to emphasize again, that modularity is the main feature of our model why extendibility, robustness, ease of use and other qualities can be achieved. We have already implemented first prototypes of either the Build-time and the Run-time Architecture of the MOBILE Workflow Management System.
We could utilize the knowledge we gained in developing MOBILE in various customer projects. It was interesting to realize that applying this experience, specifically following strictly the principle of modularity, eases either the process of finding a (workflow) model and a (workflow) architecture for specific problem scenarios, although we had to use commercial WFMS products in those projects.

Our next steps will be to complete the implementation of MOBILE, to investigate the issue of reliability in the context of MOBILE, and to investigate the link between business process modeling and workflow modeling.

10. References


INFORMATION SYSTEMS MODELLING USING LOOPN++,
AN OBJECT PETRI NET SCHEME

Chris Keen and Charles Lakos
Department of Computer Science
University of Tasmania
Hobart, Tasmania 7001, Australia
Fax: 61-02-202913
e-mail: Chris.Keen@cs.utas.edu.au, Charles.Lakos@cs.utas.edu.au

Keywords: Object-oriented languages, object petri nets, information systems modelling, ontological expressiveness

Abstract

There have been significant recent developments in high level specification schemes for modelling information systems using formalisms based on petri nets. In particular, the application of object-based design principles have produced high level petri nets which can be used to model large complex systems. This paper discusses a new language and modelling scheme, known as LOOPN++, which fully integrates the concepts of object-oriented language design and model building into a coloured petri net system. The completeness and clarity of LOOPN++ are discussed with respect to its use in the dynamic modelling of information systems.

1. Introduction

Petri Nets have been popular as a formal modelling technique and a practical tool for the description, analysis and simulation of concurrent computer systems (Jensen 1990; Reisig 1985; Reisig 1992). Much of the work on petri nets has focussed on their mathematical properties and their ability to accurately model components of systems. Relatively little work has been undertaken on scaling up petri net specification schemes to conveniently and elegantly represent large, complex information systems. An object petri net language and modelling scheme, LOOPN++ (Lakos and Keen 1994), is presented in this paper. This language fully integrates the concepts of object-oriented language design and modular design into a coloured petri net specification, and is based on several years' experience by the authors with previous modular petri net schemes. LOOPN++ specifications are amenable to formal analysis because they are founded on the semantics of coloured petri nets (Lakos 1994).

This paper contributes to the development of dynamic modelling tools by presenting an object-based petri net scheme, LOOPN++, and then evaluating that scheme against accepted measures of ontological expressiveness (Wand and Weber 1993). The authors have completed the syntactic and semantic design of the LOOPN++ language, and are now proceeding with an implementation of analysis and compilation tools. Research is also being conducted into systems development methodologies which are applicable to the analysis, design and prototyping of information systems using LOOPN++.

Conventional petri nets separate control and data in their specification. Data is modelled through places, and the assignment of tokens to places. A state of a petri net model is represented by a
configuration of tokens assigned to places. Control is modelled through transitions which are linked to input and output places by communication paths, represented by directed arcs. All state transformations in a petri net occur as discrete events which are modelled as the firing of transitions. Each transition firing will consume at least one token from an input place and produce at least one token at an output place, thus changing the model's state.

The expressive power of petri nets has been enhanced by the use of coloured tokens, which permits tokens to be assigned data types and carry values through the network. An example of a system for coloured petri nets is Design-CPN (Jensen et al. 1992) which has been employed to model a wide range of process control, real-time systems and information systems (Jensen 1990). Modular and hierarchical petri nets (Huber at al. 1990; Reisig 1986) have permitted the decomposition of the system models into discrete classes or modules, and the reuse of petri net modules. However, the development of coloured petri nets, hierarchical and modular petri net systems has focussed on increasing the expressive power of petri nets by the providing more complex control structures. Decomposition is primarily used as a mechanism to break up the program space of a model into manageable modules. Coloured Petri Nets have also increased the data modelling power of petri nets by permitting tokens to have associated data types, but this falls well short of the rich data types required to fully model complex information systems.

While a range of high level petri net models have been proposed, little attention has been given to the extent to which such models are sufficient for the task of modelling real information systems, and the extent to which their component concepts are necessary for developing conceptual models of information systems.

Wand and Weber (Wand and Weber 1990; Wand and Weber 1993) have considered system development methodologies and their associated system modelling schemes. They have attempted to formally consider the question of assessing how well information systems analysis and design methodologies address the three major tasks that need to be undertaken during the analysis and design of information systems:

1. Representation of the real world according to the views held by individual users. They have proposed a list of constructs which a modelling scheme must be able to describe.

2. Faithful tracking of the real world system as it undergoes transformations from one state to another. They have stated conditions under which an information system model provides faithful state-tracking of a real information system.

3. Decomposition of the real-world model in ways which reflect the structure and dynamics of the represented real system. They have proposed a set of necessary requirements that an information system must fulfil if it is to be well decomposed.

Wand and Weber have developed an assessment of system modelling techniques using a grammatical description of the model framework. Through the concepts of ontological completeness and ontological clarity they assess the completeness of the design constructs of a modelling scheme, and ability to clearly model each real-world (ontological) construct without construct overload, redundancy or excess.

An information system modelling scheme can be expressed as a grammar which captures all legal forms of representation in that framework. The work of Wand and Weber provides an instrument by which to measure objectively the completeness and quality of an information system modelling scheme.

This paper consider LOOPN++ (Lakos and Keen 1994) which is an object-based extension to the basic petri net model. This system is particularly suited to the dynamic modelling of complex information systems because it breaks down the conventional distinctions between control and data in petri nets models, by providing object-oriented features at all levels of language. Thus places and tokens, which are the conventional data components of petri nets, can contain both control and data features in LOOPN++. Similarly, the LOOPN++ transition
can also contain both aspects of control and data. We believe that this integration has been achieved without adding complexity to the specification model, but rather the syntax of LOOPN++ is very simple and the semantics are formally defined in terms of coloured petri nets (Lakos 1994). A significant feature of LOOPN++ is the use of a single class hierarchy to represent all components, both passive (ie. token-like) and active (ie. transition-like), via a common syntactic structure. This has resulted in a specification scheme which has a very concise grammar, yet captures all of the descriptive power of modular, coloured petri nets.

The development of LOOPN++ has been based on five years' experience by the authors with a modular petri net system, LOOPN (Language for Object-Oriented Petri Nets). LOOPN is both a model specification language and an interpretive program system. A compiler translates LOOPN specifications into efficient C programs which can be executed to simulate the operation of the LOOPN petri net. LOOPN has been employed to model a wide range of information systems, and is well suited to the dynamic modelling of highly concurrent and real-time systems (Lakos and Keen 1991; Lakos and Keen 1993).

The details of LOOPN++ are described in sections 2 and 3, and its features are analysed against Wand and Weber's measures of ontological expressiveness for information systems modelling in section 4.

2. Object-Oriented Petri Net Requirements and Proposals

In this section we review some of the other approaches which have been taken in combining object-oriented structuring with petri net specifications. We identify their benefits and their deficiencies. Firstly, we identify the properties we would expect from an object-oriented petri net scheme, and the areas of application that it would be desirable to address.

In traditional petri net specification schemes, the token types determine the kind of data that can be handled by the net. In elementary nets and place-transition (PT) nets, there is only one token type which can have only one value. The only state information available is the number of tokens present in each place. In coloured petri (CP) nets, each token has a data type, such as integer, tuple, union, record or array, and tokens can have values that are consistent with these types. These token colours condense petri net structures by allowing repetitive subnet structures to be collapsed into a single subnet. The passage of tokens through the original, repeated subnets can then be differentiated by the different values carried by tokens in the coloured petri net. Flexible net inscriptions are also present in coloured petri nets to allow more powerful conditions for the selection of input tokens, and expressions for the generation of values for output tokens.

However, despite the range of token types provided in coloured and hierarchical petri net models, tokens are still passive data items. Their life cycle and set of acceptable operations are not encapsulated within a token, but are external to the token and given by the interconnection of petri net components and the inscriptions on arcs and transitions. In other words, the net forms a global control structure in the same way that procedures form the global control structure in traditional imperative programming languages. Tokens are simply value holders to be manipulated by this control structure.

This distinction between control and data is contrary to the design philosophy of object-oriented languages in which a single syntactic entity, the class, encapsulates both control and data. The term class is used here to refer to both a data type and also the set of all instances, or objects, of that class type. All actions in an object system can then be modelled as message exchanges in which a source object transmits a message object to a target object and may receive another object as a response.
2.1 Desirable Applications of Object-Oriented Petri Nets

It would be desirable, for example, to have a simple way of modelling multi-level simulations, where the components that flow through the system have their own internal life cycles. Real-life situations generally have a number of layers of data and activity. For example, in modelling a traffic intersection, the cars which move through the intersection can be considered as data objects. But cars also have internal activities such as the consumption of petrol and the operation of the engine, which may be of interest in a simulation. One could continue by considering the people in the car as systems requiring modelling at a more detailed level.

Another desirable area of application is that of object-oriented operating systems, where the entities handled by the system encapsulate their own protocols. For example, we might consider a situation where an item of electronic mail encapsulates the logic to control interaction with its contents. Thus, different parts of the document may be visible, depending on the security clearance of the viewer, or the display of the message may vary depending on the screen technology available. Similar features are exhibited by Electronic Data Interchange, where a message encapsulates not only the original message but also related documents. Thus, a bill of lading might incorporate a request for a letter of credit, a request for insurance cover, etc.

A third desirable area of application is that of producing prototypes from object-oriented design methodologies. Such methodologies produce models that include a set of objects, each of which encapsulates its own data and interactions. For example, the Shlaer-Mellor methodology (Shlaer and Mellor 1992) produces objects which encapsulate their own life cycle, and which can be dynamically created and discarded. Without the flexibility of intermixing data and control, the production of prototypes from such designs is not simple (Martin and Santanach 1993).

2.2 Review of Existing Petri Net Models

With a range of application areas in mind, it is instructive to consider some of the proposals which have been made to incorporate object-oriented structures into petri net modelling schemes. One system, which was the precursor to this work, was the textual language for object-oriented petri nets called LOOPN (Lakos 1992). This language supported two class hierarchies: one for tokens and another for subnets or modules. Inheritance, overriding and polymorphism for token types led to some interesting results in the modelling of layered network protocols, so that one protocol layer could be designed to pass any kind of message token (Lakos and Keen 1991). Inheritance, overriding and polymorphism of subnet or module types allowed the derivation of more complex modules from simpler ones, thereby encouraging software reuse (Lakos and Keen 1993). An unusual feature in the context of petri net models was the provision of module access functions which allowed access to some internal aspect of the state of a subnet without modifying that state. These facilities proved to be extremely beneficial in the development of clean module interfaces (Lakos and Keen 1991). However, LOOPN still retained a rigid separation of token types and subnet types, with the petri net being a global control structure and tokens being essentially passive data items. As a result of this language design, it is not easy to apply LOOPN to the application areas identified in section 2.1.

Researchers at the University of Aarhus, who have been intimately involved with the development of the CP-net formalism and the associated Design/CPN tool (Jensen 1992; Jensen et al. 1992) have also been experimenting with the integration of an object-oriented language BETA (Kristensen 1991) into Design/CPN (Christensen and Toksvig 1993). The BETA language is used to declare the token types and to annotate the transitions with BETA code segments. However, there are other contexts where BETA cannot be used and there is no
intention of departing from the traditional approach with the petri net as a global control structure.

Van Hee and Verkoulen (van Hee and Verkoulen 1991, Verkoulen 1993) have considered the modelling of information systems in petri net models. In specifying such systems they identified the need for specification schemes which integrated three aspects of a system - the state space (described by data structures), the interaction structure (between system components) and the operations (or local state transformations of the components). In focussing on the petri net model, they observed: "Petri Net formalisms do not support a real data-oriented view of systems, such as is provided by modern object-oriented data models". In response to this, they present a two-level data model, called the SimCon Object Model. This model consists of simplexes or simple objects, which have object identity, attributes and relationships with other objects, and complexes or container objects, which contain structured sets of simplexes and which have object identity, location, time stamp, and a reference structure giving access to the simplexes in the complex. This data model is integrated into the SimCon Net Model where places hold particular complexes and where transitions transfer, modify, delete and generate complexes, as specified in an object algebra, called the SimCon Algebra. There are many attractive features to this system including an enhanced facility in information modelling, the ability to specify class-dependent methods and cardinality constraints, and the ability to verify some constraints automatically. However, it appears to fall short of a complete integration of object-oriented ideas, with data and functions separated, rather than encapsulated together and with the life cycle of a complex being external to the complex and not integrated within the object.

Buchs and Guelfi (Buchs and Guelfi 1991) define object-oriented nets in terms of abstract data types where the externally-accessible methods are transitions that can be synchronised with the transitions of other objects. They allow the specification of methods as components of the abstract data types, with the synchronisation still being defined globally.

In summary, all the above proposals have, to some extent, focussed on the incorporation of object-oriented structuring into the definition of token types. They have not addressed the encapsulation of data and functions, but retain the traditional petri net style with the tokens as passive data items and their life cycles specified by the global control structure of the net.

3. LOOPN++: A Language for Object-Oriented Petri Nets

This section specifies the syntax and informal semantics of LOOPN++. It first presents the basic grammar and then the following subsections consider the various components of the language in turn: classes, fields, functions, actions, transitions and places.

The design of LOOPN++ attempts to avoid the purpose-specific constructs of other petri net specification schemes and instead provides a minimal set of constructs with orthogonal combinations. It may well be appropriate to provide additional syntactic structures which resembles traditional constructs, but it is intended that the underlying constructs should be more general and should be able to be combined in arbitrary ways. The orthogonal combinations will also support concepts which were not previously available, such as place types, substitution places, and synchronous interaction between subnets.

The basic grammar of LOOPN++ is given in figure 1. It assumes the provision of some underlying language with basic, predefined types together with associated operators and functions. The following metasymbols and conventions are used:

```plaintext
[...] the enclosed construct(s) is optional
(...) the enclosed construct(s) can be repeated zero or more times
Keywords are shown in upper case.
```
### Figure 1. The basic grammar of LOOPN++

#### 3.1 Classes and Instances

A LOOPN++ program consists of a finite set of class definitions. One class is designated the root class for the purposes of the compilation, and a single instantiation of this class constitutes the main program.

A class defines a type and a set of objects, which are the set of all instances of that class. The term type is used interchangeably with class. Types may be classes defined by the user or basic, predefined types such as boolean, integer, real, string.

A class specification consists of attributes or fields, functions and actions. Each component or feature of a class is a field, function or action. Each component is created and may be bound on instantiation of the enclosing object.
A class may be declared to *inherit* the features of one or more *parents*, in which case all the features of the parents, together with the additional features declared within the class constitute the features of the new sub-class. In inheriting from a class it is possible to rename inherited feature identifiers, as in Eiffel (Meyer 1988). It is not permitted to override one feature by another of a different kind, and hence one cannot override a field by a function or an action.

The *export* clause of a class specifies which features are externally accessible. It does so simply by listing the identifiers to be exported. Any field or function, whether declared locally or inherited, may be exported.

### 3.2 Fields

#### field → type ident [ = value ]

#### value → const

  → ident

  → ' [ ident : value, ... ]'

  → ident ' [ ident : value, ... ]'

  → ' [ value, ... ]'

  → ident ( exprs )

A *field* is a class component of some type which normally holds data, and hence determines part of the state of that object. The type of a field may be any built-in, predefined type or a user-defined class. The type may also be of the form `type*` which indicates a multiset, or bag, of objects each of type, and is referred to as a *multiset type*. The form `type**` is used to declare petri net places. By implication, `type***` is also possible, but not necessarily useful. Each type identifier is either a basic predefined type or a class identifier.

A value of a field is bound and fixed when the object which contains it is instantiated. The syntax rules for values indicate that they may be specified in a number of ways: by a literal constant, by a copy of an existing object, by a new object with the values of individual fields specified, by a copy of an existing object with certain fields modified, by a list of values (for a multiset type), or by a value computed from a function call.

For example, the declarations in figure 2 come from the modelling of the dining philosophers problem in LOOPN++ (Lakos and Keen 1994).

```plaintext
Class DPhil
  Export id;
  integer n = 5;
  integer id;
End DPhil

Class DTable
  integer n = 5;
  DPhil* thinking = [[id:1],[id:2],[id:3],[id:4],[id:5]];
  DPhil* hasLeft, hasRight, eating;
  integer* fork = [1,2,3,4,5];
End DTable
```

**Figure 2.** LOOPN++ declarations in the dining philosopher problem
In this example, the class \texttt{DPhiI} contains two fields, \texttt{id} and \texttt{n}, both integers. The field \texttt{n} is initialised to the value 5 and is not exported, so it will assume this value in every instance of this class. The field \texttt{id} is not initialised but is exported, so it can be initialised wherever this class is instantiated. The class \texttt{DTable} contains a number of places holding \texttt{DPhiI} type tokens. The places \texttt{hasLeft}, \texttt{hasRight} and \texttt{eating} have no initialisation and are therefore assumed to be empty. The place \texttt{thinking} does have initialisation which indicates that it will contain five tokens with \texttt{id} fields as specified.

3.3 Functions

\begin{verbatim}
func → type ident (params) = expr
\end{verbatim}

A \textit{function} defines a parameterised expression, which returns a value based on the other features, and hence state, of an object. Functions will commonly be defined to take parameters and return values of some basic, predefined type, but they may be defined to take parameters and return values which are objects or even multisets of objects.

For example, we could extend the definition of dining philosopher tokens (given in section 3.2) to include functions which return the left and right fork numbers, and also a function to indicate whether a philosopher is hungry:

\begin{verbatim}
Class DPhil
    Export id, left, right, hungry;
    integer n = 5;
    integer id;
    integer left() = id;
    integer right() = id mod n+1;
    boolean hungry() = true;
End DPhil
\end{verbatim}

\textbf{Figure 3. The class of dining philosophers}

Functions can also use quantifiers: \texttt{forall}, \texttt{exists}, \texttt{count}, \texttt{select} to determine a value from the multiset of tokens resident in a place.

3.4 Actions

An \textit{action} is the fundamental construct by which state changes can be specified in LOOPN++. A single action alone in LOOPN++ is not very useful, but a cluster of synchronised actions can be used to define a transition.

\begin{verbatim}
action → type ident <- value [ | expr ]
       → type ident → value [ | expr ]
       → ANON proc-call
\end{verbatim}

An action is the only construct available in LOOPN++ for changing the state of an object. Actions may be input actions, output actions, or anonymous actions. Such actions are the fundamental building blocks of state changes in LOOPN++ in contrast to the more normal petri net approach of making transitions the fundamental components. As discussed in section 3.5, transitions can be considered as synchronised sets of actions. In fact, a transition can be considered to be an instance of a class which contains actions but no data. This implies that the actions which are immediate components of an object are always synchronised with each other, but not necessarily with the actions contained within other component objects.
An input action obtains a value from an object and is written:

\[
\text{type } x \leftarrow p \text{ [ } | \text{ condition } \]
\]

where the restriction "| condition" is optional. For such an input action to occur, the value of 
\( x \) obtained from \( p \) must be of an appropriate type and satisfy the condition, if any. The value \( x \) 
is called a token (or tokens), while the object \( p \) is called an input place. The condition is a 
boolean expression which may contain terms of the form:

\[
x = \text{value}
\]

This is interpreted as testing that the components of \( x \) match those specified by the value.

An output action deposits a value into an object and is written:

\[
\text{type } x \rightarrow p \text{ [ } | \text{ condition } \]
\]

where "| condition" is optional. For such an output action to occur, the value of \( x \) must 
be acceptable to \( p \). Again, the value \( x \) is called a token (or tokens), while the object \( p \) is called an 
output place. The condition is a boolean expression which will be of the form:

\[
x = \text{value}
\]

This is interpreted as generating an object with components matching that of the value.

It is important to note that output tokens are always newly-generated objects or newly-generated 
copies of existing objects. In this way, it is not possible to carry a reference to a remote object 
(such as a place) around a petri net, because within each output action the newly-generated 
output tokens must be bound to values which are within the scope of reference of that output 
action.

An anonymous action interacts with the environment of the petri net and is of the form:

\[
\text{action function-calls;}
\]

An example of an anonymous action may be a call to an output function to write a value to the 
standard output stream. In order to support formal analysis it is necessary for anonymous 
actions to have no side effects on the firing of transitions in the petri net.

As an example of a set of actions, we might consider the set of actions (in the context of the 
example given in sections 3.2 and 3.3) which would remove a DPhil token from place 
thinking provided there are the appropriately-numbered token in place fork and add a copy of 
the same DPhil token to place hasLeft:

\[
\begin{align*}
\text{DPhil } & x \leftarrow \text{thinking;} \\
\text{integer } & f \leftarrow \text{fork } | f = x.\text{left}(); \\
\text{DPhil } & y \rightarrow \text{hasLeft } | y = x;
\end{align*}
\]

The token removed from place thinking is given the identifier \( x \). The token removed from 
place fork needs to match the result of function left on DPhil token \( x \). Finally, the token \( y \) 
which is added to place hasLeft is a copy of token \( x \).

3.5 Transitions and places

Traditional petri net formalisms include the fundamental concepts of transitions and places. In 
LOOPN++, these are generalised and built from more basic components. A transition is a 
special case of an object which consists solely of actions. It is written in the form:
Trans ident;
  type x <- p | ...; — zero or more input actions
  type y -> q | ...; — zero or more output actions
  Anon function-calls — zero or more anonymous actions
End;

In other words, a transition is an object consisting of a synchronised set of actions, together
with appropriate export of place identifiers. An example of this would simply be to enclose the
set of actions given in section 3.4 inside Trans ... End brackets.

A place is any object which supports input and output actions, by supplying or accepting
tokens. In the simplest case, a place is an object of multiset type and is declared in the form:

    type* ident

In this case, the support for input and output actions is predefined.

In LOOPN++ it is also possible to define more complex objects with place properties, which
are then referred to as substitution places (Huber et al. 1990). Such an object, p, may supply
or accept tokens by including the output/input actions of the form:

    y -> self or y <- self

These actions may be considered as transferring tokens to or from the object boundary. Such
actions need to be synchronised with external actions which remove or supply respectively
those tokens. These external actions may be of the form:

    x <- p or x -> p

Often the internal actions of object p will be grouped in a transition, say f. In this case, the
external input/output action will specify the particular transition as the interface point:

    x <- p.f or x -> p.f

Note that since input/output actions of p may be synchronised with other actions of p, the
acceptance or production of tokens by p may be controlled by the internal state and logic of p.
This provides a convenient way of supporting capacity places and place filters.

Examples of the use of these substitution places are given in section 3.6.

3.6 A LOOPN++ Example Specification: an EDI System

In order to demonstrate the features of LOOPN++ we present an example of a model of
Electronic Data Interchange. We model a system where there are a number of mail subscribers
and a number of documents. Each subscriber generates, receives, reads and replies to mail
documents. Each document contains a number of information fields which we assume is
supplied by a class definition called Field. A document may also be associated with a number
of related documents, which are determined by the information fields of the original. A
document may expect a standard reply, or in a more complex example, a bill of lading might
have a paper trail including a request for a letter of credit, a request for insurance cover, etc.

We start by defining a basic document. As well as data fields indicating the kind of document,
the specification of sender and receiver and the information fields, a document encapsulates
functions to generate the standard replies, to determine which information fields (from an
incoming reply) are not already included in the document, and to determine the subdocuments
of the current document. The appropriate class definition is given in figure 4.
Since an EDI document is associated with a number of other subdocuments, it is important to maintain the status of the paper trail associated with a particular document. This is modelled in the class of umbrella documents, given by the class Umbrella. This class is a subclass of Doc because it gives the status information associated with a particular document. It includes places holding the subdocuments which are still to be posted, those which have already been posted, and the expected replies. The class includes transitions to generate the subdocuments associated with the current document, to receive an expected reply, and to post subdocuments. The class definition of Umbrella is given in figure 5.

**Figure 5. Class definition for umbrella documents**
Note that the Umbrella class uses a predefined class Null. This is the class used for colourless tokens, and is indirectly inherited by all other classes. While containing no data fields, it includes functions for determining the status of a token in a place.

<table>
<thead>
<tr>
<th>Class Subscriber</th>
<th>-- Class for an EDI subscriber</th>
</tr>
</thead>
<tbody>
<tr>
<td>Export receivemail, sendmail, readmail, replymail;</td>
<td>-- address of the subscriber</td>
</tr>
<tr>
<td>Address own;</td>
<td>-- mail box of messages belonging to this subscriber</td>
</tr>
<tr>
<td>Doc* mbox;</td>
<td>-- receive mail and add it to the mail box</td>
</tr>
<tr>
<td>Trans receivemail;</td>
<td>-- the document must be addressed to me</td>
</tr>
<tr>
<td>Doc x &lt;- self</td>
<td>-- save the document in the mail box</td>
</tr>
<tr>
<td>Doc y -&gt; mbox</td>
<td>-- method of new document generation is not specified</td>
</tr>
<tr>
<td>End;</td>
<td>-- generate a new document and send it</td>
</tr>
<tr>
<td>Trans sendmail;</td>
<td>-- examine a document in the mail box without replying</td>
</tr>
<tr>
<td>Doc x -&gt; self</td>
<td>-- the document does not expect a reply</td>
</tr>
<tr>
<td>End;</td>
<td>-- examine a document in the mail box and reply appropriately</td>
</tr>
<tr>
<td>Trans readmail;</td>
<td>-- the document has an associated reply</td>
</tr>
<tr>
<td>Doc x &lt;- mbox</td>
<td>-- send the reply</td>
</tr>
<tr>
<td>End;</td>
<td>-- examine a document in the mail box</td>
</tr>
<tr>
<td>End Subscriber</td>
<td>-- don't discard the subscriber</td>
</tr>
<tr>
<td>Class System</td>
<td>-- subscriber accepts an EDI document from an umbrella</td>
</tr>
<tr>
<td>Subscriber* subs;</td>
<td>-- the EDI subscribers</td>
</tr>
<tr>
<td>Umbrella* umb;</td>
<td>-- the umbrella documents</td>
</tr>
<tr>
<td>Trans send;</td>
<td>-- add umbrella to a subscriber-generates EDI document</td>
</tr>
<tr>
<td>Subscriber s &lt;- subs;</td>
<td>-- document sent by the subscriber</td>
</tr>
<tr>
<td>Doc d &lt;- s.sendmail;</td>
<td>-- the umbrella has the same basic fields as the document</td>
</tr>
<tr>
<td>Umbrella u -&gt; umb</td>
<td>-- the umbrella supplies the subdocument</td>
</tr>
<tr>
<td>End;</td>
<td>-- document must be accepted by the subscriber</td>
</tr>
<tr>
<td>Trans receive;</td>
<td>-- retain the subscriber</td>
</tr>
<tr>
<td>Subscriber s &lt;- subs;</td>
<td>-- retain the umbrella</td>
</tr>
<tr>
<td>Umbrella u &lt;- umb;</td>
<td>-- subscriber replies to an EDI document</td>
</tr>
<tr>
<td>Doc x &lt;- u.post;</td>
<td>-- reply sent by the subscriber</td>
</tr>
<tr>
<td>Doc y -&gt; s.receivemail</td>
<td>-- the umbrella must accept the reply</td>
</tr>
<tr>
<td>Subscriber t -&gt; subs</td>
<td>-- retain the subscriber</td>
</tr>
<tr>
<td>Umbrella v -&gt; umb</td>
<td>-- retain the umbrella</td>
</tr>
<tr>
<td>End;</td>
<td>-- retain the subscriber</td>
</tr>
<tr>
<td>Trans reply;</td>
<td>-- retain the umbrella</td>
</tr>
<tr>
<td>Subscriber s &lt;- subs;</td>
<td>-- retain the umbrella</td>
</tr>
<tr>
<td>Umbrella u &lt;- umb;</td>
<td>-- retain the umbrella</td>
</tr>
<tr>
<td>Doc x &lt;- s.replymail;</td>
<td>-- retain the subscriber</td>
</tr>
<tr>
<td>Doc y -&gt; u.receive</td>
<td>-- retain the subscriber</td>
</tr>
<tr>
<td>Subscriber t -&gt; subs</td>
<td>-- retain the umbrella</td>
</tr>
<tr>
<td>Umbrella v -&gt; umb</td>
<td>-- retain the umbrella</td>
</tr>
<tr>
<td>End;</td>
<td>-- retain the umbrella</td>
</tr>
</tbody>
</table>

End System

Figure 6. Class definitions for subscribers and the EDI system
Finally, the EDI system consists of subscribers and documents and the communication between the two. The management of the EDI system can be modelled in many different ways. In figure 6, we model the system as some global mail-clearing house by defining the classes Subscriber and System.

The ability to encapsulate data and actions in the one class and to dynamically generate instances of such a class is fundamental to the above solution. For example, the transition receive in class System accesses an umbrella u and then extracts the document from u. This multilevel access makes the above solution very natural but it is not supported by conventional petri net specification schemes.

There is a pleasing simplicity about both of the above solutions. Solutions in a traditional petri net scheme are often far from obvious. It is therefore strongly suggested that, as with CP-nets, the mapping from the more descriptive scheme into a simpler one should be provided by an implementation and not forced on the user of a model specification scheme. As already observed, the added descriptive comfort could be the difference between a practitioner being able to propose a solution to a problem, or considering that problem impractical.

4. Ontological Expressiveness of LOOPN++

Wand and Weber (Wand and Weber 1993) have addressed the issue of the extent to which a modelling scheme can completely and succinctly build models of real world systems. They consider the grammar of information systems modelling schemes and define the ontological expressiveness of that scheme as the ability of that scheme to describe all ontological constructs both completely and clearly.

4.1. Ontological Completeness

An information system modelling scheme is defined by Wand and Weber (Wand and Weber 1993) as being ontologically complete if that scheme contains a design construct to represent every component in a given set of ontological design constructs. That is, the mapping from a given set of ontological constructs to the set of design constructs provided by an information system modelling scheme is total. Wand and Weber give the list of ontological design constructs shown in table 1 as those expected for an information system modelling scheme to be ontologically complete.

<table>
<thead>
<tr>
<th>Ontological construct</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thing</td>
<td>A thing is the elementary unit in our ontological model. The real world is made up of things. A composite thing may be made up of other things (composite or primitive)</td>
</tr>
<tr>
<td>Properties</td>
<td>Things possess properties. A property is modelled via an attribute function that maps the thing into some value. A property of a composite thing that belongs to a component thing is called a hereditary property. Otherwise it is called an emergent property. A property that is inherently a property of an individual thing is called an intrinsic property. A property that is meaningful only in the context of two or more things is called a mutual or relational property.</td>
</tr>
<tr>
<td>State</td>
<td>The vector of values for all attribute functions of a thing is the state of the thing.</td>
</tr>
<tr>
<td>Conceivable state space</td>
<td>The set of all states that the thing might ever assume is the conceivable state space of the thing.</td>
</tr>
<tr>
<td>State law</td>
<td>A state law restricts the values of the properties of a thing to a subset that is deemed lawful because of natural laws or human laws. A law is considered a property.</td>
</tr>
<tr>
<td>Lawful state space</td>
<td>The lawful state space is the set of states of a thing that comply with the state laws of the thing. The lawful state space is usually a proper subset of the conceivable state space.</td>
</tr>
<tr>
<td>Event</td>
<td>An event is a change of state of a thing. It is effected via a transformation (see below).</td>
</tr>
</tbody>
</table>
Event space  The event space of a thing is the set of all possible events that can occur in the thing.

Transformation  A transformation is a mapping from a domain comprising states to a co-domain comprising states.

Lawful transformation  A lawful transformation defines which events in a thing are lawful.

Lawful event space  The lawful event space is the set of all events in a thing that are lawful.

History  The chronologically ordered states that a thing traverses are the history of the other thing.

Coupling  A thing acts on another thing if its existence affects the history of the other thing. The two things are said to be coupled or interact.

System  A set of things is a system if, for any bipartitioning of the set, couplings exist among things in the two subsets.

System composition  The things in the system are its composition.

System environment  Things that are not in the system but interact with things in the system are called the environment of the system.

System structure  The set of couplings that exist among things in the system and among things in the system and things in the environment of the system is called the structure of the system.

Subsystem  A subsystem is a system whose composition and structure are subsets of the composition and structure of another system.

System decomposition  A decomposition of a system is a set of subsystems such that every component in the system is either one of the subsystems in the decomposition or is included in the composition of one of the subsystems in the decomposition.

Level structure  A level structure defines a partial order over the subsystems in a decomposition to show which subsystems are components of other subsystems or the system itself.

Stable state  A stable state is a state in which a thing, subsystem, or system will remain unless forced to change by virtue of the action of a thing in the environment (an external event).

Unstable state  An unstable state is a state that will be changed into another state by virtue of the action of transformations in the system.

External event  An external event is an event that arises in a thing, subsystem or system by virtue of the action of some thing in the environment on the thing, subsystem or system. The before-state of an external event is always stable. The after-state may be stable or unstable.

Internal event  An internal event is an event that arises in a thing, subsystem or system by virtue of lawful transformations in the thing, subsystem or system. The before-state of an internal event is always unstable. The after-state may be stable or unstable.

Well-defined event  A well-defined event is an event in which the subsequent state can always be predicted given that the prior state is known.

Poorly defined event  A poorly defined event is an event in which the subsequent state cannot be predicted given that the prior state is known.

Class  A class is a set of things that possess a common property.

Kind  A kind is a set of things that possess two or more common properties.

| Table 1. Ontological Constructs in the Bunge-Wand-Weber representational model |
|---|---|

The ontological completeness of the LOOPN++ modelling scheme can be assessed against the set of ontological design constructs in table 1 (Wand and Weber 1993). This assessment is based on the ontological model developed by Bunge (Bunge 1979).

**Thing**

The *thing* construct corresponds to an object in LOOPN++. An object is an instance of a class. Depending on its class definition, an object may correspond to a token, a place or a transition in a conventional petri net.
Properties

The properties of a thing correspond to the attributes or fields of an object in LOOPN++. Each attribute maps an object into a value. An attribute may be a simple field, to which a data value is bound during the operation of the LOOPN++ net, or it may be a function which returns a value derived from other attributes in the LOOPN++ object.

A field in LOOPN++ may be one of two states:

- *unbound*, in which case it is undefined, or
- *bound*, in which case it is defined and bound to a value or a LOOPN++ object.

Initially all fields are unbound. Some fields may be bound during the instantiation of the containing object, while other fields may be bound when actions within the containing object are performed.

State

The state of a thing corresponds to a set of bindings of values and objects to the fields of an object in LOOPN++. At any time point this set of bindings is well defined in LOOPN++ and can be derived from the field type definitions, function definitions and history of a LOOPN++ object.

Conceivable State Space

The conceivable state space corresponds to the set of all states to which a LOOPN++ object can be bound. This state space is well defined and can be derived from the type and function definitions of the fields of a LOOPN++ object, and the environment of that object.

The semantics of LOOPN++ are based on those of coloured petri nets, for which state space enumeration is well defined and computable. Any LOOPN++ specification can be converted into a coloured petri net specification, and hence state space enumeration of LOOPN++ specifications is also well defined and computable.

State Law

Within LOOPN++ a state law can be represented via a type definition for integer attributes by specifying a restriction in allowable values to a given range. For non-integer attributes a state law cannot be expressed via a type definition.

A general state law condition can be defined using the internal components of a LOOPN++ class. A boolean function, say valid, can be defined over all attributes of the class, and can be employed to restrict access to the contents of only those objects for which the function is true. This enables the environment to determine whether the current state of an object is valid or invalid.

Lawful State Space

Given the limitations of LOOPN++ outlined above for defining state laws, the lawful state space, can be derived as that subset of the conceivable state space for which objects are in a valid state, namely for which the value of the function valid is true.

Event

An event corresponds to a transition firing in LOOPN++. The semantics of LOOPN++ ensure that each transition firing is a well defined, discrete event.
The basic unit of transformation in LOOPN++ is an action. The normal mode of construction of LOOPN++ specifications is to incorporate synchronised set of actions into a transition. The firing of a transition is then an atomic event, even though it may consist of the performance of several actions.

**Event Space**

An *event space* of a *thing* corresponds to all transition firings which can affect an object in LOOPN++. For each object the set of transition firings which affect that object is well defined and can be derived from a LOOPN++ specification.

**Transformation**

A *transformation* corresponds to a transition object in LOOPN++. The semantics of all transition firings in LOOPN++ are well defined so that the state space mapping of transformations from a domain of states to a co-domain of states can be derived from a LOOPN++ specification.

**Lawful transformation**

A *lawful transformation* is defined in LOOPN++ through the boolean conditions attached to the input actions of a transition. A transition will only fire when all of the conditions attached to its input actions are satisfied.

**Lawful event space**

A *lawful event space* corresponds to all lawful transition firings which can alter the states of LOOPN++ objects. This event space is well defined and can be derived from a LOOPN++ specification.

**History**

The *history* of a LOOPN++ object can be recorded as the sequence of states that the object has traversed in time. LOOPN++ incorporates the concept of time so that each transition firing is ordered with respect to this time axis and the chronological sequence of states bound to each object can be recorded.

**Coupling**

LOOPN++ has a iconic graphical representation based on conventional petri net diagrams. This graphical representation explicitly shows which components of a LOOPN++ model are coupled through the use of directed arcs between places and transitions.

Examples of *coupling* between LOOPN++ objects are:

- the token objects in a place are coupled to a transition object to which that place is an input place
- the token objects in a place are coupled to a transition object for which that place is an output place

**System**

A *system* corresponds to a connected LOOPN++ subnet. Systems can be modelled as LOOPN++ classes.
**System composition**

The LOOPN++ class definition provides a syntactic structure for encapsulating all of the components of a *system composition*.

**System environment**

The *environment* of a LOOPN++ object corresponds to all of those components of a LOOPN++ specification which are external to the object, but which are accessible to, or interact with that object. The environment of a LOOPN++ object is well defined through the scope rules associated with the specification of LOOPN++ objects, and the explicit export of fields and functions from within LOOPN++ objects.

**System structure**

The *system structure* of a LOOPN++ object is defined by the components of that object and its interaction with its environment. For a transition object the environmental interaction is defined via its input and output action clauses. For a token object the environmental interaction is defined via the set of exported identifiers which are available for reference outside of that object.

**Subsystem**

A *subsystem* of a LOOPN++ object A can be represented by the specification of another object B within the context of A. The structure and composition of object B are then subsets of the structure and composition of object A. Object A forms part of the environment of object B.

**System decomposition**

LOOPN++ provides for *system decomposition* by the decomposition of classes in a class hierarchy. A single class hierarchy covers all class types in LOOPN++, including transition, token and place types. LOOPN++ permits decomposition of both:

- active, or control systems, specified as transition objects. A single transition object can be decomposed into a combination of sub-objects, each interacting within the petri net which defines the operation of each firing of the transition.
- passive, or data systems, specified as token objects. A single token object can contain attributes which are either simple data values or functions, or sub-objects which contain petri nets to define their operations.

**Level structure**

The *level structure* of each LOOPN++ object is well defined through the syntactic structure of the class definition. Each object instance is either a distinct object (at the level of a main program) or that of a sub-object, contained within its parent object declaration. Hence there is a well defined partial ordering of objects in a LOOPN++ specification.

**Stable and Unstable states, Internal and External events**

The petri net component of a LOOPN++ object can model both *stable and unstable states*. Unstable states are associated with the internal operation of an object. Any state which can be changed through the firing of a transition internal to an object is an unstable state.

The interface to the environment of a LOOPN++ object is well defined, and varies between object types.
• a transition object interacts with its environment according to the specifications in its input and output actions. When a transition object fires it will get tokens from its environment and can pass these tokens to its internal petri net. A transition therefore is in a stable state immediately prior to a firing, and then may pass through a sequence of unstable states as its internal petri net undergoes a series of transformations.

An external event in a transition object corresponds to the firing of that transition, triggered by tokens being placed at its input places.

• a token object may be considered to have a passive relationship with its environment. The exported fields and functions of a token object can be accessed by the environment without changing the state of the token object. If a token object contains a petri net component which can exchange tokens with its environment, then that exchange must occur synchronously. Both the environment and the target token object must be prepared to exchange message tokens in a synchronous manner.

An external event in a token object A corresponds to the synchronous transmission of a token B into that token object A.

Internal events correspond to the internal transformations which occur when the internal petri net of a LOOPN++ object fires its transitions.

Well-defined event

A well-defined event corresponds to a time point in which only one transition in a LOOPN++ specification is capable of firing, and does fire. The state changes effected by that transition firing can be determined from the state of the input places to that transition and the details of the transition specification.

Poorly defined event

A poorly defined event corresponds to a conflict situation when there is a non-deterministic firing sequence. Such a firing can occur when several transitions are capable of firing, and the semantics of LOOPN++ define that the choice of which transition to fire is random. A specific example occurs when two transitions have the same input place(s) and both transitions are capable of firing. The firing of one of these transitions may subsequently inhibit the other transition from firing.

Class

A class corresponds to a LOOPN++ class.

Kind

A kind corresponds to a subset of LOOPN++ classes.

The above discussion covers all ontological constructs listed in table 1. With the reservations expressed about the modelling of a state law, the LOOPN++ modelling scheme may be considered to be ontologically complete.

4.2. Ontological Clarity

Wand and Weber (Wand and Weber 1993) use the term ontological clarity to refer to the degree to which each design construct in a modelling scheme is semantically clean, and is not subject to misinterpretation as to the meaning of that construct. They distinguish three situations in
which constructs may lack clarity: construct overload, construct redundancy and construct excess.

**Construct overload**

Construct overloads occurs when one design construct corresponds to two or more ontological constructs.

The class structure in LOOPN++ may be considered to suffer from construct overload when compared with conventional petri nets. Within the one design construct of a class, there is the provision to model active objects (i.e., transition objects) and passive objects (i.e., token and place objects). This is contrary to conventional petri nets where the active and passive constructs are strictly differentiated.

However, when viewed with respect to ontological constructs, LOOPN++ objects are simply things. It is considered a significant advantage, both syntactically and semantically, that LOOPN++ has a single unified class hierarchy in which all objects are expressed with the same syntactic construct. All objects can contain both passive (i.e., data or fields) components and active components (i.e., code or petri nets) in accordance with the desirable encapsulation of code and data in object-oriented schemes. The differentiation between a token object and a transition object is based solely on the mode of interaction between the object and its environment.

When interpreting ontological constructs one is faced with the machine-materials duality (Kreuzer 1986). That is, which components will be modelled as actors (or machines) that play an active role in manipulating information, and which components will be modelled as passive data (or materials) to be manipulated by the actors. It is often fruitful to consider switching roles, so that the former materials become actors which act upon and control the former machines. In conventional petri net modelling schemes the machine-material roles needs to be decided at a very early stage, due to the differentiation of control and data structures within such schemes. In LOOPN++ one can develop an object model in which choices of active versus passive roles of objects can be deferred until the internal specifications of classes are developed. Hence the construct overload associated with LOOPN++ classes can be seen as an advantage in developing flexible modelling specifications.

The construct overload associated with LOOPN++ classes is an example of a design construct which abstracts from the ontological constructs. As suggested by Wand and Weber (Wand and Weber 1993), this situation has lead to the introduction of the additional syntactic structure, the transition (see section 3.5) as a special case of a class definition.

**Construct redundancy**

Construct redundancy occurs when two or more design constructs are used to represent a single ontological construct.

A design goal of LOOPN++ has been to keep the number of design constructs to a minimum. An example is the provision of a single class construct to representing all things. In the previously proposed system, LOOPN, a distinction was made between transition objects and token objects. This may be considered as redundant modelling of the thing construct.

There are no cases of construct redundancy in the LOOPN++ scheme.

The use of abstract design constructs, such as classes and actions, in LOOPN++, is intended to provide the basic building blocks on which petri net schemes could be constructed. These basic design constructs are not themselves specific to petri net modelling schemes. This approach does require the designer who is familiar with petri net models to consider how the design constructs provided in LOOPN++ can be employed to build conventional petri net models. Research is being undertaking by the authors into appropriate system development methodologies for use with LOOPN++. 
Construct Excess

Construct excess occurs when a design construct does not correspond to any ontological construct.

LOOPN++ has well defined semantics, expressed in terms of coloured petri nets. This permits LOOPN++ specifications to be executed as conventional programs. Hence specific interpretations are placed on the way in which various operations in LOOPN++ are performed. For example, the firing of a transition and the obtaining of tokens from the input places of a transition. One may consider that this is a form of construct excess in that a specific mode of message interchange is being specified, which may not be in accord with the designer's ontological view of communication. This aspect of the LOOPN++ is part of its inheritance from a petri net modelling scheme and cannot be avoided in the LOOPN++ specification scheme without adopting an alternative model for its operational semantics.

5. Conclusions

This paper has presented a discussion of the current development of high level petri nets, and has presented a modelling scheme which fully integrates object-oriented structuring into a petri net specification system. The style of specification of LOOPN++ makes it convenient to model a wide range of systems, especially highly concurrent information systems which contain many levels of modelling. LOOPN++ can be employed as a formal specification scheme for the precise description of information systems.

The semantics of LOOPN++ has been defined in terms of the existing Design CPN petri net model. Further, LOOPN++ can be interpreted and so provides a vehicle for rapid prototyping of systems specifications. The use of object-oriented constructs permits the accumulation of extensible class libraries and the reuse of existing software components.

The key features of LOOPN++ which make it well suited to the dynamic modelling of information system are:

- the ability to define complex state laws within an object and so protect that object from external events which would otherwise result in the object entering an invalid state.
- the specification of lawful transformations as an intrinsic property of transition objects.
- the explicit display of all control coupled relationships via a graphical representation of a LOOPN++ model.
- system decomposition which is based on a single class hierarchy and well established rules for object composition. LOOPN++ permits the designer to decompose both active object (ie transitions) and passive objects (ie tokens), depending on which view is most convenient. This makes LOOPN++ well suited as the specification language for a wide variety of system development methodologies.

The constructs of LOOPN++ have been evaluated against the criteria of ontological completeness and ontological clarity. LOOPN++ does not fully satisfy the requirements of ontological clarity, but it can be seen to perform better than many other modelling techniques, including conventional petri nets.

While Wand and Weber have addressed the issues of ontological completeness and clarity, they have not considered the ability of an information systems design grammar to support one or more information systems development methodologies. The complexity of object-based design requires multiple stages, at which different ontological constructs are identified and interpreted (Henderson-Sellers 1992, Keen and Lakos 1993). The early stages of such systems development methodologies focus on building object-information and object-role models,
independent of implementation details. The design of LOOPN++ has considered the support required in the language for systems development, and has provided a single class hierarchy, so that classes may be introduced to model ontological things. It is then possible to discuss the relationships and roles of objects of these classes, without the constraints of a programming style. This is not the case with conventional petri net specification schemes in which the differentiation between control and data structures needs to be addressed at a very early stage in the design process.

The development of LOOPN++ is an ongoing project which aims to fully implement the language and provide efficient execution, and convenient petri net design and analysis tools.

References


A general formalism to describe processes in an organization should allow for the expression of real-time constraints. A real-time object specification logic (RTOSL) is motivated and presented. The logic is a linear-time point-based formalism with real-numbered time and a (syntactic) variant of the explicit-clock-variable approach. The main difference to previous real-time extensions of temporal logic is that this one is embedded in a logic for the specification of object behavior. The object paradigm has turned out to be advantageous to model arbitrary facets of organizations. RTOSL is on the one hand an extension of the Object Specification Logic presented in (A. Sernadas et al. 1992), on the other hand it is an application of approaches from the field of real-time temporal logic.

1. Introduction

In this paper a logic to specify and reason about objects with real-time features is presented. With such a formalism arbitrary dynamic aspects of organizations can be modelled. On the one hand, this framework, called RTOSL ('Real-Time Object Specification Logic'), is an extension of the object specification logic OSL in (A. Sernadas et al. 1992). OSL allows the description of...
objects' structure and behavior. Its first-order version provides the facility of aggregating objects
of the same type in classes. Furthermore, interaction between different objects can be described,
and there is also a subtyping mechanism. OSL, however, has not been designed with the objective
to capture hard real-time constraints. Sample constraints on actions might be 'Action A lasts
three time units at the most.', 'Action B begins five time units after the end of action C at
the latest', 'Action D happens at least three times in 20 time units.', 'Action E starts at 5 p.m.' and
so on. RTOSL, the logic to be presented in this paper, has been designed with the objective that
hard real-time constraints can also be expressed. Expressing real-time constraints must be
doable in order to capture all facets of activities in systems and organizations (cf. Koymans

On the other hand, there already exists a variety of real-time extensions of temporal logic.
(Emerson 1990; Alur and Henzinger 1992) give an overview of existing approaches, together
with further references. RTOSL is an application of concepts from this area. It is a linear-time,
point-based formalism with real-numbered time and a syntactic variant of the 'explicit-clock-
variable'-approach (Alur and Henzinger 1992; Pnueli and Harel 1988; Ostroff 1989; Ostroff
1990). The underlying structure is a sequence of points of time.

As opposed to previous approaches, with this formalism the means to specify real-time
constraints are embedded in a formalism for object behavior specification. The following three
features of our logic are fundamental prerequisites for its high expressiveness:

• References to absolute time are possible (cf. Alur and Henzinger 1992; Pnueli and Harel

• Rather than discussing whether all actions have a duration of time or may be instantaneous
our framework allows the specification of both instantaneous actions and actions with a
duration of time. In the second case, the begin instant of that action and its end instant are
different. Hence, each (real-world) action is related to two instantaneous actions: The begin
action occurs at the begin instant of that action, the end action at the end instant.

• The distinction between actions and action instances is necessary. For example, if 'battle' is an
action then 'Battle of Hastings', 'Battle of Waterloo' are examples of action instances. In
RTOSL, however, action instances are not discerned by names. Instead, they are numbered,
and the numbering reflects their temporal order. With that distinction, it is possible to express
constraints such as 'Action c must occur n times within t time units.'

With only the first item the expressive power of the formalism would be smaller. It seems that in
some of the related work cited previously the other two features of our logic are not dealt with.
These three items are, by the way, the main differences between OSL and RTOSL.

In the literature we are not aware of another specification mechanism for both object-
orientation and real-time using logic. There are formal specification methods for the object-

---

1 Another kind of (hard) real-time constraint occurs in real-time databases. Consider a database in
which temperature values are stored together with the point of time they were measured. A
typical absolute time constraint may be that for every point of time there must be a temperature
value measured one time unit before that point at the earliest and one time unit after that at
the latest. That kind of constraint typically occurs in process engineering applications. However, it can
be reduced to a constraint on actions (as the ones before) and we do not dwell on this here.
oriented approach. One of them is (Breu 1991); it also contains a summary of further formalisms. But they do not have a real-time facet and are not based on logic as our approach. Using logic, however, it is possible to reason about the specifications. A deductive system for RTOSL has been developed. In (Böhm and A. Sernadas 1993) this proof system is presented. In this paper, we give the precise definitions of our logic, i.e. the institution RTOSL is presented. An institution is a precise generalization of the informal conception of a logic system (Goguen and Burstall 1984) based on category theory (Adámek et al. 1990; Goldblatt 1979). Institutions are the right technical framework where the composition of specifications and theories can be discussed. Hence, they can be applied to study modularization of specifications. This, however, will not be laid out in this article. Besides that, the proofs that the logic is an institution is in the appendix. To better unveil the behavior-description mechanism of RTOSL, i.e. its real-time facet, we limit ourselves to the propositional fragment of RTOSL in this paper.

We are not aware of other work on real-time temporal logic that is institutional. (Koymans 1992), to give an example, is a relevant related work which is not. There, a more abstract general model of real-time, namely metric spaces, is used. Using the taxonomy from (Alur and Henzinger 1992), with the logic in (Koymans 1992) timing constraints are integrated into the language by means of bounded temporal operators, while with this logic this is accomplished with the explicit-clock-variable approach.

The remainder of this paper has the following structure: In the next section we give an introductory example. Section 3 contains all relevant definitions. Section 4 contains a brief summary. Most of the proofs that the logic is an institution are in the appendix.

2. Introductory Example

In the terminology used here, an object has observations and actions. Observations correspond more or less to the general notion of properties according to the object paradigm, actions correspond to methods. Within the context of RTOSL there is a differentiation between actions and action instances. An object's observations may not be modified by another object's actions. A modification of that kind can occur only mediately: That action must trigger an action of the object whose attribute is to be modified. This mechanism is called message passing.

An object's specification is composed of an interface and a set of axioms constraining the behavior. In the sequel, the interface is also referred to as signature. As an example consider a flip-flop with one stable state, i.e. its output may look like Figure 1. There is, of course, an endless number of examples from the area of organization or information system modeling, but this a fairly simple one.
This flip-flop has two observations: status=hi and status=lo. It must be part of the specification that the observations status=hi and status=lo are mutually exclusive. There are two actions: up and down. Only actions alter the observations. Clearly, the actions may only occur at particular instants of time, i.e. must be constrained by enabling conditions. It is differentiated between the actual occurrence of an action and its enabling. For instance, '◊ down' reads "down is possible." 'V_pup_n' reads "(The n-th instance of 'up' happens."). The triangle ('V') is the occurrence-operator, the diamond ('◊') the enabling-operator. The index 'p' at actions' occurrence-operator signifies that the action is instantaneous, i.e. has no duration of time. 'V_pup_n' would read "(The n-th instance of 'up' begins."). 'V_eup_n' would read "(The n-th instance of 'up' ends."). The other index (the one after the action symbols, in this example 'n') identifies the action instance. The following axioms, formulated in the propositional fragment of RTOSL, complete the specification of that kind of flip-flop.

1. \( (V_{pup_n}\to (X \diamond status=hi)) \)
2. \( (V_{pdown_n}\to (X \diamond status=lo)) \)
3. \( (◊ down \Rightarrow ◊ status=hi) \)
4. \( (◊ up \Leftrightarrow ◊ status=lo) \)
5. \( (◊ status=hi \land ◊ status=lo) \)
6. \( (V_{pup_n}:t \Rightarrow (\exists n'(V_{pdown_n}':t+t_1) \land (0<t_2<t_1 \Rightarrow \neg V_{pdown_n}':t+t_2))) \)
7. \( (V_{pup_n} \Rightarrow V_{eup_n}) \)
8. \( (V_{pdown_n} \Rightarrow V_{eup_n}) \)

Axiom (1) states that if (the n-th instance of) up is happening then in the next situation the observation status=hi is possible, i.e. status is hi. 'X' is the operator referring to the next state. So axioms (1) and (2) state an effect of the respective action. Free variables in a formula are quantified universally by convention. n, in this case, is an example of a free variable. Due to axiom (3) the enabling of status=hi is a requirement for the occurrence of down. down can only happen if status is hi. The converse ('⇐') is not true, because down can only happen when status is hi and a certain amount of time (t1) has elapsed since it became hi. At any instant of time status is either hi or lo, but never both of them (5). In the full version of RTOSL axioms like this are superfluous because with that framework observations are functional. The temporal sequence of the actions is laid down in (6): The formula 'V_pcn:t' reads '(The n-th instance of) c
happens at instant $t$. According to this notation, if (the $n$-th instance of) up occurs at instant $t$ then $t_1$ time units later (an instance of) down occurs. The second part of the axiom states that down does not happen before that. Axioms (7) and (8) state that the actions are instantaneous: The axioms state that the begin always coincides with the end.

There are a couple of general axioms that we expect to hold for every interpretation structure. They are listed in chapter 3.4. They need not be included in proper specifications due to their universal character.

3. Real-Time Object Specification Logic

A signature contains a set of action symbols and a set of observation symbols. Thus it is taken into account that actions are treated in a way different from observations. The fundamental idea to distinguish between event enabling and event occurrence is taken over from (A. Sernadas et al. 1992). An interpretation structure is a sequence of positions, and each position relates to the set of events that are occurring and the ones that are enabled. Moreover, each position is associated with a point of real-time. There are a couple of universal properties that must hold for each interpretation structure.

3.1 The Data Environment

A data signature is a pair $\Sigma=(S, OP)$ where $S$ is a set and $OP$ is an $S \times S$-indexed family of sets. The elements of $S$ are called sorts. For each $(w,s) \in S \times S$, $OP_{w,s}$ is the set of operation symbols with parameter sort list $w$ and result sort $s$. Let an $S$-indexed family $Z$ of sets of variables be given. For each $s \in S$, $T_{\Sigma,s}(Z)$ denotes the set of terms of sort $s$ that may be inductively built with the variables in $Z$ over $\Sigma$, i.e.:
- $z \in T_{\Sigma,s}(Z)$ provided that $z \in Z_s$;
- $h(t_1, \ldots, t_n) \in T_{\Sigma,s}(Z)$ provided that $h \in OP_{s_1 \ldots s_n,s}$ and for $i=1, \ldots, n$, $t_i \in T_{\Sigma,s_i}(Z)$.

An equation over a signature $\Sigma$ is a triple $(Z, t, t')$ where $Z$ is an $S$-indexed family of sets and $t, t' \in T_{\Sigma,s}(Z)$ for some $s \in S$. A data specification is a pair $\text{spec}= (\Sigma, \text{Eq})$ where $\Sigma$ is a data signature and $\text{Eq}$ is a set of equations over $\Sigma$.

In connection with the propositional fragment of RTOSL, we assume as given a specification $\text{spec}_{n,r}= (\Sigma_{n,r}, \text{Eq}_{n,r})$ with $\Sigma_{n,r}= (\{\text{nat}, \text{real}\}, OP_{n,r})$. $OP_{n,r}$ contains inter alia:

- $(OP_{n,r})_{\text{nat}; \text{nat}} = \{\oplus\}$
- $(OP_{n,r})_{\text{nat}; \text{nat}} = \{\text{succ}\}$
- $(OP_{n,r})_{\text{nat}; \text{nat}} = \{\text{+n}\}$
- $(OP_{n,r})_{\text{real}; \text{real}} = \{\text{+r}\}$

Terms serve the following purposes:
• Terms of sort real stand for instants of absolute time. The type real is our model of time.
• Terms of sort nat are used to identify action instances.

In this context for methodological reasons it is forbidden to build terms of sort nat from terms of sort real and vice versa. Since terms of type real are only used for the purpose identified above, and terms of type nat for another well-defined purpose, it is not necessary to allow terms of one sort to be built from terms of the other sort. - We concede that the denotation 'propositional' is misleading to a degree since we do allow terms, variables and quantification. However, because parameterization of predicate symbols is not part of the notions presented in this paper the designation 'propositional' is justified.

3.2 The Signature

Definition 3.2.1 A signature is a pair \( \Phi = (OBS, ACT) \) where OBS and ACT are sets.

The elements of OBS are called observation symbols, the elements of ACT action symbols. An event symbol is either an observation symbol or an action symbol. EVT denotes the set of event symbols, i.e. EVT=OBS \cup ACT.

Definition 3.2.2 A signature morphism \( \sigma: (OBS, ACT) \rightarrow (OBS', ACT') \) is a pair \( (\sigma_{OBS}, \sigma_{ACT}) \):
• \( \sigma_{OBS} \) is a map of type OBS \( \rightarrow \) OBS;
• \( \sigma_{ACT} \) is a map of type ACT \( \rightarrow \) ACT.

In the introductory example, \( \Phi_{\text{flip-flop}} = (OBS_{\text{flip-flop}}, ACT_{\text{flip-flop}}) \) with OBS_{\text{flip-flop}}={status=hi, status=lo}, ACT_{\text{flip-flop}}={up, down}. Now, consider a gate with three inputs A, B and C. After an input signal on either A, B or C it is displayed for \( t_1 \) time units at which input the signal has occurred, e.g. by turning on either a red light, a green one or a blue one. If a light is on, then, during these \( t_1 \) time units, other input signals are not considered. A gate such as this one might be used in a game show to convey which candidate out of three has reacted first, to give an intuitive example. The corresponding signature might be as follows: \( \Phi_{\text{display}} = (OBS_{\text{display}}, ACT_{\text{display}}) \) with OBS_{\text{display}}={A, B, C, none}, and ACT_{\text{display}}={display_A, display_B, display_C, clear}. Note that in this article the propositional fragment of the logic is described. With the full version parameterization is possible. In this case, an arbitrary number of inputs would be allowed. - A morphism between these signatures would be as follows: \( \sigma_{OBS} \) would map A, B, and C to status=hi and none to status=lo. \( \sigma_{ACT} \) would map display_A, display_B, and display_C to up and clear to down.

Proposition 3.2.3 The signatures together with the signature morphisms make up a category.
treated differently: In the introduction of this section we have mentioned three features of our logic that are necessary to describe real-time object behavior:

- Relating actions’ occurrences to instants of absolute time,
- distinction between begin and end instants of an action occurrence,
- action instances.

These concepts have been applied only to actions, not to observations. The reason is that observations do not alter the objects’ states. Hence, referring to individual observation instances, relating their occurrences to time instances etc. has no relevance.

3.3 The Language

**Definition 3.3.1** The propositional, linear real-time language $L_{\Phi}$ induced by $\Phi=(OBS, ACT)$ is inductively built as follows:

- $t_1=t_2$ if $t_1, t_2$ are both terms of type real or both of type nat;
- $\Diamond o, \Box o \in L_{\Phi}$ provided that $o \in OBS$;
- $\forall_p c_n.t, \forall_e c_n.t, \forall_b c_n.t : L_{\Phi}$ provided that $c \in ACT$ and $n$ is a term of type nat;
- $\forall_p c_n : t, \forall_e c_n : t, \forall_b c_n : t : L_{\Phi}$ provided that $c \in ACT$, $n$ is a term of type nat and $t$ a term of type real;
- $(-o) e L_{\Phi}$ provided that $o \in L_{\Phi}$;
- $(f \Rightarrow g) e L_{\Phi}$ provided that $f, g \in L_{\Phi}$;
- $(\forall x f) e L_{\Phi}$ provided that $f \in L_{\Phi}$ and $x$ is a variable either of type nat or of type real;
- $(X f) e L_{\Phi}$ provided that $f \in L_{\Phi}$;
- $(f \cup g), (f \cap g) e L_{\Phi}$ provided that $f, g \in L_{\Phi}$.

In the introductory example (1)-(8) are elements of the language $L_{\Phi_{flip_flop}}$ induced by $\Phi_{flip_flop}$. Formulae being elements of $L_{\Phi_{display}}$, which might be part of a display gate’s specification, are $(\forall p_{display} \Rightarrow (X \Diamond A)), (\forall p_{clear} \Rightarrow (X \Diamond none))$.

Occurrences of observations can be expressed in the language. To understand this, think of observations as activities, namely the activity of observing something. ‘$\forall status=hi$’ would read ‘The observation that status is hi is currently occurring’. It is nearby that such an observation does not change the object’s state.

We are now in the position to identify the features of the language that are included to model real-time behavior, as opposed to OSL:

- The actions’ occurrence-operator is indexed with either ’b’, ’e’ or ’p’. Index ’b’ means that the action’s occurrence is beginning, ’e’ that it is ending, respectively. ’p’ stands for ‘pointwise’. This means that the action is instantaneous, i.e. beginning and ending at the same instant of time.
- The index of the action symbol identifies an action instance. This is the only time within this framework when terms of type nat are used. The OSL-counterpart of the RTOSL-formula ‘$\forall g\in c$’ with $c$ being an action symbol is just ‘$\forall c$’. Namely, OSL does not have the concepts of begin/end instants and action instances.
**The formula 'VeCn:t' means that the n-th instance of action c begins, ends or occurs instantaneously (whatever is expressed by Oe (h, e, p1) at instant t. Since the type real is our model of time, t is a term of that type. Terms of type real are not used elsewhere in the framework. A formula of kind 'VeCn:t' does not have a counterpart in OSL.**

**Definition 3.3.2** The translation map \( \sigma: L_\Phi \rightarrow L_\Phi \) induced by a signature morphism \( \sigma: \Phi \rightarrow \Phi' \) is inductively defined as follows:

- \( \sigma((t_1=t_2)) = (t_1=t_2) \)
- \( \sigma((O o) = O oB oS(o), \sigma(V o) = V oB oS(o) \) provided that \( o \in O b s \)
- \( \sigma(V eC_n) = V eC_n e L_\Phi \) with \( \theta \in \{b, e, p\} \)
- \( \sigma((O e) = O A C t(c) \) provided that \( ce A C t \)
- \( \sigma(V eC_n:t) = V eC_n:t \) provided that \( VeC_n:t \in L_\Phi \) with \( \theta \in \{b, e, p\} \)
- \( \sigma(\neg f) = \neg \sigma(f) \)
- \( \sigma(f \circ g) = \sigma(f \circ g) \)
- \( \sigma((x f) \circ g) = (x \sigma(f) \circ g) \)
- \( \sigma(f \circ g) = (f \sigma(g)) \)

The translation map \( \sigma: L_\Phi \rightarrow L_\Phi \) maps the formulae \( (V p d i s p _A n \Rightarrow (X \circ A)) \) and \( (V p d i s p _B n \Rightarrow (X \circ B)) \) to \( (V p d i s p _n \Rightarrow (X \circ status=bi)) \), to continue the example.

**Definition/Proposition 3.3.3** The functor \( Gram \) maps each signature \( \Phi \) to the language \( L_\Phi \) and each signature morphism \( \sigma: \Phi \rightarrow \Phi' \) to the function \( Gram(\sigma): L_\Phi \rightarrow L_{\Phi'} \). 'Gram(\sigma)' maps each \( f \in L_\Phi \) to \( \sigma(\Phi f) \).

**Definition 3.3.4** A specification is a pair \( \text{spec}=(\Phi, Ax) \) where \( \Phi \) is a signature and \( Ax \subseteq L_\Phi \). In this context, the elements of \( Ax \) are called (local) axioms.

**OhioFlop**, together with the axioms from the previous section, is a specification. \( \Phi\text{Display}, \) together with the translation of these axioms induced by the above sample morphism, is another specification. Note that with this specification it is not forbidden that \( \text{Display}_A, \text{Display}_B \) or \( \text{Display}_C \) occur at the same time.

### 3.4 The Semantics

Terms of type \( n a t \) are interpreted in natural numbers, terms of type \( r e a l \) in real numbers. If \( t \) is a term then \( [t] \) denotes its interpretation in the natural/real numbers.

**Definition 3.4.1**. Let \( \Phi=(O b s, A c t) \) be a signature. Let \( PRD o b s, o := \{o | o \in O b s\} \), \( PRD o b s, v := \{vl | vl \in OBS\} \), \( PRD a c t, o := \{c | c \in A C T\} \), \( PRD a c t, v := \{vcn | vc \in A C T, \theta \in \{b, e, p\} \) and \( n \in N\} \), \( PRD o b s := PRD o b s, o \cup PRD o b s, v \), \( PRD a c t := PRD a c t, o \cup PRD a c t, v \) and \( PRD := PRD o b s \cup PRD a c t \). Furthermore, \( \lambda o b s, o \) is a function of type \( Z \rightarrow 2^{PRD o b s, o} \), \( \lambda o b s, v : Z \rightarrow 2^{PRD o b s, v} \), \( \lambda a c t, o : Z \rightarrow 2^{PRD a c t, o} \), \( \lambda a c t, v : Z \rightarrow 2^{PRD a c t, v} \), \( \lambda a c t : Z \rightarrow 2^{PRD} \) with \( \lambda a c t(z)=\lambda a c t, o(z) \cup \lambda a c t, v \)
(z) $\cup \lambda_{\text{OBS},c}(z)$ for all $z \in Z$, and $\lambda : Z \to 2^{\text{PRD}}$ with $\lambda(z) = \lambda_{\text{OBS},c}(z) \cup \lambda_{\text{ACT}}(z)$ for all $z \in Z$. An
interpretation structure over $\Phi$ is a pair $asp=(\lambda, \tau)$ where:

- $\lambda : Z \to 2^{\text{PRD}}$;
- $\tau : Z \to \mathbb{R}$ is a strictly monotonic function

such that the following universal properties hold:

(A1) If $\forall b \forall c \forall n \in \lambda_k$ then $\forall c \forall n \in \lambda_k'$ for some $k' \leq k$.
(A2) If $\forall b \forall c \forall n \in \lambda_k'$ then $\forall b \forall c \forall n \in \lambda_k'$ for some $k' \leq k$.
(A3) If $\forall b \forall c \forall n \in \lambda_k$ and $\forall b \forall c \forall n \in \lambda_k'$ then $k = k'$ (with $\theta \in \{b, e, p\}$).
(A4) If $\forall b \forall c \forall n \in \lambda_k$ then $\exists c \forall n \in \lambda_k$ if $\forall b \forall n \in \lambda_k$ then $\forall b \forall n \in \lambda_k$.
(A5) For any $n \in \mathbb{Z}$, if $\forall b \forall c \forall n \in \lambda_k$ then for any $n'$ with $1 \leq n' < n$ $\forall b \forall c \forall n \in \lambda_k$ for some $k' \leq k$.
(A6) If $\forall b \forall c \forall n \in \lambda_k$, then $\forall b \forall c \forall n \in \lambda_k'$ then
   (i) If $k < k'$ then there is $k^* \in \mathbb{Z}$ with $k < k^* < k'$ and an action symbol $c$ such that $\forall b \forall c \forall n \in \lambda_k^*$.
   (ii) If $k > k'$ then there is $k^* \in \mathbb{Z}$ with $k < k^* < k'$ and an action symbol $c$ such that $\forall b \forall c \forall n \in \lambda_k^*$.
(A7) $\forall b \forall c \forall n \in \lambda_k$ and $\forall b \forall c \forall n \in \lambda_k$ iff $\forall b \forall c \forall n \in \lambda_k$.
(A8) If $\forall b \forall c \forall n \in \lambda_k$ then $n \geq 1$.

In the sequel the arguments of $\tau$ and $\lambda$ will be referred to as positions.

If (an instance of) an action begins it will come to an end eventually (A1). Analogously, an
action instance cannot end without having begun at some instant before (A2). Action instances
are unique (A3). An action has to be enabled when an instance of it begins; analogously, an
observation has to be enabled when it occurs (A4). The instances' indexing reflects their order of
occurrence (A5). The so-called frame-rule states that events' enabling can only be altered by
actions. In more detail, a begin action only stops events from being enabled; an end action does
the converse (A6). An action is instantaneous if it begins and ends at the same instant of time
(A7). Finally, action instances are indexed with positive numbers (A8). These assertions hold
for any interpretation structure due to their universal character. So it would be redundant to
include axioms that can be derived from those properties in a proper specification.

Note that with a point-based logic it is possible to define the next state and refer to it using the
next-state operator $X$. With an interval-based logic we do not see how the next state could be
defined, making that kind of temporal logic unsuitable for the expression of various kinds of
conditions necessary for object-behavior specification.
Definition 3.4.2 Given interpretation structures $I_1=\langle \lambda_1, \tau_1 \rangle$, $I_2=\langle \lambda_2, \tau_2 \rangle$ over $\Phi$, we say that there is an interpretation-structure homomorphism $f: I_1 \rightarrow I_2$ iff there is a strictly monotonic function $f^*: \mathbb{Z} \rightarrow \mathbb{Z}$ where

1. $\lambda_1 \text{ ACT}_{\lambda_1}(k) \neq \emptyset \iff f^*(k)$ is defined;
2. $\lambda_2 \text{ ACT}_{\lambda_2}(k) \neq \emptyset \iff k \in f^*(\mathbb{Z})$;
3. $f^*(k)$ is defined $\Rightarrow \lambda_1 \text{ ACT}(k) = \lambda_2 \text{ ACT}(f^*(k))$ for any $k \in \mathbb{Z}$;
4. $f^*(k)$ is defined $\Rightarrow \tau_1(k) \leq \tau_2(f^*(k))$ for any $k \in \mathbb{Z}$.
The structure of an interpretation structure very much depends on the actions' occurrences: I.e. only the positions where actions occur are relevant for the homomorphism (cf. (1) and (2)). On the other hand, observations' occurrence does not have an impact on that structure. This is the reason why in (3) \( \lambda_{ACT} \) is considered instead of \( \lambda \). The structure of an interpretation structure is independent of the numbering of the positions. This is reflected in the definition of the interpretation-structure homomorphism. The definition also reflects the fact that the delay of events likewise does not alter that structure from our point of view.

Figure 2 contains three interpretation structures over \( \Phi_{flip\_flop} \). In this graphical notation the arrows represent the mapping of positions to points of time and to action-occurrence predicates. (Other predicates have been omitted in this graphical representation as event enabling can be inferred from action occurrence, and interpretation structure homomorphisms abstract from observation occurrence.) From (b) to (a) there is an interpretation structure homomorphism that is as follows: \( f^*(1)=2, f^*(2)=4 \) etc. In the other direction, there likewise is a homomorphism \( (f^*(2)=1, f^*(4)=2 \) etc.) On the other hand, there is a homomorphism from (a) to (c), but not in the other direction.

Obviously, the interpretation structures over \( \Phi \) together with the interpretation-structure homomorphisms over \( \Phi \) form a category: The identities are arrows in this category. The composition of two arrows is again an arrow in the category. The category will be referred to as \( ISt_\Phi \).

**Definition/Proposition 3.4.3** Given a signature morphism \( \sigma: \Phi \rightarrow \Phi' \), the functor \( \sigma^*: ISt_\Phi \rightarrow ISt_\Phi' \) is defined as follows:

- The reduct \((\lambda, \tau)\) of an interpretation structure \((\lambda', \tau')\) in \( ISt_\Phi' \) is the following:
  - \( \lambda \) is obtained from \( \lambda' \) by replacing every event symbol with its inverse image via \( \sigma \), i.e. for \( p' \in \lambda' \):
    - if \( p'=\text{\( \emptyset \)O}' \text{ with } \emptyset \in \text{OBS}' \text{ then } \lambda_k \text{ contains the set } \{ \emptyset \in \text{OBS}' -1(\emptyset) \}; \)
    - if \( p'=\text{\( \emptyset \)C}' \text{ with } c \in \text{ACT}' \text{ then } \lambda_k \text{ contains the set } \{ c \in \text{ACT}' -1(c) \}; \)
    - if \( p'=\text{\( \emptyset \)evt}' \text{ with } evt \in \text{ACT}' \text{ UOBS}' \text{ then } \lambda_k \text{ contains the set } \{ \text{\( \emptyset \)evt}' \in \text{OBS}' \text{ or } evt \in \text{ACT}' -1(evt) \text{ if } evt \in \text{ACT}' \}. \)
  - \( \tau \) is identical with \( \tau' \).
- Let \( f_1: (\lambda_1', \tau_1') \rightarrow (\lambda_2', \tau_2') \) be an interpretation-structure homomorphism. The reduct of \( f_1 \)' is the interpretation-structure homomorphism \( f_1^*: \sigma^*(\lambda_1', \tau_1') \rightarrow \sigma^*(\lambda_2', \tau_2') \) given by the map \( f^* \) being identical to \( f^{**} \).
Figure 3 contains an interpretation structure over $\Phi_{\text{display}}$, making use of that graphical notation again. The notation means that display_A, display_B and display_C occur at the same time. It is easy to see that the interpretation structure (a) from Figure 2 is the reduct of this one, based on the sample signature morphism from above.

**Definition/Proposition 3.4.4** The functor $\text{Sem}$ maps each signature $\Phi$ to the category $\text{IS}_\Phi$ and each signature morphism $\sigma: \Phi \rightarrow \Phi'$ to the functor $\sigma^*: \text{IS}_\Phi \rightarrow \text{IS}_{\Phi'}$.

Diagram 4 displays the relations between the different categories and functors introduced in this chapter.
3.5 The Satisfaction Relation and the Satisfaction Condition

Definition 3.5.1. Given an interpretation structure \( \gamma=\langle \text{OBS}, \text{ACT} \rangle \) over \( \Phi=(\text{OBS}, \text{ACT}) \), a function \( \text{asg} \) relating the variables of type \( \text{nat} \) and \( \text{real} \) to a value in \( \mathbb{N} \) and \( \mathbb{R} \), respectively, and a position \( k \in \mathbb{Z} \), the satisfaction by \( \gamma \) for \( \text{asg} \) at \( k \) is inductively defined as follows:

\[
\begin{align*}
(1) & \quad \gamma, \text{asg}, k (t_1=t_2) \iff [t_1]=[t_2]; \\
(2) & \quad \gamma, \text{asg}, k \circ \text{evt} \iff \circ \text{evt} \lambda_k \text{ for } \circ \text{evt} \in \text{EVT}; \\
(3) & \quad \gamma, \text{asg}, k \lambda \circ \text{OBS} \iff \lambda \circ \lambda_k \text{ for } \lambda \in \text{OBS}; \\
(4) & \quad \gamma, \text{asg}, k \forall \theta \in [n] \in \lambda_k \text{ for } \forall \theta \in \text{ACT} \text{ (for } \theta \in \{b, e, p\}); \\
(5) & \quad \gamma, \text{asg}, k \forall \theta \in [t] \iff \text{for some } k' \in \mathbb{Z} \forall \theta \in [n] \in \lambda_{k'} \text{ for } \forall \theta \in \text{ACT} \text{ and } [t]=t_{k'}; \\
(6) & \quad \gamma, \text{asg}, k (\neg f) \iff \neg \gamma, \text{asg}, k f; \\
(7) & \quad \gamma, \text{asg}, k (f \Rightarrow g) \iff \gamma, \text{asg}, k f \Rightarrow \gamma, \text{asg}, k g; \\
(8) & \quad \gamma, \text{asg}, k (\forall x f) \iff \gamma, \text{asg}, k f \text{ for any assignment } \gamma' \text{ x-equivalent to } \gamma; \\
(9) & \quad \gamma, \text{asg}, k (X f) \iff \gamma, \text{asg}, k+1 f; \\
(10) & \quad \gamma, \text{asg}, k (f \cup g) \iff \gamma, \text{asg}, k f \text{ for some } k' \geq k \text{ and } \gamma, \text{asg}, k f \text{ for every } k'' \text{ such that } k<k''<k'; \\
(11) & \quad \gamma, \text{asg}, k (Y f) \iff \gamma, \text{asg}, k-1 f; \\
\end{align*}
\]

That is, giving the same value as \( \text{asg} \) to every variable except possibly to variable \( x \).
\[ (12) \text{asp} \models_{\text{asp},k} (f \doteq g) \iff \text{asp} \models_{\text{asp},k'} g \text{ for some } k' \leq k \text{ and } \text{asp} \models_{\text{asp},k} f \text{ for every } k'' \text{ such that } k' < k'' \leq k. \]

**Definition 3.5.2.** Let \( \text{asp} = (\lambda, \tau) \) be an interpretation structure over \( \Phi = (\text{OBS}, \text{ACT}) \). We say that \( \text{asp} \) satisfies a formula \( f \in L_\Phi \), written \( \text{asp} \models f \), iff \( \text{asp} \models_{\text{asp},k} f \) for every assignment \( \text{asp} \) and every position \( k \in \mathbb{Z} \). We say that \( \text{asp} \) satisfies a specification \( \text{spec} = (\Phi, A_\lambda) \), written \( \text{asp} \models \text{spec} \), iff \( \text{asp} \models f \) for every formula \( f \in A_\lambda \).

**Definition 3.5.3.** Let \( \Phi = (\text{OBS}, \text{ACT}) \) be a signature. We say that a formula \( f \in L_\Phi \) is valid, written \( \models_{\Phi} f \), iff \( \text{asp} \models f \) for every interpretation structure \( \text{asp} \) over \( \Phi \).

**Theorem 3.5.4** For every \( \sigma : \Phi \to \Phi' \) in \( \text{Sig} \), \( \xi \in L_\Phi \), \( m' \in \text{IS}_\Phi \):
\[ m' \models_{\Phi'} (\text{Gram}(\sigma))(\xi) \iff (\text{Sem}(\sigma)(m')) \models_{\Phi} \xi. \]

For illustration purposes, consider our sample signature morphism. Let \( m' \) be the interpretation structure \( (a) \) from Figure 2, \( \xi = (\text{Vp}_{\text{display} \cdot \text{AN}} \Rightarrow (X \circ A)) \). \( (\text{Gram}(\sigma))(\xi) = (\text{Vp}_{\text{up}} \Rightarrow (X \circ \text{status} = \text{hi})) \), and \( m = (\text{Sem}(\sigma)(m')) \) is the interpretation structure from Figure 3. \( m \) satisfies \( (\text{Vp}_{\text{display} \cdot \text{AN}} \Rightarrow (X \circ A)) \), and \( m' \) satisfies \( (\text{Vp}_{\text{up}} \Rightarrow (X \circ \text{status} = \text{hi})) \).

4. Conclusions

We have presented the propositional fragment of a logic to specify real-time object behavior. On the signature level there is a differentiation between observations and actions that is due to different treatment in the language. Besides that, it is distinguished between event (i.e. observation or action) occurrence and enabling. The semantics is a set of life-cycles, the underlying structure is a sequence of positions. The map \( \tau \) takes each position to an instant of real time.

With the first-order version of the logic parameterization over individuals is possible. Moreover the notion of 'object class' is more distinct. Here, we did not present inheritance, i.e. both specialization inheritance and aggregation inheritance, for lack of space, i.e. for the sake of focusing on the behavioral facet with real-time features.

References

A Logic to Specify Real-Time Object Behavior

Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya, 159-174.


Acknowledgement. We have very much appreciated the comments of Cristina Sernadas.

Appendix

The presentation of the institution RTOSL is completed by definitions and proofs that have been omitted from the main text. First, an institution (Goguen and Burstall 1984) is defined as follows:

**Definition 3.1** An institution is a quadruple $(\Sigma, \text{Gram}, \text{Sem}, \vdash)$ such that

- $\Sigma$ is a category (the category of signatures);
- $\text{Gram}$ is a functor from $\Sigma$ to $\text{Set}$ (the grammar functor - the image of a signature $\Sigma$ is the language $L_\Sigma$);
- $\text{Sem}$ is a functor from $\Sigma$ to $\text{Cat}_{\text{UPP}}$ (the semantic functor - the image of a signature is the category of aspects for that signature, the arrows in this category will be called *aspect homomorphisms* within this context);
- $\vdash$ is a $|\Sigma|-$indexed family of relations: Each $\vdash_\Sigma$ is a subset of $1_{\text{Sem}(\Sigma)} \times \text{Gram}(\Sigma)$ (the satisfaction relation);
- for every $\alpha: \Sigma \rightarrow \Sigma'$ in $\Sigma$, $\alpha$ in $\text{Gram}(\Sigma)$, $m'$ in $1_{\text{Sem}(\Sigma')}$: $m' \vdash_{\Sigma'} \text{Gram}(\alpha)(\alpha)$ iff $\text{Sem}(\alpha)(m') \vdash_\Sigma \alpha$. □
**Definition/Proposition 3.3.3** The functor $\text{Gram}$ maps each signature $\Phi$ to the language $L_\Phi$ and each signature morphism $\sigma: \Phi \to \Phi'$ to the function $\text{Gram}(\sigma):L_\Phi \to L_{\Phi'}$. 'Gram($\sigma$)' maps each $f \in L_\Phi$ to $\sigma^\wedge(f)$.

Proof: To prove that $\text{Gram}$ is a functor from $\text{Sig}$ to $\text{Set}$ it must be verified:

(a) $\text{Gram}(\Phi)$ is a set.

(b) $\text{Gram}(\sigma)$ is a map.

(c) $\text{Gram}(\text{id}_\Phi)$ is the identity on $L_\Phi$. (id$_\Phi$ is the identity signature morphism on $\Phi$.)

(d) $\text{Gram}(\sigma \circ \sigma') = \text{Gram}(\sigma) \circ \text{Gram}(\sigma')$.

(a) and (b) are trivial.

(c) Clearly $\text{Gram}$ preserves the identity. (id$_\Phi : \Phi \to \Phi$ is mapped to id$_{L_\Phi} : L_\Phi \to L_\Phi$.)

(d) This becomes obvious by induction on $\sigma^\wedge$.

**Definition/Proposition 3.4.3** Given a signature morphism $\sigma: \Phi \to \Phi'$, the functor $\sigma^\#: \text{IS}_\Phi \to \text{IS}_{\Phi'}$ is defined as follows:

- The reduct $(\lambda, \tau)$ of an interpretation structure $(\lambda', \tau') \in \text{IS}_{\Phi'}$ is the following:
  - $\lambda$ is obtained from $\lambda'$ by replacing every event symbol with its inverse image via $\sigma$, i.e. for $p' \in \lambda'_k$.
    - if $p' = \text{Vo}'$ with $o' \in \text{OBS}'$ then $\lambda_k$ contains the set $\{\text{Vo} | o \in \sigma \text{OBS}'^{-1}(o')\}$.
    - if $p' = \text{Vc}n'$ with $c' \in \text{ACT}'$ then $\lambda_k$ contains the set $\{\text{Vc}n | c \in \sigma \text{ACT}'^{-1}(c')\}$.
    - if $p' = \text{evt}'$ with $\text{evt} \in \text{ACT}'$ $\cup \text{OBS}'$ then $\lambda_k$ contains the set $\{\text{evt} | o \in \sigma \text{OBS}'^{-1}(\text{evt}')\}$ if $\text{evt} \in \text{OBS}'$ or $\text{evt} \in \sigma \text{ACT}'^{-1}(\text{evt}')$ if $\text{evt} \in \text{ACT}'$.
  - $\tau$ is identical with $\tau'$.

- Let $f: (\lambda_1', \tau_1') \to (\lambda_2', \tau_2')$ be an interpretation-structure homomorphism. The reduct of $f$ is the interpretation-structure homomorphism $f^\#: (\sigma^\#: (\lambda_1', \tau_1')) \to (\sigma^\#: (\lambda_2', \tau_2'))$ given by the map $f^*$ being identical to $f^\wedge$.

It has to be shown that

(A) The reduct of an interpretation structure in $\text{IS}_{\Phi'}$ is an interpretation structure in $\text{IS}_{\Phi}$.

(B) The reduct of an interpretation structure homomorphism in $\text{IS}_{\Phi'}$ is an interpretation structure homomorphism in $\text{IS}_{\Phi}$.

(C) $\sigma^\#$ is a functor.

(A) Clearly, $\lambda$ is of the correct type.

- Trivially, $\tau$ is strictly monotonic.
- The general axioms hold for $(\lambda, \tau)$, given that they hold for $(\lambda', \tau')$.

(1) Assume that there is $V_{b'c'n'} \in \lambda'_k$. From the reduct's definition it follows that $V_{b'c'n''} \in \lambda'_k$. (A1) holds for $(\lambda', \tau')$, thus $V_{b'c'n''} \in \lambda'_k$ for some $k \geq k'$. The definition of the reduct yields $V_{b'c'n''} \in \lambda'_k$, i.e. $V_{b'c'n'} \in \lambda'_k$.

(2) (2) is shown analogously to (1).

(3) Let us assume that $V_{b'c'n'} \in \lambda'_k$, $\lambda'_k'$ for $k < k'$. Then, analogous to (1), $V_{b'c'n''} \in \lambda'_k'$, $\lambda'_k$ for $k = k'$. (A3), however, states that $k = k'$ must hold. So the assumption is not true.

(4) The argumentation is as in (1).

(5) (5) is proven like (1). Instead of the indices $(b, c)$ one deals with indices $(n, n')$. □
(6) Suppose $\circ \circ \lambda_k$ and $\circ \circ \lambda_k'$. W.l.o.g. $k < k'$. Then $\circ (\text{OBS}(o)) \in \lambda_k'$ and $\circ (\text{OBS}(o)) \in \lambda_k'$. Due to (A6) there must be a begin action in between. The rest is as before.

(7) This is shown as (1).

(8) This is true because $\sigma^\#$ does not alter terms of type nat.

(B) $f^*$ is identical to $f^\#$. Hence, (1)-(4) are fulfilled for the reduct if they are fulfilled for $f_1^*$, $f_2^*$ and $f^\#$.

(C) We show that $\sigma^\#(f_1; f_2) = \sigma^\#(f_1); \sigma^\#(f_2)$ (with $f_1; (\lambda_1', \tau_1') \rightarrow (\lambda_2', \tau_2')$, $f_2; (\lambda_2', \tau_2') \rightarrow (\lambda_3', \tau_3')$).

This is true because the $\tau$'s are not altered by $\sigma^\#$, and from $f_1^*$ and $f_2^*$ $(f_1; f_2)^*$ is obtained unambiguously.

**Definition/Proposition 3.4.4** The functor $\text{Sem}$ maps each signature $\Phi$ to the category $\text{IS}_\Phi$ and each signature morphism $\sigma: \Phi \rightarrow \Phi'$ to the functor $\sigma^\#: \text{IS}_\Phi \rightarrow \text{IS}_{\Phi'}$.

**Proof:** $\text{Sem}$ is a functor because

- $\text{Sem}(\text{id}_\Phi) = \text{id}_\Phi^\#$ with $\text{id}_\Phi^\# : \text{IS}_\Phi \rightarrow \text{IS}_\Phi$, and $\text{id}_\Phi^\#$ obviously is the identity on $\text{IS}_\Phi$.
- $\text{Sem}(\sigma; \sigma') = \text{Sem}(\sigma); \text{Sem}(\sigma')$:
  
  We look at signature morphisms $\sigma: \Phi \rightarrow \Phi'$, $\sigma': \Phi' \rightarrow \Phi''$ and their composition $\sigma''$.

  First, let $\text{asp}'' = (\lambda'', \tau'')$ be an interpretation structure over $\Phi''$. We show that $(\text{Sem}(\sigma''))((\text{asp}'')) = (\text{Sem}(\sigma'))((\text{Sem}(\sigma''))((\text{asp}''))).

  $\lambda$ obtained via $\text{Sem}(\sigma')$, $\text{Sem}(\sigma)$ is the same as $\lambda$ obtained via $\text{Sem}(\sigma'')$: W.l.o.g. suppose that we have obtained an $\circ o$ in $\lambda_k$ via $\text{Sem}(\sigma')$, $\text{Sem}(\sigma)$. Because $o'' = o^\circ \text{OBS}(o) = o^\circ \text{OBS}(o)$, it follows that we obtain $\circ o$ in $\lambda_k$ also via $\text{Sem}(\sigma'')$. The other direction is analogous.

  Second, let $f_1''$ be an interpretation structure homomorphism over $\Phi''$. The images via $\text{Sem}(\sigma)$, $\text{Sem}(\sigma')$ and via $\text{Sem}(\sigma'')$ are trivially the same.

**Theorem 3.5.4** For every $\sigma: \Phi \rightarrow \Phi'$ in $\text{Sig}$, $x \in L_\Phi$, $m' \in \text{IS}_{\Phi'}$:

$m' \vdash_{\sigma'} (\text{Gram}(\sigma))(x)$ iff $(\text{Sem}(\sigma))(m') \vdash \sigma(x)$.

**Proof:** The proof is done inductively over the structure of $x$. In the following, $x$ is $(\lambda', \tau')$, $\text{asp} = (\text{Sem}(\sigma))(\text{asp}') = (\lambda, \tau)$. The numbering is in accordance with definition 3.5.1.

(1) Formulae of kind $(t_1 = t_2)$ do not impose any problem because the interpretation of terms (either of type nat or of type real) is independent of the interpretation structure.

(2) Let $x = o$. Then $x = \circ \text{OBS}(o)$. $\text{asp} \vdash_{\text{asp}^k} \circ \text{OBS}(o)$ iff $\circ \text{OBS}(o) \in \lambda'_k$. Due to the definition of $\sigma^\# = \text{Sem}(\sigma)$, $\circ \text{OBS}(o) \in \lambda'_k$ iff $o \in \lambda_k$. Because $\circ o \in \lambda_k$ iff $\text{asp} \vdash_{\text{asp}^k} \circ o$, $\text{asp} \vdash_{\text{asp}^k} \circ \text{OBS}(o)$ iff $\text{asp} \vdash_{\text{asp}^k} \circ o$.

(3), (4) The proof is the analogous to the one for (2).

(5) The proof follows from (1) and (4). Recall that the instant at which an action happens is modified neither by the grammar functor nor by the semantics functor.

(6)-(12) Once the equivalence has been shown for $f$, $g$ it also holds for $\neg f$.□
TEMPO:
A SUPPORT FOR THE MODELING OF OBJECTS WITH DYNAMIC BEHAVIOR

Noureddine Belkhatir & Walcélio L. Melo
LGI
BP 53
38041 Grenoble France
e-mail: {belkhatir, melo}@imag.fr

Abstract

Recent developments have shown the need for an integrated view of the large-scale software development environment that takes account of how software products are managed and produced. To fulfill these requirements, the activities carried out during software development must be managed. In this paper, we describe the work we are carrying to support software process definition and enacting.

We shall discuss an approach to process modeling known as an object-oriented software process modeling language. Special attention is paid to how object-oriented concepts, the role concept, and trigger rules can be integrated to describe software process models.

1 Introduction

There is broad agreement that software engineering environments (SEE's) should provide explicit support for capturing and controlling software processes. Software processes should be represented explicitly and, in part, automatically executed by a SEE [16]. To satisfy this requirement, several research programs have explored data integration and centralized control using integrating platforms. These platforms provide support for product structuring, versioning, software configuration, and other engineering processes. On the basis of this experience, the entity-relationship-attribute data model (ERA), extended with Object-Oriented (OO) concepts, has been successful used as a conceptual framework for process definition. Nevertheless, although the basic OO concepts are a good starting point, they alone are not sufficient for capturing all the complexity of software engineering processes and their evolution. We claim that major capabilities such as multiple object behavior and modeling of activities with long duration, including long events and the time concept, need to be added to such a conceptual framework.
We have addressed these problems in our work in the framework of the Adele project. Adele, which was initially a revision control system, has been extended with user-defined entities and relation types to support the definition and control of large software systems. To make it possible to support software processes, the Adele language has been extended with event-condition (ECA) rules, and the Adele kernel has been extended with a trigger mechanism. This new system is called Adele 2 [3]. Adele 2 is now a commercial software product and is used for automatic software configuration activities in various European companies, such as Matra-Space, and ESPRIT projects, such as REBOOT. On the basis of our experience working with Adele 2, we have concluded the O.O. data model, even when integrated with ECA rules, is not sufficient for defining software process models. Once ECA rules are fragmented among data and relation types, it becomes difficult to control process enacting and to manage changes in processes. To overcome such drawbacks, we have initiated the Tempo project [5] to support software development processes. Tempo has enable an accurate set of software activities to be aggregated in process types and the static and behavioral description of objects, manipulated by such activities, to be re-defined according to their roles in a process step. As time plays an important role in any SEE, because we deal with long transactions, we have also studied how temporal information can be represented and how it can be exploited in the software process evolution. As we shall show later, we have decided to extend our ECA formalism with temporal logical operators. We have modified the Tempo process engine to make it possible to interpret temporal ECA rules. This paper describes these extensions and shows examples of how they are used for modeling, evolving, and enacting software processes.

1.1 Rationale

Adele/Tempo is a Process-Oriented Software Development Environment (POSE) [6] that focuses on the following capabilities:

- management of resources shared by a team for enforcing cooperative work, by providing objects with roles;
- activity coordination and traceability of activity execution, by providing activity management.

1.1.1 Object roles

Using object-oriented technology, we can model software process steps (or sub-processes) using complex active objects. As software objects associated with sub-processes also provide operations (methods), the combination of these two kinds of facilities could be used to describe statically (process model) and dynamically (process enactment) the software processes of a particular environment or company.

As the only structuring concept supplied by the traditional O.O. paradigm is the classification concept, however this model supports only one object behavior description, which can be refined by specialization using the class structure. All applications are supposed to conform to that structure and consequently to that behavior. This mono-behavioral belief, which reigns in the O.O. world, impacts directly on the difficulties involved in pro-
cess management. If we allow an object to behave differently depending on where, when, and how is it used, and enable it to be seen through different prisms, we claim it will be possible to better manage process changes. Using this approach, we could modify software processes by creating new ways to see and manipulate already-instantiated objects in harmony with old definitions.

1.1.2 Activity management

In an O.O. approach, objects interact by executing and exchanging messages via methods that support the active part. A number of mechanisms have been suggested for controlling and synchronizing interaction between methods. The most influential is the trigger mechanism based on Event-Condition-Action rules. ECA rules have been proposed to manage communication by extracting actions specific to method driving from method definition.

The trigger mechanism is most integrated with mechanisms supporting only short transaction (in the sense of database management systems). In the framework of software process, however activities have a long duration (long transactions). Thus, to coordinate method execution and control activity evolution, we need mechanisms that keep track of activity chaining and trace its execution. Therefore, we must introduce concepts and mechanisms for dealing with time. These requirements lead to:

1. a log database that contains capabilities for tracing object states;
2. capabilities to reason with temporal events.

1.2 Outline

Section 2 presents an overview of the Adele/Tempo system. Section 3 presents an overview of the Tempo software process modeling language. We present our conclusions in section 4.

2 An overview of Adele/Tempo

The Adele/Tempo system consists of three basic parts (see figure 1):

- A resource manager using Adele database as a persistent object base for storing objects and activities and for tracing the project’s progress. ADL-DB supports an entity-relationship data model which is extended with object-oriented concepts like inheritance, methods and encapsulation. Simple and composite objects with attributes and relationships can be described and managed. This component of Adele/Tempo architecture is responsible for the data integration according to conceptual model proposed by [21].

- An activity manager which is the responsible for the control integration in our platform. This activity manager is driven by Temporal-event-condition-action rules (TECA) and supported by a trigger mechanism.
A process manager which offers definition concepts for activity structuring by the process and role concepts. Process occurrences are supported by work environments (WE) wherein software activities are performed. The process manager, based on the activity manager, manages communication and synchronization between teams and between agents involved in a same project. It also controls the consistency of complex objects used simultaneously in different work environments by different software processes occurrences and agents. This component represents the conceptual component responsible for process integration in the Adele/Tempo architecture.

Figure 1: The kernel of Adele/Tempo environment.

3 The Tempo software process modeling language

Tempo [5] is an executable formalism for describing and enacting software process models. It uses an object-oriented approach extended by the addition of a multi-behavioral facility. The multi-behavioral facility is a major problem currently being researched in a wide range of fields. The problem arises when developing large, complex systems characterized by the presence of several agents, working on shared resources and using multiple representations and multiple development strategies. In this context we need a way of expressing relationships between multiple viewpoints.

There are three sides to our approach:

- modeling of software activities by software process types;
- analysis of the various viewpoints and software component life cycle states using the role concept;
- describing software temporal constraints by temporal-event-condition action rules (triggers rules). ECA rules are extended, using a temporal modality, to support long transactions (long duration activities). The temporal modality is applied to events, and it allows reasoning in relation to past activities.

3.1 The Adele data model

The Adele data model is derived from an entity-association model and integrates object-oriented concepts [3]. The basic entities of the model are object type and relationship type. Each entity (object and relationship) possesses static (attributes) and dynamic (methods, event-condition-action temporal rules) properties. Relationships are binary.
The data model supports complex objects referred to as aggregates. An aggregate is an object linked to its components by relationships. For example, a Pascal module can consist of an interface and an implementation. Consequently, the Pascal module object can be represented as an object linked to two other objects by two types of relationship, possesses-interface and possesses-implementation. Aggregate semantics are defined by the dynamic properties of the relationship linking the aggregate to its components. The semantics are defined by the user; any aggregate can thus be defined using its own semantics and consistency constraints.

In Adele, a type is defined by an interface part, and an implementation part which describe type instance properties.

The notions of interface and implementation are similar to those used in programs written using languages such as ADA and MODULA, or indeed certain object-oriented languages. The interface part contains the type properties that are visible and are exported. The implementation part contains private properties and the implementation of visible methods.

3.2 Software process types: modelling software process models

TEMPO describes and executes software processes. A software process model of considerable size may thus be written by a group of various software process types. A software process type can aggregate other software process types.

For example, an activity to check a module design document comprises two sub-processes:

1. A sub-process that models the modification activity that makes changes to the design document.

2. A sub-process that models the revision activity that approves design document modifications that have been made.

MonitorDesign ISA PROCESS;
   CONTROL md;
       sub = ModifyDesign;
   CONTROL rd;
       sub = ReviewDesign;
END_OF MonitorDesign;
ModifyDesign ISA PROCESS;
   ATTRIBUTES
       begin_date = DATE := now();
       end_date = DATE;
       deadline = DATE;
   METHODS ... 
   RULES ... 
END_OF ModifyDesign;
The example above shows the software process type MonitorDesign, composed of the sub-processes ModifyDesign and ReviewDesign. The activity coordinating the module design document modification is represented by the MonitorDesign type. ModifyDesign is the type which describes the design document modification process, and ReviewDesign is for revising this modification.

It is possible, for every process type, to define attributes, methods, and temporal constraints by using the event-condition-action rules.

### 3.3 The temporal contraints

Due to the long life duration of software processes, we also need management mechanisms for time constraint and traceability. To support process evolution, we also need to be able to reason about execution sequences. Thus we have introduced temporal reasoning capability in Adelej/Tempo system to plan and schedule activities in software process. Temporal constraints can be used either to schedule activities or to aid synchronization and cooperation between activities. To integrate temporal constraints in the Tempo language, we are incorporating temporal features in the event-condition-action rules (ECA). After we have extended the trigger mechanism of the Adelej/Tempo system to support reasoning about time. This new formalism combines of standard ECA rules and temporal logic predicates. In this way, we are adding a new dimension to the Adelej/Tempo system for managing process evolution using temporal knowledge in the field of software process management.

#### 3.3.1 Temporal event-condition-action rules

Temporal contraints are described by temporal-event-condition-action (TECA) rules. TECA rules are similar to Alf/Pcte [8], Damokles [9], and HiPAC [12] trigger rules. Interpretation and execution of these rules are based on trigger mechanism integrated to the object management system of Adelej/Tempo [3]. For example:

```
ModifyDesign ISA PROCESS;
  ATTRIBUTES
    begin_date = DATE := now();
    end_date = DATE;
    deadline = DATE;
  METHODS
    continue_execution;
    ...
  RULES
  (1) AFTER WHEN deadline_arrived
      DO stop_execution;
  (2) PRE WHEN continue_execution
      IFPAST not deadline_changed
```
1. The rule described in line 1 specifies the design document modification activity must stop when the date foreseen has been reached.

2. The rule in line 2 states that resumption of the activity (it hasn’t been completed yet) first requires the termination date be changed.

### 3.3.2 TECA rules execution module

TECA rules are defined in the data model (not shown in this article) and in the software process module. They are inherent in the hierarchy of object types and software processes. In the data model, the TECA rules describe integrity limitations that are independent of the object’s usage context. On the other hand, these rules are used to express the software development strategy used in the software process model: order of activity execution, activity synchronization, and software resource usage limitations.

A TECA rule is expressed in the following manner: "WHEN temporal-event DO Method", where "temporal-event" is the temporal predicate expressing:

1. an event in the present state of the development environment (e.g., objects, tools and agents states) or
2. an event about the past state of the development environment which has been stored in the database of the Adele/Tempo system.

Method is an instruction sequence.

```plaintext
DEFEVENT delete_obj = [ !cmd = rmobj ] ;
```

The `delete_obj` event is defined in this example as being the event that survives whenever the current command (!cmd) is an object removal command (rmobj).

A method is a program written in simple, direct language similar to the Unix shell.

```plaintext
METHOD delete ;
   IF [state = stable] THEN ABORT
   ELSE "rmobj %name ";
END delete;
```

This method allows for object removal in an unstable state.

Triggers rules can be defined to control the execution of methods. Some triggers will be executed before the methods, acting as pre-conditions, others after the method execution, as post-conditions. Since triggers are (originally) intended to enforce consistency, any inconsistency found by a trigger must be able to undo (roll-back) the method execution. Thus for any method the following instructions will be executed:
PRE list of triggers
   Action (Method)
POST list of triggers

The whole execution is always a single transaction, even if the triggers or the methods send messages to other objects. The execution of a primitive ABORT, anywhere in a block (PRE/Action/POST) will undo everything that was done in this block. AFTER triggers are executed after transaction committing; they are used to execute actions when sure that the transaction succeeded, as for example sending notifications, or to execute new actions whose failure must not undo the main action.

If the transaction failed, ERROR triggers are executed.

Thus for each object and relation type, there are five blocks:

PRE  list of triggers
   METHOD list of methods
POST list of triggers
AFTER list of triggers
ERROR list of triggers

3.3.3 Related work

Other SEEs use ECA rules, such as AP5 [13], Alf/Pcte [8] and Appl/A [20]. Alf and Tempo provide four TECA rules execution modes (PRE, POST, AFTER and EXCEPTION) whereas AP5, Marvel, Triad and Appl/A support only one mode of execution, the mode POST. All these systems do not provide concepts for handling Temporary constraints.

Alf's event-condition-action rules are similar to those offered by Adele/Tempo. However, Alf does not have the concept of method. Such as in Marvel, all the actions must be defined by the operators (production rules according to the MASP formalism). The execution of an operator can trigger a forward chaining process in the user's private space (ASP in the Alf terminology). ECA rules are defined elsewhere and executed by another mechanism called "trigger". In Tempo, we adopted only one concept to define both the constraints on the execution of methods and the constraints on the utilization of objects, i.e, the TECA rules.

Some other systems like AP5 [13] and ODE [11] also, in a way, provide ECA rules concerning time. However, these systems do not allow the specification of conditions about past actions. They limit themselves to specifying that for example, some actions (mainly methods) must be executed at an absolute/particular time in the future, for example, every day at 9 a.m or tomorrow at 6 p.m. etc.
3.4 Object with roles

Multiple perspectives or viewpoints often occur during a software product life cycle. Several users treat objects concurrently, using different views of the objects with limited, controlled actions specific to their activity. These users, controlled by multiple development strategies, handle different models of the same product. A SEE should provide a work environment that can describe and control these various aspects.

Owing to the role concept, Tempo language allows each software process occurrence to have local constraints and properties for each object treated [4].

Roles have a defined type. A role type may reference different types of objects. This strategy allows the integration of various types of behaviour and properties, coming from different types of objects, within a single perspective. This strategy unifies thus the treatment of a heterogeneous set of objects. The advantage of this approach is that a set of object types with different static and dynamic characteristics can be viewed, using the role concept, during a specific software process execution step in a coherent, homogeneous fashion. This coherence is maintained by using the multiple heritage rules available in object-oriented models. The principal difference is based on the extent of the roles. At the definition level, a role type is viewed as the specialisation of the types it contains. However, at the instance level:

1. Objects created from a role are not included in the role’s specialised type extensions.
2. A subset of objects pertaining to these types may belong to the role.

A software process type may have several role types; a software process becomes a list of roles whereby each object type may have different roles. Consequently, two objects of the same type may be controlled differently within the same software process. At the same time, an object can play roles within different software processes. For example:

```
ReviewDesign ISA PROCESS;
  ROLE under_review;
    derived_from = specification_document;

  ROLE requested_change;
    derived_from = cc_request;

END_OF ReviewDesign;
```

3.5 Role discussion

One may claim that this kind of contextual behavior can be achieved by standard object-oriented techniques. Roles and type look similar. This raises the question: Can roles be implemented in terms of typing and sub-typing? Is the concept of role needed? We claim the role concept has the following properties:
1. Prevent type explosion.
A role, as well as a type, is a template applied to a set of instances sharing the same definition (static and behavioral). A given object instance can be simultaneously a member of different roles (classes). Both roles and types can be seen as a viewing mechanism since a given object instance has a different description depending on the role (class) from which it is managed. One would need to create a sub-type for all the possible combinations of roles for a single type, and to change instance type dynamically each time a new role is applied to it. However, there is a fundamental difference:

The association between an instance and its type is defined statically at instantiation time, while an instance can be bound dynamically to an arbitrary role at any time.

Furthermore, since the instance can be shared and play different roles simultaneously, dynamic typing cannot be used. We introduce the possibility of changing type dynamically. In an OO system the type definition is created first, and then the instances of the types. In the Adele/Tempo system, on the other hand, the instances usually are created first, and are associated dynamically, for a while, to a (set of) role(s).

2. Identity is not altered.
Since a given object can be simultaneously a member of different roles, there are compatibility rules between the roles allowed for shared objects. In TEMPO, objects can change behavior depending on the context without changing identity.

Schema evolution support is an important facility for a software engineering environment, because we need to make it possible to evolve the characteristics of software objects manipulated during the software processes. The role concept naturally integrates a type evolution facility, since role types are similar to object types in O.O. languages. The role concept offers two kinds of evolution:

(a) role definition can change generating role version;
(b) objects can change their roles dynamically.

3.6 Related work

Other SEEs have recently integrated concepts similar to the role concept proposed in Tempo, for example, ES-TAME [14] and Alf/PcTe with its “Work Scheme” concept [8].

ES-TAME allows dynamic change of object type during the execution of software activities. As a result, object attributes and methods can change depending on the activity under which this object is handled. This flexibility of type modification can be taken as the implementation mechanism for the role concept proposed in Tempo.

In Alf/PcTe [8], the same object can have different properties (attribute and relations) depending on the “Work Scheme” under which it is handled. Although Alf/PcTe offers also trigger rules [15], but time dimension is not taken into account.
In the field of databases, the problem related to the modelling of object roles tends to be used as a means of improving the description of object evolution phases or object utilization facets. Several studies have been and are still on this problem. In general, the strategy used is to adopt a persistent object-oriented language and then extend it with the view concept, like the Aspect languages [18], Fibonacci [2] and Views [19]. Object handling is therefore carried out via its view. The limits presented by these approaches as compared to ours are the following:

- They offer no concepts for modelling activities where objects can be handled. Therefore, the way in which an object is perceived and handled is described without considering the context in which the said object will be used.
- The choice of the use of a view is left to the application’s programmer. This decision is based on the description, in the program, of the view that must answer to the messages sent to the object.
- In general, these languages, do not offer concepts for aggregation of different points of view.

4 Conclusions

In this work, we have tried to resolve some difficult aspects of software process modelling and execution. The main goal of Adele/Tempo is to provide an executable object-oriented formalism permitting the description of cooperative software process models and an open architecture to support this formalism. We will now highlight the contributions and results of this work.

4.1 The contribution of Adele/Tempo

4.1.1 The description of software process models

The object-oriented approach seems to be the central element in very research areas like: database management systems, e.g O2 [1], programming languages (C++, Eiffel), software engineering environments, e.g. Ipse 2.5 [7] and PMDB+ [17]. We have adopted the object-oriented approach in the context of software process modelling. As such, we have designed an executable formalism allowing the description of software activities in an object-oriented way.

A software process model in Tempo is a set of cooperating activities where each activity is represented by a software process type. Therefore, a software process model can be described by a set of software process types, in which each type can represent one or more phases/tasks of a particular life cycle. To a specific software process can be attached attributes permitting the description of its characteristics and methods and rules for controlling the execution of such methods. Occurrences of software processes are treated as active objects. The software process type can also be specialized using a multiple inheritance mechanism. To be able to split up a complex phase into simpler ones, a
software process type can assemble many fragments of less complex processes.

4.1.2 The roles of the objects

Each software process can specify which objects may be consumed, modified and created during its execution. Every object handled by an occurrence/instance of a software process can have its characteristics (attributes) changed and its behavior (the methods and their corresponding execution constraints) changed as well depending on the role played by the object within the instance. This is possible due to the role concept offered by Tempo. The role concept has two advantages compared to the classical inheritance techniques:

1. Software process models evolve more easily. Roles can be added or subtracted without having to re-organize the whole database.
2. Roles can be used as a dynamic inheritance mechanism. Dynamic modification of methods and attributes of an active object is also possible.

4.1.3 Supporting evolution

Software process changes make it possible to incrementally develop software process models. This evolution is achieved by dynamic addition of role types. A role can either be an object view or a sub-process (role aggregate). An object view is a definition of a set of encapsulated rules in a consistent unit, or in other words a sub-process. In this way the evolution consistency of the whole depends on adding roles with transitions implemented by Temporal rules, controlling the flow of role instances. Flow can be intra-process (evolution of a role inside a sub-process) or inter-process by adding sub-processes.

Developing a software process language based on an object-oriented approach, extended by the role concept, makes evolution an inherent part of software process models. The major asset resides in the fact that role concepts and Temporal rules allow evolution to be expressed naturally, including the transfer of the affected instances.

4.2 Further studies

Other research and development activities centered on Adele/Tempo have begun from the present work. Here is a summary of the major future studies considered:

- Realization of an object, user-friendly, graphic interface to enable users to execute activities by means of graphic support.
- Tempo and the management of products and software processes. Since software engineering activities are long lasting, coordination and synchronization policies can change during execution. The integrated model allowing for the description of software processes and products should also offer evolution and adaptation capacities.
- Tempo and the Adele data model. Here, the objective is to unify the role concepts so as to structure products and software processes. For the latter, we should plausibly
go for solutions to models based on object roles in order to support products. This would enable the unification of visions of process and product. The process level is a level of abstraction above that of the product. It thus supports activities by organizing them via the role concept.

Acknowledgements

We would like to express our thanks to Barbara Swain and Carolyn Seaman for suggesting substantial and helpful revisions to the original text.

The authors would like to thank one of the anonymous referees for his valuable comments and suggestions.

References


DYNAMIC PERFORMANCE EVALUATION
OF OFFICE INFORMATION SYSTEMS

Vesa Savolainen

University of Jyväskylä
Department of Computer Science and Information Systems
P.O. Box 35, FIN-40351 Jyväskylä, Finland
e-mail: vesa@cs.jyu.fi

Keywords: dynamic modelling, information systems, office systems, performance evaluation, evaluation criteria, value of information systems

Abstract

The evaluation of an office information system (OIS) is a dynamic process if an OIS is defined to include both technical computerized systems and the changing and developing social office environments where the information system supports different functions, activities and tasks. We analyze the dynamic nature, structure and criteria for the performance evaluation of the OIS. Our analysis includes several viewpoints, such as the office and the information system supporting its activities, the development and use of the OIS, the OIS development process and its results, the internal efficiency and external effectiveness of the OIS, the organizational, social, economic and technical criteria for the evaluation, and the evaluation practices and user participation aspects. We draw guidelines for the classification of the criteria, for choosing the right criteria for the performance measurement objects, and for conducting the whole dynamic evaluation process in office environments. Petri net-based Ossadic tools are applied in the description of the evaluation process.

1. System performance and changing user preferences

The evaluation of an office information system (OIS) is a dynamic process because its requirements and forms change with the changing office environment. Changes, exceptions and interruptions are characteristic of today's offices, and often the changes are driven by information technology
(Leppanen and Savolainen 1989). An OIS is an information system for supporting office functions, activities and tasks in order to improve the productivity and working conditions of the office workers (Conrath and Dumas 1989, Schaefer 1988). The development of an OIS thus includes the improvement of both the OIS technology and the dynamic social office environment. The performance evaluation should be an essential part of an OIS development methodology for many reasons as indicated e.g. by Dumas and Charbonnel (1990), Kauffmann and Weill (1989), Olle et al. (1982), Pulst et al. (1990), OSSAD (1986), Savolainen (1991a), Sherwood-Smith (1989), and Warren and Stott (1992).

The launching of an OIS development project is triggered by the decisions based on the perceptions about the unfit state of the existing system, e.g. an office, and these perceptions are obtained by performance evaluation actions. The changes in business processes cause changes in performance objectives. This, together with the IT development, changes user preferences so that current OIS performance no longer satisfies them (Nieisen and Levy 1994), though maintenance actions are made. See Figure 1.

![Figure 1. The contrast between user preferences and OIS performance](image-url)
During the development of an OIS, the work process and its results are controlled by the evaluation made by the analysts, designers and users alongside the project management and the internal auditors. After the project, the new system is continuously monitored by the predefined procedures (Conrath and Dumas 1989, Dumas and Charbonnel 1990). However, it is apparent that performance evaluation plays different roles during the information system life cycle. Its meaning is reduced especially after a decision on launching a project for designing a new OIS. The OSSAD (Office Support Systems Analysis and Design) methodology is maybe the only information system design methodology which explicitly includes a performance evaluation function. This methodology guides its users towards balanced solutions of social and technical problems in dynamic office environments, in order to accomplish an acceptable, effective and efficient OIS in a way that is best suited to its development environment (Conrath and Dumas 1989). The OSSAD methodology also provides principles, procedures, modelling tools and languages for the organizational and technical design and specification of an OIS and its user environment.

Basically, the performance evaluation addresses a problem about the evaluation of the value of the current OIS: does the utilization of the products of the OIS (information and services) support the object system (i.e. the office activities) as was planned when constructing the OIS? Are the positive impacts of the OIS more valuable than the investment in its development and use? This is, however, only one side of the performance evaluation, and may be seen as the *external effectiveness* of the OIS. The performance evaluation also has a technical side, which it may be called the *internal efficiency* of the OIS. A most central area of criteria on both sides can be derived from the user interface because the OIS’s ability to serve the user is offered through the interface. The contrast between the user’s preferences and OIS performance is often measured by the characteristics of the interface.

There are many difficulties in the performance evaluation of OISs (Ang et al. 1991, Banker and Kauffman 1989, Coccia 1985, Jordan 1990, Sherwood-Smith 1989). Performance seen by managers is like a kaleidoscope: everyone has a different conception stemming from individually different views of work and the business environment (Pulst et al. 1990). It is laborious to find relevant, common and acceptable *measurement units*, to define suitable *criteria* and to collect valid data. It is difficult to analyze, combine and compare results of evaluation. It is difficult to find *tools for the evaluation of dynamic systems*, though some do exist (Ang et al. 1991, Burch and Grudnitski 1989, Di Febbraro and Minciardi 1993, Jin et al. 1991, Jordan et al. 1990, Slevin et al. 1991). For these reasons, it is typical to totally exclude this part from the description of an OIS development methodology, or to use misleading generalizations and simplifications when discussing the evaluation (cf. Kauffman and Weill 1989, Leppanen and Savolainen 1989, Olle et al. 1982, Savolainen 1991b, and Warren and Stott 1992).
The performance evaluation is bound to many organizational activities. In each activity specific criteria and procedures are used. In spite of this, there is a set of general concepts, criteria and activities common to all the contexts of evaluation. The purpose of this paper is to present the main concepts, principles and activities needed for the performance evaluation in a uniform way (Section 2). Specific concepts and contextualizations of activities follow those presented in the OSSAD methodology (Conrath and Dumas 1989, Dumas and Charbonnel 1990, Saastamoinen and Savolainen 1991, Savolainen 1989).

Many performance evaluation activities need to be performed during the OIS development process itself, and in this paper we restrict our analysis mainly to these activities. During the development process the performance evaluation is usually based on the system descriptions, and in this paper we try to connect evaluation criteria to each level of OIS models. In the OSSAD methodology these models are products of the development functions which in information systems literature are generally known as the OIS requirement specification (resulting in OSSAD Abstract Models), the systems design (resulting in OSSAD Descriptive Models), and systems technical specification (resulting in OSSAD Technical and Organizational Specification Models).

Section 3 includes a holistic picture of the dynamic performance evaluation process. The process descriptions are presented as Ossadic, Petri net-based graphs. Section 4 analyzes the aspects of the systems evaluation practices and the user participation, and Section 5 offers some concluding remarks.

2. Concepts and criteria

2.1. Main concepts

Generally speaking, the purpose of the performance evaluation is to explore the accomplishments of the subject matter in terms of qualitative and quantitative criteria (Jordan et al. 1990, pp. 346-373). To get a deeper view of the issues of evaluation explicit definitions of the main concepts are needed.

Performance means the accomplishments expressed in some terms. Evaluation, in systems development terms, refers to the broad range of activities leading to the choice of an IS from a range of alternatives (Jordan et al., p. 321); it may consist of activities for making comparisons between the actual and desired OIS performance in office resource utilization, office activities and services produced in the office. Performance is evaluated by using a set of criteria. A criterion is an aspect of reality considered relevant for the goals and meaningful for the evaluation. It helps to classify one or more realities in reference to its value. For instance, quality of service can be used as a criterion when evaluating the office from the viewpoint of a customer. A measurement dimension is chosen as a good indicator for related
criteria for which data can be gathered. In the banking field, the average length of queues is one of many dimensions for the quality of the service. A measure is a quantitative (at least ordinal) data associated with the measurement dimension. A proper unit (e.g. number of customers) is associated with each measure.

A criterion can be expressed in qualitative or quantitative terms. More difficulties are faced with the application of qualitative criteria, for example in finding a common understanding of them, and therefore they are too often underestimated. For that reason, evaluation may be neither realistic nor profitable. Criteria can also be employed as the requirements or standards for the development work (e.g. a standard for the length of an office service queue would be at most 4 people).

2.2. Evaluation criteria connected with Ossadic models

The application of the OIS development methodology called OSSAD produces three kinds of models (Conrath and Dumas 1989, Saastamoinen and Savolainen 1991, Savolainen 1989). *Abstract Models* are constructed to describe and find out the essentials and imperatives of an organization, i.e. what needs to be done in order to accomplish the organizational goals and objectives. These models reflect the management strategy and reveal the functions and activities, as well as the information flows or packets between them. *Descriptive Models* describe both the existing office in terms of dynamic performance and behaviour, and the proposed alternative configurations of the new organizational system and its OIS. It works on the level of tasks, procedures and operations, organizational units, roles and actors, resources and facilities, job assignments, priorities and controls, dynamic aspects and synchronization, etc. *Specification Models* present the details of technical and organizational systems used for supporting the activities of people. The technical OIS specification actions produce models for the user interface; software; databases, files and knowledge bases; data media, hardware and other facilities; systems interconnection; and quality and control. The design process and the construction of these Ossadic models mainly proceed from the Abstract Models, through the Descriptive Models, to the Specification Models. The models are presented as various schematic diagrams and description forms. *Evaluation during the development work is based mainly on these descriptions*, for example by simulation techniques as proposed by Warren and Scott (1992) or utilizing the OOM technique of Ang, Conrath and Savolainen (1991). A set of criteria can be attached to each class of models. See Table 1.

Relevant criteria for the Abstract Models pertain to the goals, functions and outcomes of an organization. The criteria most commonly used for this purpose are effectiveness and profitability. Effectiveness means how well the outcome corresponds to the ultimate goals set for it. Measures of effectiveness are service availability, service quality, service timeliness, and reliability.
<table>
<thead>
<tr>
<th>Concept</th>
<th>Performance Criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Abstract model</strong></td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Effectiveness, Timeliness, Correctness</td>
</tr>
<tr>
<td>Packet</td>
<td>Profitability, Validity, Ease of use</td>
</tr>
<tr>
<td><strong>Descriptive model</strong></td>
<td></td>
</tr>
<tr>
<td>Task</td>
<td>Effectiveness, Efficiency</td>
</tr>
<tr>
<td>Procedure</td>
<td>Ease of use, Effectiveness</td>
</tr>
<tr>
<td>Operation</td>
<td>Efficiency, Timeliness</td>
</tr>
<tr>
<td>Unit</td>
<td>Productivity</td>
</tr>
<tr>
<td>Actor</td>
<td>Degree of motivation, Capacity</td>
</tr>
<tr>
<td>Role</td>
<td>Productivity</td>
</tr>
<tr>
<td>Resource</td>
<td>Validity, Reliability, Relevancy, Timeliness</td>
</tr>
<tr>
<td>Facility</td>
<td>Ease of use, Portability, Capacity, Flexibility, Cost, Efficiency, Recovery, Reliability, Ease of acceptance</td>
</tr>
<tr>
<td><strong>Technical Specification model</strong></td>
<td></td>
</tr>
<tr>
<td>User Interface</td>
<td>Ease of use, Acceptability, User-friendliness, Flexibility</td>
</tr>
<tr>
<td>Software</td>
<td>Effectiveness, Efficiency, Implementability, Testability, Updatability, Recovery, Ease of application, Portability</td>
</tr>
<tr>
<td>Databases, Files and Knowledge Bases</td>
<td>Validity, Accuracy, Integrity, Correctness, Consistency, Ease of access, Integrity, Timeliness</td>
</tr>
<tr>
<td>Data Media, Hardware and Facilities</td>
<td>Ease of use, Portability, Safety, Capacity, Flexibility, Efficiency, Recovery, Compatibility, Ease of extension</td>
</tr>
<tr>
<td>Systems Interconnections</td>
<td>Compatibility, Flexibility</td>
</tr>
<tr>
<td>Quality and Control</td>
<td>Auditability, Ease of control, Safety, Ease of protection, Quality</td>
</tr>
</tbody>
</table>

Table 1. Relevant performance criteria of office information systems expressed in the connection of Ossadic concepts
The Descriptive Models contain more detailed information about roles, tasks, operations and facilities, and their coordination. Thus, relevant criteria are e.g. efficiency (of tasks and operations), productivity (of roles and units, often measured by quality), reliability and acceptability (of facilities) and information value (of resources). Measures of efficiency include office throughput (the amount of work that can be performed during a given period of time), utilization (a ratio of what is used to what services are produced or available) and cost.

The criteria for the Specification Models concern technical or organizational details. Criteria such as efficiency, usability, flexibility, extendability, portability, accuracy and timeliness are commonly used. In this level, timeliness for example, is often related to such measures as response time of the OIS, waiting time, length of queue and backlog.

2.3. Evaluation of the OIS development process and its results

The OSSAD Methodology covers, in fact, two levels of actions: development of office work and development of an OIS for which the models and facilities are constructed by the development work. In the following, these are considered in separate contexts which have effects on the interpretation of the general criteria mentioned in the previous section. In the former, one considers the profitability and quality of the office work; in the latter, it is a question of the profitability and quality of the development work. The levels are interdependent in many ways. One of the main links can be perceived as follows. on both levels, the evaluation can be directed towards either the process or its results. The results of the OIS development work contain the models and constructions to be followed in the OIS implementation and to be utilized in the office work. The criteria used for the evaluation of the results of the development work are shared by the evaluation of the office work. For example, the development work should produce the Specification Models which lead, after the implementation, to enhanced productivity in the office work. See Table 2.

The two levels and their similarities in terms and constructions have partially been recognized in (Conrath and Dumas 1989), and therefore the same set of Ossadic terms are used here for describing them. In the following, we analyze the evaluation criteria in the connection of Ossadic models on two levels.

The level of development: In the OIS requirements specification, the Abstract Model is concerned with the goals, functions and results of the organization. Effectiveness refers, in this case, to the extent to which the development process can contribute to beneficial change in an office, with an acceptable amount of resources. In OIS development, the criteria for the Descriptive Model are concerned with the efficiency of development tasks and operations, the productivity of the project organization, the precision, completeness,
Level of development | Evaluation criteria
--- | ---
- development of an office work
- development of an OIS

Development process | productivity and quality
Results of development process | quality

Level of office work

Work process | productivity and quality
Results of work process | quality

Table 2. Aspects of evaluation

timeliness and relevance of information collected and produced during the development work, and the acceptability and usefulness of development facilities. In the OIS specification for the implementation, the criteria for the Specification Models refer to the efficiency of the software, usability and user friendliness of the system and its interface, quick availability of information, reliability and efficiency of hardware, interoperability of systems, and several kinds of system quality and control issues (cf. Table 1).

The level of office work: The Abstract Model describes the goals, functions and results of the office. Thus, effectiveness means the degree to which the office can reach the ultimate goals set for it. Similarly, efficiency, productivity, precision, completeness, timeliness, relevance and acceptability can be used to evaluate the elements of an office seen from the viewpoint of the Descriptive Model. The Specification Model leads to the application of criteria such as flexibility, extendability and portability for the evaluation of the technical details of an OIS.

The criteria are interrelated in many ways. In the literature, it is usual to present them as a hierarchy where one or two criteria are uppermost and the others are ingredients of them or of each other. However, in each organization, the traditions and the competitive situations form a local criterion hierarchy. In order to provide a more in-depth view of the criteria in relation to OSSAD concepts, Table 1 presents the main Ossadic concepts and relevant OIS performance criteria. The dynamic nature of performance evaluation can also be seen in this list, e.g. in such criteria as timeliness, flexibility, ease of recovery and updatability.
2.4. Organizational, social, economic and technical criteria for the evaluation

The evaluation criteria are traditionally divided into four domains: organizational, social, economic and technical criteria. In the following we briefly analyze them according to this classification. Organizational criteria pertain to the way the organization pursues its goals and objectives.

Social criteria concern human beings as workers or managers, their job satisfaction, motivation, attitudes, and capabilities as an individual or a group. Economic criteria are employed to measure the productivity of an organization or part of one in terms of cost-benefit analysis. The purpose of the evaluation is to obtain quantitative financial assessments. Technical criteria are suitable only for the evaluation of the features of a technical system. Figure 2 clarifies the relations of the domains and the Ossadic models. It shows that the organizational criteria are mainly connected to the Abstract Model, the social and economic criteria interplay with the Abstract, Descriptive and Specification Models, and the technical criteria are mainly related to the Specification Model. In general, they all have relations to each other.

![Figure 2. Relations of the domains and the Ossadic models](image-url)
3. Framework for dynamic OIS evaluation

Like the evaluation criteria, the evaluation activities can also be applied on two levels: for producing assessments about the OIS development work and about the utilization of the working OIS in office functions. The activities naturally have features which are specific to a level, but level-independent characteristics also exist. In this section, a framework for this part of features is presented. The relationships of the basic concepts in the OIS performance evaluation, defined in Section 2.1, are presented in Figure 3.

![Diagram of relationships of basic concepts in OIS performance evaluation]

**Legend**
- Facility: support used to perform work
- Operations, tasks and functions
- Unit or role to control operations and tasks
- Resource: data or objects which are inputs to or outputs from operations, tasks and functions
- Resource and facility connector
- Sequencing connector

**Figure 3.** Relationships of the basic concepts in OIS performance evaluation
The evaluation consists of

(a) the definition of objectives for the OIS evaluation,
(b) the identification and delimitation of the objects under study,
(c) the establishment, i.e. the selection and customization, of evaluation criteria and dimensions,
(d) the organization of the OIS evaluation,
(e) collection and analysis of data on performance, and
(f) the development of conclusions and recommendations.

See Figure 4 for the illustration of the steps, and dynamics of the OIS performance evaluation process. These activities can be on a large scale if the object of the evaluation is broad and the study is thorough. There are also cases where the activities are simple and they are performed in an ad hoc way. Whether the evaluation is directed towards OIS development work or office work there is a need to have a set of predefined procedures for accomplishing the activities. In the context of the OIS development work this is important for the project management as well as for the analysts and the designers.

4. Evaluation practices and user participation

OIS performance evaluation should be a regularly scheduled activity (Hussain and Hussain 1984, p. 225-241). The analyzers who are responsible for the evaluation should be appointed, a budget for the evaluation activities drawn up, and the purpose and scope of the evaluation publicized (Ang et al. 1991, Coccia 1985, Mumford 1983). When performance fails to measure up to the prescribed standards, corrective actions followed by re-evaluation are required (cf. Figures 1, 3 and 4). A satisfactory performance rating indicates no immediate need for change but is no grounds for complacency. Nowadays, offices are not static environments. New IS technology, an altered business or administration climate, increased load in office work places, or demand for new types of office services can suddenly turn contented office clients into frustrated users of office services (Hussain and Hussain 1984, p. 225). For this reason, the evaluation cycle should be scheduled at regular intervals so that office system weakness can be identified and rectified before they become chronic, and so that new objectives for the evaluation can be set in time.

Evaluation should also be initiated whenever problems arise or new OISs are designed and implemented. Normally performance evaluation is, above all, a mechanism of control, both on the OIS development level and on the office work level. Performance evaluation has close connections to quality control, privacy and security considerations and auditing of the office information system. The office head or the Ossadic team leader is responsible for implementing performance objectives, though these objectives can come from several sources, and change for several reasons (Hussain and Hussain
Figure 4. Steps in the performance evaluation process
A part of the objective specification can be done by corporate management, steering committees, planning groups, OIS development teams or office personnel. Evaluation criteria and dimensions must be established by consensus in the whole office environment and in conformity with the Ossadic principles of user participation. Evaluation should be conducted openly, not secretly, with the evaluator known to office workers (Hussain and Hussain 1984, p. 234). Employees can be helpful to evaluators in identifying and diagnosing poor OIS performance, and their co-operation is needed when corrective procedures are initiated.

When monitoring the performance in OIS systems in general there are three broad categories that need to be evaluated: resource utilization, office functions and tasks, and services. The first of these can be divided into the following evaluation activities: personnel performance, office information systems performance, and office equipment and facilities performance. The second includes, from our point of view, the evaluation of the development of the OIS and its internal performance (efficiency), and the third the evaluation of the use of the OIS, i.e. its external performance (effectiveness, support given within its utilization environments). The exact breakdown and scope of subcategories in a given office will depend on the complexity of office services and on managerial choice.

In regular office system performance evaluation, it is important to study user satisfaction. Suitable performance factors for this might be such criteria as the timeliness of office services, the validity and completeness of office products, the achievement of predetermined acceptable levels of office services, the frequency of errors in the OIS, the time required to meet requests of office clients, the effectiveness of measures to control data and protect both security and privacy of information, the reliability of the OIS, the frequency of unscheduled downtime of the technical OIS and this time as a percentage of total scheduled time, the availability and usefulness of the documentation of the OIS, the age of the technology in the system, the quality and turnover of personnel, the user perception of the quality and service of the OIS, and the openness of communication lines between OIS users and IS specialists (see Hussain and Hussain 1984, p. 238-239). The reviews, reports and recommendations made by the OIS performance evaluators should include follow-up and decision information about the OIS and office service levels, priorities, workloads, forecasts of resource and personnel needs, acceptance levels, relevance and obsolescence, cost effectiveness, exceptional performance, technical performance, financial aspects, cost-effective innovations (both technical and organizational), and corrective actions to improve performance.
5. Concluding remarks

In this paper we defined and analyzed basic concepts of information system evaluation in dynamic office environments. The office information systems were defined to include both technical computerized systems and office environments where the information systems support different functions, activities and tasks. We analyzed the nature, structure and criteria for the OIS performance evaluation. This analysis was conducted from several viewpoints, such as those of the office and the information system supporting its activities, the development and the use of the OIS, the OIS development process and its results, the internal efficiency and the external effectiveness of the OIS, and the organizational, social, economic and technical criteria for the evaluation. We offered guidelines for the classification of the criteria, for choosing the right criteria for the performance measurement objects, and for conducting the whole dynamic evaluation process in office environments. Finally, we analyzed the systems evaluation practices and user participation viewpoints. We applied Ossadic Petri net-type description tools for illustrating the dynamic nature of measurement procedures.

References


T-ORM: EVOLVING OBJECTS AND ROLES

Elisa Peressi    Angelo Montanari
Dipartimento di Matematica e Informatica
Università di Udine
via Zanon, 6 - 33100 Udine, Italy
Phone: +39 432 272 [215,218], telefax: +39 432 510755
e-mail: [peressi,montana]@dimi.uniud.it

Barbara Pernici
Politecnico di Milano
piazza Leonardo da Vinci, 32 - 20133 Milano, Italy
e-mail: pernici@elet.polimi.it

Abstract

Support for temporal data is a requirement posed by many database applications: therefore several different time models have been proposed to manage temporal knowledge. Most of these models are extensions of the relational model and confine themselves to temporal aspects of properties of an entity. In order to deal with dynamic aspects, we adopt an object-oriented data model augmented with the concept of roles which an object can play during its lifetime, supporting the representation of temporal object evolution and querying object history. The case in which the object becomes a member of different classes or roles during its lifetime is considered. We allow objects to change class, to play new roles, defining new constructs for a query language which allow us to retrieve the history of an object.

Keywords: object-oriented temporal model, query language, object evolution, migration.

1 Introduction

Nowadays many highly advanced application areas, such as CAD/CAM, office automation, knowledge representation, software engineering, are insufficiently supported by traditional relational DBMSs since they require semantically richer data models. These applications need the possibility of modeling composed entities, different types of relationships among entities (e.g. uses, is-part-of), special attribute domains, relationships
among entity types (e.g. is-a). As a consequence, powerful operators are needed to manipulate such data. Many of these applications deal with a reality that evolves in time. To support modeling and possibly interactions with an evolving environment without losing past information, the current state of the database must be updated, preserving past states. Conventional databases providing only information about the current state of the world are therefore inadequate and temporal databases maintaining information about past, present and (possibly) future states are being proposed (Tansel et al. 1993).

In the last ten years, several time models have been proposed to manage temporal knowledge in database systems. Most of them are extensions of the relational model (Gadia 1988; Snodgrass 1987); work on the addition of time dimensions in object-oriented databases has focused mainly on representing time evolution of object properties (Clifford and Croker 1988; Dayal and Wu 1993). A consensus characterization has not yet been achieved even in the simpler context of temporal relational models, and in the more complex realm of temporal object-oriented databases the problem of a consensus characterization has not been addressed at all. In object-oriented approaches, there is mainly a concern about the representation of state evolution. A more general representation of object evolution is needed, to represent and be able to query about modifications of the structure of an object during its lifetime.

In fact, the notion of object migration is emerging as an important functionality that should be supported by object-oriented database systems (Su 1991): in object-oriented databases, objects belong to hierarchically structured classes, and in many application domains it is natural to allow objects to dynamically change the class(es) they belong to. For example, it seems fairly acceptable to allow an object created as a member of the class PERSON to migrate to the subclass ADULT. Only few papers deal with object migration in object-oriented databases (Su 1991; El-Sharkawi and Kambayashi 1990; El-Sharkawi 1991), while object migration has not been addressed at all in the context of temporal object-oriented databases.

This paper presents a temporal extension of an object-oriented data model to deal with both local and global histories of modeled entities. The temporal data model we adopt (T-ORM: Temporal Objects with Roles Model) extends the object-oriented conceptual model ORM (Pernici 1990) with primitives for modeling time. It can be seen as a generalization of the concepts proposed in (Clifford and Croker 1988; Segev and Shoshani 1987) on time sequences to represent the history of attributes. Furthermore, the proposed extension to the ORM model is complementary to the temporal extension presented in (Edelweiss et al. 1993; Edelweiss et al. 1994), focusing on the specification of temporal requirements for information systems.

Several extensions of SQL to pose queries about object properties have been proposed in the literature (Dayal and Wu 1993; Rose and Segev 1991; Snodgrass 1987; Snodgrass 1993; Tansel et al. 1993): in this paper, we define the T-ORM query language based on SQL, focusing in particular on constructs to formulate queries about the global history of an object.

The organization of the paper is as follows. Section 2 illustrates the T-ORM model, based
on the concepts of class and role. Section 3 introduces and discusses the notion of class migration and discusses lifespans associated to classes and roles. Section 4 presents the T-ORM query language.

2 The T-ORM data model

2.1 Classes and roles

The management of object behavior is one of the main problems in real-world modeling. Most of the efforts in this area have been limited by static schema definitions, supplying objects with methods which operate on object states. Recently, it has been suggested to incorporate rules within objects for expressing object behaviors. Besides the necessity of representing changes in state, another problem occurs. Many applications have the necessity of describing particular entities from different perspectives, dealing with multifaceted object states, that is, an object can play different roles and its behavior depends on the role it plays. The term role has been used in various contexts with different meanings.

Richardson and Schwarz (Richardson and Schwarz 1991) introduced roles under the name of "aspects" which act as a template from which instances may be cloned. Wieringa (Wieringa and de Jonge 1991) allows roles playing in turn other roles and allows hierarchies between roles. Su (Su 1991) considered migration patterns as a new type of dynamic integrity constraints which specify the admissible sequences of states that an object can possibly migrate through. Sciore (Sciore 1989) modeled roles with object hierarchies composed of independent objects, each with its own identity. In Papazoglou (Papazoglou 1991), an entity can play several roles at the same time, but only a single occurrence of each role type is permitted per entity.

These approaches do not treat temporal aspects in object attributes, classes and roles. Some authors have proposed to deal with time in object-oriented models. In Elmasri (Elmasri and Kourmajian 1992) a distinction is made between conceptual objects and temporal objects (entity roles). A role type can have only temporal attributes (time-varying). In TROLL (Hartmann et al. 1994), possible roles for classes are defined, although temporal aspects are considered only from the point of view of specification of possible behaviors of objects. In this paper we extend the ORM model (Objects with Roles Model) with time, associating temporal information both to attributes, and to objects, and to roles.

The ORM model allows the representation of possible object behaviors by means of the concept of role. A role is a state in which an object can be, getting new properties associated with it; we say that an object plays a given role. Specialization hierarchies are usually used in traditional object-oriented systems to model the various states which an entity may assume, and representing real-world entities as instances of the most specific class in which they can be classified. This approach implies numerous drawbacks. We refer to an example proposed in (Wieringa and de Jonge 1991) in order to explain our
Reasons.

Assume that passenger is a subclass of person and consider a person who migrates to the passenger subclass of person, say by entering a bus. This bus may carry 4000 passengers in one week, but counted differently, it may carry 1000 persons in the same week. So counting persons differs from counting passengers.

The conclusion of this observation can be stated in terms of identifiers. If PASSENGER is defined as a subclass of PERSON, then the person identifier is also the identifier of the person as a passenger. However, this is not the case in the example presented above, since persons and passengers need different identifiers. Therefore we need a different way to represent those instances. We must realize that a passenger is not identical to a person, but that it is a state of a person, or, as we define it, it is a role of the class PERSON. Roles can be instantiated several times for the same object. So, when we count passengers, we really count how often persons have been in the state of being a passenger. The reason for which roles cannot be implemented as subclasses of the class they relate to is that the inheritance mechanism does not allow multiple instantiation. As we said, a person could become a passenger more than once during a week. We cannot instantiate the same person as a passenger more than once to represent the fact that he is a passenger at different points in time. In T-ORM, we associate role types to classes, to represent the different states in which a class can be in. The roles mechanism is flexible enough to allow the representation of an object playing different roles at different times, playing more than one role at the same time, through multiple instantiations of roles. This aspect distinguishes the T-ORM model from other similar proposals. In the model proposed in (Papazoglou 1991) an entity could play several roles simultaneously, but only a single occurrence of each role type is permitted per entity. A similar approach with respect to role instantiation can be found in Troll.

There are also other reasons providing a motivation for defining roles of a given class. When only the construct of class specialization can be used, in the case in which an entity can assume different roles independently (for example, a person who is at the same time a student and an employee), we would have to define a separate class which is subclass of both classes EMPLOYEE and STUDENT. Subclasses like that must be defined for every possible combination of independent class specializations. With the construct of role, we can define both employee and student as roles of a class person, and each role can be instantiated independently.

Therefore, in T-ORM, an object assumes a certain role via a mechanism of instantiation which is analogous to that used to populate classes. We speak about role instances in the same way in which we speak about class instances. Every time that a role is instantiated, we associate to the instance a unique identifier (Role Identifier o RID) which preserves instance identity across changes of its state (i.e., changes to attribute values of the role). We assume that this identifier is unique across the database. All instances of roles evolve independently.

A class in T-ORM is defined by a name Cn and a set of roles R1, each one representing a
different possible behavior of object belonging to the class:

\[ \text{class} = (C_n, R_0, R_1, \ldots, R_m) \]

Each role \( R_i \) is a 5-uple:

\[ R_i = < R_{ni}, P_i, S_i, M_i, R_{ri} > \]

consisting of a role name \( R_{ni} \), a set of properties \( P_i \) associated to role \( R_i \) (abstract description of object characteristics), a set of abstract states \( S_i \) that the object can be at while playing this role, a set of messages \( M_i \) that the object can receive and send in this role, and a set of rules \( R_{ri} \). Each property has a property name and a domain. Domains may be either simple or complex. Simple domains are predefined domains (such as string, integer, real, boolean) or references to other classes or roles of classes; complex domains are defined as aggregations, sets (unordered collection of objects) or sequences (ordered collection of homogeneous objects without duplicates) of other domains (simple or complex). Rules fall into two categories: state transition rules and integrity rules. State transition rules define which messages an object can receive/send in each role state and the state changes these messages causes. Integrity rules specify constraints on object evolution.

Every class has a base-role \( R_0 \) that describes the initial characteristics of an instance and the global properties concerning its evolution. Those properties are visible within all the other roles. In the base-role, messages are used to instantiate, delete, suspend and resume instances of the roles; the possible states in the base-role are pre-defined (active and suspended) and describe the global state of the object; the rules define possible transitions between roles and global constraints for the class.

Finally, a class can be a subclass of one or more classes (multiple inheritance), and inherits all roles specified in the parent class(es).

Let us introduce a simple schema that will be used in the rest of the paper as a source of exemplification (see Figure 1). We consider four classes, namely PROJECT, DOCUMENT, PERSON, and ADULT, which is a subclass of PERSON. A PERSON object can become an ADULT during its life-cycle, and, if so, the property "driving licence" can be assigned a value. Objects belonging to the class PERSON can play two different roles (Employee and Student), each one characterized by its own properties. Projects are developed by PERSONs playing the Employee role. Each project has associated a set of documents written by the employees who participate in the project.

### 3 Temporal information in T-ORM

Adding the time dimension to object-oriented systems is required for modeling how the entities and their relationships may change over time (Clifford and Croker 1988). Often an object is created at a given time and is relevant to a system for only a limited period of time. Furthermore, during their existence, objects attribute values may change, as well as the roles a given object is playing, and the classes it belongs to. Temporal (object-oriented) databases proposed in the literature differ from each other both in the structure
of the underlying time domain and in the way of associating time information to database entities. In the T-ORM model, time is associated both with single attributes, and with class and role instances of an object.

With respect to the association of time with data, object attributes can be partitioned in \textit{time-varying} and \textit{constant} ones (Tansel et al. 1993), depending on the fact that their value may change or not over time. The values of time-varying attributes are usually time-stamped at specific time points or intervals; therefore, in general, we do not know their value at a time where there is no specific entry. In T-ORM, we assume that values of attributes are step-wise constant. An object attribute value is defined as a \textit{time sequence} (TS) (Segev and Shoshani 1987), i.e., a sequence of values associated with a different time intervals. An interval \([s_i, e_i]\) is defined as the set of time points from a given time point \(s_i\), the starting time of the interval, to a time point \(e_i\), the ending time of the interval, which is not part of the interval. Due to the bidimensionality of time, time sequences are constituted of a set of triples \(<\text{attribute value}, \text{valid \text{-} time interval}, \text{transaction \text{-} time interval}>\). We assume that, given a transaction time point, the validity time intervals for an attribute do not overlap and they can be totally ordered. Finally, for complex attributes, e.g. aggregates, sets, lists, we assume that valid and transaction times are associated both with the whole structure and with its components.

Besides associating time information to attributes, object-oriented temporal databases
OOTDBs can temporally characterize the existence of objects, that is, they can specify when and how an object exists in the database. In most OOTDBs, the set of time intervals during which an object logically exists in the database is called its lifespan (Clifford and Croker 1988). This object lifespan spans from the object creation (the point in time when the object is instantiated) till its complete termination. As an object can be member of different classes, an object lifespan is the union of its lifespans in all classes in which it has participated. In historical object-oriented databases the notion of "reincarnation" is also supported, because a termination of an object is not necessarily terminal (Clifford and Croker 1988). For example, employees can be hired, fired, and subsequently re-hired. We deal with such cases both with object suspension and with different instantiations of roles.

In T-ORM, role instances are treated similarly to class instances, as far as the representation of temporal properties associated to them. They have associated a lifespan which spans from their instantiation till their complete termination and can be temporarily interrupted when the role instance is suspended or when the corresponding object is suspended. All role instances are terminated when the corresponding object is terminated. The lifespans of roles instances and those of the correspondent objects are linked by specific constraints: the lifespans of role instances are always contained in the lifespan of the correspondent object. Formally, let \( o \) be an object instance of a class \( C \), \( p(C) \) a function that maps \( C \) to the set of the roles defined for that class and \( r(o,R) \) a function that maps an object \( o \) to the set of its role instances of role \( R \). The following constraint must hold:

\[
\forall R \in p(C) \forall r_i \in r(o,R) \\
(r_i,LIFESPAN) \subseteq (o,LIFESPAN)
\]

Since \( (r_i,LIFESPAN) = \{[s_1^i, e_1^i], ..., [s_n^i, e_n^i]\} \)
and \( (o,LIFESPAN) = \{[s_1, e_1], ..., [s_m, e_m]\} \)
the given constraint states that

\[
\forall k = 1, ..., n \ \exists j \in \{1, ..., m\} \text{ such that} \\
[s_k^i, e_k^i] \subseteq [s_j, e_j]
\]

4 Evolution of objects

4.1 Class migration

In most object-oriented data models proposed in the literature an object is created as an instance of a class with some attribute values and operations associated to it and remains an instance of that class till its deletion from the database. This restriction limits the expressiveness of those models, because as it is possible in the real world to change entity status, so it should be possible for an object during its life cycle. For example, a person
can be an employee for some years, then attend a course playing the role of a student keeping also the role of employee at the same time and then have second employment at the same time. Part of these problems are overcome by using the concept of role in T-ORM, that allows us to have a person who plays the same role (e.g. employee) more than one time, modeling a kind of multiple instantiation, but still preserving the single object identity. Other issues, however, cannot be solved by the concept of role alone, such as letting an object migrate from a class to another class maintaining its identity (its oid). That is, we want that an object representing a person maintains its oid if it migrates to the class ADULT.

Class migration may assume different forms. In particular, it is possible:

- to let the object migrate to a different class (the object becomes an instance of the new class);
- to migrate an object to a subclass (the object becomes an instance of the subclass, but remains a member of the original class);
- to migrate an object to a superclass (the object becomes an instance of the superclass and it is no more a member of the original class);
- to dynamically add new class memberships to an object, so that it can be an instance of more than one class at the same time;
- to dynamically delete classes from an object;

In (El-Sharkawi and Kambayashi 1991) three types of object migration are distinguished: generalization, specialization, and generic migration. **Generalization** is done when an object moves to one of its superclasses. **Specialization** is done when an object moves to one of its subclasses. **Generic migration** is done when an object migrates to a class which is neither its super nor subclass.

In T-ORM, we deal only with object generalization and specialization, that is object migration is allowed only along a unique class hierarchy. This is not an excessive restriction, since the data model allows the definition of a common root for all class hierarchies. In fact, in that case, using an appropriate combination of generalization and specialization operators, we allow an object to migrate everywhere. However we think that in general object migration does not make sense when it occurs between different hierarchies because it would completely change the nature and the structure of an object.

### 4.2 Constraints on object migration

Due to inheritance, we have to impose additional semantic constraints on object migration operators. Let us assume that an object \( o \), instance of class \( C_i \), migrates to class \( C_j \). Consider the four cases illustrated in Figure 2.
case 1) specialization with single inheritance: non-inherited properties defined for class \( C_j \) should be added to the object and their values should be given by the user, or considered to be null;

case 2) generalization with single inheritance: all properties that are specific for \( C_i \) have to be dropped from the object;

case 3) generalization with multiple inheritance: all properties inherited from class \( C_k \) (in general, from superclasses of \( C_i \) different from \( C_j \) and which are not superclasses of \( C_j \)) and all properties specific for \( C_i \) have to be dropped from the object;

case 4) specialization with multiple inheritance: all properties of class \( C_i \) that are inherited from a superclass \( C_m \) of \( C_j \), where \( C_m \) is not a superclass of \( C_i \) and all properties specific for \( C_j \) have to be added to the object and their values should be given by the user or considered to be null.

The previous constraints occur also for roles: when an object migrates to a superclass, the roles it has instantiated which are specific for its original class, are terminated; when it migrates to a subclass, it maintains all its previous roles, the base-roles for classes of which it is a new member are instantiated, and instantiation of the roles specific for the new class becomes possible.

4.3 Classes and roles lifespan

As illustrated above, during its lifetime an object can change roles and migrate along the class hierarchy. The T-ORM model stores temporal information concerning events in the object lifespan.
Consider the following example, (based on the T-ORM schema illustrated in Figure 3: an object $o_1$ is created as an instance of the class $C_1$ at time $t_1$; $o_1$ instantiates role $R$ twice ($r_1$, $r_2$), at times $t_2$ and $t_3$, respectively; then at time $t_4$ it migrates to class $C_2$; at time $t_5$ role instance $r_1$ is suspended; at time $t_6$ $o_1$ instantiates role $S$ ($r_3$), and so on, as illustrated in Fig. 4.

In T-ORM, we treat object lifespans as time sequences, in a similar way to attributes, as shown in Sect. 3.

![Figure 3: Example of T-ORM schema](image)

The lifespan of an object has different dimensions:

- The **class-lifespan** is the global lifespan of an object, and stores the history of object migration. It is the union of the object lifespans related to the various classes it is (or was) instance of. It has as a value component the set of the class types which it belongs to, during the associated interval.

  $$o_1.\text{CLASSLIFESPAN} = \langle \{C_1\}, [t_1, t_4]\rangle, \langle \{C_1, C_2, C_3\}, [t_4, +\infty]\rangle$$

- The **role-lifespan** is the global lifespan for an object, for all its roles, and it is the union of the lifespans of the single role instances the object has played during its history. It has as a value component the set of role identifiers of the active instances in the associated interval, as in the following example:

  $$o_1.\text{ROLELIFESPAN} = \langle \{r_1\}, [t_2, t_3]\rangle, \langle \{r_1, r_2\}, [t_3, t_5]\rangle, \langle \{r_2\}, [t_5, t_6]\rangle, \langle \{r_1, r_2, r_3\}, [t_6, t_7]\rangle, \langle \{r_1, r_3\}, [t_7, t_8]\rangle, \langle \{r_1, r_3\}, [t_9, +\infty]\rangle\rangle$$

- The **lifespan of an object as a member of a class $C$** stores the history of that object within that class.

  $$o_1.\text{LIFESPAN}(C_2) = \langle (o_1, [t_4, t_8]), (o_1, [t_9, +\infty])\rangle$$
An object could play different roles at the same time, we assume that the lifespan of an object for a given role type is a sequence like the previous one, whose values are the sets of the active role instances.

\[ o1.\text{LIFESPAN}(R) = \langle \{r1\}, [t2,t3] \rangle, \langle \{r1,r2\}, [t3,t5] \rangle, \ldots > \]

- The lifespan of a role instance is associated to the correspondent object and is memorized in a time sequence where the value component contains the role identifier of the corresponding role instance, so it has the following form:

\[ r1.\text{LIFESPAN} = \langle r1, [t2,t5] \rangle, \langle r1, [t6,t8] \rangle, \ldots > \]

The object migration mechanism leads us to impose some temporal constraints on the object lifespan. In particular, referring to the previous example, the following temporal constraint must hold:

\[ o1.\text{LIFESPAN}(C2) \subseteq o1.\text{LIFESPAN}(C1) \]

In fact, when an object is created as an instance of a class, it starts to be a member of all its superclasses. Moreover, if an object created as an instance of class \( C1 \) migrates to \( C2 \), it continues being a member of \( C1 \), so its lifespan as member of \( C1 \) is not terminated.

5 The query language

The complete definition of a data model requires the definition of the corresponding query and data manipulation languages. In this paper, we present the characteristics of
the T-ORM query language, providing a uniform framework to deal with temporal and object-oriented features of the T-ORM model. A description of the data manipulation language can be found in (Peressi et al. 1994). In the following we focus our presentation on those aspects which are related to object migration and time (see (Peressi et al. 1994) for a complete syntax).

5.1 Basic query structure

A query has the following structure:

RETRIEVE < target clause >
FROM < specification clause >
WHERE < qualification clause >
AS OF < as-of clause >

The target clause specifies the information to be retrieved, which could be a set of instances (the object or role identifiers are retrieved in this case), a time sequence, a set of values, or a sequence of time intervals (or points).

The specification clause specifies instance variables used in the query, linking them with the correspondent set of object (role) instances.

The qualification clause specifies conditions on time sequences to select particular information. In bitemporal databases we have three dimensions: the data dimension, the valid-time dimension and the transaction-time dimension. The T-ORM query language has operators suitable for manipulating all dimensions. In order to maintain the language as simple as possible, we chose to have only one clause (qualification clause) to specify constraints both on the value and the valid-time dimension, whereas other extensions of SQL (such as TSQL (Tansel et al. 1993)) introduce additional clauses.

The as-of clause specifies constraints on the transaction-time dimension. It is used to determine the values of object properties as they were recorded sometime in the past. In this way, we could retrieve information as known in previous states of the database.

5.2 Queries on time-varying properties

The goal of querying a temporal database is the retrieval of stored information, taking into account the modifications performed on it. Since bitemporal databases model two temporal dimensions, we can distinguish two basic types of queries: (i) queries that retrieve the sequence of historical values of time-varying information (along the valid time axis); (ii) queries that retrieve data as of a past database state (along the transaction time axis).

As regards the first group of queries, the main queries we want to be able to express are the following: (i) select an attribute value valid at a given instant, e.g. find John’s salary on 04/15/1986; (ii) select an attribute value valid at a time instant associated
to another attribute value of the same object, e.g. find John’s salary when Mary was his manager; (iii) select an attribute value valid at a time instant associated to another attribute value of another object, e.g. find John’s salary when Mary’s salary was $4000; (iv) select objects stored in the database at a given instant, e.g. find all employees in year 1992; (v) select time periods starting from attribute values, e.g. find the time period during which Mary was John’s manager.

In this work we do not discuss the second group of queries because it is less important from the modeling point of view. In T-ORM, those queries are treated in the traditional way using the rollback clause as-of.

In T-ORM time-varying attributes are modeled with a time sequence which represents all their history (see 2.1). A query language must allow the selection of a portion of that history through the specification of conditions either on time, or on attribute values, or both. We can directly select the first two components of the time sequence, with the following notation:

- \text{id.attribute-name.value} for the value component
- \text{id.attribute-name.vtime} for the v_time component

For instance, a path expression of the kind \text{e.salary.vtime} retrieves the time intervals which form the history of the specified attribute (salary) associated to the retrieved object, identified by the object identifier \text{e}. We must provide our query language with operators which allow one to select portions of that history. To do that we can use in the where clause predicates with relational operators involving time. Such operators are those of Allen’s interval logic (Allen 1983) (that is PRECEDES, MEETS, OVERLAPS, STARTS, ENDS, INCLUDES, their inverse, and EQUAL), those between time points (i.e. <, = and >) and those between time points and intervals (i.e. BEFORE, BEGINS, ENDS, IN, AFTER, and their inverse).

For instance, in the following query values of the attribute salary, valid before April, 10, 1987, from employee roles in the database are selected:

\text{EX1: “Find John’s salary before that of 04/10/1987”}

\begin{verbatim}
RETRIEVE s.value
FROM (e,Employee)
WHERE e.name == “John”
AND (04/10/1987 AFTER s.vtime)
AND NOT(EXISTS(j,e.salary):
(s.vtime MEETS j.vtime)
AND (04/10/1987 AFTER j.vtime))
\end{verbatim}

A path expression which refers to a set of values (such as \text{e.salary}) can be quantified using either the existential (EXISTS) or the universal quantifier (FORALL). Quantification cannot be made on the variables of the target clause, which are free. We assume that all operators, when applied to a set, distribute on its elements (in the previous example the operator \text{AFTER} distributes on the elements identified by \text{s.vtime}). Particular elements of a sequence can be selected with the following operators:
HISTORY

SUBHISTORY

SINGLE ELEMENTS

SINGLE COMPONENTS

TIME

VALUE

\( \langle v_1, [s_1, e_1]), ..., (v_n, [s_n, e_n]) \rangle \)

\( \text{seq} = \langle (v_{i_1}, [s_{i_1}, e_{i_1}]), ..., (v_{i_m}, [s_{i_m}, e_{i_m}]) \rangle \)

\( \text{FIRST}(s, \text{seq}) = \langle v_{i_1}, [s_{i_1}, e_{i_1}] \rangle \)

\[ \forall m \geq n \text{ and } \forall j \leq i \leq m \exists k \leq n \text{ such that } (v_{ij}, [s_{ij}, e_{ij}]) = (v_k, [s_k, e_k]) \]

Figure 5: Operators on time sequences

- **FIRST**(s,e.salary) \( \mapsto \) retrieves the first element in the sequence and assigns it to the variable s

- **CURRENT**(s,e.salary) \( \mapsto \) retrieves the current element in the sequence

- **LAST**(s,e.salary) \( \mapsto \) retrieves the sequence whose element is the last element in the given sequence

- \( <n>-\text{TH}(s,e.salary) \mapsto \) retrieves the n-th element in the sequence

**EX2:** "Find John’s present salary"

```
RETRIEVE c.value
FROM (j,Employee)
WHERE j.name == "John"
AND CURRENT(c, j.salary)
```

Our model is based on time intervals, however we could also select the endpoints of intervals using the functions **BEGIN** and **END** which can be applied to a unique interval or to a sequence of intervals, returning a single time point or a sequence of time points. We have defined selection operators on time sequences, which act at different levels of detail, as shown in Figure 5.

### 5.3 Queries on the history of an object

The other important aspect related to time in object-oriented databases concerns the history of an object as a whole, in addition to considering the history of single attributes as in the previous section.
In our model, we distinguish between \textit{local histories} and \textit{global object histories}. The local history regards single classes and roles. The global object history refers to the history of variations of state of an object as a member of different classes and instance of various roles.

5.3.1 \textit{Local history}

Due to object migration the set of objects belonging to a class varies over time, so it is necessary to have some primitives which allow us to denote the set of instances of a certain class at a specific time point.

A user may be interested in formulating a query on a particular object when it was an instance of a certain class (i.e. before a migration) to know a particular attribute value, or the roles it played, and so on. Our query language allows us to address instances of a class (role) in three different ways, according to the portion of the history of the class we consider:

1. the set of all class (role) instances (i.e. current and terminated instances) with the notation used in the previous paragraph:
   
   \verb|(<object-variable>,<class/role name>)|

   \textbf{EX3}: Find all employees (current and past)
   
   \verb|RETRIEVE e|
   \verb|FROM (e,Employee)|

2. the set of instances valid during a specified interval are retrieved through appropriate conditions on the valid time dimension in the \textit{WHERE} clause:

   \textbf{EX4}: Find John's salary when he was an employee during 1975 and his manager was Mary
   
   \verb|RETRIEVE s.value|
   \verb|FROM (e,Employee)|
   \verb|WHERE e.name == "John"|
   \verb|AND e.manager.name == "Mary"|
   \verb|AND EXISTS(s,e.salary):|
   \verb|(91/01/1975,31/12/1975) INCLUDES s.vtime|

3. the set of current class (or role) instances

   \verb|(<object-variable>, CURRENT(<class/role name>))|

   \textbf{EX5}: Find all present employees
   
   \verb|RETRIEVE e|
   \verb|FROM (e,CURRENT(Employee))|
5.3.2 Migration history

The operators defined to retrieve information about attribute time sequences can be applied to lifespans to retrieve information about the state of an object with respect to class membership and role instantiation.

For instance, the following types of queries are considered:

EX6: When did Mary become an employee? (role instantiation)
EX7: When did Mary become an adult? (class migration)
EX8: When did Mary change class?
EX9: Which roles did Mary play at 15/03/1992?
EX10: Which roles did Mary play during [11/07/1987,24/7/1989]?

The answer to those questions can be easily found by appropriate queries on the various dimensions of the object lifespan with the use of temporal functions like BEGIN and END.

For instances, queries EX8 and EX9 are formulated as follows:

EX10: RETRIEVE p.CLASSLIFESPAN.vtime
FROM (p,PERSON)
WHERE p.name == "Mary"

EX11: RETRIEVE s.value
FROM (p,PERSON)
WHERE p.name == "Mary"
AND EXISTS(s,p.ROLELIFESPAN):
(15/03/1992 IN s.vtime)

Predicates on object history can be used also inside the qualification clause, like in the following example:

EX13: "Find the salary of the employees who became employees before becoming adults"
RETRIEVE c.value
FROM (e,Employee)
WHERE EXISTS
(a,e.LIFESPAN(Employee)):
(s.vtime PRECEDES
FIRST(e.LIFESPAN(ADULT)).vtime
AND CURRENT(c,e.salary)
6 Concluding remarks

We presented a temporal extension of an existing object-oriented conceptual model (the ORM model), focusing our attention in particular on object evolution and on a query language to pose queries both about the object state and about its changes of states, i.e., its evolution in time.

The constructs we considered for representing and querying about class migration do not depend on the particular object-oriented model we chose, but can be extended to other object-oriented data models. However, the concept of role is essential to be able to express correctly the evolution in time of object, in particular when an object would be an instance of the same class more that one time. In the paper, we discussed the examples of a person being a passenger in a bus and of a person being employed by more than one company at the same time. In the T-ORM model, such aspects of object evolution can be represented.

Some remaining open issues concern the evolution of objects not only in terms of objects changing classes, but also in terms of changing schemas, i.e. changing class definitions. Further work is also needed to model temporal aspects in complex objects, in particular with respect to the varying composition of objects in time.

References


Acknowledgments

This work has been partially supported by P.A.O.L.A. Consortium and the Italian National Research Council. The authors would like to thank Nina Edelweiss for her suggestions.
DYNAMIC MODELLING AND SIMULATION OF INFORMATION SYSTEMS USING MULTI-AGENT SYSTEMS

C. Roche, F. Deloule, J.C Profizi
L.G.I.S
Université de Savoie
Campus Scientifique
73 376 Le Bourget du Lac cedex
France

mail: roche@univ-savoie.fr , deloule@univ-savoie.fr , profizi@univ-savoie.fr
tel: (33) 79.75.87.79 fax: (33) 79.75.88.88

Key words
Dynamic modelling, information systems, open systems, multi-agent systems, analysis, simulation, object-oriented concurrent programming, message sending.

Abstract
The present article is intended to demonstrate the advantages to multi-agent systems in modelling processing in information systems. First, the objectives of modelling processing and the formalisms commonly used for such systems are briefly recalled. Then, a general presentation of open systems, i.e. actor languages, actor groups and multi-agent systems is presented. We will see that there are advantages to multi-agent systems for three different aspects. The first is analysis. Indeed, the concepts of the agent, the society of agents, and communication via messages permit modelling that is very close to the real world. The second aspect concerns the open and evolving nature of the system. The control and communication mechanisms are not fixed in advance and can be defined or specialised according to the application at hand. The third and last aspect is that of computer simulation, which makes it possible, for example, to understand the circulation of information between agents and to supervise their activation.

Then our approach is presented through the BAAL multi-agent system (BAsic Agent Language) and a look at two of its basic principles. The first is the equivalence between "agents" and "societies of agents" that permits a hierarchical breakdown of the agents. The second principle is BAAL's reflexive structure that uses dedicated agents to specify the interface and control. This kind of reflexivity makes it possible to define new communication protocols and control systems.

Finally, the article concludes by presenting an example of modelling using BAAL. We focus on a simple example of flexible workshop management where all the components, manufacturing islands, wire-guided trucks, and network nodes are represented by cooperating agents.
1. Modelling processing in Information Systems

1.1 Objectives

The main methods in Information Systems ([Chen 76], [Marca 87], [Yourdon 79], [Tardieu 83-85], [Peterson 81], etc.) rely on a separation of data from processing. This is justified in particular by the fact that data is more stable in an organisation than the processing that is applied to it. This separation makes changes in the Information System easier, and especially avoids re-constructing the existing system architecture when changes must be made. It corresponds perfectly to the conventional procedural languages that serve in programming Information Systems.

We thus obtain a model of data that gives a good description of the static aspects of the system through the description of the most stable elements (invariants in the system) that compose it.

Modelling processing, on the other hand, makes it possible to visualise the dynamic aspects of the system under study, describing the actions taken on the data and showing their sequencing. It reveals events that trigger or result from an action, constraints on achievement, or necessary supports for the progress of processing.

Of course, the models of the data and processing must be coherent and validate each other. However, in this article, we deliberately restrict ourselves to representations of processing, meaning the dynamic aspects of an Information System.

1.2 Modelling processing

Before trying to model, at a more or less detailed level, all of the processing to which the data handled in an application may be subjected, it is often interesting to look at the data flow diagram (DFD) [Codd 79] or the Merise conceptual communication model (MCC) [Tardieu 83-85]. These models are quite similar and both can be considered as a common source for modelling on one hand and processing on the other.

Such models attempt to bring to light how information circulates (exchanges of data flows or messages of defined types) between actors in the real world; in this case an actor means an element (human being, department in a company, etc.) that is capable of ensuring a management function. The models can be used in particular to identify the events that will trigger processing (arrival of an order, production request, etc.) and to follow the itinerary and transformations of the information concerned.

So those models are interesting for the obvious assistance they provide in building processing models, especially by supplying the initial processing events and the expected components of the final result. But although they show existing communication patterns between the actors, as defined above, they do not show the communication that occurs between the various processes, permitting the system to evolve.

We will now briefly set out the main representations of processing used in modelling information systems, i.e. SADT actigraphs, Petri nets [Peterson 81], and the Merise conceptual processing model.

The SADT method [Marca 87] uses a descending technique, towards ever greater degrees of detail, to give a modular approach to a system. This method is strictly organised in a hierarchy: a model always refers to another model with finer granularity. But all actigraph
type models are built along the same principle: for each operation or set of operations, represented by a verb set in a « box », SADT gives:
- the data (or object-event) that triggers the processing,
- the data (or object-result) that results from it,
- the constraints (or decision framework) governing accomplishment of actions,
- the resources needed to accomplish them.

Sequencing of the actions is represented by the link created between flows of results that trigger new operations or apply constraints to other actions. There is no simple representation of synchronisation (constraints) or time in such a representation. Consistency with data models (datagraphs) can only be guaranteed by the designer’s vigilance.

Petri nets [Peterson 81] are essentially devoted to representing the various state changes that manifest changes in the system. The occurrence of events makes it possible to get through transitions, meaning to activate processing that itself will generate result events and cause a change in the system’s state. Sequencing of processing, synchronisation and constraints are easily represented in a simple formalism of places (with circles representing events), arcs with attributed values indicating the number of occurrences of an event required to trigger an operation, and transitions representing the triggered processes. However, these processes must be described textually, in the form of comments that are crucial for graphic representation. The great disadvantage to this representation is that it does not allow a close link with the data description, which must be supplied from another source and whose coherence and consistency are totally left up to the operator’s care.

The Merise Conceptual Model [Tardieu 83-85] aims to define the operations that are essential in the field under study, by revealing exclusively semantic links with no concern for organisation. The representation formalism is simple. The various processing operations are presented preceded by the events that trigger them, and followed by the events resulting from them, which in turn may trigger processing. Triggering events can be linked by a synchronisation condition that will be expressed by a Boolean condition using the logical operators AND, OR and NOT. The result events generated by an operation can be conditioned by emission rules (by default of any given rule, the rule is « always »).

Dynamic sequencing and synchronisation (if any) of all the operations making up a function are therefore formalised.

An operation (production of an information flow) is made up of actions (or elementary functions) that take place in sequence and refer to one or several management rules, rules that are textually described at the beginning and provide frameworks for accomplishment of the main processing operations.

An operation may make use of one or several entities and/or associations for actions to create, modify, delete or browse. These references ensure coherency between the data model and the processing model.

The Merise model can represent, in a manner quite similar to Petri nets, all the dynamic aspects of a system (sequencing and synchronisation), and creates the link with the data in a more explicit way. However, it relies on this separation of data and processing and cannot represent the exchanges and communication between data and processing modules.

1.3 Limitations

Computer modelling of the problem space is performed in terms that are easy to transcribe into a programming language: continuity between the models of the system life cycle
(SADT), software life cycle methods (SA and SD) and programming languages (structured languages). But those terms are remote from the actors and objects of the real world and the criticism that is sometimes voiced about conventional procedural languages is also applicable to these methods for designing information systems. In fact, the separation of data from processing, the hierarchical breakdown of processing into subprocess cycles, and exhaustive determination of their call links, mean that existing systems are difficult to define, to understand, to maintain and to improve. The present trend toward the concept of the object encourages communication (interface and message exchange) rather than data and processing and the DFD or MCC, even though they do give the «entrance points» to the various processing operations and therefore an initial idea of the exchanges between actors, do not describe internal communications and exchanges between modules at all [TABOURIER 91]. The contribution of the concepts of the object and of encapsulation provide a different viewpoint that might solve some of the problems relating to that separation.

2. Multi-agent systems

As we have just seen, the various models of the dynamic aspects in Information Systems are intended to identify the processing applied to entities of the real world, their sequencing and their breakdown into more basic elements.

This kind of approach relies on the separation of processing from data, as used in conventional programming languages that are employed to construct computer systems. Such an approach corresponds to a computer view of the problem space. Now, there is another approach that is closer, from a metaphorical standpoint, to the real world, and that emphasises real actors, their organisation and communication. This approach defines what are called multi-agent systems and makes up an independent research topic. In some disciplines, such as Distributed Artificial Intelligence or Concurrent Object Programming, it holds a place of its own. This section is intended to briefly present and describe such systems, their advantages for Information Systems and their current limits.

2.1 Presentation and Objectives

We can note that any system can be considered as being made up of a set of concurrent, relatively autonomous, entities that can cooperate to accomplish one or several tasks and reach one or several objectives. On the basis of this realisation, associated with the fact that parallel machines have become a reality, three families with increasing granularity have appeared that can be grouped under the generic term of open systems. Actor languages, actor groups (organisational programming) and multi-agent systems aim at modelling and simulating systems based on the cooperation between parallel and communicating entities.

2.2 Modelling and Examples

The actor model originates in the problems of exploiting parallelism and in distributed problem-solving [Hewitt 79], with the first languages appearing at the beginning of the 1980s [Lieberman 81]. Actors are self-contained computing entities with weak granularity which can be processed in parallel. They interact by sending unidirectional and asynchronous messages. Their processing generally results in creation of many short-lived actors. Examples of such systems are the Act2 language [Theriault 83] and the Gul Agha model [Agha86].
The notion of groups of actors was introduced in order to obtain entities with stronger granularity. This makes it possible to enlarge the actor model to entities with more complex behaviour, for example taking into account more complex messages [Jong 91].

Multi-agent systems [MAAMAW'92] for their part are focused on distributed problem-solving entrusted to a set of agents. An agent is an entity with a certain autonomy and a partial representation of its environment. It is capable of communicating with other agents and of taking action on itself and its environment. The solution can be defined as the result of interaction within the set of agents. The modes of cooperation then take on a preponderant role. Let us quote some methods from distributed artificial intelligence [Bond & Gasser 88], such as the blackboard [Hayes-Roth 88] and the contract net [Smith 80].

2.3 Advantages

The advantages of multi-agent systems (MAS) for Information Systems touch on three different levels.

The first is that of analysis. On this point alone, we obtain several benefits. Modelling is no longer a question of processing that must be broken down and structured, an approach that means the problem must be first understood and "solved" from a functional standpoint. Rather, it is done in terms of agents, societies of agents, and communication. This kind of modelling allows analysis that is very close to a real situation, with agents modelling the actors in the real world. This approach is modular by nature, since each agent corresponds to a different module. In this case we have advantages that are equivalent to those of object-oriented technologies in comparison to more conventional systems. Analysis is clearer and more intelligible. Finally, previous analyses can be re-used, either by re-using some agents in their existing form or by specialising them.

The second level concerns the open character of the system. Control and communication mechanisms are not fixed in advance. It is then possible, according to concrete cases, either to define new modes of cooperation, or to use predefined ones (Contract net, Blackboard). Similarly, it can be possible to define the structure of the information exchanged in a complex manner.

Finally, the third level is that of computer simulation, whose importance is increasingly recognised in the analysis phase. It allows not only the validation of a specification, but also, for example, comprehension of the circulation of information between agents and supervision of their activation.

2.4 Limitations

While the MAS approach is in fact very interesting and represents a promising path for research in Information System modelling, there are still several problems with it. We could cite in particular a problem of methodology that cannot help but recall object-oriented languages. There is also a problem of formalisms, that must permit expression of both organisational structures and communication or control protocols (since everything cannot be expressed from the point of view of an agent, a functional analysis or more conventional control structures must be easily describable). Finally, it is vital to have operational MAS to simulate the analyses made. Existing actor languages have too weak granularity to be directly usable for analysis of a system.
3. The BAAL Agent Language (BAsic Agent Language)

Our objective is to model and simulate systems that can be seen as a society of autonomous, concurrent, communicating and cooperating agents.

BAAL is intended to be a development environment for multi-agent systems in which the designer is invited to define and then test his own societies of agents as well as his own communication and control protocols. BAAL is the continuation of the work that was begun as part of an ESPRIT project and that resulted in a first agent language, LRO3 [Roche & al 93].

3.1 The BAAL agent

A BAAL agent is an entity that is:
- autonomous to the extent that it possesses the resources needed for its accomplishment,
- concurrent, since it is executed in parallel with the other agents,
- communicative by means of protocols that may be complex,
- cooperative by inserting itself in organisations to define more complex entities with higher granularity.

The originality of the BAAL language mainly lies in the diagram defining agents and in how its elements are organised. Two remarks led us to adopt two basic principles in defining the BAAL language.

The first was that an agent can have complex types of competence, and each of them can be an agent in itself. The first principle is therefore the equivalence between the "agent" and the "society of agents" that makes it possible to define an agent through the society of its competence agents. This principle corresponds to a descending and hierarchical breakdown of agents, and requires the introduction of primitive agents that cannot be broken down.

The second remark concerned the diversity and complexity of an agent's communication and control protocols. This led us to introduce a meta-level to specify the interface and communication. Each of these functions is then the responsibility of one particular agent. This kind of reflexivity in the system is a crucial condition for its evolution. Indeed, the interface agent and the control agent only need to be modified, for example, to define different communication protocols.

3.2 An agent as a society of agents

A BAAL agent is defined as a society of cooperating agents spread out over two different levels. Its behaviour is defined in terms of the cooperation and collaboration between those agents.

The competence level
The competence level gathers the agents called competence agents which define the "know-how" of the agent society to which they belong. These competence agents share common data called the universe of the agent. Let us remark that these competence agents are similar to the methods in object-oriented languages, just as the universe is to instance variables.
The meta-level
The meta-level describes how an agent handles received messages and how it handles the competence agents. So it contains two different agents in charge of these functions.

The interface agent
Generally communication functions are simple, fixed mechanisms (object-oriented languages, actor languages). Now, the complexity of the problems to be solved and the diversity of possible solutions require communication protocols between agents that can be complex and variable from one agent to another. For example, a given agent will take a given priority into account whereas another one will not have to. The communication function is therefore entrusted to a special agent of the meta-level: the interface agent.

Any message sent to an agent is then automatically re-dispatched to its interface agent, which in its turn re-directs the message to its own interface agent. This process is repeated until reception of the message by a primitive agent whose competence is to actually process the message received.

The control agent
Management of competence agents, that are autonomous and concurrent, is linked to the interpretation of high-level messages, and makes the control of an agent complex and variable from one application to another. This is why a meta-level agent, the control agent, is in charge of this function. Its role is to process messages received and give them an appropriate treatment according to the work underway in the competence level.

The meta-universe
Like competence agents, meta-level agents share a common set of data: the meta-universe. For example, its role may be to stock messages received.

![Figure 1: Reflexive structure of a BAAL agent](image-url)
3.3 The sending of messages

The basic communication protocol of BAAL is asynchronous message passing. This means that the sending agent does not wait for a reply to continue its activity and that the receiving agent does not stop to handle the message. The asynchronous approach requires the existence of a continuation agent to which the answer to the message should be sent.

Here is the syntax of the BAAL sending message.

\[ \text{Send } \text{dest: agent1 mess: message cont: agent2} . \]

where \text{agent1} is the receiver and \text{agent2} the continuation

From such a basic sending message, one can set up more complex communication modes. To do that, complex message structures can be defined including a name, parameters, priority, goal, and so on. All this information will be handled under the control of the receiver agent. Such an example will be presented in 3.7 below.

Let us note that message sending is done in an environment where the receiver knows the sender agent, the society to which it belongs to and its universe.

3.4 The primitive agents

This recursive definition of agents in terms of societies of agents means that there must be undividable primitive agents. These are the very first agents, from which agents with more complex behaviour will be created.

The interface and control of a primitive agent are as simple as possible. All the primitive agents have the same interface and control, i.e. they receive and manage their messages in the same way. They only differ in their behaviour and their state. This competence is unique and corresponds to a procedure in the traditional sense.

The competence level

Competence is a program written in the host language that may contain BAAL instructions, including sending of asynchronous messages. To make a parallel with object-oriented languages, competence corresponds to an object that has only one method.

The universe defines the agent’s state. It is made up of a set of remanent data that only the primitive agent’s competence has access to. The universe is the equivalent of instance variables in object-oriented languages.

The meta-level

The meta-universe stores all the messages received by the primitive agent. It will be considered by the interface and by control as a queue managed by FIFO principles.

The interface allows a primitive agent to accept any message it receives, even if it is in the midst of executing its competence. The message received is stored according to the order in which it arrives in the meta-universe.
Control considers each message according to the order in which it arrives, activates the competence with the contents of that message as parameter, and sends the result to the following stages.

![Primitive Agent Diagram](image)

**figure 2**: primitive agent

### 3.5 Types of Agents

The definition of an agent, whether primitive or not, relies on a preliminary definition of a type of agents, similar to the classes of the object-orient languages. Then it is possible to define classes of agents hierarchically. However, the resulting inheritance relationship is different from that of object-oriented languages because the agents are independent and must have all the resources needed for their processing.

### 3.6 Implementation

The BAAL language has been implemented in a computer system designed according to an object-oriented methodology (O.O.A [Coad & Yourdon 91]) and written in the object-oriented language Smalltalk80 [Goldberg & Robson 85] by ParcPlace Systems, release 4.1, on an Intel workstation. The choice of Smalltalk80 was motivated by the many advantages that language offers, in particular for time-slicing.

The implementation of BAAL was broken down into four stages: implementation of non primitive agents, implementation of primitive agents, management of message sending, and achievement of time-slicing. In this section we deal only with management of message sending and achievement of time slicing.

**Message sending**

Management of messages, i.e. sending itself and then distribution, is achieved through the object Send, the only instance in the class Mailbox. This object corresponds to a SharedQueue to store the messages received.
Processing strictly speaking of the messages stored in the mailbox consists in re-directing them to the interface agent of the addressee agent if the latter is not a primitive agent. If it is, the interface method of the primitive agent is executed with the message received as parameter.

Achievement of time-slicing
Primitive agents are the only active entities. In order to ensure the concurrence of primitive agents, a Smalltalk80 process is associated with each request for execution of a competence by a primitive agent. All the processes have the same priority. Time-slicing is achieved using a scheduler, a process that always has higher priority than the processes associated with the primitive agents. The scheduler executes the following tasks in succession and indefinitely:
- asks the mailbox to process the stored messages,
- selects a primitive agent process (by default from a circular list)
- suspends itself for several dozen milliseconds to permit the selected process to be executed.

Class hierarchy
The core of the BAAL language is made up of the five following Smalltalk80 classes: Agent, PrimitiveAgent, BAALMessages, MailBox, TimeSharing and 2 instances Send and TS.
To build a new application, the programmer must, depending on his needs, first define subclasses of the classes Agent and PrimitiveAgent, and then instances of those new classes for simulation.

3.6 Example
As we have just seen, a primitive agent processes messages received according to the order of arrival. It is impossible to change that order, for example to take into account a priority associated with the message.

The following example shows how one can construct a non-primitive agent whose interface and control enable it to handle message in a more elaborate mode, for example taking into account a priority. We would like to define an agent, let us call it QueueAgent, able to handle the oldest message of highest priority first. Such an agent would receive messages of the form: `<aMessage>` ::= (priority content)

A QueueAgent is a non-primitive agent defined by its competence level and its meta-level.

Competence level
The competence level is made up of a single competence agent. This competence agent will receive the content of the oldest message of highest priority sent by the control agent of the QueueAgent.

Meta-level
The interface and control agents on the meta-level must cooperate via the meta-universe to manage messages received according to their priority. This is why the meta-universe is a collection of queues whose indexes correspond to all the possible priorities of received messages.
The interface is a primitive agent whose the competence consists in adding the incoming message to the appropriate queue of the meta-universe (i.e. the queue with the same index as the priority of message).

From BAAL for Smalltalk 80
PrimitiveAgent subclass: #QueuePrimitiveInterface
interface: aMessage
  "inherited from PrimitiveAgent"
control
  "inherited from PrimitiveAgent"
competence: aMessage
  Semaphore forMutualExclusion critical:
    [society metaUniverse at: (aMessage priority)
      addLast: (Message content: aMessage content cont: continuation)]

The control component is also a primitive agent whose skill consists in getting the oldest stored content of message from the highest priority queue and sending it to the agent of the competence level of the QueueAgent.

From BAAL for Smalltalk 80
PrimitiveAgent subclass: #QueuePrimitiveControl
interface: aMessage
  "inherited from PrimitiveAgent"
control
  "inherited from PrimitiveAgent"
competence: aMessage
  |message| "message is a local variable"
  [true] whileTrue:
    [ Semaphore forMutualExclusion critical:
      [message := society metaUniverse getFirstMessage ].
      message isNil ifFalse: [Send dest: (society competence)
        mess: (message content)
        cont: (message continuation) ]].

This example shows that a class of QueueAgent can be defined, whose meta-level will always be made up of instances of the primitive agents QueuePrimitiveInterface and QueuePrimitiveControl. The QueueAgent class is easily created by the following Smalltalk80 statements:

From BAAL for Smalltalk 80
Agent subclass: #QueueAgent
interface
  "Creation of a new interface agent"
  ExamplePrimitiveInterface new
control
  "Creation of a new control agent"
  ExamplePrimitiveControl new
metaUniverse
4. The flexible workshop example

To illustrate our approach, we present a model, in terms of BAAL agents, of a simplified example of flexible workshop management. A flexible workshop is composed of a set of agents of different natures cooperating to manufacture a set of parts. There are therefore "manufacturing island" agents in charge of manufacturing the parts, whose behaviour and cooperation with the other agents are simple. "Wire-guided truck" agents have the function of transporting parts from one island to another. They access the network in a conflictual manner and must therefore cooperate, and in particular be synchronised thanks to "node" agents. The "node" agents are in charge of managing the network on which the manufacturing islands are located. They are the agents that authorise circulation of the "wire-guided truck" agents.

Let us see in greater detail the two main classes of agents, which are the node agents and the truck agents.

The network is made up of segments. These segments are not agents but passive objects described by a name, coordinates and a state. The state of a segment indicates if the segment is occupied by a truck or not. As the state of a segment can be modified by several nodes, it is necessary to protect it using a semaphore.
A node is a primitive agent, instance of the Node class. A truck sends a node a message asking it to return a free segment in order to be closer to the island it has to reach. So the continuation of the message is the truck itself.
note: the ^ symbol in a Smalltalk80 program means the return function.

From WorkShopExample for Smalltalk 80
PrimitiveAgent subclass: #Node

Interface: aMessage
  "inherited from PrimitiveAgent"

control
  "inherited from PrimitiveAgent"

competence: aMessage
  "the received message is a list made up of the asking-truck segment and of the island segment to be reach"

listOfSegments := ((aMessage getTruckSegment) to: (aMessage getIslandSegment)).

"listOfSegments is an ordered list made up of all the segments reachable from the node"

listOfSegments do: [:seg |
  Semaphore forMutualExclusion critical:
  [seg isStateFree ifTrue: [ seg stateOccupied.
    ^ Send dest: continuation
    mess: #('moveTruck' seg)
    cont: nil ] ]].

"if there is no free segment"
Send dest: continuation mess: #('moveTruck' nil) cont: nil

The behaviour of a truck is quite simple. In fact it has only to ask a node to give it a free segment in order to reach its manufacturing island. If no free segment is available, the truck asks again until a segment is free.

But the core of the BAAL language does not provide synchronous message sending, so we are obliged to introduce different agents to achieve such a mechanism. The following figure illustrates how to implement it.

* a synchronous sending message does not need continuation which is the sender itself

figure 4: from synchronous to asynchronous sending message
It is the reason why a truck is a non-primitive agent made up of two different competence agents, one for asking a node for a free segment, the second one to move the truck.

The meta-level of a truck is simple. As a primitive agent, the meta-universe is a queue and the interface agent stores the incoming messages in the order of their arrival in the queue. The contents of messages must be structured according to the following frame: `<name-of-agent> <parameters>` where the name-of-agent is the name of one of the two competence agents. The control agent handles the oldest messages and sends the parameters of the message to the corresponding competence agent. Such interface agents and control agents are useful, so there are predefined primitive agent classes which achieve this behaviour.

```
From WorkShopExample for Smalltalk 80
Agent subclass: #Truck
  interface
    "Creation of a new interface agent"
    PredefinedSimpleInterface new
  control
    "Creation of a new control agent"
    PredefinedSimpleControl new
  metaUniverse
    SharedQueue new
  competenceLevel
    competence add: AskSegment new.
    competence add: MoveTruck new
  universe
    universe at: #truckSegment put: nil.
    universe at: #goalSegment put: nil "the manufacturing-island segment to reach"
```

Let us just see the primitive agent class MoveTruck. The behaviour of a MoveTruck agent is to move the truck to the next segment and to free the previous segment it occupied.

```
From WorkShopExample for Smalltalk 80
PrimitiveAgent subclass: #MoveTruck
  interface: aMessage
    "inherited from PrimitiveAgent"
  control
    "inherited from PrimitiveAgent"
  competence: aMessage
    "the received message is a name of a free segment or nil"
    seg := aMessage.
    "if I reached the manufacturing island, I stop"
    (seg = (universe at: #goalSegment))
    ifTrue: [] Semaphore forMutualExclusion critical:
      [(universe at: #truckSegment) stateFree].
      universe at: #truckSegment put: seg.
      ^ society stop ].
    "otherwise and if there is no free segment, I send my society a message asking it
```
5. Conclusion

The implementation of a BAAL-type agent language and its use on examples such as that of the flexible workshops have shown us the advantages to multi-agent systems for modelling and simulating systems. Therefore, when faced with a complex problem where a functional approach would be difficult and rigid, multi-agent systems make it possible to break down the complexity into simpler elements, the agents. Modelling in terms of agents is easier to understand, as it is closer to the real world. Furthermore, simulation of such systems not only makes it possible to validate the analysis, but also to follow step by step the future behaviour of the system and circulation of information. It is noteworthy that, in the case of our flexible workshop example, the system achieved is capable, by virtue of its very design, to function in the impaired mode in the event, for example, that a segment should be definitively out of access consecutive to a breakdown.

Nonetheless, despite these advantages, many problems persist. The first is that of a methodology of multi-agent systems that recalls that of object-oriented languages. The second problem is that of the formalism in the agent language, which must be able to express different communication, control and organisation modes. The reflexive approach is one of the solutions to this problem, and it is the one we adopted in designing the BAAL language. It offers the advantage of a simple and powerful formalism, with the extensions of the BAAL language written in BAAL. It also has the inconvenience of its simplicity. Indeed, it becomes very difficult to understand precisely how non primitive agents behave, and to correctly separate the behaviour of the meta-level agents from that of the competence agents.

Multi-agent systems are still in the research stages, and define a very promising focus for work.

References


[Chen 76] Chen P.: "The Entity-Relationship Model - Toward a Unified View of Data", AMC transactions on database systems, Vol.1 n°1 mars 1976


[Goldberg & Robson 85]: "Smalltalk-80, the language and its implementation", Addison Wesley, 1985


[Roche & al 93] Christophe Roche, Yves Dumond & Serge Stinckwich: "LRO3 : un modèle pour la programmation par objets parallèle" revue Génie Logiciel, n°32, septembre 1993


AUTOMATED DESIGN OF ELECTRONIC TRADE PROCEDURES USING DOCUMENTARY PETRI NETS

Ronald M. Lee, Roger W.H. Bons, Clive D. Wrigley, René W. Wagenaar
Erasmus University Research Institute for Decision and Information Systems (EURIDIS)
Erasmus University, PO Box 1738, 3000 DR Rotterdam
The Netherlands, tel. 31 (0) 10 4082601 email rlee@fac.fbk.eur.nl

Keywords: Open-edi, Petri Nets, Documentary Petri Nets, Procedure Constraint Grammars

Abstract.

Global information infrastructures can offer tremendous opportunities for small and medium enterprises to do global business electronically. But communication networks alone are not sufficient to enable international electronic trade. Parties have to know about each others' "way of doing business" before they can start exchanging data electronically; they have to agree upon the trade procedure they are going to follow.

Since the costs to reach such an agreement are very high, electronic linkages have been hardly realized in 'open' or short-term trading relationships. A solution to this problem could be the availability of standardized trade procedures. Two steps must be taken when developing such procedures: a common language should be defined and next, industry groups should specify standard trade procedures using this language.

In this paper the Documentary Petri Net representation is introduced, which can be used as a language to describe electronic trade procedures. Furthermore, the paper demonstrates how these trade procedures can be designed within the Case/Open-edi environment using this representation.

It is shown that both top-down and bottom-up approaches are supported in the design process. Both these approaches will be necessary since these industry groups might either start with the definition of the procedure for all the individual parties and then try to integrate these into an overall procedure (bottom-up), or they might start on a global level and then gradually refine the procedure to the specifications of the individual parties.
1. Introduction.

Global information infrastructures are rapidly becoming a reality. Such worldwide networks help companies to operate not only on a local or regional level, but also on a global level. Especially for small and medium enterprises (SMEs) this would offer tremendous opportunities to do global business electronically. However, communication networks alone are not sufficient to enable international electronic trade.

In the past it has been shown that the introduction of Electronic Data Interchange (EDI) can have tremendous benefits for the efficiency of trading both between and within organizations (see for instance the proceedings of the annual EDI conference in Bled, Slovenia). On the other hand it can also be shown that in many cases long and costly negotiations are necessary between the trading partners before they can exchange their first EDI message\(^1\).

As a result, most successful EDI implementations have been realized in what could be called 'closed trading relationships', i.e. long-lasting trading relationships, involving a high number of transactions, between parties that have a high level of trust and possibly a close coordination of the parties' business processes (Table 1). In these kind of relationships, parties can gain extra benefits by closely coordinating each others' actions, thus compensating for the extra start-up costs stemming from detailed trading partner negotiations. This process is an example of business process redesign or re-engineering.

However, when the partnership is established for a limited period, covering a few transactions only and on an "at arms' length" basis, EDI linkages are seldom observed since the costs of the necessary negotiations cannot be recovered from the benefits. These shorter-term partnerships could be called 'open trading relationships' (Table 1). The main aim of our research is to contribute to the lowering of the barriers for using EDI in these open trading relationships.

<table>
<thead>
<tr>
<th>Level of Trust</th>
<th>Open</th>
<th>Closed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Transactions</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>Duration of Relationship</td>
<td>Short</td>
<td>Long</td>
</tr>
<tr>
<td>Level of Coordination</td>
<td>Low</td>
<td>High</td>
</tr>
</tbody>
</table>

**Table 1. Open vs. closed trading relationships.**

One of the main reasons for the complexity of the negotiation process is the fact that parties have to know about each others' "way of doing business" before they can start exchanging data electronically. Extra knowledge about the preferred way of doing business of one trading partner has to be conveyed to the other; in other words, the parties have to agree upon the trade procedure\(^2\) they are going to follow.

We define a trade procedure as *the mutually agreed upon set of rules that governs the activities of all parties involved in a set of related business transactions*. Thus, a trade procedure controls all interactions among the roles involved. A trade procedure stipulates which actions should be undertaken by which parties, the order in which these actions should be performed and possibly the timing constraints on the performance of these actions. Actions of parties include the sending and/or receiving of goods, documents or funds.

\(^{1}\) Baker (Baker 91) gives an example of the size of such negotiations: "At one conference on EDI law, James Pitts, a purchasing manager at R.J. Reynolds, said he spent 18 months negotiating a single trading partner agreement. That left him with only 349 other trading partners to go ".

\(^{2}\) It should be noted that although we call these agreements 'trade procedures', the principle is applicable to other societal areas than trade. The main focus of this paper however is electronic commerce which explains the term 'trade' in the definition. Other terms used to describe this concept are: trade scenarios, business scenarios and business protocols (Wrigley 1992; Wrigley et al. 1994).
A major reduction in these negotiation costs could be achieved through the availability of standardized (electronic) trade procedures, specified by industry groups such as EDIFICE, SWIFT or CEFIC or international trade facilitation bodies such as UNCTAD or the International Chamber of Commerce (ICC). Although international standards for the structuring of EDI messages exist (i.e. UN/EDIFACT or ANSI X.12), such standardized trade procedures have not been developed yet. In order to build these procedures, two steps should be taken:

- The first step is the definition of a common language in which these procedures are described. This language should be formal, graphical and computer interpretable.
- Second, groups of business experts should specify these standard trade procedures. The procedures that these groups define should be then registered in a repository, governed by an international body. Computer aided support is desirable in this process, in order to assess several properties of these procedures, to simulate these procedures in a testing environment before implementing them and finally, to support the reusability of certain parts of trade procedures.

If these trade procedures are specified in a computer interpretable language such as the DPN formalism, a third step can be taken. Companies might download such procedures from the repository. Then, the information system of these companies can interpret the procedure and use this as a decision support tool, or even more effectively, let the computer system conduct most of the trading automatically. This will provide them with strong guidance during the execution of the trade procedure.

The remainder of this paper is as follows. In Section 2, the Documentary Petri Net (DPN) formalism is proposed as a representation technique for trade procedures. Section 3 shows how the design of trade procedures can be supported using the DPN formalism. It demonstrates both a bottom-up and a top-down approach. Finally, Section 4 draws conclusions and gives future research directions.

2. The Documentary Petri Net Formalism.

An ISO/IEC sub-committee (ISO/IEC JTC1/SC30) is working on the definition of standards that should minimize the set-up costs for new EDI Linkages. This initiative is called "Open-edi". Open-edi is "EDI among autonomous, multiple participants using public standards and aiming towards interoperability over time, business sectors, information technology systems and data types, capable of multiple, simultaneous transactions, to accomplish a explicit shared business goal" (ISO 1994). These standards include both technical and business aspects of interoperability. In the business aspects work is being conducted on the specification of a standard formal description technique for trade procedures. The authors of this paper participate in these standardization efforts.

In order to maintain coherence with these Open-edi standardization efforts, a representation formalism should include the modeling primitives as proposed by this sub-committee ("Open-edi scenarios", "Roles", "Information Parcels" and "Scenario attributes" (ISO 1994)). However, our research has shown that further requirements should be posed on a representation language. These requirements are listed below. It should be noted that this list is preliminary and may be extended as research progresses. In this stage, a distinction can be made between formal requirements, notational requirements and verification requirements.

- Formal requirements include the possibility to express concurrency, choice (internal to a party) and contingency (external to a party) and the representation of deontic and/or legal relationships and changes thereof. It should also be possible to explicitly model time, both absolute and relative.
- Notational requirements include the possibility to represent trade procedure designs in a graphical way. Also, there should be a way to hierarchically decompose a trade
procedure into a number of levels. This is also reflected in the need to be able to model roles as proposed by SC30.

- Finally, automated verification and/or performance evaluation of the models should be possible. This verification includes, but is not limited to, properties such as boundedness and liveness of a trade procedure, but also constraints such as the legal soundness of a procedure and measures whether insufficient or superfluous controls are established in the trade procedure.

We found Petri Nets as being one of the few acceptable candidates that offer both a graphical representation and a formal basis for the verification of various properties of these nets. The main advantage of the Petri Net formalism, in addition to its capability to graphically model both concurrency and choice, is that it offers various kinds of both formal and informal analysis methods, which make Petri Nets especially suitable for modelling "Discrete Dynamic Systems" (Van der Aalst 1992a). In the remainder of this section, we introduce the Documentary Petri Net representation: an extension to the classical Petri Net formalism we developed to satisfy the requirements previously mentioned.

Classical Petri Nets (Petri 1962; Peterson 1981) satisfy the need for expressing concurrency and choice. A classical Petri Net is a bi-partite, directed graph. It has two disjunct sets of nodes: places (represented as circles) and transitions (represented as bars). Arcs connect places with transitions or vice versa (it is not allowed to connect two places or two transitions). The dynamic behavior of the modelled system is represented by tokens flowing through the net (represented as dots). Each place may contain several tokens (the marking of the place); a transition is enabled if all its input places (i.e., arcs exist from those places to the transition) contain at least one token. If this is the case, the transition removes one token from each input place and instantaneously produces one in each output place (i.e., an arc exists from the transition to the place). This is called the 'firing' of a transition. Transitions in Documentary Petri Nets are labeled in order to identify the role that brings about the transition. The syntax of these labels is Role(s) : Action.

Classical Petri Nets only allow the modelling of relative time, but not absolute time. However, it should be possible to specify timers, e.g. for modelling contractual deadlines. This can be referred to as 'timed Petri Nets' (Van der Aalst 1992a; Peterson 1981). Documentary Petri Nets allow the specification of timers in the following manner. Setting a timer is modelled by putting a token in a place labeled X:TimerSet. This place is the input place of a transition that is labeled timer: Timer_condition. This transition has an extra constraint on its firing rule: the timer condition has to be satisfied, in addition to the availability of tokens in each of the input places. It then fires a token into a place labeled X:TimerExpired. This place will be in most cases one of the input places of a transition representing the action to be taken if the timer expires. An example of such a timer condition is timer: current_date >> expiry_date.

The classical Petri Nets only allow one kind of token. In order to distinguish between different types of information parcels, different types of tokens have to be distinguished. This is referred to as 'colored Petri Nets' (Peterson 1981; Jensen 1990). A similar extension of the classical Petri Nets are the Predicate/Transition nets, in which logical predicates are associated with transitions (Genrich and Lautenbach 1979 & 1981). Documentary Petri Nets use colors and predicates to specify the different information parcel types, goods, funds and deontic states.

The exchange of information is based upon the concept of information parcels in the Open-edi reference model. Information parcels are modelled using document places. A document place is represented by a square box. These kind of places have labels that identify the information parcel type. Information parcels can be sent or received by a specific role. Sending such a parcel is represented by a transition labeled X to Y: D, in which X identifies the sender, Y the receiver and D the type of parcel that is exchanged. This transition has a document place of type D as an output place. It will be
part of the sub-net describing the behavior of role X. Conversely, receiving an
information parcel is modeled by a transition labeled \textit{Y from X: D}. This transition
has a document place of type \textit{D} as an input place. This will be part of the sub-net for
role \textit{Y}.

- Goods are represented as cube places. A description of the goods, including quantity,
weight, volume, quality etc. may be added. The transfer of goods among parties is
modeled using the same primitives \textit{X to Y: G} and \textit{Y from X: G} as used in the
modelling of information exchanges, but in this case \textit{G} refers to the goods description.

- The exchange of funds is modeled similar to the modeling of information. Since the
concept of money is closely related to documents (a 100 dollar bill is a performative
document), we use the document places to denote funds transfer. In the description of
these documents, the amount and currency are specified in the structure of these
documents.

- The deontic states of each individual role, as seen by the other roles, are modeled
using the classical Petri Net control places and tokens. They are represented by circular
places, and labeled with a description of the deontic state. An example of such a
description is \textit{oblig(X,A)} which means that party X has an obligation to perform
action A.

One important aspect of modeling complex scenarios is the ability to model the view of each
party as a separate Documentary Petri Net. This allows the decomposition of a trade procedure
into a number of logically separate sub-nets. This modeling style results in a clear
"geographical" separation between the roles. As the role description is a sub-net in the scenario
description, designers have some flexibility for experimenting with different role descriptions
within the overall scenario constraints.

A state transition is enabled by receiving an information parcel, goods or funds, or the
expiration of an internal timer (events). Firing a transition can lead to sending information
parcel(s), goods or funds and/or setting an internal timer (actions). An example of a
Documentary Petri Net model is presented in Figure 1. Upon receiving a certain information
parcel of type \textit{D} from role \textit{Y}, role \textit{X} is obliged to send goods \textit{G} to role \textit{Y}.

![Figure 1. An example of a Documentary Petri Net model.](image)

3. Supporting the Standardization of Trade Procedures.

This section discusses how the Documentary Petri Net formalism can be used in the
standardization efforts of the industry groups. For this purpose Case/Open-edi was developed

\footnote{All figures in this paper are prepared using the Case/Open-edi software running on a Apple Macintosh Quadra
900 computer.}
by Lee (Lee 92). Case/Open-edi offers a graphical interface to specify trade procedures in the DPN formalism. The graphical specifications are compiled into executable code which is used to simulate these trade procedure specifications in a laboratory setting. Furthermore, several analyzing techniques are built in to support the validation of proposed trade procedures. The interested reader is referred to the Dynamic Modelling III proceedings for further details on this Case tool.

In this section, two modelling approaches are presented. The first, a bottom-up approach, is perhaps most typical of practice where organizations involved develop their own trade procedures relatively independently. Here, we use the DPN representation as a canonical language for representing those separate procedures. Then these separate procedures have to be combined into an integrated trade procedure applicable to all parties involved.

The second modelling approach is a top-down one. Here we make use of Procedure Constraint Grammars (PCGs) to produce a single, integrated Petri net of the trade procedure. Following that, separate Documentary Petri Nets are automatically generated for each of the roles.

Case/Open-edi supports both modelling approaches. Potential users of the tool are user groups formed from business experts. These groups might be on an industry level (SWIFT, EDIFICE, CEFIC) or governed by an international organization such as the United Nations (UN/CE, UNICID, UNCITRAL), the ISO or the International Chamber of Commerce.

The Documentary Petri Nets representing the behavior of each of the parties involved can be easily distributed over multiple machines running Case/Open-edi. The exchange of information, goods and funds among these parties is represented as tokens exchanged among the machines through an electronic network. This setting can then easily function as a testing or gaming environment for trade procedures.

3.1 Bottom-Up Design of Electronic Trade Procedures.

Most current inter-organizational trade procedures were gradually constructed over the last decades and sometimes even over the last centuries. As technology evolves, these procedures have to be adjusted or redesigned in order to benefit from new possibilities offered. As pointed out in Section 2, Open-edi is such a development that might dramatically change the common business practice.

In the coming decade, user groups will be formed to define new common ways of doing business with participants from different industries, for example banks, transportation and production companies. In these groups the participants will have to specify their preferred way of doing business. Then, these specifications have to be combined in order to define the total trade procedure for all parties involved. This process might involve multiple iterations, since all parties involved need to reach consensus on the final integrated trade procedure.

Using our modelling approach, this means that the DPNs of the several roles have to be connected. This can be simply done by connecting the roles at their communication points: the document and goods places. For example, in the role description of role X there is a transition labeled X to Y: D with an output document place of type D. In the description of role Y there should be a transition labeled Y from X: D and an input place of type D. The two roles can now be connected by replacing the two document places of type D in the sub-nets by one of the same type, with an incoming arc from the transition in role X and outgoing arc to the transition in role Y.

3.1.1 Analytical Verification.

When a total trade procedure has been constructed based on the individual specifications of the procedures of the parties involved, two formal properties play an important role: liveness and boundedness.
Liveness can be defined as the absence of deadlock states. A trade procedure is said to be in a deadlock state when two or more parties are waiting for each other's actions, but these actions will never be performed. In a Petri Net representation this means that the sub-nets of some parties are blocked without reaching their final state and without the possibility of this ever happening. An example of deadlock is given below in Section 3.1.4.

Boundedness can be defined as the absence of endless loops in a procedure. If parties keep exchanging the same documents without making progress the procedure will never be completed. In a Petri Net, this can be translated as a situation in which the markings (i.e. states) of the nets are in a cycle that can never be exited, leading to a situation in which the final states of some parties can never be reached.

The difference between liveness and boundedness is that, although in both cases parties cannot complete their procedures, in a deadlock state they are waiting for each other whilst in the case of an unbound procedure they keep 'running around in circles'.

In literature many ways to analyze these formal properties are proposed (see for instance (Murata 1987; Jensen 1990)).

3.1.2 Heuristic Verification: Audit Daemons.

In addition to these analytical techniques, we also propose heuristic detection of control weaknesses in the procedures. These heuristic techniques are based on a device we call *audit daemons*. Audit daemons are drawn graphically using the Petri net notation described above. However, they do not represent entire procedures, but rather match segments or regions in the procedure. Numerous audit daemons are applied to the object procedure, each looking for a possible weakness or fault. When an audit daemon pattern matches a part of the procedure, the user is given a diagnostic message. The auditing of procedures is thus somewhat similar to syntax checking in programming language compilers. However, the faults identified here are not simply syntactic, but focus on the semantic character or business requirement of the procedure. For example, a business rule may stipulate that two specific places may not be marked simultaneously, as this could lead to collusion among roles. The interested reader is referred to (Chen and Lee 1992) for a more detailed discussion of audit daemons.

3.1.3 Simulation and Animation.

In order to allow parties to check the efficiency of a trade procedure design it should be possible to simulate trade procedures measuring throughput time etc. Furthermore, animation and/or gaming are useful tools to make a proposed design understandable for the participants of such user groups, since several intangible properties of EDI linkages cannot be validated using quantitative techniques (Streng 1992). These properties may be assessed through gaming experiments, using a laboratory setting of such a trade procedure. The parties involved can then experiment with these procedures. Examples of such inter-organizational gaming environments are the Port of Rotterdam game (Wagenaar 1992) and the IOS-game (Wrigley 1994).

3.1.4 Example of Bottom-Up Design.

To illustrate the design of a trade procedure we use a small business example. It includes a buyer, a seller and potentially a carrier and/or a bank. The buyer and seller exchange goods for money. We limit this example to three possible payment modes: pre-paid sale (seller gets money before he ships the goods), post-paid sale and payment through an escrow. An escrow is a service offered by the bank, which offers some extra security to the buyer and seller. The buyer puts the amount payable on an account at the bank. The bank notifies the seller of this account, upon which the seller starts the shipment of the goods. As soon as the goods are delivered, the buyer instructs his bank to transfer the money to the seller.
Furthermore, we distinguish three transport modes in this example. The seller might either deliver the goods himself, the seller might hire a carrier or the carrier might be hired by the buyer. Although this is very simplified example it will illustrate the difficulties that might arise in the bottom-up design. The three possibilities for payment mode and transport mode result in a total of nine possible trade procedures for both the buyer and the seller. If a carrier is involved in the procedure, he might follow two possible procedures, depending on who is paying him. Finally, since we assume that the bank is only involved in case of the escrow payment, there is only one possible procedure for the bank. Figure 2 shows some examples of these role descriptions in the Documentary Petri Net formalism.

To illustrate the problems that might arise when the individual role descriptions of the parties have to be combined into an integrated trade procedure we regard the simplest case combining a buyer role with a post-paid sale and a seller role with a pre-paid sale, both without a carrier involved (Figures 2.2 and 2.3 respectively). Figure 3.1 shows that the combination of these two roles leads to a deadlock situation at once (the transitions @buyer from @seller: goods and @seller from @buyer: money will never be enabled since they are waiting for each other). Of course in this simple case an easy solution can be negotiated, for example the buyer settles for a pre-paid sale as well (Figure 2.1), yielding the integrated procedure of
Figure 3.2 which is deadlock free. However, it will be clear that if the procedures of the individual roles get more complicated and the amount of roles increases, the complexity of the integrated procedure increases and automated tools are necessary to analyze liveness and boundedness.

Figure 3. The integration of individual role descriptions into a trade procedure

3.2 A Top-Down Approach of Inter-Organizational Procedures Design.

An alternative to the bottom-up approach is a top-down one where the parties first agree on a single, integrated trade procedure, and then transform that into the various role procedures. (Thus, a role procedure might be regarded as a 'view' of the integrated one). However, the issue arises as to how the parties might effectively collaborate on the design of this integrated procedure. One might imagine that standardized procedures are made available. For example, the parties might download such a standard procedure from a central library, and customize it as needed for their purposes in an automated Joint Application Design (JAD) session.

3.2.1 Procedure Constraint Grammars.

In order to be able to make the necessary abstraction for a top-down approach, a more general representation is needed. Below the procedure constraint grammar (PCG) formalism is presented, which can automatically generate the integrated procedure described above. The PCG has a certain resemblance to the 'definite clause grammar' representation often used in AI for natural language parsing or sentence generation. However, rather than generating textual sentences, our use of the PCG is to generate procedures, in the form of Documentary Petri nets. Rules in procedure constraint grammars have the following form:

\[
\begin{align*}
\langle call \rangle & \Rightarrow \{ \langle selection\_conditions \rangle \} \langle calls \rangle \\
\langle call \rangle & \Rightarrow \{ \langle selection\_conditions \rangle \} \langle DPN\_pattern \rangle \\
\langle call \rangle & \Rightarrow \langle action \rangle
\end{align*}
\]
Each of these rules is regarded as a constraint on the graph to be generated. The `<selection_conditions>` are a list of parameter comparisons, indicating tests to determine if the constraint is applicable. The `<calls>` of the first rule are one or more references to the `<call>` of other rules, separated by commas. The second rule form is not textual but rather contains a documentary Petri net graph (usually with variable arguments). The third form is a basic action, using the action notation described earlier for documentary Petri nets. Interested readers are referred to (Lee 1992) for a full description of the PCG syntax.

### 3.2.2 Example of Top-Down Design.

To illustrate the top-down design, we use the example used in the bottom-up approach (Section 3.1.4). In this case we first define the PCG including the three possible payment and transport modes. Figure 4 shows the resulting PCG. Using Case/Open-edi, such a PCG can be instantiated. In this example we instantiated the PCG with the options `@paymode = postpaid_sale` and `@transportmode = paid_by_seller`. Figure 5 shows the integrated procedure generated by Case/Open-edi. The last step to be taken in the top-down design is the generation of the individual role descriptions from the integrated procedure. This can be done automatically using Case/Open-edi. The result for this example is given in Figure 6.

![Figure 4. A PCG for the top-down design of a trade procedure.](image-url)
Figure 5. The DPN for the integrated post paid, carrier paid by seller procedure.

Figure 6. The individual role descriptions as generated by Case/Open-edi.
4. Conclusions and Further Directions.

Most previous research in the area of EDI and EDI modelling has been focusing on partnerships that can be characterized as 'closed trading partnerships'. In these kind of partnerships the costs of rethinking the way of doing business will pay off in the longer term benefits from the new way of doing business. However, in shorter term relationships or 'open' relationships, these costs cannot be compensated due to the less frequent interactions in the relationship. Therefore, standardized trade procedures are necessary to lower these transaction costs sufficiently to enable open electronic trading.

Two steps should be taken to build these trade procedures. First a common language should be defined and second user groups should then specify standard trade procedures using this language. If these trade procedures are specified in a computer interpretable language such as the DPN formalism, a third step can be taken. Companies might download such procedures from the repository. Then, the information system of these companies can interpret the procedure and use this as a decision support tool, or even more effectively, let the computer system conduct most of the trading automatically. This will provide them with strong guidance during the execution of the trade procedure.

In this paper we have proposed two approaches to the automated design of Open-edi trade procedures. The assumption has been that, once developed, these trade procedures would be made available to parties in a publicly accessible electronic repository. The paper shows that the specification of trade procedures using Documentary Petri Nets and Procedural Constraint Grammars offer possibilities for the development of productivity enhancing CASE tools. Complex procedures can then be assembled from predefined components (bottom-up), or already standardized procedures may be customized to satisfy the needs of a specific group of business users (top-down).

Future research directions include the investigation of criteria to improve the efficiency and security of trade procedures which can be included as audit daemons in Case/Open-edi for automated validation. Furthermore, since these trade procedures are specified in a computer interpretable formalism, new ways of doing business might occur. A Documentary Petri Net specification might be interpreted by the computer system of an organization, thus supplying a strong guide line or decision support tool. Finally, it can be imagined that certain functions of organizations might be fully automated as computational agents based on the Documentary Petri Net specification.
References.


NEWSPAPER BUSINESS REDESIGN: ORGANIZATION MODELS AND ECONOMIC CONSEQUENCES

Annikka Tuukkanen
Institute of Industrial Automation
Helsinki University of Technology
Otakaari 4, 02150 Espoo
Finland
Fax: 358-0-4513665, e-mail: annikka.tuukkanen@hut.fi

Abstract

The object of this study was to research the relations between newspaper organizations and their performance and their effect on profitability measured in terms of economic indicators using modelling technology. The early results of the study proves that by reorganizing the processes better productivity and profitability can be achieved. The empirical research and use of modelling in practice gives new information and support when redesigning the business process.

1. Introduction

Newspapers in the Scandinavian countries have long traditions with stable business environment and financial standing. Until the end of 1980's the market conditions were favourable and the business was profitable enough. Then, because of the general economic recession, the advertising volumes drastically decreased, which caused economically hard times for most of the newspapers. This again forced the newspapers to search for new models of organizing the business. In order to support the newspapers in business process re-engineering and redesign the collaboration organization of Scandinavian newspapers has financed two projects both to be carried out at Helsinki University of Technology (HUT).

This paper describes the methods used and results obtained in two projects where newspaper business reengineering was studied. The objectives of these projects were to research the relations between organizations and their performance, in more specific, alternative workflows and their effect on profitability and performance measured in terms of economic indicators using modelling techniques. The emphasis is on the production processes of the newspaper, but the fundamentals of the administration and marketing functions are included in the organization models.
In the research projects described we concentrated mainly on the "meso"- and to some extent on the "macro"- level business process redesign, not much on the "micro"-level task oriented problems /Verbraeck et al. 1994/.

2. Description of the problem

The newspaper production processes have changed very much during the recent years. The printing process is highly automated together with the postpress operations. The prepress operations, i.e. production of articles, advertisements and pages, are even more based on integrated computer systems. These new technologies make it possible to organize the work and workflows in different ways, e.g. by either distributing or centralising the work in new creative ways.

The newspaper production is very information intensive and the framework for managing complex workflow processes can be done in relation with newspaper prepress processes. In comparison with other fields there are two main characteristics differentiate newspaper work processes from many others: a much shorter workflow cycle and the changing structure of the product produced /Alasuvanto et al. 1993/. In other industries the life-cycle of a production process is longer, and the drastically last minute changes quite exceptional.

Business process re-engineering has given us concepts to organize business, administrative and production processes. However, the methodologies to analyse the economic consequences of the alternatives have been less systematically studied. This is a central flaw because the economic performance is central for redesigning the organizations because much of the motivation for all the work comes evidently from economic advantages to be gained.

3. Research objectives

The first research project on flexible newspaper production had as the main objective to find out the most productive and profitable production methods in idealised conditions, e.g. forgetting encumbrances of traditions or other attitudes as limitations. This project was finished by the end of 1993 and the results proved to give fruitful bases to continue /Björkell 1994/.

The second project tries to continue from where the first one ended trying to enlarge the scope to include administrative and marketing aspects but still with strong emphasis on prepress workflows. The main objective is to study the relationship between the processes, i.e. organization and structure of the production, and the economic indicators, e.g. productivity and profitability within the newspaper framework /Tuukkanen et al. 1994/.

One of the main goals is to develop tools that can be used in individual newspapers for measuring, evaluating and redesigning the business processes and production.
4. Research Methods

The method of designing an ideal, flexible newspaper production system was based on several approaches. The newspaper business and production processes were analysed using concepts and principles of controllability analysis, lean management and activity based management. The material for modelling the ideal production was collected by inquiries and case studies. The best practices were searched for by benchmarking techniques.

In the second phase of research project the experiences of the previous study have been used. Also the previous studies on dynamic modelling and simulation of the newspaper and printing business have been used as a framework in designing new approaches /Tuukkanen et al.1993, 1994/.

The common known concepts of business process reengineering and redesign /Hammer et al. 1993/ have been applied to the extent it has been possible or as expressed in a slogan by Hammer: "Re-engineering is the search for new models of organizing work".

During the summer the models developed so far have been tested in six newspapers in Finland, Sweden, Norway and Denmark. In these case studies the method of modelling the newspaper has been tested. At this phase we still believe that it is possible to describe the "meso"-level organization and production system of the complex newspaper adequately to make conclusions as for the economic consequences of dynamically changing conditions.

5. Case studies and benchmarking

The pilot newspapers were selected by the criterion of being interested in or in the process of redesign or re-engineering of their business processes. The pilot studies have been carried out by predesigned models adapted to each pilot newspaper describing the organizations and workflows before and after the business process redesign. The data of economic indicators and other parameters was collected by structured inquiries. The data is processed with proper calculating tools and analysed by experts and previous experiences. The benchmarking has been used to find the best practices within the newspapers but also with corresponding results of other industries. A research project of value-adding factory, a part of a Finnish national productivity program, currently being carried out at HUT gives proper data for these kinds of comparisons /Eloranta 1993/.

6. Conclusion

The results of the previous projects and the results of the field study prove that by reengineering and redesign of the newspaper business processes better productivity and profitability can be achieved. Several reasons stand behind this, such as better utilization of resources, better
management of complex workflows, balancing uneven production load and using lean management principles not only in production but also in administrative functions. The empirical research and use of modelling in practise gives new information and support to newspaper management when redesigning the business process.

References


Acknowledgement

This research was carried out at Helsinki University of Technology by the Media Technology Group. It was conducted by professor Reijo Sulonen and professor Nils Enlund.
DYNAMIC MODELLING OF A MANUFACTURING FIRM’S INFORMATION SYSTEM WITH OLYMPIOS-DESIGN

Ramzi Guetari and Frédéric Piard
LLP/CESALP - Université de Savoie
BP. 806 - 74016 Annecy CEDEX - FRANCE
Tel: (+33) 50.66.60.80 - Fax: (+33) 50.66.60.20.
e-mail: guetari@esia.univ-savoie.fr.piard@esia.univ-savoie.fr

Keywords : Object oriented design, industrial information, algebraic specifications, object structure

Abstract :

This paper presents a model, OLYMPIOS, for designing manufacturing firms information and decision systems (IDS). This model proposes a designing method which respects the model uniformity and ensures the continuity between different stages of the IDS life cycle. This method allows us to pass automatically from one stage to the following, so as to preserve the coherence obtained in the first stage. After the presentation of OLYMPIOS’ concepts, we present the different stages of an IDS life cycle according to our model, i.e.: analysis, specification, design and implementation. Then we present the automatic translation from one stage to another. Finally we illustrate the method with an industrial example.

1. Introduction

An industrial Information and Decision System (IDS) is typically dynamic, not only because of the real time- and resource sharing- constraints at the shop floor level, but also because of the new emergent post-mass production paradigm, according to which manufacturing will have to be more and more reactive and adapt their organisations and strategies to a more and more rapidly changing environment (markets, technologies, politics,...). This paper presents a design method dedicated to manufacturing firms’ IDS. It has been developed to help small and medium enterprises to follow a “CIM” (Computer Integrated Manufacturing) approach. So our work is a contribution to dynamic modelling in the very particular field of manufacturing firms’ IDS. We refuse to present OLYMPIOS as a general model for dynamic information systems, for two main reasons. Firstly, OLYMPIOS targets a particular type of organisations: small and medium manufacturing firms. It entails several features: “reasonable” size organisation (i.e. where the existing world can be analysed by interviewing people), and which can be modelled by a net of exchange relationships based on the customer-supplier paradigm. Secondly, the “dynamic” side of such systems can be represented by specific “temporal properties” (Piard and Haurat 1994): periods, deadlines, strict and relative constraints, determinist and non-determinist choices,... In the case of other dynamic systems (robotics, concurrent or real-time programming,...) the pertinent temporal properties are different. However, the formal description techniques used in OLYMPIOS can be extended to other applications. But in fact, what is really important, and what we want to focus on, is the philosophy of our works, in two words: coherence and continuity. The coherence of a model, which is the main factor of quality, can be obtained by the means of mathematical specifications (logic or algebra). But to match the specification to reality, and to
preserve coherence until realisation and maintenance, one must manage the system life cycle in a homogeneous and continuous approach. For example, OLYMPIOS uses object oriented design techniques to cope with the complexity of industrial information. Then we have tried to take the object concept into account, as early as possible in the life cycle. We had also to cope with time (the "dynamic" side) so we propose to introduce temporal features in objects. And we propose a continuous life cycle, with partly automatic translations from analysis to design, and soon to code generation. One can find in real time software engineering (TOCCATA (Martin and al 1992)) and in robotics (ORCCAD (Simon and al 1993)) advanced object oriented design methods for dynamic systems. But we have found no method taking into account at the same time:

- coherence proofs (simulation is often preferred to formal techniques);
- the homogeneity of the model ("time" and "data/treatments" are always separate at the design level);
- the continuity of the system life cycle (for example, TOCCATA starts from SA/RT specifications (Hatley and Pirbhai 1991) to target an object oriented design level).

2. The OLYMPIOS model concepts

The information processed in an enterprise, which we call industrial information, is a complex datum. An information and decision system (IDS) must take this complexity into account. We propose to represent industrial information in four main facets:

- data, describing the different entities handled by the IDS and the actions that they can perform or be subjected to;
- temporal properties of the different kinds of processes (including traceability of information);
- organisation, considered through information flows;
- economic facet, which describes the means of performance appraisal in relation to enterprise environment and objectives.

The OLYMPIOS model is a methodology to construct the IDS of a class of operators in a particular enterprise taking into account the previous facets. It covers the different stages of such a system's elaboration (Figure 1), and proposes original solutions for analysis, specification, design and realisation. For analysis and specification, OLYMPIOS essentially uses an algebraic approach for the four facets of industrial information, so as to obtain a coherent (i.e. sufficiently complete and consistent) specification. The design stage enables us to conceive the information system from specification and by analysing the "existing" system of the enterprise and its objectives. The result of this stage is a representation of the IDS with structured entities.

The IDS designed with OLYMPIOS (Beauchêne and al 1993) is able to describe the activities' execution with relation to the objectives assigned to an operator, taking into account resource availability. These characteristics are modelled by:

- Consumer-Supplier Information Systems (CSIS). A CSIS represents an atom of IDS organisation. It generalises the "customer-supplier" exchange relationship to all the participants in the enterprise (workforce, machines, software). Every CSIS is bound to a user objective, and emits a "satisfaction level". Objectives are the projection of enterprise finalities on each participant's space. The satisfaction level results from the comparison between the objective and its realisation, and is appraised thanks to performance indicators.
- an Objectives Management System (OMS), which creates a graph whose nodes are expressed objectives. Every node is bound to a CSIS. We represent objectives by triples (resource name, indicator and expected domain, date of appraisal);
- a Resources Management System (RMS), which represents the activities management. It has to manage resources and products, which are only seen in OLYMPIOS under an
informational aspect (whether they are informational or physical);
- an Activation System (AS), which elaborates an action plan, taking into account the application and temporal constraints. This elaboration is split into two steps: a first plan is generated from the graph of objectives, without paying attention to resources. The final plan then integrates resource sharing by introducing synchronisation.

![Diagram of the OLYMPIOS Model](image)

**Figure 1.** The life cycle management in the OLYMPIOS Model

To illustrate the life cycle management of an IDS according to OLYMPIOS, we will apply the method to a simple resource-entity: a stock of products by an automotive subcontractor using sales management by E.D.I. This example is drawn from our laboratory’s demonstrating case.

The situation is the following: the subcontractor has a contract for each product to deliver to a particular customer. The customer sends a delivery instruction daily (or “delins”, according to the European standard “GALIA” on electronic data exchanges). The delins specifies the quantities of the product to deliver and dates of deliveries. Quantities are fixed for the following week and estimated for the following 3 months. These quantities are updated everyday by a new delins. The main resources-entities in this particular model are DELINS, STOCKS, DELIVERIES and EXPEDITIONS. For simplicity’s sake we consider that a stock is dedicated to a kind of product, has an infinite capacity and has no particular management rule (stock “in-bulk”).
3. The Analysis Stage

In the analysis stage, we collect the relevant information for the data, the temporal, the organisational and the economic facets. The result of the data facet analysis consists of the description of the data handled in the system to design and the set of operations that can be realised upon each datum. This static description can be translated into a finite state automaton in which every node represents a state of the datum in question and every transition an operation which produces a new state (Nkongo 1990). This translation consists of identifying, from the observation of the real world, the significant states of entities, and the actions changing their state.

The following figure (Figure 2) represents the automaton obtained from the analysis of a resource Stock (infinite capacity). It has two states (empty and intermediate). One can add a part in the stock from every state, but one cannot remove a part from an empty stock. The “Create” transition allows the creation of a particular stock dedicated to a given product. The stock is empty when created.

![Figure 2. Automaton of resource Stock](image)

The analysis of the organisational aspects of the manufacturing firm, results in a set of interactions between the different agents of the enterprise in the form of exchange relationships. By interviewing each of these agents we enumerate on the one hand, the exchange relationships in which he is consumer, and on the other hand, the relationships in which he is supplier and performs certain functions. For each relationship, we identify the objectives and the necessary resources. Starting from this information, we can establish a knowledge base about the different ways to decompose objectives, and a knowledge base for the needed resources for each basic operation. These knowledge bases will help us, in addition to the predefined structure of such an exchange relationship, to define the enterprise’s organisation.

The analysis of the temporal facet provides a dynamic description of the system. It enables us to describe the temporal behaviour of different agents and resources of the system and their interactions. A graphic method to help this part of analysis has been developed (not described in this paper). It proposes a user-friendly way of describing temporal rules.

As far as the economic facet is concerned, we are actually working on the evaluation of the performance of industrial systems.

4. The Specification Stage

The data and organisation facets are specified using ASAT (Algebraic Specification of Abstract Type) (Gutttag and Horning 1978; Liskov 1987; Jacquenet and Lescanne 1986) in order to have efficient and simple proof techniques at our disposal. An ASAT enables the expression of an entity behaviour in a high level formalism. In the OLYMPIOS MODEL ASATs are directly issued from automata (Nkongo 1990), which are the results of the analysis stage.
The specification of the "organisation" facet starts with the analysis of the "existing" system, which results (inter alia) in the identification of participants, their functions and objectives. Specifying organisation consists of formally expressing identified objectives (in the "triple" form), and in constructing their associated CSIS from standard parametrized ASAT of organisation (which have been defined once for all in (Beauchêne 1993)). Simultaneously, one must elaborate the different graphs of objectives.

ASAT does not take into account the temporal facet of an information system. The specification of the industrial information temporal facet uses asynchronous process algebra, directly derived from the Synchronous Calculus of Communicating Systems (SCCS) by R. Milner (Piani and Braesch 1993). We specify four kinds of processes with this language:
1- chronological and event-based clocks, essential to specify synchronisation and to measure temporal intervals;
2- behaviour of data facet entities, which are not completely determined by ASAT axioms;
3- behaviour of CSIS;
4- activation plans, elaborated by the activation system from graphs of objectives and resources to schedule the CSIS.

The economic facet cannot be specified independently of data and organisation. Indeed it is shared between them, and the most important part is included in the organisation facet. Work continues to sharpen the economic view of OLYMPIOS on the information system (with the help of performance indicators, fuzzy logic and a project-based management approach).

4.1. Algebraic Specification of Abstract Types

For a given entity, an ASAT is a triple <Ω,Σ,Α>, where:
- Ω is a set of domains, containing the domain of the entity values, written "TI" (type of interest). They are recursively enumerable sets.
- Σ is a set of operations applicable to the entity. This set can be divided into three subsets:
  - O: set of observers. An observer is an operation whose codomain does not contain TI. Observers allow us to define the abstract data type semantics (Guttag and Horning 1978).
  - C: set of constructors,
  - S: set of simplifiers.
- T = C ∪ S is called the set of transformers. A transformer is an operation generating TI values, or transforming them in other values of TI. If a value generated by an operation σ of T can be expressed by other operations σ1, σ2, σ3..., then σ is a simplifier, else it is a constructor.
- A is a set of equations (axioms and preconditions) concerning the operations of Σ. They define the entities' behaviour and their relations.

In the OLYMPIOS model, ASAT are automatically built from the entities' description by automata.

4.2. The Synchronous Calculus of Communicating Systems

Our need of a formalism with an abstraction high level, coherent with our algebraic approach and allowing formal proofs, directed us to the process calculus. We use, as basic operators, those of SCCS (Milner 1989), and our derived operators are chosen, according to the situation, from SCCS or MEIJE (Boudol 1985). The SCCS is based on a double structure:
- a set Act of atomic actions (i.e. temporally indivisible and lasting "one instant"). Milner defines the simultaneous composition of actions "\cdot".
- a set of agents, which can be looked upon as the behaviours of processes.

Milner defines the following "basic" operators in the set of agents:
- The action operator prefixes the behaviour of an agent E with an action a.
- Sum: it expresses the non determinist choice between a set of agent behaviours.
- Product: or synchronous parallel composition.
- Restriction: inhibits all the actions not belonging to the set A.
- Recursive operator: It allows the designation of a family of agents, mutually and recursively
defined by a system of equations.
Morphism : It permits to rename actions thanks to an endomorphism \( \Phi \).
We also use derived operators which are expressible with basic operators.

5. The Design Stage

OLYMPIOS uses, in the design stage, an object oriented technique. We have developed a design method dedicated to industrial information systems in the framework of OLYMPIOS (Guètari and Piard 1994). This method enables us, through pre-defined elementary organisations (CSIS, Objectives, ...), to accurately translate the specification, and to suit the domain of industrial information. The result of the design stage is an organisation of structured entities, which are independent of the implementation language : OLSEN (OLympios Structured ENtities). Our choice is justified by two reasons :
- its expressive power, which is close to human perception of the environment, and allows the advantages of reusability, modularity, etc.
- the continuity between ASAT and Objects as mentioned in (Meyer 1990) “Object design is the construction of software, represented as a collection of implementation of abstract data types”. In another way the translation from ASAT to Objects is automatic in the OLYMPIOS model.

The syntax and the internal organisation of OLSEN (close to the "class" model) have also been predefined so as to take all the characteristics of the specification stage into account. Especially, the behaviour of entities in their environment (definition of the events that trigger message sending and of the target-entities, partial determinism of behaviours, ...), rarely taken into account at the design level, are an originality of OLSEN. Moreover we have automated with algorithms the transformation of the specification layer (ASAT and SCCS) into an OLSEN organisation. These algorithms have been implemented in an OLSEN software generator.

An OLSEN (Guètari and Piard 1994) is an extension of the Class/Instance model. It is composed of a part inherited from the standard class model and a part called “scenario”, indicates the interactions which it has with its environment. The difference between an OLSEN and a classical object is the scenario which describes the temporal behaviour generally absent in the standard class model. The OLSEN model is a “design object” where instances are mono-task and can be active at the same time.

The class part is composed by a structure (a set of attributes which can themselves be OLSEN) and a method dictionary which indicates operations that can be applied to its structure. An OLSEN is a model which breeds individual entities (its instances) by a mechanism of instantiation. OLSEN are organised by aggregation and inheritance relationships. Inheritance is applied to structure and behaviour as for classes (Wegner and Zdonic 1988 ; Masini and al. 1989). OLSEN communicate using messages. A message is an event carrying a semantic. This semantic concerns the service that the receiving OLSEN must do to the emitting OLSEN (a message must normally trigger a service by the receiving OLSEN).

An OLSEN is a model which breeds individual entities (its instances) by a mechanism of instantiation. OLSEN are organised by aggregation and inheritance relationships. Inheritance is applied to structure and behaviour as for classes (Wegner and Zdonic 1988 ; Masini and al. 1989). OLSEN communicate using messages. A message is an event carrying a semantic. This semantic concerns the service that the receiving OLSEN must do to the emitting OLSEN (a message must normally trigger a service by the receiving OLSEN).

The scenario of a given OLSEN \( O \) is composed of two parts : a declarative part and a dynamic part.
- The declarative part enumerates significant events that occur during an instance life of the OLSEN \( O \). It enumerates the set of OLSEN that can do a service to the instance \( O \) (its suppliers OLSEN). We can distinguish three kinds of events : environment events where occurrences are diffused to all system entities and are used to manage the global organisation of the system, communication events where occurrences are bound to the message passing and allow the synchronisation between OLSEN, and conditional events where occurrences are bound to some conditions about the state of the system data.
- The dynamic part describes the using of processes if an event occurs, and especially communication and synchronisation. This part joins to each event \( \varepsilon \) the description of the behaviour of the OLSEN if an occurrence of \( \varepsilon \) happens. This description of the behaviour is the script joined to the event \( \varepsilon \) in the OLSEN \( O \). The script shows the progress of the method linked to this event. The dynamic part of an instance \( O \) of an OLSEN is composed
Dynamic Modelling of a Manufacturing Firm's Information System with \textsc{OLYMPIOS-Design}

by the set of scripts associated to events introduced in the declarative part. The script is a sequence of predefined instructions, message sending and environment events generation. Parallel message sending to different \textsc{OLSEN} is possible.

6. The translation from analysis (automata) to Specification (ASAT)

This stage consists of describing some data types using automata. We must first insist on the fact that every entity cannot be described by an automaton, but only if it has successive states and if it is concerned by actions passing from one state to another. Here we don't use here the automata as a specification tool but as a tool allowing us to shape the evolution of some kind of data type over a set of states.

In this kind of automata, each transition represents a changing of states and each node is one state of the entity. The automata can have many transitions corresponding to the same operation, however, each state is the one and only possible. A particular state called "starting state" must always exist. It is the extremity of the transition corresponding to the operation which creates the type of interest (TI) values.

The entities described by automata are distinguishable by the successive states that they can have. The order in which different states are occupied is well defined. The graph of state changing is oriented and has a starting state from which we can observe the evolution of the entity. This graph allows us to distinguish the constructor operations using a single method : the corresponding transitions have, as extremities, nodes which are reached from the starting state by only one path of the graph. The construction of axioms is done in two parts : the construction of the left parts of axioms and the construction of the right parts.

The construction of left parts of axioms :

The construction of axioms' left parts consists of building the following sets, where $y^*$ is a list of variables not belonging to TI and $x$ is a variable belonging to TI :

- $C_T = \{c(y^*), c \in C\}$ (constructor applied to $y^*$)
- $O_T = \{o(x, y^*), o \in O, x \in C_T\}$ (observer applied to a constructor)
- $S_T = \{s(x, y^*), s \in S, x \in C_T\}$ (simplifier applied to a constructor)

$O_T$ and $S_T$ contain the left parts of specification axioms. Axioms which define the semantic of the abstract data type have their left parts into $O_T$ set and axioms which show the simplification of terms of $T(Q, l)$ have their left parts in the $S_T$ set.

The construction of right parts of axioms :

The entity's finite state automaton (cf. § 3.) provides :

1- $\Omega = \{T_1, STATES\}, STATES = \{E_1, E_2, E_3, \ldots\}$ (the automaton's states give the domain "STATES")

2- $\Sigma = \{\text{state}, \sigma_1, \sigma_2, \sigma_3, \ldots, \sigma_n\} = 0 + C + S, \ T = S + C = \{\sigma_1, \sigma_2, \sigma_3, \ldots, \sigma_n\}$ is the set of operations which create or transform the values of TI (represented in the automaton by transitions), $O = \{\text{state}\}$ contains a single observer.

3- Left parts of axioms by building of $AC, AO, AT$ from $O, C$ et $T$.

4- Right parts ($y$) of axioms in the form $state(c(x^*)) = y$, where $c \in C$, and $y$ is the name of the extremity of the path from the starting state to $c(x^*)$. If there are several possible paths, the $y$ term will be expressed in the form of if...then...else ...

5- Right parts ($y$) of axioms in the form $s(c(x^*)) = y$, where $s \in S$ is a simplifier and $y$ corresponds to the canonical form of the node represented by $c(x^*)$, i.e. the shortest path's expression between the starting state and the state represented by $c(x^*)$. In other terms, these axioms are represented in the automata by simple circular paths. If there are many of these paths then the $y$ term will be expressed in the form of if...then...else ...

6- Preconditions related to the state of arguments (membership of TI) of each operation, which are expressed by the restrictions of the domain of this operation before its execution. These restrictions are issued from the state which is the origin of the arc representing the operation.
The application of this method on the example gives the ASAT shown in the figure 3. For the temporal facet, our graphic method produces the following SCCS formula to describe the entity’s behaviour:

\[
\text{BEHAVIOUR (STOCK)} = \text{fix} (X = \delta (\text{preSTOCK} : \delta \text{postSTOCK} : X + \text{preREMOVE} : \delta \text{postREMOVE} : X + 1 : X)) \text{fix} (X = \text{EMPTY} : X + 1 : X)
\]

Here we use the following SCCS operators: unspecified delay $\delta$, recursion fix, non determinist choice $+$, sequential composition of actions "::" and synchronous concurrent composition $\times$ (for more details see (Milner 1989)).

Significant events (or SCCS “actions”) are the beginning and the end of stocking, removing products (identified by the “pre” and “post” prefix), and the signal of empty stock. The duration of stocking or removing a product is not calculable: they are asynchronous transformational processes, so we introduce the delay operator between pre- and post- actions.

The fix operator expresses that the behaviour has been recursively defined. The first $\delta$ operator indicates that the order to stock or remove comes from “elsewhere” (another entity).

We try now to provide automatic proofs on the SCCS formulae produced by our software prototype of SCCS graphic generation.

\[
\begin{align*}
\text{ASAT (STOCK)}; \\
\text{DOMAINS} \{\text{STOCK, STATE = \{IS, ES\}, PRODUCT, INTEGER}\} \\
\text{OPERATIONS} \{ \\
\quad \text{Create} : \text{PRODUCT} \to \text{STOCK}; \\
\quad \text{Add} : \text{STOCK} \times \text{INTEGER} \to \text{STOCK}; \\
\quad \text{Remove} : \text{STOCK} \times \text{INTEGER} \to \text{STOCK}; \\
\quad \text{State} : \text{STOCK} \to \text{STATE}; \\
\quad \text{Products} : \text{STOCK} \to \text{PRODUCT}; \\
\quad \text{Level} : \text{STOCK} \to \text{INTEGER}; \\
\} \\
\text{PRECONDITIONS} \{ \\
\quad s : \text{STOCK}; \\
\quad q : \text{INTEGER}; \\
\quad \text{Remove} (s, q) = \text{state} (s) = \text{I AND Level} (s) >= q; \\
\} \\
\text{AXIOMS} \{ \\
\quad \text{prod} : \text{PRODUCT}; \\
\quad s : \text{STOCK}; \\
\quad q : \text{INTEGER}; \\
\quad \text{state} (\text{Create} (\text{prod})) = \text{ES}; \\
\quad \text{state} (\text{Add} (s, q)) = \text{IS}; \\
\quad \text{state} (\text{Remove}(s,q)) = \\
\quad \quad \text{IF} (\text{Level}(s) - q = 0 \text{ THEN (E)} \\
\quad \quad \quad \text{ELSE (I)} \\
\quad \quad \quad \quad \text{products} (\text{Create} (\text{prod})) = \text{prod}; \\
\quad \quad \quad \quad \text{products} (\text{Add} (s, q)) = \text{products} (s); \\
\quad \quad \quad \quad \text{products} (\text{Remove} (s, q)) = \text{products} (s); \\
\quad \quad \quad \quad \text{Level} (\text{Create} (\text{prod})) = 0; \\
\quad \quad \quad \quad \text{Level} (\text{Add} (s, q)) = \text{Level} (s) + q; \\
\quad \quad \quad \quad \text{Level} (\text{Remove} (s, q)) = \text{Level} (s) - q; \\
\} \\
\end{align*}
\]

Figure 3 - The ASAT generated from the Stock automaton
This formula means that a delins is read everyday at the strike of the hour $\&l_1$ (for example 7 o'clock), then is selected when it mentions a product to deliver in the afternoon, then the quantity to deliver is calculated and included in a grouping of products to dispatch. The receipt of a new delins of the same kind inhibits the current one (I is the "end" state).

$$\text{BEHAVIOUR (DELINS)} = \delta \text{preREAD} : \delta \text{postREAD} : X; \delta \text{preSELECT} : \delta \text{postSELECT} : (\text{NONDATE} = X + \text{DATE} = \delta \text{preCALCULATE} : \delta \text{postCALCULATE} : \delta \text{preINCLUDE} : \delta \text{postINCLUDE} : X) + s_1, \text{NEWDELINS} \Rightarrow 1$$

It means that the quantity of product to deliver is calculated, then adjusted in function of other customers' requirements, and finally products are removed from the stock and included in the grouping of products to deliver.

7. The translation from the Specification stage to the Design stage

The transition from the specification stage (ASAT and SCCS) to the design stage is done automatically in two steps. The first step consists of taking ASAT one by one and translating each one into standard class. The second step is a global one and permits the organisation of the communication between the obtained classes. The benefit of this automation is to preserve the coherence obtained in the specification stage.

7.1. The standard class generation

The class attributes and methods are generated from the ASAT operations. This is done using the following rules (We note an operation : $\sigma : \Omega_1 \rightarrow \Omega_2$. $\Omega_1$ is the set of domains and $\Omega_2$ is the set of codomains. "TT" is the data type that we specify):

We distinguish three kind of operations:
- Case 1 : $\sigma : \Omega_1 \rightarrow \Omega_2 / \text{TT} \notin \Omega_1$ and $\Omega_2 = \{\text{TT}\}$. This kind of operation corresponds to a particular constructor. For each one of these constructors, we generate a method "New"
with parameters of type $\Omega_1$.

- **Case 2**: $\sigma : \Omega_1 \rightarrow \Omega_2 / \Omega_1 = \{TI\}$ and $\Omega_2 = \{\omega \neq TI\}$. This kind of operation corresponds to observers. The class structure is obtained from these observers. For each observer we generate an attribute of type $\Omega_2$ and a method to read it.

- **Case 3**: $\sigma : \Omega_1 \rightarrow \Omega_2 / TI \in \Omega_1$ and $TI \in \Omega_2$. This case corresponds to a general one. For each operation of this kind we generate a method with in parameters of type $\omega \in \Omega_1 / \omega \neq TI$ and out of parameters of type $\omega \in \Omega_2 / \omega \neq TI$.

This algorithm applied on the ASAT Stock gives the OLSEN shown in the figure 5. This automatic generation follows a syntactic analysis of the ASAT, which detects possible errors (lacking parameters, non declared types,…).

```c
enum STATE {IS, ES} ;

Class STOCK
  Global :
    Public :
      New (PRODUCT) ;
  Local :
    Private :
      att10_State : STATE ;
      att11_Products : PRODUCT ;
      att12_Level : INTEGER ;
    Public :
      Add (INTEGER) ;
      Remove (INTEGER) ;
      State () : STATE ;
      Products () : PRODUCT ;
      Level () : INTEGER ;
  Assert :
    s : STOCK ;
    q : INTEGER ;
    s.Remove (q) if s.State () == IS AND s.Level () >= q ;

End
```

**Figure 5.** Result of the passing from specification to design: “class” part

### 7.2. The scenario generation

The scenario of an OLSEN is issued from SCCS formula. An SCCS formula contains several deterministic parts. Each part provides one script in the OLSEN’s scenario. The scenario generation is realised in three steps: the first two provide the declarative part of a scenario, the third one provides the dynamic part. For each OLSEN, we identify the deterministic parts of the corresponding BEHAVIOUR (separated by a “sum” operator). For each part, we execute the following three steps:

- **Event detection.** This step permits the detection and declaration of the different kinds of events. The type of each event is deduced from the SCCS syntax. A communicational event appears in at least two BEHAVIOURs, once preceded by the delay operator $\delta$, and once without this operator. An environmental event is identified by the existence of a clock emitting this event. An event is conditional if its complementary event appears at least once in a BEHAVIOUR. When all events are declared, we proceed to the unification of the communicational events. This unification is based on the observational equivalence (Austry and Boudol 1984) and consists of giving the same name to two synchronously successive events in a SCCS formula.
• **Identification of the set of suppliers.** For each communicational event, we define its receiving OLSNes which BEHAVIOURs contain this event, preceded by the delay operator 8. Any OLSN responding to this event, by applying one of its methods, must be added to the suppliers list of the treated OLSN.

• **Script generation.** A script is generated for each determinist part. Each event described in the formula is replaced by one or several simultaneous dispatches of messages. The recipients of these messages are the suppliers defined in step 2.

The application of this algorithm gives the following results for the OLSNes "DELivery" and "STOCK". We have introduced an external entity "SUPERVISOR", whose role is to receive the signal "empty" and to supply the stock with products.

```plaintext
Class STOCK
...
SCENARIO :
  communication event (Stock, before (PRE_STOCK), after (POST_STOCK)) ;
  communication event (Remove, before (PRE_REMOVE), after (POST_REMOVE)) ;
  conditional event (att12_LEVEL = 0, EMPTY) ;

select (Spv : SUPERVISOR)

PRE_STOCK (in Q : INTEGER) : Add Q to att12_Level ;
PRE_REMOVE (in Q : INTEGER) : Sub Q from att12_Level ;
EMPTY : send Spv Supply () ;
END

Figure 6. The Scenario of the OLSN Stock

Class DELINS
...
SCENARIO :
  communication event (Include, PRE_INCLUDE) ;
  PRE_INCLUDE : // Script
...
END

Class EXPEDITION
...
SCENARIO :
  communication event (Include, PRE_INCLUDE) ;
  PRE_INCLUDE : // Script
...
END

Figure 7. Scenarios of OLSNes "DELINS" and "EXPEDITION"

Note : according to the BEHAVIOUR of the DELIVERY entity, the end of the "remove" operation is followed by a "preINCLUDE" event. This event also appears in the BEHAVIOURs of DELINS and EXPEDITION, and generates a parallel message sending (sign @).
Class DELIVERY

...  
SCENARIO:
communication event (Calculate, PRE_CALCULATE)

... select {Stck : STOCK, Dlns : DELINS, Expd : EXPEDITION, ...}

PRE_CALCULATE : send self Calculate () ;
send self Adjust () ;
send Stck Remove (Q) ;

POST_REMOVE[Stck] : send Dlns Include @ send Expd Include () ;

END

Figure 8. Result of the passing to design: "scenario" part of the OLSEN DELIVERY

8. The code generation

As the "organisation" facet of the IDS as not yet been treated in the design level, we are not yet able to automatically generate databases.

At the present time, we are able to generate from the OLSEN organisations Object-programming source code or ADA source code. Objects depend on the target programming language structure and philosophy. The result of the translation from OLSEN organisation to classes organisation is different from the translation from OLSEN to actors organisation or to ADA source code.

8.1. Class Generation.

The translation from an OLSEN to a class is realised in two steps. In the first step we copy the class part specification by changing the syntax into the corresponding language syntax. The second step consists of the generation of methods allowing the communication between objects. These methods are issued from the OLSEN scenario and contains a sequence of message sending.

At the present time, our target programming language is C++ (Figure 9) because of its widely spreading in industry. The attributes are private to the class. Methods which are linked to a communication event are public and allow the generation of applications programs. Methods which are not linked to a communication event must be triggered only inside a public method, and then must be private to prevent a direct access. The inheritance relationship is recovered as it is in OLSEN.

As the target language does not dispose of concurrent programming primitives, we must rebuild them with the available primitives (this is an axis of our future work). The OLSEN STOCK results in the following C++ class:

8.2. ADA source code Generation

Each OLSEN class part is translated into an ADA package. Each scenario is transformed into an ADA task. The package contains the OLSEN structure and methods which are not linked to a communication event. Each of other methods is replaced by an entry of the generated task.
enum STATE {IS, ES};
class STOCK {
protected :
    STATE att10_State;
    PRODUCT att11_Products;
    int att12_Level;
public :
    STOCK ();
    void Add (const int);
    void Remove (const int);
    PRODUCT Products () {return att11_Products;}
    int Level () {return att12_Level;}
};

STOCK::STOCK () {
    // Develop Constructor body
}

void STOCK::Add (const int Q) {
    att12_Level += Q;
}

void STOCK::Remove (const int Q) {
    assert (Q <= att12_Level);
    att12_Level -= Q;
}

Figure 9. The result of the C++ code generation

package P_STOCK is
    type STATE is (IS, ES);
type STOCK is private;
function Create return STOCK;
procedure Add (s : in out STOCK, Q : in INTEGER);
procedure Remove (s : in out STOCK, Q : in INTEGER);
function Products return PRODUCT;
function Level return INTEGER;
EXCEPT Level : exception;
private
    type STOCK is record
        att10_State : STATE;
        att11_Products : PRODUCT;
        att12_Level : INTEGER;
    end record;
end P_STOCK;

task T_STOCK is
    entry Add (s : in out STOCK, Q : in INTEGER);
    entry Remove (s : in out STOCK, Q : in INTEGER);
end T_STOCK;

Figure 10. The result of ADA source code generation (specification part).

We don’t present the package body of P_STOCK but we give the source code of the generated task body of T_STOCK. The exception EXCEPT_Level is raised into the procedure Remove of the package P_STOCK.
task body T_STOCK is
begin
  loop
    select
      accept Add (s : in out STOCK, Q : in INTEGER) do
        P_STOCK.Add (s, Q);
      end Add;
      or
      accept Remove s : in out STOCK, Q : in INTEGER) do
        P_STOCK.Remove (s, Q);
      end Remove;
    end select;
  end loop;
end T_STOCK;

Figure 11. The source code of the task body T_STOCK

The instruction select ... or ... in the task body T_STOCK shows that the Stock behaviour is non determinist.

8.3. Actor Generation

The inheritance relationship is generally absent in the actor model. It exists only in some languages designated by "hybrid languages" as the programming language HYBRID (Nierstrasz 1987). So, inheritance is replaced by the delegation. In the delegation relationship, an actor knows another actor called its proxy to which it delegates messages that it does not understand (Masini and al 1989). The relation OLSEN O_sub inherits from an OLSEN O_super becomes O_super proxy of O_sub.

The actor structure is obtained by copying the OLSEN structure and the actor script is issued from the OLSEN scenario.

9. Conclusion

OLYMPIOS can be summarised in several striking points:
- an object oriented approach to cope with industrial information complexity;
- a project-based approach of management and organisation (this is not the subject of this paper);
- the use of algebraic specification techniques to dispose of formal proofs;
- a homogeneous design model called OLSEN, which extends the standard class model with scenario;
- a continuous life cycle, with automatic or partly automatic translations from design to realisation.

The OLYMPIOS model provides the means to analyse and specify coherently an industrial information and decision system. It allows to design the specified IDS by preserving the coherence obtained in the specification stage by using algebraic techniques. The continuity and uniformity claimed by the Olymios model is the result of two factors:
- the use of algebraic tools to specify all the components of an IDS like the data facet, the organisation facet or the temporal facet,
- the use of ASAT to specify data and Objects to design them.

This care of continuity and uniformity has lead us to develop algorithms (and parts of a future CASE-Tool) to automatically generate a coherent object organisation from the analysis. Our objective is to generate a maximum of code for applications.

We insist on the fact that OLYMPIOS is no "miraculous" method: we have solved a lot of problems by shrinking the application domain to something we know: small and medium manufacturing firms.
Design objects (called OLSEN) are independent of the target language. This kind of object takes all the industrial information features into account, especially temporal aspects. OLSEN are automatically deduced from ASAT and from the SCCS behaviour description. All that it aims at is preserving the coherence obtained in the specification stage.

References


EXPRESSING PRODUCTION CONTROL PRINCIPLES IN THE DEMO COMMUNICATION MODEL

Nardo B.J. van der Rijst and Jan L.G. Dietz
Department of Information Systems
Delft University of Technology
P.O. Box 356, NL-2600 AJ Delft, The Netherlands
E-mail: {n.vanderrijst|dietz}@is.twi.tudelft.nl
Fax: +31(0)15786632

Key Words: Production Control, Production Management, System Analysis

Abstract

In the field of production and logistics several control principles exist that suggest typical control structures for typical production situations like e.g. job shop and chain production. In this paper the question is addressed whether and how these principles are expressed in the high abstraction level communication model that results from the analysis of an organization with the DEMO methodology. One practical conclusion is that generic DEMO communication models of typical production situations are very suited to serve as reference models for the development of the specific communication model of a specific organization.

1. Introduction

Due to the recent developments in production management many companies strive to a reconstruction and redesign of the production line and planning structure. In general it is argued that information technology plays an important role in redesigning business processes (Keen 1991). This paper aims at the analysis of business processes concerned with production control. For this reason we propose to analyse this type of business processes in a structured way. By means of reference models, based on logistic principles, we try to reach this goal, applying the models as a guideline for the analysis process using the DEMO analysis method.

The result of the analysis of an organization with DEMO is a dynamic model that is independent of the actual informatical structure, i.e. it does not show the way information is stored (in data bases, archives etc.), transmitted (through flows) or processed (by human information processors or computer applications). Moreover, it also abstracts from the actual organizational structure, i.e. from the way tasks or functions are assigned to people and from the way people are grouped in organizational units. These outstanding features of DEMO enable one to get a comprehensive understanding of the essential characteristics of an organization.

From reflecting upon the possibilities such an organization model offers, the general research question emerges how it compares with approaches to the modelling of organizations outside the field of informatics. In this paper that question is addressed with regard to the modelling approaches that are common in the field of logistics. The general question thereby is refined into two specific questions.

The first one is whether and how characteristic differences between typical production situations (i.e. the job shop and the flow shop) are expressed in a DEMO model, and also whether and how
the relationship with the corresponding material flows can be represented. This question is addressed in section 3. In this section we will also show a comparison with the GRAI methodology (Doumeingts et al. 1989) used for the analysis and design of production management systems.

The second one is whether and how the important logistic notion of the Customer Order Decoupling Point can be dealt with in a DEMO model, and in what way such a model can contribute to the understanding of that notion. This question is addressed in section 4.

Because many readers may not be familiar yet with DEMO, the basic theoretical roots and the main practical features are explained in section 2. In section 5 we provide some provisional conclusions based on the research that is reported upon in section 3 and 4.

2. Modelling organizations with DEMO

"DEMO" is an acronym for "Dynamic Essential Modelling of Organizations". It is the name of a cross-disciplinary theory about the dynamics of activities in organizations, as well as of an analysis method based on that theory. The disciplines on which it draws are the philosophical branches of semantics and scientific ontology (Bunge 1979), and the social theory grounded in language philosophy (Searle 1969), (Habermas 1981). Next to these it incorporates the discrete dynamic system theory as described in (Dietz 1990).

Because a comprehensive description of DEMO in one paper or book is not yet available, we have to refer the reader to fragments of such a description. A relevant set of fragments is constituted by (Dietz 1990), (Dietz 1992), (Dietz 1994a), (Dietz 1994b) and (Reijswoud and Rijst 1994). In the next two subsections a description is provided, as comprehensive as possible, of the aspects of DEMO that are relevant to the scope of this paper.

2.1 The theory

The major motivation behind the development of DEMO was the strongly felt need for information systems analysts to have a theory about organizations (and, by way of analogy, about discrete dynamic systems in general) that is only and purely based on the role of information, taken in a very general sense. Having such a theory at one's disposal would make it possible to understand organizations in a new and original way. The theory would be independent of other theories, in particular the economic theories about organizations, like the Value Chain Theory (Porter 1985), the Agency Theory (Eisenhardt 1989) or the Transaction Cost Theory (Williamson 1985). It would have its own value and just be complementary to those other theories. Because of this property, DEMO is most valuable for providing an original information contribution to Business Process Redesign and Reengineering, next to and in harmony with contributions from other disciplines, e.g. the economically founded discipline of business administration.

The reason for the above mentioned need and for feeling it so strongly is constituted by personal experiences in practice on the one hand, and by a critical evaluation of many current information system methodologies on the other hand. We intentionally refrain from providing a thorough comparison between DEMO and these methodologies for two reasons. First, the attempts in the past to compare information system methodologies have not really led to more insight in the similarities and differences between them, but mainly to quarrels and to the proposal of frameworks for comparison and evaluation that chiefly seem to please their developers. A substantial lot of work in this area is performed in a series of so-called CRIS-conferences (Olle et al. 1982; Olle et al. 1983). Second, with the exception of BDT (Flores and Dunham 1990) and a few recent derivatives, like the one described in (Scherr 1994), there would be no use in making comparisons because the point of view taken by DEMO differs too much from the points of view taken by current methodologies. As for the few comparable approaches, the language-philosophical basis on which they are founded was already criticized elsewhere (Dietz and Widdershoven 1991).
If informatics is understood to be the interdisciplinary field of science that deals with information and communication and their role in the functioning of dynamic systems, especially organizations, then the DEMO theory can most appropriately be called an informational theory.

So far then for the general remarks, we will now start to explain the theoretical pillars of DEMO more specifically. The first point to make is that one should profoundly understand that an organization is a social system. This means that it is composed of social individuals, or subjects as we prefer to call them. These subjects influence each others behaviour through communication. In order to abstract from the particular individuals and to concentrate on the behaviour exposed by them, we introduce the notion of actor. So, an actor is a particular function or activity to be performed, ultimately by a subject, and an organization is a system of communicating actors.

It is crucial to adopt this basic notion of an organization fully. Its rightness can easily be demonstrated by imagining that there would be no human beings. Obviously, there also would be no organizations. It is also crucial to recognize that this notion holds for any degree of automation of the activities in an organization (i.e. of the actors). As we will see in a moment, some activities can easily and without any risk or harm be automated, some however never can. To avoid making mistakes in modelling an organization it is therefore very helpful to forget for a while about all automation actually present in an organization and thus to look at it as it was (or would have been) in the pre-automation era.

The actors in an organization communicate about some world, which we prefer to call the object world. It is important to grasp the distinction between the system of communicating actors, or the subject system as we prefer to call it, on the one hand, and the object world where the communication is about on the other hand. What constitutes the object world therefore is easily to determine: it encompasses everything where the communication in the subject system is about. The subject system as well as its corresponding object world are at every moment in a particular state. The state of the subject system represents the progress made in performing activities, the state of the object world represents the results of these activities. We come back to these issues after having introduced the concept of transaction.

A well-known distinction in levels of abstraction when studying organizations from the perspective of informatics is the distinction between the documental and the informational level. At the documental level an organization is viewed as a system of actors that produce, store, transport and destroy documents (Note. By a document we mean any information carrying substance, like a sheet of paper containing text or a drawing, or a magnetic disk containing bit patterns).

At the informational level one abstracts from the substance and the syntactic aspect in order to focus on the semantic aspect of information. What one observes now is a system of actors that emit and receive messages (semantic meanings) to and from each other. This is the level where most current methods and techniques (like e.g. the DFD and the ER-model) aim to be helpful, or, in fact can only be helpful, in spite of farther reaching claims.

It appears to be possible to abstract even further by focusing on the pragmatic meaning of these messages, i.e. on their role in carrying on the business activities. Language philosophy provides the necessary instruments for analyzing an organization at this level of abstraction, which we prefer to call the essential level. What one observes when focusing on the pragmatic aspect is a system in which the actors carry on units of communication that have a particular effect. We call these units conversations. More specifically, they are performative conversations, i.e. conversations resulting into an actual change of the state of either the subject system or the object world. We distinguish between two kinds: actagenic conversations, resulting in agreements about future actions (agenda of the subject system) and factagenic conversations, resulting in the establishment of facts in the

---

1 Communication is here to be understood in its original human sense, not in a technical sense. A good definition therefore would be: sharing mental states.
2 We only take into account the business communication in an organization. So, chats, expressions of emotions, and conversations not related to the business at hand, are excluded.
object world. Because only in performative conversations, original new things are accomplished, we consider these conversations to represent the essence of an organization. Furthermore, we call the actions that are agreed upon in actagenic conversations and the results of which are established in factagenic conversations, essential actions, and the conceptualization of the system observed the essential model of the organization.

Because of the very nature of an organization, these essential conversations and actions can only be performed by responsible, authorized subjects. Every other activity could as well, and often much more efficiently, be performed by artefacts. This holds especially for all actions that are purely and only informational. There are two kinds of such actions. One kind is the reproduction of already existing information, thus the actions usually performed by data base systems. The other kind is the production of information that is only a derivation from other information by means of mathematical or logical computation, otherwise said the processing of information.

![Figure 1. The documental, the informational and the essential level of abstraction](image)

The relationship between the documental, the informational and the essential level of abstraction is depicted in figure 1. This figure must be interpreted as follows. For any organization there exists at any moment one documental model, one informational model and one essential model. In principle, one may conceive of a number of documental models, all realizing the same informational model. Otherwise said, there is a freedom of choice, and consequently a choice problem. Choosing and implementing a documental model is what Information System (Re)engineering is about. The choices are determined by the available information technological possibilities. Likewise there is a freedom of choice when transferring from the essential level to the informational level. The choice concerns the purely informational actors, i.e. actors that only reproduce or derive information, and the particular messages by which the essential actors communicate in order to carry on their performative conversations. Choosing an informational model is what Information System (Re)design is about. It is part of the more encompassing activity of Business Process (Re)engineering.

Having conceived of the essential level of abstraction and having discovered and understood the crucial role it fulfils in the design of information systems, it is rather astonishing to find that current methodologies do not encompass that level!

The core modelling concept in DEMO is the concept of the (essential) transaction. A transaction is considered to be the basic pattern of organizational behaviour. It evolves in three phases: inception, execution and conclusion. Figure 2 exhibits this pattern.

During the inception phase agreement is reached between actor 1 and actor 2 about the future execution of an action by actor 2. This phase consists of an actagenic conversation, initiated by actor 1, starting at $t_1$ and ending at $t_2$. The result is an agendum (singular of agenda) for the execution of an essential action by actor 2. During the execution phase this essential action is executed by actor 2, somewhere between $t_2$ and $t_3$. During the conclusion phase actor 1 and actor 2 reach agreement about the facts that have been accomplished as a result of the execution by actor.
2. It consists of a factagenic conversation, starting at \( t_3 \) and ending at \( t_4 \). Actor 1 is called the \textit{initiator} of the transaction and actor 2 the \textit{executor}.

\begin{figure}[h]
\centering
\includegraphics[width=0.8\textwidth]{transaction_pattern.png}
\caption{The pattern of a transaction}
\end{figure}

The behaviour of any organization thus is conceived as consisting of carrying through transactions. Every (essential) action is embedded in a transaction and every established fact is the result of the successful carrying through of a transaction. This constitutes the DEMO \textit{transaction paradigm}.

Carrying through a transaction can be viewed as a discrete event process, and can thus be modelled by means of a state transition diagram (Dietz 1994b). Every state of a transaction process then is a state of the subject system, as opposed to a state of the corresponding object world, which is the set of facts established as the result of the successful carrying through of transactions. For example, buying a car or a house is a (transaction) process that may proceed through a large number of distinct subject system states; its successful completion results into the transfer of property which is a fact in the object world.

2.2 The methodology

On the theoretical foundations explained above, a practical methodology is developed for the informatical analysis of organizations, in particular for deriving the essential model of an organization.

\begin{figure}[h]
\centering
\includegraphics[width=0.8\textwidth]{essential_model.png}
\caption{The partial models of the essential model}
\end{figure}

This essential model is an integrated whole of several partial models, as summarized in figure 3. The communication model contains basically the identified transaction types and the actors that are involved as initiator or executor. The fact model is a specification of the fact types and the
constraints that together constitute the state space of the object world. The process model is a specification of the possible transaction processes for every transaction type. The action model is the specification, as far as possible or known, of the behaviour rules of the actors. Only the communication model is used in this paper; it will be further explained below. The double arrows represent one-to-many relationships. So for instance, to one and the same communication model a number of process models may belong, and also a number of fact models. Otherwise said, the communication model is the most generic model. This is indeed the case; in fact, an organization can be identified by its communication model. A change in its communication model therefore is equated with a redefinition of the organization. Modifying any of the other partial models is considered to be a redesign of some business process(es).

The communication model of an organization is the specification of the influencing by the actors of each other's behaviour. It is said to represent the interaction structure and the interstriction structure between the actors. By interaction structure is understood the mutual influencing through being initiator or executor of transactions. By interstriction structure is understood the mutual influencing by means of the subject system and object world state elements that serve as data in the condition part of the behaviour rules that are executed in carrying through transactions.

A communication model can graphically be represented by means of a communication diagram (CD). Figure 4 shows the graphical elements of the CD.

![Communication Diagram](image)

**Figure 4.** The graphical elements of the communication diagram

An actor is represented by a box, identified by an actor number (A...). A transaction type is represented by a disk, identified by a transaction type number (T...). The operational interpretation of a disk is a store for the statuses through which the transactions of that type pass in the course of time. The disk symbol therefore is called a **transaction bank**.

The facts in the object world that are created as a result of the successful carrying through of a transaction, are considered to be represented by true propositions stored in a **fact bank**. A fact bank is represented by a diamond. To show that the facts of some fact bank are the result of carrying through transactions of the same particular type, the diamond symbol is drawn 'behind' the disk symbol.

The combined symbol of a transaction bank and its corresponding fact bank is suited to represent internal fact types and boundary fact types. However, the actors in the system may also need to know external facts, i.e. facts created outside the system. The combined symbol is too specific to represent the external fact types, because the corresponding transaction types fall outside our scope of interest: they are not known. Therefore, the external fact types are arbitrarily put together in external banks, which are represented by only the diamond symbol. They are labelled by an external bank number (E...).

The actor who is the initiator of a transaction type is connected to the transaction bank by an **initiator link**. It is represented by a plain line. The actor who is the executor of a transaction type is
Expressing Production Control Principles in the DEMO Communication Model

connected to the transaction bank by an *executor link*. It is represented by a plain line with an arrow head at the side of the actor box, pointing to that box. Interstriction is represented by dotted lines (data links) between actors on the one side and fact banks and transaction banks on the other side. Figure 5 shows possible constructions in a CD.

![Diagram](image-url)

**Figure 5.** Possible constructions in a communication diagram

The grey-coloured ‘roundangle’ represents the system boundary. The actors A0.1 and A0.2 are external actors, the actors A1.1, A1.2 and A1.3 are internal actors. The transaction types T1 and T2 are boundary transaction types, the transaction types T3 and T4 are internal transaction types. The bank E1 is an external bank.

The meaning of the example CD in figure 5 can be explained as follows. The external actor A0.1 is the initiator of transaction type T1; the internal actor A1.1 is the executor of this transaction type. In executing the transactions of this type and in carrying on the performative conversations, actor A1.1 uses data from the external bank E1. As a result of the execution of a transaction, it states one or more facts, which are stored in the bank of T1, and it initiates a transaction of type T4. Actor A1.3 is the executor of this transaction. In executing the transactions of type T4 and in carrying on the performative conversations, it uses data from the transaction bank of transaction type T3 (transaction statuses). This transaction type has the same actor (A1.2) as initiator and as executor. We call this auto-activation of actor A1.2 (Note. This is the way to model periodic activities). In executing the transactions of this type and in carrying on the performative conversations, actor A1.2 uses elements of the contents of the fact bank of T1 and the transaction bank of T4 as data. As a result of the execution of a transaction, it initiates a transaction of type T2. The executor of this transaction type is the external actor A0.2.

The distinction between *auto-activation*, e.g. transaction type T3, and *allo-activation*, e.g. transaction type T2, needs special attention, because it is a major point in the discussion of production control structures in section 3. Allo-activation is the direct way for an actor to influence the behaviour of another actor: actor A0.2 only becomes active if and when actor A1.2 initiates a transaction of type T2. Compared to this, there seems at first sight to be no influencing by another actor in the case of auto-activation. The behaviour of actor A1.2 however is certainly influenced by other actors, be it indirectly. According to figure 5 the contents of the fact bank of transaction type T1 as well as the contents of the transaction bank of transaction type T4 are taken into account when actor A1.2 is active, i.e. when it executes transactions of type T3. This indirect way of mutual influencing was called interstriction earlier, expressing that it restricts the action or decision space of the actors.
The practical meaning of the distinction between interaction and interstriction is that, in general, interaction is the fastest way to perform a chain of subsequent activities (a business process) at the cost of efficiency (lower occupation rates of resources). Interstriction on the other hand offers maximum independence between the 'chained' resources, and therefore facilitates high efficiency at the cost of higher throughput times.

3. Production Control Structures

The DEMO methodology described in the previous section is the basis for the construction of so called reference models for typical production control structures. Particularly the DEMO communication models are presented. Reference models can enhance significantly the modelling of the logistics of organizations as well as reduce the development time of these models. After determining the type of production control structure, one of the reference models is chosen that matches the general case of such a production control structure. Then the specific attributes of the organization to be modelled are added and incorporated in the chosen reference model to create a more specific model of the organization.

In the field of CIM (Computer Integrated Manufacturing) and the development of intelligent manufacturing systems, the GRAI method has been developed (Doumeingts 1989). This method is a typical example of an approach aimed at the development of a production management system, combining artificial intelligence tools, production management techniques and system analysis methods. More specifically, the GRAI method is meant for the decisional system as part of the larger production management system. It consists of several modelling techniques. The first one, called grid represents decision centres in a hierarchical manner, describing their activities. Decision centres are based on different planning horizons (e.g. one year to three months, one month to one day etc.). The second are so-called nets, based on Petri Nets, representing the way activities are performed. For the application of the GRAI method different reference models are developed that are used as the basis for the analysis of the application domain. One of these models, called 'skeleton' represents the ideal functional structure of the MRP production management system.

The GRAI method is not comparable with DEMO because it only covers the informational level (cf. section 2.1). So the GRAI reference models are also on the informational level. The development of reference models on the essential level in DEMO will be shown below.

We make a distinction between two major types of production situations. The first one is the job shop production and the second one is the chain production. In the job shop production each order has a specific routing on the different machines available, while in chain production, also known as a pipe line, all orders follow the same routing. Each of these two production control structures is represented in a communication model. We will also show, where appropriate, the correspondence between the essential control structure and the material flows.

The job shop production control structure is depicted in the CD of figure 6. The figure can be explained as follows. The actor Customer is initiator of transactions of type T1 (order delivery). The actor Order Acceptance is executor of this transaction type. After the successful completion of the inception phase of a transaction of this type, the customer order is accepted. Engineering is responsible for the preparation of the order and determines the order routing on the different work stations. A work station is defined as the collection of machines performing the same type of operation (e.g. all milling machines are combined in one Work Station). The actor Engineering is both initiator and executor of transactions of type T2 (check-order-engineering). This is e.g. done on a daily basis. Engineering uses the data about the order from transaction type T1 (the dotted line from T1 to Engineering) as conditions for the execution of transaction of type T2. This holds also for the actor Scheduler. This actor initiates (and executes) transactions of type T3 (check-order-scheduling) and uses data about the order (e.g. delivery dates) from T1 and T2 for the execution of these transactions. As a result of scheduling an order the Scheduler initiates the transactions for the production of the order at the Work Stations WS1 through WS4 (Note: we arbitrarily have chosen four work stations). After the successive production steps have been finished at the Work Stations
(in conformance with the order routing), the order can be shipped to the customer (not shown in figure 6).

Figure 6. Communication diagram of job shop production control for regular orders

Next to the customer orders, so called forecast orders may also be scheduled on the work stations. These orders can be viewed as internal customer orders with the same order dispatch as the (external) customer orders. Both kinds are collectively called 'customer orders' hereafter.

What is expressed in figure 6 is the production control structure for so called regular customer orders. In most job shops a second kind of order is identified, called rush orders. A rush order is an order that gets priority over all regular orders at every work station. To control these rush orders a new actor is introduced, called the Chasseur. It plays the combined functions of Engineering and Scheduling for these orders. This situation is represented in figure 7. After the acceptance of a rush order (T1), Order Acceptance directly activates the Chasseur (via T8) in order to deal with it. The same type of transactions (T4', T5', T6' and T7') are initiated as in the case of the regular orders. However, now the Chasseur is the initiator. All Work Stations are directed by the Chasseur to perform the actions, and all regular work in progress is overruled.

We now come to the second major type of production situation, the chain production. The basic difference between job shop production and chain production is that all orders have the same order routing. For each of the Work Stations a separate Scheduler deals with the scheduling of the incoming orders. Because of the independent operation of the Work Stations and the variance in processing time, intermediate stocks are created between them. Orders are flowing from one Work Station to the next till the end of the production line has been reached. Figure 8 presents the CD of the chain production control structure. The figure can be explained as follows.
The actor Order Acceptance is responsible for the acceptance of incoming customer orders (T1). In this particular example four work stations are involved in the production process (WS1 through WS4). The data about the order (from T1) are used by Scheduler 1, to schedule the operations for Work Station 1 (by means of the data link between the transaction bank T1 and Scheduler 1)³. This actor is both initiator and executor of transactions of type T2 (check-scheduling-1). After the schedule has been made, the scheduler directs the Work Station 1 to perform the requested operations. Finished work is stored in an intermediate stock point for the next Work Station (in this case WS2). Scheduler 2 uses, beside the data from T1 (about the order), data from T6 about the progress in the production at Work Station 1. The same holds for Scheduler 3 and Scheduler 4.

³ Although the order has been accepted, the delivery still has to take place. Until the delivery has taken place, this transaction is still pending. For this reason a data link exists between the transaction bank and the scheduler.
In figure 8 we combined the essential control structure with the material flow between the Work Stations. Finished work from one Work Station is stored in an intermediate stock point between the Work Stations. These stock points are depicted in grey (at the bottom of figure 8).

With respect to rush orders as has been described for the job shop control structure, the same kind of control can be found in the case of chain production. It is depicted in figure 9. After acceptance of the order from a customer (T1), order acceptance directs the first Work Station in line to produce the order. After finishing the operations, Work Station 1 will direct Work Station 2 to continue with the order and so on. Because of the variance in processing time, waiting queues (instead of stock points) may be formed between the Work Stations. These waiting queues are depicted (in grey) at the bottom of the figure.

4. Extension with the Customer Order Decoupling Point

In the previous section four different types of reference communication models have been presented, based on two production control structures. In this section an extension to these communication models is made using the so-called Customer Order Decoupling Point (Hoekstra and Romme 1992). The Customer Order Decoupling Point (CODP for short) specifies the influence of a customer on the products to be produced and divides the business activities of the organization in a customer oriented and a planning oriented part. The choice for the position of the CODP is decisive for the organizational structure, including the logistic organization and the risks the company will accept. Figure 10 shows the five possible CODP's.

The uppermost part of figure 10 shows a general production line with stock points represented as triangles. The lower part of the figure depicts the five different CODP's as stock points from which customer orders are distributed. For example, an organization producing for stock only and delivering customer orders from this stock point can be characterized with the first CODP (physical distribution from stock). An example of such an organization is a ceramics producing firm. The customer is restricted to the product range available from the distribution stock point. The other extreme (CODP 5) shows that every order is customer specific (engineer to order). An example of such an organization is a satellites producing firm.
A direct relationship exists between the communication models from section 3 and the CODP. Every communication model has been constructed with the idea in mind that orders are specific and only limited by the capabilities of the firm in question and the available product range. While both CODP 1 and 5 show extreme (rare) cases, the communication models constructed so far can be characterized with CODP 4. Orders from customers initiate the production process. In the other cases, the production process can be divided in a part concerned with the production of semi-manufactured materials and a part that is customer order driven. The difference in control structure as a consequence of the position of the CODP can also be represented in the communication models.

The introduction of the notion of the CODP in the communication models of section 3 will result in the following changes. The changes in the diagram will be discussed using the chain production
for regular order handling as an example. For this case it is observed that the customer order driven activities follow the same line as the diagram of figure 8. But the planning driven activities, which are part of the regular control structure, are depending on the CODP, restricted by a different actor. The actors Scheduler 3 and Scheduler 4, performing scheduling operations now only initiate the transaction types that are customer order driven (T8 and T9). A different actor, called Planner restricts the planning driven transaction types. Figure 11 shows the case of CODP 3 (assemble to order).

In the case of CODP 4 (shown in figure 8) all schedulers of the work stations are restricted by the data from transaction type T1. The CD of figure 11 shows that only those actors that are involved in the customer order driven part are still restricted by the same data. The other actors (Scheduler 1 and Scheduler 2), concerned with the production of semi-manufactured products are restricted by the data from the Planning (T10). There exists a relationship between the two parts (planning driven and customer order driven production): the data from transaction type T10 are used by Scheduler 3 for the execution of the scheduling activities.

In the case of production on stock (CODP 2) a further shift in responsibilities can be noticed. Data produced for the planning is used by the Scheduler 1, Scheduler 2 and Scheduler 3 in this case. Work Station 4 must now be viewed as a distribution point.

5. Conclusions

The first research question, stated in section 1, was whether and how the characteristic features of typical production situations could be expressed in a DEMO model. This question has been dealt with in section 3, with respect to one of the partial models, namely the communication model. It has been answered first for job shop production and then for chain production. In both cases a distinction has been made between regular orders and rush orders.

Probably the most clarifying property of the DEMO communication model is the distinction between interaction and interstriction. From the CD’s presented several issues become at once clear. The first one is that in job shop production there is a need of a central scheduler that directly activates all distinct work stations. In contrast with this, the scheduling of operations in a chain production situation is decentralized (in principle at least; one may assign the scheduling tasks to one department, but that does not change the essential character of independence!). The second issue is that rush orders have minimal waiting times. For the case of chain production this is very evident when comparing figure 8 and figure 9. Rush orders are never put into intermediate (work-in-progress) stocks, at most in waiting queues of rush orders. There is in principle no distinction between this case and the case of job shop production. In the last case the Chasseur takes, so to speak, the work under his arm and waits only for other rush orders.

The second research question was whether and how the notion of CODP could be dealt with in a DEMO model. This question has been addressed in section 4, for the case of chain production. The CD of figure 11 shows very clearly that a shift of the CODP from position 5 'downwards' introduces the need of a planning function that generates production orders independently of the acceptance of customer orders. The span of control of this actor Planner includes all work stations that are in the material flow located before the CODP (in figure 11 between WS2 and WS3).

One of the main advantages of reference models for the purpose of redesign is that they enhance significantly the modelling of the logistics of organizations as well as reduce the development time of these models. In this way they provide a better support for the analyst and removes much of the subjectivity which is often present in the first phase of the analysis. Above all, the reference models have the major advantage already shown in other DEMO analyses (see e.g. Rijst and Dietz 1993): a coherent modelling structure between the data, process and behaviour perspective. The use of reference models is in the first place meant for the first phase in the analysis process. Each predefined transaction type in the communication model can be discussed and elaborated, if needed, in the discussions between analysts and managers.
The research in the near future is aimed at a further exploitation of the use of reference models in modelling practice. Up till now two case studies have been analysed on the basis of the proposed communication models. Currently a new case study is performed with the reference models used as the basis for the analysis process. A major advantage of the new case study is its difference with the other two case studies. The company under study is only producing on customer's request in a highly competitive market. Other parts of the research are aimed at the development of reference models for other types of organizations like e.g. departments with a high information load and intrinsically difficult interactions between the organization and its environment (like e.g. customers and suppliers).

Lastly, some general conclusions can be drawn concerning the application of DEMO. It has been shown that it is possible to acquire an understanding of the business processes of an organization at a higher level of abstraction than is currently provided by most methodologies. This understanding is complete and sufficient in the sense that it contains everything needed to (re)design and (re)engineer the information systems and the information infrastructure. Moreover, it has become clear that (re)designing information systems in fact can only start from an understanding that abstracts from the informational level (cf. figure 1).

The compactness of an essential model also offers a very practical advantage. The reduction in size compared to a model at the informational level (e.g. by means of DFD's and ERD's), typically a factor 5 to 10, may already be a reason to apply DEMO. Much more important however is the new, original, way of looking at an organization. To this we can happily add our recurrent experience in practice that the transaction paradigm is appealing to as well as easily grasped by the people that are actually performing the business processes in an organization.

References


SIMULATING INDIVIDUAL AND TEAM EXPERTISE IN A DYNAMIC DECISION MAKING ENVIRONMENT

Michael D. Coovert and David W. Dorsey
Department of Psychology and Institute for Human Performance,
Decision Making and Cybernetics, BEH 339
4202 E. Fowler Avenue
University of South Florida
Tampa, Florida 33620-8200, USA
Fax: 813-974-4617, e-mail: coovert@esrvm (BITNET)

Keywords: Petri nets, simulation, decision making

ABSTRACT

Traditional methodologies for modeling and measuring individual performance do not perform effectively in rapid tempo decision making environments. Furthermore, if the performance of an aggregation of individuals -- such as a team -- is of interest, traditional approaches are also found lacking. Our goal is to understand the impact of such variables as expertise on individual and team performance in combat information centers aboard naval warships. We have found Petri nets an effective tool for modeling the performance of individuals and teams, in these fast tempo and information overloading environments. As we present here, simulation models of performance can be constructed to determine the impact of such variables as expertise on performance variables such as time to perform a task, the amount of time engaging in optimal behavior, and estimates of operator workload.

1. INTRODUCTION

During the late 1980s the United States Navy was involved in several incidents which had unfortunate and grievous consequences. The most well known of these incidents involved the Aegis class guided missile cruiser Vincennes. Out of these tragic incidents grew several research programs whose goal is, in part, to better understand human performance in complex decision making environments. Once this understanding takes place, interventions such as training, and decision aids can be developed to augment human performance for individuals and teams in these environments.
Early work by the first author demonstrated that Petri nets provide a useful modeling capability for representing individuals as they interact with various computer and control systems in combat information systems. That is, we can represent in a graphical manner, individuals performing various tasks and duties. The present paper extends that work by: 1) providing analytical evidence regarding the advantages of Petri nets as a diagnostic tool in assessing human performance in a dynamic team environment, 2) extend the notion of Petri net modeling and computer simulation as a means for testing psychological theories such as decision making, and 3) demonstrate how specific variables of interest, such as expertise, can be modeled and analyzed using simulations.

In task environments as complex as a combat information center (CIC), having techniques that go beyond conventional methodologies -- in terms of accurately specifying and measuring human performance -- are essential. In the case of the CIC team, analyzing and then correcting performance deficits can be life-critical.

1.1 A Brief Overview of Petri Nets

Conventional tools such as flow diagrams, narrative descriptions, and time-line analysis, are useful for a variety of modeling problems; however, they impose constraints on the representation and therefore are not powerful enough for certain types of tasks. Specifically, those tools make it difficult to expose critical time dependencies, task concurrencies, and behavior which is event driven. Petri nets are a very useful and powerful modeling tool and overcome the aforementioned shortcomings.

The methodology was proposed by C. A. Petri (1962) in his dissertation as a general purpose modeling tool for asynchronous systems. Systems which have been successfully modeled by Petri nets have such diverse characteristics as being: distributed, asynchronous, concurrent, and stochastic. The approach is especially useful since individuals at work often perform tasks in an asynchronous manner with stochastic properties. Furthermore, since we have the capability of modeling parallel activities (and conflict) with this methodology, aggregates of individuals (sub teams and teams) can also be modeled.

Petri nets have both graphical and mathematical properties. The graphical aspect makes them a useful tool for representing a system in a visual manner, making it an excellent communication medium.

As a mathematical tool, algebraic equations, state equations, or other mathematical models are established which control the behavior of the system. The mathematical underpinnings of the nets allow for rigorous analyses of various types. We will not discuss the mathematics of Petri nets here, the interested reader is referred to Reutenauer (1990).
A good overview of recent developments and extensions of Petri nets can be found in the book edited by Jensen & Rozenberg (1991). Readers interested in special issues are encouraged to explore the literature (cf., Jensen & Rozenberg, 1992; Peterson, 1981; Reisig, 1992) on the topic of interest, and to employ various software packages for the analysis of their models (cf., Alphatech, 1990; Chiiola, 1989; Metasoft, 1992; Perceptronics, 1992).

1.1.1 Basic components. The basic components of the nets are quite few; yet, we can use these limited building blocks to construct and represent very complex and powerful models. There are three basic elements in a Petri net. The first is a representation of an active component of the system we are modeling. Active components are represented as rectangles or squares (square). Active components are used to represent agents or events, and are generally referred to as transitions. Passive components are the second system type, and they are represented as circles (round). Passive components also represent preconditions, or post conditions and are generally referred to as places. Connections between the active and passive system components are made through arrows (arrow), with the direction of the arrow indicating the direction of the relationship (e.g., the flow of information). For example, an active component connected to a passive component and the passive connected to a subsequent active one is represented as: square → circle → square.

Petri nets can be an excellent tool for communication because of their graphical nature but these nets go beyond flow charts and block diagrams in that they incorporate tokens, which are used to simulate dynamic and concurrent activities of a system. Tokens reside in places and move throughout the net as the transitions "fire." The firing of a transition is controlled by rules associated with the transition. In the simplest case, a transition is enabled and fires as soon as a token resides in the place that precedes it. Tokens are used to represent abstract or non-abstract entities within a model and are usually depicted as a solid circle .

Formally, the structure of a Petri net is a bipartite directed graph, $G = [P, T, A]$ where $P = \{p_1, p_2, ..., p_n\}$ is a set of finite places, $T = \{t_1, t_2, ..., t_m\}$ is a set of finite transitions, and $A = \{P \times T\} \cup \{T \times P\}$ is a set of directed arcs. The set of input places of a transition $(i)$ is given by $I(i) = \{p | (p, i) \in A\}$, and the set of output places of transition $(i)$ is given by $O(i) = \{p | (i, p) \in A\}$.

So, in the simplest case, only three elements are used to construct a Petri net model; transitions reflect active model components, places represent passive components, and directed arcs add structure in terms of possible linkages between: places-to-transitions and transitions-to-places. Tokens represent the current state of the system (e.g., where ever a token resides, that activity is being performed or that place is being occupied). The mathematics of the system underlie the structure and controls the behavior of the system.
2.0 METHOD

2.1 Jobs

The three jobs modeled in this study are the Tactical Information Coordinator (TIC), Identification Supervisor (IDS), and Electronic Warfare Supervisor (EWS). These jobs are a subset of the larger combat information center (CIC) team on board U.S. Navy Aegis ships. The first job, Tactical Information Coordinator, is responsible for all air and surface tracking and identification. The second job, Identification Supervisor, is responsible for establishing and entering identification parameters, monitoring the assignment of identification to tracks, and resolving identification conflicts. The third job, the Electronic Warfare Supervisor, oversees the operation of certain radars and sensors to ensure, among other things, the proper reporting, correlation, and triangulation of tracks. (A track is any contact identified by the various radars and sensors).

2.2 Specification of Behaviors

In order to determine what actions, behaviors, or decisions might occur in each of the three positions for a given environmental cue, a declassified version of a scenario entitled "Desert Shriek" was employed. This scenario models similar events to those involved in the Vincennes' incident; specifically it includes 122 discrete identifiable events that are important to the team. The information comes in the form of radar data, voice reports, link data (from other platforms), or electronic warfare data. Figure 1 provides in the form of a timeline, a general overview of some of these events.

Subject matter experts provided us with the list of appropriate behaviors that each operator should perform for each event throughout the scenario. This information included: 1) how critical the event is within the scenario (e.g., a missile launch toward ownship is more critical than a friendly merchant ship fading from radar); 2) what are the pre- and post-event behaviors occurring for each of the events; and 3) what is the likelihood that an expert and a novice would engage in the described behavior. For example, listed on the timeline in Figure 1 as an Air Report at time 38 minutes and 00 seconds in which 1 hostile aircraft changes course. The experts provided to us a criticality index for the event; those behaviors that each individual (TIC, IDS, EWS) should be performing given what has happened up to that point in the scenario and what actions the operators should perform given that the event just occurred; and the likelihood that an expert and novice, respectively, would engage in the appropriate behavior for that event. We used this information to build the models of operator behavior for the simulations.
Figure 1. Event timeline
2.3 Apparatus

Percnet/HSI software (Perceptronics, 1992) was utilized and the simulations executed on a SUN SPARCstation. Percnet is a knowledge-based graphical simulation environment and uses Modified Petri nets (MPNs), as the primary modeling tool.

2.4 Procedure

Subsequent to gathering the data from the subject matter experts, Petri net models of each of the three positions in the team were built with Percnet. The specific architecture of these models will be discussed shortly. After constructing the basic net structures, control structures were added that allowed the manipulation of independent variables such as level of expertise (i.e., novice versus expert) and informational ambiguity. Dependent measures including behavioral latencies (i.e., time), behavioral frequencies, and workload measures were collected based on simulation runs of the different conditions.

2.5 Independent and Dependent Variables

The independent variables chosen for manipulation are a small representation of variables that probably play a role in CIC performance. However, subject matter experts agree these two variables are among the most important. The variables are: expertise, which is operationally defined in terms of a dichotomy based on amount of experience, either expert or novice; and informational ambiguity, which is defined and manipulated in terms of a set of events involving conflicting or vague information. A concrete illustration of informational ambiguity is, the possibility of unreliable identification/sensor data due to adverse atmospheric conditions. In the simulations, informational ambiguity corresponded to a delay in operator behavior, as well as an increase in the probability of performing a non-optimal behavior. Both expertise and informational ambiguity are hypothesized to be key factors in dynamic, naturalistic, decision making environments (Klein, 1989).

The dependent variables generated from a simulation include: behavior latencies (mean duration of activities -- how long it takes an individual to perform a task), behavioral probability (frequency of behavior -- the probability an operator performed the correct behavior in reaction to a scenario event), and the amount of individual operator and overall team workload. Workload is defined according to the Multiple Resource Model (MRM) of Wickens (1988). This workload model measures the individual demands on a series of resource channels (e.g., visual processing, auditory processing, verbal processing, spatial processing, continuous motor, and discrete motor) and also takes into account concurrent activities in terms of resource conflict. Individual resource demand values for each activity were determined by subject matter experts.
2.6 Design of the Operator Nets

Figure 2 presents the basic architecture for the models of operator behavior. It should be emphasized that what is being modeled is not the scenario events themselves but the behavioral or "problem" space of the operators (Anderson, 1993). As the figure illustrates, the modified Petri nets in the Percnet implementation are similar to traditional Petri nets.

The activities (in this representation are noted as circles) represent a specific activity that is taking place. In the Percnet implementation the events (noted as rectangles) mark the end of specific activities and the events "fire" to denote that a specific event has taken place or "happened" (Percnet, 1992). Along the top of the net is a series of two acts (labeled ACT1 and ACT2) that an operator might engage in after having received cues that a specific event has just occurred. These acts can be thought of as the appropriate actions that need to be taken in response to the event. The data collected from the subject matter experts outlines what each of the appropriate actions should be.

Also, inspection of this net indicates that instead of engaging in the appropriate series of acts, the operator might make an error -- either at the first or second step in the series -- and engage in another act (labeled as ERROR1 and ERROR2). These represent non-optimal performance, or behaviors performed in response to a scenario event that were specified as non-optimal by the subject matter experts. These behaviors are modeled as a function of the independent variables discussed earlier. For example, the probability that any of the events labeled with an e in the figure will fire might be very low for an expert (e.g., .01) and very high for a novice (e.g., .70). Also, note that even if an operator does make a mistake initially, he might still engage in the next appropriate behavior. In modeling these error activities an assumption had to be made about what the error activities might be. For practical purposes these error activities were modeled as normal operating procedures (unique for each operator) that would have a high likelihood of occurring, given the context of the scenario at that time.
Petri nets allow the measurement of performance at different levels of a system hierarchy, such as team, position, duty, or task. Figure 3 provides an example of how the Petri net model for the EWS operator is constructed in a hierarchical fashion. Each of the dotted line arrows indicates that the net in the level below is a subnet, or elaboration, of the original net. The activities with the double circles indicates that the net contains a subnet.

Figure 3. Hierarchical series of elaborations

Level 1 in Figure 3 corresponds to the overall team with the three positions. Level two is the duty level and is the level of our main focus here. Level three corresponds to the task...
level, and the nets could go even further down to elemental levels of operator behavior corresponding to such actions as specific button pushes. [We have, in fact, constructed nets at this high level of detail for other purposes (e.g., to model human-computer interaction with the Command and Decision, Weapons Control, and SPY radar systems)]. We choose the duty level as the main level of focus for performance measurement, because it most clearly indicates the outcomes of operator decision making. The task level is basically the sequential steps that the operator has to take in order to complete a specific activity. To provide the reader a feel for the complexity of the problem, Figure 4 provides the duty level net for the behaviors which could be performed by the TIC in response to scenario events 25 through 60.

3. RESULTS

This section presents simulation results based on workload, behavioral frequency (i.e., what is the operator doing), and behavioral duration (i.e., how long does it take the operator to perform the action). Each of these events is really a behavior taken in response to some type of informational event in the scenario; voice, link, ESM, or other sensor data (see again, Figure 1).

Based on Wicken's MRM model, Table 1 shows workload measures for both an expert and a novice TIC. The table contains information for the total time to complete the scenario, the overall workload levels, that point where the workload was the highest, and the activities that caused the highest workload levels.

As expected, the expert TIC completes the scenario in less overall time, and has a slightly higher workload level. The activities causing the highest levels of workload are, however, the same for the expert and novice TIC. Interestingly, the novice operator reached those high levels of workload much later in the scenario, most likely because he "got behind" and was unable to accomplish the correct tasks in the correct order and in a timely manner.

Tables 2 and 3 provide the same information for the IDS and EWS positions. Interestingly, except for the experts having a higher overall workload than the novices, the TIC pattern of results does not hold for the other two positions. First of all, there is very little time differences in the overall length of time to perform the duties for the experts and novices. Also, the time for the highest level of workload is slightly longer for the experts here than for the novices. Also, the point that causes the highest levels of workload are different for the novices than the experts. These differences from the TIC position are probably due to the novices spending more time in "error" activities than are the experts.
Figure 4. TIC Events 1-24
### TIC Duration, and Workload Data For Expert and Novice TIC

<table>
<thead>
<tr>
<th>Activities</th>
<th>TIC - Expert Team</th>
<th>TIC - Novice Team</th>
</tr>
</thead>
<tbody>
<tr>
<td>Occurring</td>
<td>SUB ID STATE 5 - Ensure ID is correct</td>
<td>SUB ID STATE 5 - Ensure ID is correct</td>
</tr>
<tr>
<td>During</td>
<td>TIC NOS 5 - Ensure ID is correct</td>
<td>TIC NOS 5 - Ensure ID is correct</td>
</tr>
<tr>
<td>Largest</td>
<td>COORD/COMM 2 - Comm. with other operator</td>
<td>COORD/COMM 2 - Comm. with other operator</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time:</th>
<th>TIC - Expert Team</th>
<th>TIC - Novice Team</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1253.417 sec.</td>
<td>1730.407 sec.</td>
</tr>
<tr>
<td>Workload</td>
<td>.265</td>
<td>.204</td>
</tr>
<tr>
<td>(Mean for TIC):</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Largest Workload:</td>
<td>2.00</td>
<td>2.00</td>
</tr>
<tr>
<td>(Occurs at time:</td>
<td>528.36 - 531.43)</td>
<td>769.00 - 771.41)</td>
</tr>
</tbody>
</table>

### IDS Duration, and Workload Data For Expert and Novice IDS

<table>
<thead>
<tr>
<th>Activities</th>
<th>IDS - Expert Team</th>
<th>IDS - Novice Team</th>
</tr>
</thead>
<tbody>
<tr>
<td>Occurring</td>
<td>IDS ID STATE 1 - ID all contacts within radar</td>
<td>COORD/COMM 1 - Comm. with TIC</td>
</tr>
<tr>
<td>During</td>
<td>COORD/COMM 1 - Comm. with AAWC</td>
<td>IDS (error) 1 - Coord./Comm. with TAO, AAWC, TIC</td>
</tr>
<tr>
<td>Largest</td>
<td>1.48</td>
<td>1.51</td>
</tr>
<tr>
<td>Workload:</td>
<td>(Occurs at time: 172.76 - 173.42)</td>
<td>(Occurs at time: 259.69 - 260.79)</td>
</tr>
<tr>
<td>Time:</td>
<td>669.061 sec.</td>
<td>651.625 sec.</td>
</tr>
<tr>
<td>Workload</td>
<td>.454</td>
<td>.397</td>
</tr>
<tr>
<td>(Mean for IDS):</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
A final point of interest regarding behavioral differences at the individual operator level is the considerable difference in the amount of time the various operators spent in activities the subject matter experts indicated were non optimal, given the context of the scenario. Table 5 presents this information, and it can be seen that as expected, the novices spent much more time in non optimal activities than did the experts.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Expert Team</th>
<th>Novice Team</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIC</td>
<td>496.14 (40% of total time)</td>
<td>1495.72 (86% of total time)</td>
</tr>
<tr>
<td>IDS</td>
<td>143.05 (21% of total time)</td>
<td>232.68 (36% of total time)</td>
</tr>
<tr>
<td>EWS</td>
<td>0.00 (0% of total time)</td>
<td>82.74 (13% of total time)</td>
</tr>
</tbody>
</table>

Table 5. Total Time In Error Activities (seconds)

3.1 Team Level Analysis

The same information examined for the operators can also be analyzed for the team as a whole. Again, this is one of the strengths of the approach -- being able to model and analyze behavior from multiple levels of abstraction. Table 6 presents the team level data for expert and novice teams. The expert team is able to perform each of the behaviors elicited by the scenario events in a much quicker overall time than is the novice team. The workload levels are higher for the expert team, on both average workload and for the highest workload level. As would be expected, the behaviors the individual operators are engaged in during the periods of highest workload, are different for the experts and novices.

Activity frequencies in terms of workload range can also be compared for the two team types. Figures 5 and 6 present graphs of the different workload ranges for the expert and novice team, respectively, at different time segments of the scenario. The patterns are quite different from one another, indicating the very different levels of workload impacting on operators of the expert and novice teams. Statistically, these values can be evaluated with the chi square to determine if they are significantly different; and they are, based on 88 team activities $\chi^2 = 185$, $p < .01$.

A final result is that the expert team spent on average 20% of their time in non optimal behaviors while the novice team spent 45% of its time in non optimal behaviors.
### Table 6. Workload Measures For Expert and Novice Teams

<table>
<thead>
<tr>
<th>Activities Occurring During Largest Workload</th>
<th>EXPERT TEAM</th>
<th>NOVICE TEAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>- COORD/COMM 1 - Comm. with AAWC</td>
<td>1.215</td>
<td>.738</td>
</tr>
<tr>
<td>IDS - SUB ID STATE 3 - Make sure ID is correct</td>
<td>4.75</td>
<td>4.26</td>
</tr>
<tr>
<td>- ID STATE 1 - Use ESM data correlate contacts</td>
<td>(Occurs at time: 183.39 - 183.75)</td>
<td>(Occurs at time: 199.37 - 200.35)</td>
</tr>
<tr>
<td>TIC - TIC NOS 2 - Continue monitoring inbound missile</td>
<td>IDS - IDS ERROR 3 - Comm. with other operator</td>
<td>IDS - IDS ERROR 4 - Comm. with other operator</td>
</tr>
<tr>
<td>- TIC NOS 3 - Ensure tracks valid (no duals)</td>
<td>TIC - TIC NOS 4 - Ensure tracks are valid (no duals)</td>
<td>TIC - TIC ERROR 15 - Ensure ID's are correct</td>
</tr>
</tbody>
</table>
Figure 5. Novice workload values
Figure 6. Expert workload values
4. DISCUSSION

Our work demonstrates that Petri nets provide a useful tool for modeling human performance in dynamic decision making environments. Furthermore, models can be constructed to determine the impact of various individual difference variables, such as expertise, on the performance of individual operators and teams of individuals. However, the question remains as to what extent the mapping between our Petri net representation and the real-world situation provides utility. There are really several aspects of this issue, and we briefly address two.

First, it is obvious that the models validity, and the computer simulation results, are heavily dependent on real-world data, which serve to frame the model and drive the accompanying simulations. This is essentially the problem of "garbage in - garbage out" that is familiar to all researchers working with computer based models. Only to the extent that the subject matter experts are providing veridical information can the outcome of the simulations provide useful information.

The second issue is how to handle the growth of complexity within a network model. We need to capture enough detail in the model so all critical parameters are represented, but in a complex environment where many individuals are interacting with several different computer systems, the model rapidly becomes very large. This question is related to what is commonly called "Bonini's paradox" (Lewandowsky, 1993). The paradox asks the question: does the simulation or model turn out to be no easier to understand than the real-world processes (Dutton & Starbuck, 1971)? While our Figure 3 is easy to comprehend, Figure 4 is approaching a level of complexity beyond what is desired -- and it is representing only part of the behaviors for one operator, not all the behaviors for the team!

From a less pessimistic perspective, it is clear that variables such as expertise can be manipulated and tested in Petri net models. This is promising for hypothesis testing and modeling specific theories and also for generating hypotheses. For example, in this study we have generated some interesting questions concerning the levels of workload experienced by both experts and novices. Is it possible that the types of errors made by novices can effect workload and might this have an overall effect on performance?

The flexibility of Petri nets to allow both global and local performance measurement is a clear advantage of our approach. One can consider the measures related to a specific event-activity pairing, or consider the overall picture of activity within a set of nodes. This type of data can also address problems related to the flow of work behavior. For example, is there a specific activity where operators are taking too much time in terms of relaying information, thus creating a bottleneck for the entire team's performance? Also, a given vector of node measures could be validated against some criterion measure, for example a vector of the same activities as performed by an expert team on an important outcome criterion. Some preliminary work indicates certain approaches hold promise,
including linear statistics (Mahalanobis distance) and nonlinear analyses (neural networks).

Another question we might address with these models concerns team composition: how does a team with a mix of both expert and novice operators react in comparison to a team of all expert operators? Other more directly applied issues indicate the need for advanced modeling tools. For example, how can you concurrently measure such aspects of team performance as communication, psychomotor activities, workload fluctuations, and information flow? Also, how are errors propagated throughout the team's information processing network? These are the types of issues that the Petri net modeling and simulation hold great promise in addressing. If researchers hope to address the dynamic nature of behavior, measurement techniques have to move away from conventional analytic approaches, to techniques that address the moment-to-moment interdependencies that exist in a system.

5. REFERENCES


Authors Note

This research was supported in part by Contract No. DAAL03-86-D0001, Delivery Order 2212 of the Scientific Services Program. The views, opinions, and/or findings contained in this paper are those of the authors and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.
AGENT-ORIENTED REQUIREMENTS ENGINEERING
A CASE STUDY USING THE ALBERT LANGUAGE

Eric Dubois, Philippe Du Bois, Frédéric Dubru and Michaël Petit
Institut d’Informatique, University of Namur
Rue Grandgagnage, 21 – B-5000 Namur (Belgium)
Fax: +32 81 72.49.67 – Phone: +32 81 72.49.80
Email: {edu, pdu, fdb, mpe} @ info.fundp.ac.be

1 Introduction

In this paper, our aim is at reporting some preliminary results on the introduction of formal methods and object-orientation paradigm in the development of distributed real-time systems. More specifically, in this paper, we address the requirements phase of the system lifecycle where the customers wishes have to be captured and analysed.

Requirements Engineering (or Requirements Analysis) is now widely recognised as a critical activity in the context of information system and software development. Besides classical semi-formal languages that are in use for the modelling and the analysis of requirements (like, e.g., SADT, MERISE, SSADM, etc), two recent trends can be identified in some more recent languages designed for that purpose: on the one hand, languages based on mathematical/logical theoretical grounds are now emerging (see e.g. CIM [Bubenko80], RML [GBM86], GIST [Feather87], MAL [FP87], ERAE [DHR91] and TELOS [MBJK90]); on the other hand, the object-orientation paradigm, originally used at the design and implementation levels, is now being investigated and promoted at the requirements level (see e.g. [CY91, SM88]). Along these directions, within the framework of the Esprit ICARUS project, we have developed the
(ALBERT, Agent-oriented Language for Building and Eliciting Real-Time requirements) language which aims at the expression of (i) statements about real-world entities, (ii) performances requirements and (iii) visibility and reliability requirements. More precisely, the language is based on a specialization of the object concept with the following features [DDP93b]:

- An agent must not be considered in isolation but through its relationships with other agents. The society of agents is involved in a joint problem solving characterized by some goals to be achieved within the system.

- An agent is characterized by an internal state (by analogy with human agents, the internal state is sometimes referred as ‘mental state’). The state contains the knowledge of the agent about the external world. This knowledge is represented in terms of data which correspond to the information level.

- An agent is responsible for actions. They are characterized through the changes that they bring to its internal state. They support the description of the activities level. Moreover, there are restrictions on the agent capabilities for performing actions in given situations. These supports the description of the behavioural level.

- An agent will not act autonomously but will require cooperation (and thereby communication) with other agents. These aspects cover part of the resources level by making possible to describe commitments given by an agent to another one in terms of actions to be performed or required accesses to some piece of information.

This paper aims at an in-depth presentation of the \textit{ALBERT} language and of its use in an industrial context. In Sect.2, we briefly recall some specificities of the Requirements Engineering activity and we precise the specific characteristics of the designed language. Then, in Sect.3, the language itself is fully presented through the handling of two case studies related to Computer Integrated Manufacturing applications. On top of a language, some methodological guidance is definitively required for an incremental elaboration of a complex requirements document. In Sect.4, we briefly describe three general strategies that we have experimented. Finally, Sect.5 concludes with a brief report on the development of supporting tools for the \textit{ALBERT} language and on future researches perspectives.

2 Requirements Engineering

In this section, we briefly outline the specificity of the requirements engineering activity (RE). We conclude with the motivations underlying the \textit{ALBERT} language.

2.1 The Specificity of the Requirements Engineering activity

Since a few years, there is a large consensus in the Software and Information System (IS) communities, on the necessity to make distinct three different documents in the IS development lifecycle at the Design Engineering (DE) level (as depicted on bottom of Fig.1): (i) the software specification product where are precisely defined the behavioural properties (the
What) expected from the final code, (ii) the design product where is presented the logical design of a modules architecture describing an abstract solution for the problem described by the specification and (iii) the final code where is presented the physical design of programs.

The IS lifecycle described above covers well the activities leading from specifications to programs, but does not address the activity of obtaining the initial specifications, i.e. Requirements Engineering (RE). This activity starts from informal wishes expressed by one or several customers and elaborate a so-called requirements document where the system to be developed is defined in a precise way. In the context of the development of information systems, the requirements document should include not only specifications on the software piece to be implemented but also on the environment (made of devices, hardware, humans, etc) around this software as well as the interactions taking place between both. Such systems are sometimes referred as composite systems [Feather87] or Open Information Systems [Hewitt91].

![Figure 1: IS. Development Activities](image)

On the top of the Fig. 1, two important products of the RE phase are made distinct.

1. the goals of the system to be developed. For example, in the context of a CIM application, one may think to a goal as “when a production order is issued, a bolt has to be produced within 10 minutes”;

2. the requirements document where are described the different components of the system as well as the set of requirements attached to each individual component. For example, in the CIM example, the requirements document will identify the different components (like, e.g., a Controller, a Lathe machine, a Robot) and the way they will interact together for producing the requested bolt within the appropriate delay.

The RE products presented above are quite similar to the DE products presented before. One may consider that, on the one hand, the Goals product (at the RE level) is analogous to the Specification product at the DE level and, on the other hand, the Requirements document product (at the RE level) is analogous to the Design product (at the DE level).

To conclude and summarise Fig.1,

- the requirements document is specifying a solution for the problem expressed in terms of goals associated with the whole system;
- the code is describing a solution for the problem expressed in terms of a specification associated with the software system.
One may imagine a similar implementation lifecycle for the other components of the requirements document (e.g. the implementation of a robot). This aspect is not depicted on the figure.

2.2 About the Design of the Language

Basically, the design of the language has been done according to three directions: agent-orientation, formality and expressiveness.

2.2.1 An “Agent-Oriented” Paradigm

As a first approximation, “agent-oriented” can be understood as “object-oriented”. This means that the language meets the main OO principles, namely encapsulation of data structures and actions on them in one unit called here an agent. The word “agent” has been preferred to the one of object for the following reasons: as the requirements document is intended to describe the contractual behaviour attached with the different components of a system, we feel that the word “agent” is well appropriate for denoting components having responsibilities and perceptions within the system.

Since the requirements document is a central document resulting from an agreement between customers expressing their wishes about admissible agent behaviours and designers in charge of implementing them, we feel important to use a rigorous specification language supporting a “natural” mapping between all kinds of things of interest and the various language concepts being available. With respect to that aspect, usual OO specification languages tend to propose a too operational style (based, in most cases, on Petri nets). In \( \text{language} \), we have tried to support a more declarative style for expressing requirements.

2.2.2 Formality

\( \text{language} \) is formal. It has a formal semantics giving a precise meaning to all specifications written in this language. The existence of these rigorous rules of interpretation provides support to the analyst in its modelling task by giving him/her some hints in the application of the language constructs. Besides, rules of deductive inference are also available for supporting automatic reasoning on the written specification in order to help the analyst to analyse it (e.g., for discovering inconsistencies and incompletenesses) and to validate it (e.g., through its animation).

2.3 Expressiveness

Several formal specification languages have been proposed for the purpose of describing software components (e.g., Z, VDM, Larch). The \( \text{language} \) goes along these lines but with a greater expressiveness, i.e. a large variety of requirements can be easily encoded without the introduction of any over-specification. This variety is illustrated in the rest of this section.
Information Structures Modelling. The ERA (Entity-Relation-Attribute) model is now considered as standard for the data modelling aspect. However, in the case where the number of data is large and/or where the complexity of data is important (e.g., the plan of an aircraft), it is recognised that the ERA model suffers from a lack of basic structuring mechanisms (like, specialization, aggregation and classification mechanisms). It offers the possibility of mapping data structures in an ERA-like way but makes also possible to support more structured descriptions, expressed in terms of predefined mechanisms used for the elaboration of complex data types.

Historical Data Management. Traditional RE approaches rely on a “snapshot” view of the information state. This information state, at a time t, mirrors the real-world state of information at this time but also records information related to the past, which entails the risk of introducing over-specifications according to the way these historical information are represented. For example, in an ERA context, the mapping of a requirement like “a book can only be borrowed once by a reader” will lead to the introduction of a somewhat artificial data structure for keeping trace of the “borrowed books”.

In the language, we have chosen to represent, in an implicit way, the historical sequence of information states in order to make possible the expression of a variety of requirements referring information at different moments of time.

Effects of Actions. In some traditional approaches, actions, which alter information states, are described in an algorithmic way (pseudo-code, decision tables, etc). In contrast, adopts a “functional” characterization style where the effect of an action is expressed in terms of a mathematical relationship among two successive information states.

Causalities among Actions. Actions triggering is usually ensured through ECA (Event-Condition-Action) rules, i.e., at any moment, when an event occurs and if a condition on the current information state is met, then the action happens. Thereby, the supported specification style is an operational one since, at any moment, we need to evaluate the set of occurred events and the set of fulfilled conditions in order to determine the set of candidate actions.

In many cases, this style may lead to the introduction of over-specifications at the information state level in order to keep track of actions occurrences and of specific causalities among them. Consider a statement in a lift management system like “the push on the button at a floor is followed by the visit of the lift at this floor”. The use of an operational specification style results in a mapping of this statement where an extra piece of state information (like “pending requests”) will be introduced for recording the happening of events.

In , like in some other recent specification languages, the reactive nature of a system is expressed without any over-specification, in terms of processes (or transactions), viz. sequence of events.

Real-World Non-Determinism. Most of the existing formal specification languages have been designed with the purpose of modelling the behaviour of a software component. In particular, in such languages, the assumption is that the description of state changes implicitly defines
the system behaviour. Implicitly, it is assumed that actions occur when their preconditions are satisfied.

When dealing with the modelling of real-word things (e.g., a human behaviour), we need to introduce elements of uncertainty associated with occurrences of actions. Primitives are offered in \[\text{Alm}\] to support the modelling of deterministic happenings (things that must happen) as well as non-deterministic ones (things that may happen).

**Application Scope.** Unlike most of RE approaches focussing on the modelling of centralised business systems (i.e. non distributed systems), the scope of the \[\text{Alm}\] language is real-time cooperative systems.

The real-time aspect stems from our system modelling in terms of histories (i.e. sequences of time-stamped states and actions) allowing the expression of statements like “a book cannot be borrowed by a reader for more than 20 days” or “the push on the button at a floor is followed by the visit of the lift at this floor within the next two minutes”.

The cooperative aspect is covered through the modelling of distributed systems in terms of agents, each of them being characterized with time-varying communication/information possibilities.

### 3 \[\text{Alm}\] : the Language

In this section, the \[\text{Alm}\] language is presented. After an intuitive description of the models associated with a specification, each constructs of the language is introduced. The last part of this section is devoted to comments on a real-size case study: the Chessmen Making Shop.

**Running example**

The \[\text{Alm}\] language will be introduced using parts of a small toy example: the Bolts Manufacture.

The Bolts Manufacture is composed of production cells and a manager. Each cell has two stocks: one for rivets, one for bolts. The manager is in charge of supplying the cells with rivets. When the manager gives a production order to a cell, the cell takes a rivet from its stock and produces a bolt from it within 10 minutes. The manager is also in charge of taking bolts out of the cells stocks.

#### 3.1 Models of a Specification

The purpose of our requirements language is to define admissible behaviours of a composite system. This description, which must abstract of irrelevant details, is usually called a model of the system. A specification language is best characterized by the structure of models it is meant to describe.
The rules for deriving the set of admissible models from a given specification expressed in the formal language is beyond the scope of this paper, which will remain informal.

In order to master their complexity, models of a specification are derived at two levels:

- at the agent level: a set of possible behaviours is associated with each agent without any regard to the behaviour of the other agents;
- at the society level: interactions between agents are taken into account and lead to additional restrictions on each individual agent behaviour.

The specification describes an agent by defining a set of possible *lives* modelling all its possible behaviours. A life is an (in)finite alternate sequence of *changes* and *states*; each state is labelled by a time value which increases all along the life (see Fig.2).

![Figure 2: A possible life of the Cell agent](image)

The term "history" refers to the sequence of changes which occur in a possible life of the agent. A change is composed of several occurrences of simultaneous *actions* (the absence of action is also considered as a change). In our terminology we use the word ‘action’ both for denoting:

- happenings having an effect on the state where it occurs (called *actions* in some existing specification languages: e.g. [RFM91], [JSS91]);
- happenings with no direct influence on the state (called *events* in some other specification languages: e.g. [DHR91]).

The term “trace” refers to a sequence of states being part of a possible life of the agent. A state is structured according to the information handled in the considered application in terms of *state components*. In the case study, a specific state structure is associated with the *Cell* agent (see Sect.3.3).

The value of a state at a given time in a certain life can always be derived from the sub-history containing the changes occurred so far.

### 3.2 Language Constructs

Basically, the formal language that we propose is based on a variant of *temporal logic* [GB91], a mathematical language particularly suited for describing histories. This logic is itself an
extension of multi-sorted first order logic, still based on the concepts of variables, predicates and functions. In this paper, three extensions are taken into account:

1. the introduction of actions to overcome the well-known frame problem [BMR92], a typical problem resulting from the use of a declarative specification language;

2. the introduction of agents together with their properties (responsibilities for actions, for providing perceptions, ...). This object-oriented concept can also be seen as a possible way of structuring large specifications in terms of more finer pieces, each of them corresponding to the specification of an agent guaranteeing a part of the global behaviour of the whole system;

3. the identification of typical patterns of constraints which support the analyst in writing complex and consistent formulas. In particular, typical patterns of formulas are associated with actions.

Using the language involves two activities: (i) writing declarations introducing the vocabulary of the considered application, and (ii) expressing constraints, i.e. logical statements which identify possible behaviours of the composite system and exclude unwanted ones.

A graphical syntax (with a textual counterpart) is used to introduce declarations and to express some typical constraints frequently encountered. The expression of the other constraints is purely textual.

3.3 Declarations

3.3.1 Declaration of Agents

The declaration part of an agent consists in the description of its states structure and the list of the actions its history can be made of. Importation and exportation links between agents are also graphically described.

Agents are considered as specialized objects; therefore, our modelling of a state structure is largely inspired by recent results in O-O conceptual modelling (see, e.g., OBLOG [SSE89] and O* [Brunet91]).

The state is defined by its components which can be individuals or populations. Usually populations are sets of individuals but they can also be structured in sequences or tables. Components can be time-varying or constant. Elements of components are typed using:

- predefined elementary data types (like, STRING, BOOLEAN, INTEGER,...) equipped with their usual operations;
- elementary types defined by the analyst (like, BOLT and RIVET in our example), those are types for which no structure is given, they are only equipped with an equality predicate;

Operations on data types should not be confused with actions of agents: operations denote only mathematical functions, they may be used to simplify expressions in constraints but cannot be used to model the dynamic behaviour of systems (i.e. agents).
• more complex types built by the analyst using a set of predefined type constructors like extension\(^2\), set, list, Cartesian product,\(\ldots\) and elementary types; on top of operations inherited from their structure, new operations can be defined on these new types;

• types corresponding to agent identifiers. Agents includes a key mechanism that allows the identification of the different instances. A type is automatically associated to each class of agent. This corresponds to the type of agents identifiers within that class. E.g., each \textit{Cell} agent has an identifier of type \textit{CELL} \(^3\) \(^4\).

\[\text{Figure 3: Declaration associated with the Cell agent}\]

Figure 3 proposes the graphical diagram associated with the declaration of the state structure of the \textit{Cell} where:

- \textit{Input-stock} and \textit{Output-stock} are considered as two set populations, respectively of type \textit{RIVET} and \textit{BOLT};

- The \textit{Output-stock} is characterized by a \textit{Capacity} attribute of type \textit{INTEGER};

- \textit{Out-full} is an instance of type \textit{BOOLEAN};

- \textit{Produce}, \textit{Store-rivet}, \textit{Remove-bolt}, \textit{Remove-rivet} and \textit{Store-bolt} are five actions which may happen in a \textit{Cell} history. Actions can have arguments\(^5\); for example, each occurrence of the \textit{Store-rivet} action has an instance of type \textit{RIVET} as argument.

The wavy line between the \textit{Output-stock} and the \textit{Out-full} components expresses that the value of the latter is derived from the former. The value of a component may be derived from one or several others.

The diagram also includes graphical notations making possible to distinguish between internal and external actions and to express the visibility relationships linking the agent to the outside (Importation and Exportation mechanisms):

\(^2\)The extension type constructor (noted \(\ast\)) adds a special value \textit{"UNDEF"} to a data type. E.g., a variable of type \textit{BOLT}\(^\ast\) may take any value of type \textit{BOLT} or the special value \textit{"UNDEF"}.

\(^3\)When an agent is unique (like, e.g., the \textit{Manager} agent), then a constant is also automatically defined to refer to the identifier of that agent.

\(^4\)Inside the description of an agent, the \textit{self} constant refers to the proper identifier of the described agent.

\(^5\)Arguments may be regarded as input or output arguments but there is no difference on a semantics point of view.
(i) Information within the parallelogram is under the control of the described agent (the Cell) while information outside from the parallelogram denotes elements (state components or actions) which are imported from other agents of the society the agent belongs to. From the graphical declaration, it can be read that Cell has the initiative for Remove-rivet and Store-bolt actions while it lets the Manager having the initiative of Store-rivet and Remove-bolt actions 6;

(ii) For information in the parallelogram, boxes without arrow indicate that this information is not visible from the outside. Conversely, boxes with arrow denote information which is exported to the outside. From the graphical declaration, it can be read that the Manager may have knowledge of the Output-stock state and of the Store-bolt actions.

Importation and Exportation are static properties; Perception and Information are their dynamic counterparts and provides the analyst with a finer way of controlling how agents can see information inside each other (perception and information constraints will be discussed in Sect.3.4.3).

Figure 4: Declaration associated with the Manufacture agent

3.3.2 Declaration of a Society

Agents are grouped into societies. Societies themselves can be grouped together to form larger societies. In fact, a specification consists in a hierarchy of agents (a tree-like structure).

Figure 4 shows the declaration associated to the Manufacture.

The existing hierarchy among agents is expressed in term of two combinators: Cartesian Product and Set. In our specific case, the Manufacture agent is an aggregate of one Manager agent and several Cell agents (which form a "class").

3.4 Constraints

Constraints are used for pruning the (usually) infinite set of possible lives of an agent.

Unlike usual O-O design languages, the semantics is not operational. A life must be extensively considered before it can be classified as possible or not, i.e. adding new states and changes at the end of a possible life does not necessarily result in a possible life.

Figure 5 introduces the specification associated with the behaviour of the Cell agent and refers to the graphical declaration introduced in Fig.3.

---

6In the textual part of the specification, external actions will be referred prefixed with the identifier of the agent responsible for it
In order to provide some methodological guidance to the analyst, properties are classified under ten headings and grouped into three families: Basic Constraints, Local Constraints, and Cooperation Constraints.

The identification of these different headings and families has been an important of our work because, from large case studies performed by analysts, we have experimented that a logical language (although sufficiently expressive) cannot be easily used because such a language is too flat and does not include any methodological guidance (a few years ago, the same was true for assembly programming languages and resulted in the development of new programming language offering higher level constructs).

3.4.1 Basic Constraints

Basic constraints are used to describe the initial state of an agent and to give the derivation rules for the derived components. Those constraints are respectively put under the headers Initial Conditions and Derived Components.

On Fig.5, the derived components constraint expresses that the boolean value Out-full is true when the number of items in the output stock is equal to its capacity and is false otherwise. The initial conditions constraint asserts that, in the initial state of the cell, the output stock is empty.

3.4.2 Local Constraints

Local constraints are related to the internal behaviour of the agent. They are classified under four headings: State Behaviour, Effects of Actions, Causality and Capability.

State Behaviour. Constraints under this heading express properties of the states or properties linking states in an admissible life of an agent.

First of all, there are constraints which are true in all states of the possible traces of an agent (see the first state behaviour constraint on Fig.5 expressing that the input stock may never be empty). These constraints are written according to the usual rules of strongly typed first order logic.

On top of constraints which are true in all states (usually referred as invariants), there are constraints on the evolution of the system (like, e.g. if this property holds in this state, then it holds in all future ones) or referring states at the different times (see the second state behaviour constraint on Fig.5 expressing that a rivet may not stay indefinitely in the input stock). Writing these constraints requires to be able to refer to more than one state at a time. This is done in our language by using additional temporal connectives which are prefixing statements to be interpreted in different states. These connectives (◊, ◤, □, □, U, S) are inspired from temporal logic (see e.g. [Sernadas80, MP91]) and express respectively “sometimes in the future”, “sometimes in the past”, “always in the future”, “always in the past”, “until”, “since”. There are constraints related to the expression of real-time properties. They are needed to describe delays or time-outs (like, e.g., “an element has to be removed from its population within 15 minutes”) and are expressed by subscripting temporal connectives with a time period. This time period is made precise by using usual time units: Sec, Min, Hours, Days, ... [Koymans92].
Effects of Actions. Beyond this heading, we describe the effects of actions\(^7\) which may alter states in lives (see on Fig.5 how, e.g., an occurrence of the *Store-bolt(b)* action alters the output stock). Only actions which bring a traceable change are described here (for example, we do not describe the role of the *Produce* action).

In the description of the effect of an action, we use an implicit *frame rule* saying that states components for which no effect of actions are specified do not change their value in the state following the happening of a change.

The effect of an action is expressed in terms of a property characterising the state which follows the occurrence of the action. The value of a state component\(^8\) in the resulting state is characterised in terms of a relationship referring to (i) the action arguments, (ii) the agent responsible for this action (if this action is an external one, the name of the agent is prefixing the action) and (iii) the previous state in the history.

In the last statement of the effects clause on Fig.5, we express that the effect associated with the action *Store-rivet* issued by the external agent *Manager* is to add a rivet in the *Input-stock* of the *Cello.*

Causality. This heading is related to the *causality* relationship existing between some occurrences of actions.

Expressing causality rules with usual temporal connectives may appear very cumbersome (see, e.g., motivations given by [FS86]). To this end, our language is enriched with specific connectives which allow to specify, for example, that an action has to be issued by the agent as a unique response to the occurrence of another action (brought or not by the agent). A common pattern is based on the use of the "\(\rightarrow\)\" symbol which is not to be confused with the usual "\(\rightarrow\)" logical symbol. In our case, we want to denote some form of *entailment*, as it exists in Modal Logic [HC68].

In the case study, an example of causality exists between the *Produce*, the *Remove-rivet* and the *Store-bolt* actions. It relies upon the necessity of having a unique occurrence of the *Remove-rivet* and of the *Store-bolt* action (in that order) in response to each occurrence of the *Produce* action (see Fig.5).

The "\(\rightarrow\)" symbol can be quantified by a temporal operator to express performances constraints (e.g. the "\(\leq 10\)" symbol in the causality constraint on Fig.5 means that the occurrence of the *Store-bolt* action has to happen within a 10 minutes interval after the occurrence of the *Produce* action).

The right part of a commitment (the *reaction*) may only refer actions which are issued by the agent (i.e. actions which are not prefixed).

---

\(^7\)This heading contains constraints which describe both effects of internal actions and effects of actions perceived from the outside.

\(^8\)Please note that in an effect statement, derived components may not appear in the left part of a valuation.
Cell

BASIC CONSTRAINTS

DERIVED COMPONENTS

Out-full \triangleq \text{Card}(\text{Output-stock}) = \text{Capacity}(\text{Output-stock})

INITIAL VALUATION

Empty(\text{Output-stock})

LOCAL CONSTRAINTS

STATE BEHAVIOUR

\neg \text{Empty}(\text{Input-stock})

\text{In}(\text{Input-stock}, r) \implies \Diamond \neg \text{In}(\text{Input-stock}, r)

EFFECTS OF ACTIONS

\text{Remove-rivet}(r): \text{Input-stock} = \text{Remove}(\text{Input-stock}, r)

\text{Store-bolt}(b): \text{Output-stock} = \text{Add}(\text{Output-stock}, b)

\text{Manager.Remove-bolt}(b): \text{Output-stock} = \text{Remove}(\text{Output-stock}, b)

\text{Manager.Store-rivet}(r): \text{Input-stock} = \text{Add}(\text{Input-stock}, r)

CAUSALITY

\text{Manager.Produce} \xrightarrow{\phi_{\text{out}}} \text{Remove-rivet}(r); \text{Store-bolt}(b)

CAPABILITY

\mathcal{F} (\text{Store-bolt}(b) / \text{Out-full})

\mathcal{F} (\text{Remove-rivet}(r) / \neg \text{In}(\text{Input-stock}, r))

COOPERATION CONSTRAINTS

ACTION PERCEPTION

\mathcal{K} (\text{Manager.Store-rivet} / \text{TRUE})

\mathcal{I} (\text{Manager.Remove-bolt} / \text{Empty}(\text{Output-stock}))

\mathcal{X}\mathcal{K} (\text{Manager.Produce} / \neg \text{Empty}(\text{Input-stock}))

ACTION INFORMATION

\mathcal{K} (\text{Store-bolt}(b).\text{Manager} / \text{TRUE})

STATE INFORMATION

\mathcal{X}\mathcal{K} (\text{Output-stock}.\text{Manager} / \neg \text{Empty}(\text{Output-stock}))

Figure 5: Constraints on the Cell agent
Left and right parts of a commitment may be composed of one or more occurrences of actions. In case of more than one, occurrences may be composed in the following ways:

- "act1 ; act2" which means "an occurrence act1 followed by an occurrence act2";
- "act1 ® act2" which means "an occurrence act1 and an occurrence act2 (at the same time)";
- "act1 || act2" which means "an occurrence act1 and an occurrence act2 (in any order)"
- "act1 ⊕ act2" which means "an occurrence act1 or an occurrence act2 (exclusive or)"

Some more complex expressions are provided to express iterative application of actions.

**Capability.** Under this heading, we describe the role of the agent with respect to the occurrence of its own actions. To this end, we are still using an additional extension of the classical first-order and temporal logic by making possible to express permissions associated with an agent. To this end, we consider three specific connectives allowing the expression of obligations, preventions and exclusive obligations (respectively the O, the F and the XO connectives). The study of these connectives has been heavily influenced by some work performed in the area of Deontic Logic (see, e.g. [FM90], [Dubois91]).

The pattern for an obligation "O (<int-action> / <situation>)" expresses that the action has to occur if the circumstances expressed in the situation are matched (these circumstances refer to conditions on the current state).

The pattern for a prevention "F (<int-action> / <situation>)" expresses that the action is forbidden when the circumstances expressed in the situation are matched (e.g. the first constraint in Fig.5 expresses that "the cell cannot store a bolt into the stock when the stock is full", in other words, it is forbidden to the Cell to produce the Store-bolt action when the stock is full).

The pattern "XO (<int-action> / <situation>)" is used to express exclusive obligation, it is a shorthand for the combination of "O (<int-action> / <situation>)" and "F (<int-action> / ¬ <situation>)".

The default rule is that all actions are permitted whatever the situation.

Using these connectives makes possible to express the control that the agent has with respect to its internal actions.

### 3.4.3 Cooperation Constraints

This family of constraints specifies how the agent interacts with its environment, i.e. how it perceives action performed by other agents (Action Perception), how it can see parts of the state of other agents (State Perception), how it let other agents know what actions it does (Action Information) and how it shows parts of its state to other agents (State Information).

---

9 In actions expressions, the special action identifier OAC may be used as a dummy action: e.g. "a ; (b ⊕ OAC) ; c" is a shortcut for "(a ; b ; c) ⊕ (a ; c)".
As said previously in Sect.3.3, perception and information provide the specifier a way to add a dynamic dimension to the importation and exportation relationship between agents expressed in the declaration part of the specification.

**Action Perception.** Beyond this heading we define how the agent is sensitive to changes occurring in its environment, which are made available to it by other agents belonging to the same society.

Action perceptions are specified using the $\mathcal{K}$ (knowledge), $\mathcal{I}$ (ignorance) and $\mathcal{XK}$ (exclusive knowledge) connectives.

The pattern "$\mathcal{K} (\text{<ext-action>/<situation>})$" defines the situation where, if an action is issued by the external agent, the behaviour of the current agent is influenced. For example, the first action perception constraint expresses that "the cell is always obliged to take into account the rivet storage performed by the Manager" (in other words, *Store-rivet* actions occurring in the Manager’s life has necessarily to affect the history of the Cell).

The pattern "$\mathcal{I} (\text{<ext-action>/<situation>})$" defines the situation where, if such action is issued by the external agent, it has no influence on the current agent’s behaviour.

The pattern "$\mathcal{XK} (\text{<ext-action>/<situation>})$" is used to express exclusive obligation, it is a shorthand for the combination of "$\mathcal{K} (\text{<ext-action>/<situation>})$" and "$\mathcal{I} (\text{<external-action>/<situation>})$".

The default rule is that all imported actions available may be perceived whatever the situation.

**State Perception.** Beyond this heading we define how the agent sees parts of the state of other agents belonging to the same society and which are made available to it by them. State perceptions are also specified using the $\mathcal{K}$, $\mathcal{I}$ and $\mathcal{XK}$ connectives.

The default rule is that all imported state components available may be perceived whatever the situation.

**Action Information.** Constraints under this heading specify how occurrences of actions performed by an agent are made available to other agents belonging to the same society. This is also a dynamic property and is expressed using the $\mathcal{K}$, $\mathcal{I}$ and $\mathcal{XK}$ connectives introduced above.

The pattern "$\mathcal{K} (\text{<int-action>/.<agent>/<situation>})$" defines the situation where occurrences of an internal action are made available to a given agent. For example, the action information constraint in Fig.5 expresses that "the cell always tells the Manager when it stores bolt in the stock (in other words, *Store-bolt* actions occurring in the cell life are always visible by the Manager)".

The pattern "$\mathcal{I} (\text{<int-action>/.<agent>/<situation>})$" defines the situation where the occurrences of an internal action are not made visible for a given agent.

The pattern "$\mathcal{XK} (\text{<int-action>/.<agent>/<situation>})$" is used to express exclusive obligation, it is a shorthand for the combination of "$\mathcal{K} (\text{<int-action>/.<agent>/<situation>})$"

\footnote{Broadcast may be modelled by using here a free variable instead of an agent identifier.}
and "I ( <int-action>. <agent> /¬ <situation> )".
The default rule is that all exported actions may be visible by any agent to which it is exported, whatever the situation.

**State Information.** Beyond this heading we define how the agent shows parts of its state to other agents belonging to the same society. State information is also specified using the $K$, $I$ and $XK$ connectives.
The default rule is that all exported state components may be visible by any agent to which it is exported, whatever the situation.

### 3.5 Case Study: the Chessmen Making Shop

In this sub-section, we report on the performance of a more real-size case study that the one handled in the previous sub-section. Because the lack of place, the full specification is presented in Appendix and we only discuss some specific aspects illustrating the possibilities offered by the language.

In the first part, we completely rephrase the formal specification into informal terms. Comments on the formal specification are given in the second part.

#### 3.5.1 Informal Description

The **Chessmen Making Shop** is a small manufacturing unit in charge of producing chessmen from wooden cylinders. It is composed of production cells (automated units) and is supervised by a manager (human actor).

The shop manager gives a **production order** to a cell when a chessman has to be machined. The order specifies the type of chessman to be produced (king, queen, castle, bishop, knight or pawn) and a coded reference.

A cell is composed of a number of **machine tools**, a **stock**, a **robot**, a **clamping system**, an **auto-guided vehicle** (AGV) and a **controller** (computer).

The cylinders and end products are stored in the stock. Before they can be machined, cylinders have to be clamped on a pallet. This work is done by the clamping system. The robot is used to withdraw parts from the stock and furnish them for the clamping operation. Once clamped, parts are transported to one machine by the AGV. If additional machining is required, the AGV picks the partly-machined part and transports it to another machine (once or several times) until the desired chessman is obtained. The chessman is then transported to the clamping system to be unclamped and stored by the robot in the stock (at the location where the raw cylinder was withdrawn). All these activities are coordinated by a controller. The whole production process of a chessman takes less than 20 minutes.

The stock is composed of a number of locations identified by an address. Each location can contain at most one item (cylinder or chessman). Items can either manually be entered in or withdrawn from the stock by the shop manager or automatically by the robot. The manager is able to see if the stock is empty and an alarm is sent to him by the stock when it is full.
The robot is in charge of loading and unloading the clamping system. The grip of the robot is equipped with a sensor, so it can see which item is stored at a given location of the stock when its arm reaches that location. The same holds for the contents of the clamping system. The robot receives commands from the controller to load the clamping system with an item and commands to unload it. The commands specify the item to be transported and the address of the stock concerned by the operation, i.e. for a load (resp. unload) command, the location of the stock where the item has to be taken (resp. put).

The clamping system role is to clamp and unclamp the items it receives. It is an automatic device: it automatically clamps any item put on it by the robot and automatically unclamps any item brought by the AGV. Clamping and unclamping operations take at most one minute. The clamping system may handle one item at a time and must be unloaded between two operations. The clamping system sees what is carried by the robot and the AGV when these are located at the clamping system.

The AGV has two places where items can be put. But those places can never be occupied at the same time during transportation (there are only used to perform some exchange between an item already in the AGV and an item located on a device (machine tool or clamping system). The AGV receives commands from the controller asking it to carry one item from one location to another. In reaction to these commands, it has to move to the origin (if it was not yet there), pick up the item, move to the destination and deliver the item (if there was already an item on the destination device, an exchange must take place). The AGV sees the contents of a device (a machine tool or the clamping system) where it is located. The AGV notifies the devices when it picks up (resp. delivers) an item from (resp. to) it.

Machine tools are used to apply basic transformations to raw materials or partly-machined items. Each machine tool has some capabilities, i.e. the set of programs it is able to run. Machine tools receive commands from the controller. A command, addressed to a given machine tool, specifies the program to be run. Machine-tools notify the controller when the program they had to run has been completed. Machine tools of a cell are independent and may work simultaneously.

The controller is in charge of achieving orders it receives from the manager by issuing commands to the devices of the cell. The software running in the controller contains a “recipe book” which associate a recipe to each chessman type. A recipe consists in a sequence of production steps. Each production step is characterized by a machine and a program to be run on it. The controller keeps traces of the status of each order it is processing and the status of each component of the cell.

The communication between the controller and the machine-tools is not always reliable. In order to detect problems, the controller marks a machine as “down” when a command was sent to a machine tool and no end-of-work notification has been received within 10 minutes. The controller send then an alarm message to the manager specifying which machine appeared faulty. The controller does not use “down” machine tools anymore.

3.5.2 Comments on the Formal Specification

The formal specification corresponding to the informal description shown above is given in Appendix A. The process followed to build this specification will be presented in Sect. 4. Informal
comments are provided in the text of the specification to facilitate understanding.

In this part, interesting properties of some parts of the specification will be highlighted and discussed: (i) concurrent behaviour and non-determinism; (ii) fine-grain visibility control; (iii) reliability of agents.

**Concurrent Behaviour and Non-Determinism.** As other languages allows to describe systems in an operational way. But it also allows to give declarative properties, i.e. considering extensively the life of an agent.

Causality constraints are good examples of declarative properties. In the specification of the controller (see Sect.A.6), the causality relationship means that each production order has to be followed by a list of actions starting some subprocesses, but this does not preclude that nothing else may happen in the meantime. In fact, the controller may process several orders at the same time.

A common problem encountered in concurrent systems is deadlocks. Components of the cell are working in parallel so we could imagine deadlocks may happen. The role of requirements specification is just to state that deadlocks should be avoided but not to express solutions to avoid them (this will be done during design). The causality constraints mentioned above is sufficient to express that we do not want deadlocks: only lives with finishing order processing (thus without deadlocks) will be accepted as valid models of the specification.

**Fine-grain Visibility Control.** The visibility mechanism taking place among agents (i.e. the way they exchange information) is quite elaborate with precision visibility control:

- **static importation/exportation mechanism**
  This syntactic mechanism constrains which state components of one agent may be referred to in the specification of another one. See, e.g., the declaration associated with the robot (Sect.A.2) where it is stated that the Robot agent imports the Contents component from the Stock agent. This means that references to this component may be made in constraints describing the robot.

- **state information mechanism**
  This mechanism dynamically restricts what exported state component is effectively available and to which agent (i.e. to which instance(s)) it is made available. See, e.g., the state information constraint on the stock (Sect.A.1) which restricts the visibility of the robot on its contents to the location where the robot stands (if it is on the stock).

- **state perception mechanism**
  This last mechanism may restrict furthermore the quantity of information exchanged between agents by putting constraints on when imported components, which where made available to the agent by another, are indeed perceived (accessed) by the agent. See, e.g., the state perception constraints (Sect.A.2) which says that the robot looks at the contents of a location of the stock only when it is standing at that location.

---

11For this example, the perception constraints (in the robot specification) matches the corresponding information (in the stock specification) but this is not necessarily the case.
Those three mechanisms act as successive filters on channels carrying information between agents (see Fig.6).

Figure 6: Fine-grain visibility control

In an expression, the value of an imported components which is not made available or is not perceived, is the special value "UNDEF".

Reliability of Agents. A similar three levels mechanism is provided to deal with control flow between agents. This provides the specifier a nice way to model reliability of agents.

In the informal statement of the problem it is said that "the communication between the controller and the machine-tools is not always reliable". This is quite straightforwardly modelled in the specification of the machine-tool (see Sect.A.5):

- The first action perception constraint defines some situation where the Run action from the controller is not seen but nothing is said about other situations. The default rule is that it may be seen (exactly what happen through an unreliable communication channel).

- No action information constraint is given about the Work-done action which is exported from the machine tool to the controller. Here again, the default rule will apply and this means that occurrences of the Work-done action may be made visible to the controller.

This way, communication (from a control flow point of view) between the controller and the machine tool is modelled as unreliable in both directions.

As for visibility of state components, constructs here act as successive filters on the control flow and an agent has always "the last word" on control information coming from the outside.

4 Methodological Guidelines at the RE Level

In the RE field as well as in other fields, a language is not usable for real applications if it is not supported by an appropriate and effective methodology. The requirements document, in its final version, is usually a complex document due to the number of individual agents belonging to the system and the complexity of interactions taking place among agents. Therefore, one cannot imagine that the requirements document can be written in one shot and it is found essential to provide some methodological guidance in the elaboration of successive and incremental versions of the requirements document.
As depicted in Fig. 7, the elaboration of a specification can be seen as a sequence of development steps, each step being defined by the application of transformations on the current version of the requirements document and resulting in a new version of this document. The application of a specific transformation at some stage of the development depends on some strategies followed by the analyst. In the rest of this section, we briefly discuss and illustrate two possible elaboration strategies based on a progressive refinement of the specification. Another strategy has been described in [DDP93a] and consist in the possibility of reusing generic specification components.

![Figure 7: Example of RE process](image)

### 4.1 A "Goal-oriented" Strategy

The main idea behind this strategy is based on a top-down agent refinement philosophy [Bjorner92] [Dubois89]. Tackling a new problem, the analyst will have to isolate and specify the main goals of a system first, before to refine them progressively in terms of finer requirements that can be attached to finer subsystems. To be more precise, the strategy is:

1. Identify the goals of the system and specify them in terms of a unique monolithic agent.
2. Identify new sub-agents and attach to each of them their individual responsibilities so that the behaviour of the sub-agents preserves the original goals.
3. Apply recursively the step 2 on each agent up to the identification of 'terminal' agents, i.e. components for which designers agree on the implementation of their attached responsibilities.

![Figure 8: Application of the "Goal-oriented" Strategy](image)
Within the framework of the *Chessmen Making Shop* system introduced in the previous section, the *goal-oriented* strategy has been applied to result in the second version of the requirements document (see Fig.7):

- In the first version, the identification of the problem has resulted in the specification of a system made of two agents (the *Ceil* and the *Manager*). At this stage of elaboration, the *Ceil* is considered as an individual agent for which the given specification states that "a production order received by the cell should cause the delivery of an appropriated manufactured cylinder within a 20 minutes delay". This specification is considered as the *goal* of the system.

- In the second version, the application of the goal strategy defined above has resulted in a new version of the requirements document where the original *Ceil* agent has been refined in terms of six finer agents (*Stock, Robot, Clamping-sys, Agv, Mach-tool and Ctrl*, see Fig.8).

### 4.2 A "Retracting-Assumptions" Strategy

In version 2 of the requirements document, the six agents being components of the *Ceil* have been identified. However, due to the complexity of their individual descriptions, the analyst has imagined an idealized version of them which liberates him/her from too many specification details in the requirements document. In other words, the work of the analyst was based on some assumptions related to the behaviours of the involved agents. In the proposed strategy (inspired by the pioneering work of [Feather89]), we consider two kinds of possible assumptions:

- The first assumption is based on an *omniscience* hypothesis i.e. the perfect knowledge an agent has of the states and actions of other agents.

- The second assumption is based on a *reliability* hypothesis associated with each agent, i.e. its responsibility for performing the appropriate actions under the given circumstances.

The proposed strategy is the following one:

1. Specify a system where: (i) the internal state and actions of each agent are visible from each other and (ii) all agents are considered reliable.

2. Elaborate a new specification by retracting the first assumption, viz: (i) the internal state of each agent is not necessarily visible from the outside and (ii) all agents are still reliable.

3. Elaborate the final specification of the system by retracting the reliability assumption associated with each agent.

In our example, the specification, contained in the version 2 of the requirements document, has been built by the analyst with these two assumptions in mind. This means, for example, that the controller can access to the *status* of the clamping system (*omniscience hypothesis*) and that the machine tool is reliable w.r.t. its communication with the controller (*reliability hypothesis*).

The application of the step 2 of the proposed strategy resulted in a new version of the requirements document (version 3). At this level, the controller has now to maintain information
recording the status of the clamping system. This information aims at mirroring the real status of the clamping system but this duplication is needed since the controller has no longer access to the state of the clamping system and thereby has to derive its status from the control commands delivered to it.

Finally, in version 4 of the requirement document, the reliability assumption attached to agents has been removed (this version is the one fully presented in Appendix A). This, again, has the consequence to add complexity to the requirements document by having to consider and incorporate new details to deal with failures, as an example one may refer to the handling of failure in the communication between the machine tool and the controller.

5 Conclusion

Multiple experiences have shown that an engineering approach such as cannot be effectively adopted by analysts if it is not supported by tools. In the long term, we want to build an integrated requirements engineering environment for . For the moment, two basic tools are made available:

1. **Editing facilities** are offered through the use of a structural editor that will support ‘graphical’ as well as ‘textual’ syntaxes. The graphical part of this editor is already implemented using the *GraphTalk* tool which allows to manage multiple views of the same requirements specification (several figures presented in this paper have been edited using this tool).

2. **Validation facilities** are offered through an animator tool which can be used for the purpose of verifying the adequacy of formal requirements specifications with respect to what customers really want to do. Discussions around a set of logical formulae cannot be an acceptable basis and thereby an animator can be used for testing dynamically the possible behaviours of the described system (a classical prototyping approach could not be applied here due to the non-executable nature of ALBERT). More about the animator can be found in [DDD94].

Within short term, we have some plan to develop checking facilities developed on the basis of the formal interpretation rules associated with the language.

Two other researches directions are related to:

- at the practical level, the use of in some other specific application domains. Preliminary investigations are performed in domain of telecommunications, Electronic Data Interchange and office automation (workflows);

- at the method level, we are working on the elaboration of formalised methodological guidelines for writing requirements documents as well as for integrating other kinds of requirements like organizational and non-functional requirements (see, e.g., [Yu93],[BCDS93]).

---

12 *GraphTalk* is a registered trade mark of Rank Xerox
References


A ALBERT Specification of the Chessmen Making Shop (detailed version)

A.1 The Stock

![Diagram of Stock]

**BASIC CONSTRAINTS**

**DERIVED COMPONENTS**

- Empty $\triangleq$ Card(Contents) = 0
- Full $\triangleq$ Card(Contents) = Capacity(Contents)

**INITIAL VALUATION**

Contents[$.] = UNDEF

**LOCAL CONSTRAINTS**

**STATE BEHAVIOUR**

Card(Contents) $\leq$ Capacity(Contents)

**EFFECTS OF ACTIONS**

- Manager.Store(it,c): Contents = Add(Contents,it,c)
- Manager.Remove(_,c): Contents = Remove(Contents,c)
- Robot.Put(it,c): Contents = Add(Contents,it,c)
- Robot.Get(_,c): Contents = Remove(Contents,c)

**CAPABILITY**

$\bigcirc$ (Alarm-Full / Full)

**COOPERATION CONSTRAINTS**

**ACTION PERCEPTION**
\[ \forall \forall ( \text{Manager.Store}(\_, c) / \text{Contents}[c] = \text{UNDEF} \land \text{Full}) \]
\[ \forall \forall ( \text{Manager.Remove}(\text{it}, c) / \text{Contents}[c] = \text{it}) \]
\[ \forall \forall ( \text{Robot.Put}(\_, c) / \text{Contents}[c] = \text{UNDEF} \land \ldots, \text{Full}) \]
\[ \forall \forall ( \text{Robot.Get}(\text{it}, c) / \text{Contents}[c] = \text{it}) \]

STATE PERCEPTION

The stock always perceives the location of the robot.
\( \forall \forall (\text{Robot.Location} / \text{TRUE}) \)

ACTION INFORMATION

\( \forall \forall (\text{Alarm-Full.Manager} / \text{TRUE}) \)

STATE INFORMATION

The stock shows its contents, at a given address, to the robot if and only if the robot is located at that address.
\( \forall \forall (\text{Contents}[i]. \text{Robot} / \text{Robot.Location} = i) \)
\( \forall \forall (\text{Empty.Manager} / \text{TRUE}) \)

A.2 The Robot

**Robot**

**BASIC CONSTRAINTS**

**INITIAL VALUATION**

\( \text{Grip} = \text{UNDEF} \)
\( \text{Busy} = \text{FALSE} \)

**LOCAL CONSTRAINTS**

**EFFECTS OF ACTIONS**

\( \text{Get}(\text{it}_+) / \text{Grip} = \text{it} \)
\( \text{Put}(\_, -) / \text{Grip} = \text{UNDEF} \)
\( \text{Busy} = \text{FALSE} \)
\( \text{Goto}(_, -) / \text{Location} = \text{UNDEF} \)
\( \text{Arrive}(c) / \text{Location} = c \)
\( \text{Ctrl.Load-Clamp}(\_, _) / \text{Busy} = \text{TRUE} \)
\( \text{Ctrl.Unload-Clamp}(\_, _) / \text{Busy} = \text{TRUE} \)

**CAUSALITY**
Ctrl.Load-Clamp(.,e)  
\[ \begin{cases} 
\text{a. The robot is not yet located at the right address.} \\
(Goto(c) ; Arrive(c)) \oplus DAC \\
\text{b. The robot is already located at the right address.} \\
Get(i,c) ; Goto(Clamping-sys) ; Arrive(Clamping-sys) ; Put(i,Clamping-sys) 
\end{cases} \]

Ctrl.Unload-Clamp(.,e)  
\[ \begin{cases} 
\text{a. The robot is not yet located at the clamping system.} \\
(Goto(Clamping-sys) ; Arrive(Clamping-sys)) \oplus DAC \\
\text{b. The robot is already located at the clamping system.} \\
Get(i,Clamping-sys) ; Goto(c) ; Arrive(c) ; Put(i,c) 
\end{cases} \]

CAPABILITY
\[ \begin{align*} 
F( & \text{Goto(c) / Location = c)} \\
F( & \text{Get(i,Clamping-sys) / Location } \neq \text{ Clamping-sys } \lor \text{ Grip } \neq \text{ UNDEF } \lor \text{ Clamping-sys.Contents } \neq \text{ it) } \\
F( & \text{Put(it,Clamping-sys) / Location } \neq \text{ Clamping-sys } \lor \text{ Grip } \neq \text{ it } \lor \text{ Clamping-sys.Contents } \neq \text{ UNDEF }) \\
F( & \text{Put(it,c) with Is-of.ST-ADRS(c) / Location } \neq \text{ c } \lor \text{ Grip } \neq \text{ it } \lor \text{ Stock.Contents[c] } \neq \text{ UNDEF }) \\
F( & \text{Get(it,Clamping-sys) / Location } \neq \text{ Clamping-sys } \lor \text{ Grip } \neq \text{ it } \lor \text{ Stock.Contents[c] } \neq \text{ UNDEF }) \\
\end{align*} \]

COOPERATION CONSTRAINTS

ACTION PERCEPTION
\[ \begin{align*} 
\mathcal{X}K( & \text{Ctrl.Load-Clamp(.,_)/ } \neg \text{ Busy) } \\
\mathcal{X}K( & \text{Ctrl.Unload-Clamp(.,_)/ } \neg \text{ Busy) } \\
\end{align*} \]

STATE PERCEPTION
\[ \begin{align*} 
\mathcal{X}K( & \text{Stock.Contents[i] / Location = i) } \\
\mathcal{X}K( & \text{Clamping-sys.Contents / Location = Clamping-sys) } \\
\end{align*} \]

ACTION INFORMATION
\[ \begin{align*} 
\mathcal{X}K( & \text{Get(.,Clamping-sys / Location = Clamping-sys) } \\
\mathcal{X}K( & \text{Get(.,Stock / Is-of.ST-ADRS(Location))] } \\
\mathcal{X}K( & \text{Put(.,Clamping-sys / Location = Clamping-sys) } \\
\mathcal{X}K( & \text{Put(.,Stock / Is-of.ST-ADRS(Location))] } \\
\end{align*} \]

STATE INFORMATION
\[ \begin{align*} 
\text{The robot shows its location to the stock when it is on one of the addresses of} \\
\mathcal{X}K( & \text{Location.Stock / Is-of.ST-ADRS(Location)] } \\
\mathcal{X}K( & \text{Location.Clamping-sys / Location = Clamping-sys) } \\
\end{align*} \]

A.3 The Clamping System
The clamping system is an automatic device: it clamps (resp. unclamps) any item delivered by the robot (resp. a.g.v.) without waiting for any signal from the controller.

**BASIC CONSTRAINTS**

**INITIAL VALUATION**

Contents = UNDEF  
Clamped = FALSE

**LOCAL CONSTRAINTS**

**EFFECTS OF ACTIONS**

Clamping-done: Clamped = TRUE  
Unclamping-done: Clamped = FALSE  
Agv.Deliver(it,..): Contents = it  
Clamped = TRUE  
Agv.Pick-up(it,..): Contents = UNDEF  
Robot.Put(it,Clamping-sys): Contents = it  
Clamped = FALSE  
Robot.Get(it,Clamping-sys): Contents = UNDEF

**CAUSALITY**

Clamping and unclamping take at most $1'$.

Agv.Deliver(_,..) $\preceq_{st'}$ Unclamping-done  
Robot.Put(_,Clamping-sys) $\preceq_{st'}$ Clamping-done

**CAPABILITY**

$\mathcal{F}$ ( Clamping-done/ Contents = UNDEF )  
$\mathcal{F}$ ( Unclamping-done/ Contents = UNDEF )

**COOPERATION CONSTRAINTS**

**ACTION PERCEPTION**

$\mathcal{XK}$ ( Agv.Deliver(_,..)/ Contents = UNDEF )  
$\mathcal{XK}$ ( Agv.Pick-up(it,..)/ Contents = it )  
$\mathcal{XK}$ ( Robot.Put(_,..)/ Contents = UNDEF )  
$\mathcal{XK}$ ( Robot.Get(it,..)/ Contents = it )

**STATE PERCEPTION**

$\mathcal{K}$ ( Robot.Location / TRUE )  
$\mathcal{K}$ ( Agv.Location / TRUE )

**ACTION INFORMATION**

$\mathcal{K}$ ( Clamping-done.Ctrl/ TRUE )  
$\mathcal{K}$ ( Unclamping-done.Ctrl/ TRUE )

**STATE INFORMATION**

$\mathcal{XK}$ ( Contents.Robot / Robot.Location = SELF )  
$\mathcal{XK}$ ( Contents.Agv / Agv.Location = SELF )
A.4 The Auto-Guided Vehicule

BASIC CONSTRAINTS

INITIAL VALUATION

\( \text{Place1} = \text{UNDEF} \)
\( \text{Place2} = \text{UNDEF} \)
\( \text{Busy} = \text{FALSE} \)

LOCAL CONSTRAINTS

STATE BEHAVIOUR

\( \text{Location} = \text{UNDEF} \Rightarrow \text{place1} = \text{UNDEF} \lor \text{place2} = \text{UNDEF} \)

EFFECTS OF ACTIONS

\( \text{Pick-up(it,1)}: \text{Place1} = \text{it} \)
\( \text{Pick-up(it,2)}: \text{Place2} = \text{it} \)
\( \text{Deliver(it,1,x)}: \text{Place1} = \text{UNDEF} \)
\( \quad \text{Place2} = \text{x} \)
\( \quad \text{Busy} = \text{FALSE} \)
\( \text{Deliver(it,2,x)}: \text{Place1} = \text{x} \)
\( \quad \text{Place2} = \text{UNDEF} \)
\( \quad \text{Busy} = \text{FALSE} \)

\( \text{Arrive(c)}: \text{Location} = \text{c} \)
\( \quad \text{Busy} = \text{FALSE} \)

\( \text{Goto(\_)}: \text{Location} = \text{UNDEF} \)
\( \text{Ctrl.Carry(\_\_,\_,\_,\_\_\_\_\_\_)}: \text{Busy} = \text{TRUE} \)

CAUSALITY

\( \text{Goto(c)} \xrightarrow{\text{\_\_\_}} \text{Arrive(c)} \)
\( \text{Ctrl.Carry(it,1,1,2)} \xrightarrow{\text{\_\_\_}} (((\text{Goto(c1)} \land \text{Arrive(c1)})) \land \text{DAC}) \lor \text{Pick-up(it,1)} \land \text{DAC}) \)
\( \quad \text{Goto(c2)} \land \text{Arrive(c2)} \land \text{Deliver(it,\_,\_)} \)

CAPABILITY

\( \mathcal{F} (\text{Goto}(c)/\text{Location} = c) \)
\( \mathcal{F} (\text{Pick-up(it,1)}/\text{Place1} \neq \text{UNDEF}) \)
\( \mathcal{F} (\text{Pick-up(it,2)}/\text{Place2} \neq \text{UNDEF}) \)
\( \mathcal{F} (\text{Pick-up(it,\_)}/\text{Location} = \text{UNDEF} \lor \text{Location}.\text{Contents} \neq \text{it}) \)
\( \mathcal{F} (\text{Deliver(it,1,\_)}/\text{Place1} \neq \text{it}) \)
\( \mathcal{F} (\text{Deliver(it,2,\_)}/\text{Place2} \neq \text{it}) \)
\( \mathcal{F} (\text{Deliver(it,\_,x)}/\text{Location} = \text{UNDEF} \lor \text{Location}.\text{Contents} \neq \text{x}) \)
COOPERATION CONSTRAINTS

ACTION PERCEPTION

$\forall K (\text{Claim}(\text{Carry}(-,-,-,\_))/ \neg \text{Busy})$

STATE PERCEPTION

$\forall K (\text{Contents} / \text{Location} \neq \text{UNDEF} \land \text{Location} = a)$

ACTION INFORMATION

$\forall K (\text{Pick-up}(\_,-,-,\_)/ \text{Location} = a)$

$\forall K (\text{Deliver}(\_,-,-,\_)/ \text{Location} = a \lor a = \text{Ctrl})$

STATE INFORMATION

$\forall K (\text{Location} \_ / \text{Location} \neq \text{UNDEF} \land \text{Location} = a)$

A.5 The Machine Tool

BASIC CONSTRAINTS

INITIAL VALUATION

Contents = UNDEF
Busy = FALSE

LOCAL CONSTRAINTS

EFFECTS OF ACTIONS

Work-done: Busy = FALSE
Contents = new
Ctrl.Run(\_,-,-,-): Busy = TRUE
Agv.Deliver(it,-,-,-): Contents = it
Agv.Pick-up(-,-,-,-): Contents = UNDEF

CAUSALITY

ctrl.Run(\_,-,-,-)\rightarrow Work-done

CAPABILITY

$F (\text{Work-done} / \text{Contents} = \text{UNDEF})$
COOPERATION CONSTRAINTS

ACTION PERCEPTION

\[ T (\text{Ctrl.Run}(m,\text{pgm}) / m \neq \text{SELF} \lor \text{Contents} = \text{UNDEF} \lor \text{Busy} \lor \neg \text{In(Capabilities,pgm)}) \]

\[ \mathcal{X} \mathcal{X} (\text{Agv.Deliver}(_,x) / \neg \text{Busy} \land \text{Contents} = x) \]

\[ \mathcal{X} \mathcal{X} (\text{Agv.Pick-up}(it,_) / \text{Contents} = \text{it} \land \neg \text{Busy}) \]

STATE PERCEPTION

\[ \mathcal{X} (\text{Agv.Location} / \text{TRUE}) \]

STATE INFORMATION

\[ \mathcal{X} \mathcal{X} (\text{Contents.Agv} / \text{Agv.Location} = \text{SELF}) \]

A.6 The Controller

![Diagram of control system]

BASIC CONSTRAINTS

INITIAL VALUATION

Free-cyl = {}

Orders[.] = \text{UNDEF}

Status[.] = \text{Free}

Proc[.] = \text{UNDEF}

Agv-ord = \text{UNDEF}

LOCAL CONSTRAINTS

STATE BEHAVIOUR

Orders[ref] \neq \text{UNDEF} \Rightarrow q_{\leq 20'} \hspace{1cm} \text{Orders[ref]} = \text{UNDEF}

\[ \neg =_{10'} \hspace{1cm} \text{Status[m]} = \text{Working} \Rightarrow \text{Status[m]} = \text{Down} \]

EFFECTS OF ACTIONS

Load-clamp(ref,c):

\[ \text{Proc[Clamping-sys]} = \text{ref} \]

\[ \text{Loc(Orders[ref])} = \text{Clamping-sys} \]

Free-cyl = \text{Remove(Free-cyl,c)}
Carry(ref,_,c2): Loc(Orders[ref]) = Agv
  Status[c2] = Booked
  Agv-ord = ref
Run(m,\_): Status[m] = Working
  Tdl(Orders[Proc[m]]) = Tail(Tdl(Orders[Proc[m]]))
Alarm(m): Status[m] = Down
Unload-clamp(ref,\_): Orders[ref] = UNDEF
Manager.Store(_,c): Free-cyl = Add(Free-cyl,c)
Manager.Remove(_,c): Free-cyl = Remove(Free-cyl,c)
Manager.Produce(_,ref): Dest(Orders[ref]) = Clamping-sys
  Loc(Orders[ref]) = Stock
  Tdl(Orders[ref]) = Recipes(t)
Clamping-sys.Clamping-done: Status[Clamping-sys] = Done
  Loc(Orders[Proc[Clamping-sys]]) = Clamping-sys
  Dest(Orders[Proc[Clamping-sys]]) = Mach(Head(Tdl(Orders[Proc[Clamping-sys]])))
Agv.Deliver(_,\_): Status[Loc(Orders[Agv-ord])] = Free
  Proc[Loc(Orders[Agv-ord])] = UNDEF
  Proc[Dest(Orders[Agv-ord])] = Agv-ord
  Agv-ord = UNDEF
m.Work-done with Tdl(Orders[Proc[m]]) = [ ]: Status[m] = Done
  Dest(Orders[Proc[m]]) = Clamping-sys
m.Work-done with Tdl(Orders[Proc[m]]) \[ ]: Status[m] = Done
  Dest(Orders[Proc[m]]) = Mach(Head(Tdl(Orders[Proc[m]])))
Clamping-sys.Unclamping-done: Status[Clamping-sys] = Done
  Proc[Clamping-sys] = UNDEF

CAUSALITY
Manager.Produce(_,ref) \[= 0\]
  Load-clamp(ref,c)
  Carry(_,Clamping-sys,Mach(Recipes[t][1]),)
  \forall i: 2 \leq i \leq \text{Length}(
  Recipes[t])
    \{ Run(Mach(Recipes[t][i-1]),
      Prog(Recipes[t][i-1]));
    Carry(_,Mach(Recipes[t][i-1]),Mach(Recipes[t][i]));
  \}
  Run(Mach(Recipes[t][\text{Length}(Recipes[t])]),
      Prog(Recipes[t][\text{Length}(Recipes[t])]),Clamping-sys);
  Carry(_,Mach(Recipes[t][\text{Length}(Recipes[t])]),Clamping-sys);
  Unload-clamp(ref,c)

CAPABILITY
\{ Load-clamp(ref,c) / \neg \text{In}(Free-cyl,c) \vee Loc(Orders[ref]) \neq \text{Stock}
  \vee \text{Dest}(Orders[ref]) \neq \text{Clamping-sys} \vee \text{Status}[Clamping-sys] \neq \text{Free} \}
\{ Carry(_,c1,c2) / \neg \text{Proc}[c1] \neq ref \vee c1 = c2 \vee Loc(Orders[ref]) \neq c2 \vee \text{Dest}(Orders[ref]) \neq c2 \vee \text{Status}[c1] \neq \text{Done}
  \vee \text{In}(\{\text{Working},\text{Booked},\text{Down}\}) \vee \text{Status}[c2] \vee \text{Agv-ord} \neq \text{UNDEF} \}
\{ Run(m,pgm) / \text{Dest}(Orders[Proc[m]]) \neq m \vee \text{Loc}(Orders[Proc[m]]) \neq m
  \vee \text{Proc}[\text{Head}(Tdl(Orders[Proc[m]]))] \neq pgm \vee \text{Status}[m] \neq \text{Booked} \}
\{ Alarm(m) / \text{Status}[m] \neq \text{Working} \}
\{ Unload-clamp(_,\_)/ \text{Loc}(Orders[ref]) \neq \text{Clamping-sys} \vee \text{Dest}(Orders[ref]) \neq \text{Clamping-sys}
  \vee \text{Proc}[\text{Clamping-sys}] \neq ref \vee \text{Status}[\text{Clamping-sys}] \neq \text{Booked} \}

COOPERATION CONSTRAINTS

ACTION PERCEPTION
\{ Manager.Remove(p) / \text{Type}(p) = \text{Cyl} \}
$\forall K \left( \text{Manager.Store}(p) / \text{Type}(p) = \text{Cyl} \right)$

$\forall K \left( \text{Manager.Produce}(t, \text{ref}) / t \neq \text{Cyl} \land \text{Free-cyl} \neq \{\} \land \text{Orders}[\text{ref}] = \text{UNDEF} \right)$

$K \left( \text{Clamping-sys.Clampping-done}/ \text{TRUE} \right)$

$K \left( \text{Agv.Deliver}(\_,\_)/ \text{TRUE} \right)$

$K \left( \_\_\_\_.\text{Work-done} / \text{TRUE} \right)$

$K \left( \text{Clamping-sys.Unclamping-done}/ \text{TRUE} \right)$

$\forall K \left( \text{Robot.Put}(\_,\_)/ \text{Status}[\text{Clamping-sys}] = \text{Done} \right)$

**ACTION INFORMATION**

$O \left( \text{Load-clamp}(\_,\_)/ \text{Robot} / \text{TRUE} \right)$

$O \left( \text{Unload-clamp}(\_,\_)/ \text{Robot} / \text{TRUE} \right)$

$O \left( \text{Carry}(\_,\_,\_)/ \text{Agv} / \text{TRUE} \right)$

$XO \left( \text{Run}(m_1,\_)/ m_2 = m_1 \right)$

$O \left( \text{Alarm}(\_)/ \text{Manager} / \text{TRUE} \right)$

**STATE INFORMATION**

$O \left( \text{Orders}/ \text{Manager} / \text{TRUE} \right)$
'A HARD CORE FOR SOFT PROBLEMS'
A Business Engineering Case Study within the Amsterdam Municipal Police Force.

J.W. van Meel / P.W.G. Bots / H.G. Sol
Delft University of Technology
School of Systems Engineering, Policy Analysis and Management
P.O. Box 5015 NL-2600 GA Delft The Netherlands
Tel: 31 15 78 7231 / 2948 / 2318 Fax: 31 15 78 4811
E-mail: {vanmeel,bots,sol}@sepa.tudelft.nl

Abstract

In this paper, a case study with Business Engineering (BE) is described within the Amsterdam Municipal Police Force. A short review shows that literature on BE-methods, new organizational forms, organization design and information systems design offers little methodological support for BE. However, integrating aspects from different bodies of literature seems to be a promising prospect. Integration in a purely deductive way is most difficult to realize. Therefore, an exploratory case study has been carried out instead. The methods and techniques used during the case study, and the final results have been evaluated. The result is an outline of an integrated simulation approach consisting of: a SocioTechnical, engineering way of thinking, a problem solving way of working, an adaptive way of controlling and a dynamic way of modelling. The proper fit between these different approaches is examined in detail.

Keywords: Business Engineering, Organizational Impacts of IT, Social and Behavioral Sciences, Simulation Support Systems
1 Introduction

The Amsterdam Municipal Police Force is facing serious problems. In the Netherlands, crime is becoming a problem of major importance. Within the last fifty years, crime rates have actually been multiplied by ten (Department of Justice 85). From recent governmental research, it appears that every inhabitant of Amsterdam of 15 years or older has a 50% chance annually to fall victim to one or more criminal offenses (Department of Home Affairs 91). Thus, Amsterdam has one of the highest crime rates amongst European cities. Note however, that in Holland 'criminal offence' covers anything from bicycle thefts to acts of violence. Regardless of this mitigation, the Amsterdam Municipal Police has an arduous task.

To improve the performance of the Amsterdam Police Force, a large-scale reorganization was set in motion in the late eighties which is still ongoing today. Information technology is regarded as one of the building blocks of the reorganization process. Within the force, numerous plans have been formulated for restructuring the current business and information processes of the force. To judge their effectiveness, their impact on the performance of the force must be established. However, actually experimenting with police departments can be expensive and full of risk. Since the last reorganization was carried out only recently, the force's management is reluctant in announcing additional changes. The business processes and the information processes themselves are rather complex and unpredictable: most activities are carried out on a basis of experience and professionalism, and are coordinated through mutual adjustment. Few work procedures are formalized. Combined, these reasons make the restructuring of the Amsterdam Police Force a fuzzy, 'soft' problem: complex, difficult to tackle with both technical and social components.

According to the force's management, crime is not a problem of the police alone: it is becoming a problem of major social importance. This has created a special interest in cooperation with universities and other scientific institutions. In particular, the School of Systems Engineering and Policy Analysis has been asked to investigate the possibilities and impossibilities of changing and restructuring police work.

A short literature review reveals that the Amsterdam Municipal Police Force is not the only organization in the midst of a restructuring process. To survive, many organizations need to transform their business more frequently than they used to do. Business processes of these organizations are becoming more and more information intensive, and yet they are seldom designed with the possibilities of new information technology in mind. It could be fruitful to transform technology and user environment in an integral way instead of what is usually done: holding one constant and changing the other. Such integral design of both organizational structures and processes and information systems is called 'business engineering' (BE) in this paper.

The aim of this paper is to define a 'hard core' of an integrated design approach for business engineering. In section 1, the notion of BE is further elaborated and a short literature review into BE and related fields is presented. In section 2, a first research framework is described for further investigation into BE in a real life problem situation. In section 3, the results of the application of this research framework in a case study within the Amsterdam Police Force is described. In section 4, the case study results are evaluated and combined with the results of the literature review, resulting in an outline of an
integrated design approach for BE. In section 5, conclusions are drawn and implications are given for further research.

2 Business Engineering

2.1 Why transformation?

Organizations in our society are facing serious problems (Hammer 90). Most of the structures, processes and technologies used within organizations today stem from the industrial age. The design principles of Frederick Taylor have long been the favourite guidelines of American industry (Davenport and Short 90). The functional-differentiated hierarchy has long been the preferred organization model when manufacturing was predominant and the business environment was stable. With the help of new technologies, manufacturers were able to expand by improving the productivity of their employees dramatically (Drucker 91).

In our globally networked society, the assumptions and guidelines of the industrial age no longer hold. The boundaries in our environment are subject to change due to global transitions - changing in type and geography (Sol 92). The organizational environment is no longer stable, but characterized by increasing complexity, hostility and turbulence (Huber 84). 'Competitive pressures' such as global competition, risk, service and costs force major organizations to change (Rockart and Short 89). Rapid technological change, as well as shifting patterns of international trade and competition, have put intense strain on many organizations (Miles and Snow 86). Information technology in particular offers organizations new possibilities, and yet the experience with the application of this kind of technology appears to be rather paradoxical (Sol 92).

Manufacturing is no longer predominant in our globally networked society. In 1990, the service industry accounted for 75% of all jobs in the private economy. Over the past decade, the U.S. service industry has created nearly 20 million new jobs (Roach 91). Unlike manufacturing, products devised by the service industry have a high information content. Hence, an important part of the business processes of a service organization involves information processing (Dur 92). However, despite enormous capital investments in information technology (Davenport and Short 90), non-manufacturing productivity (a good proxy for productivity in the service sector) has come to a virtual standstill since the mid-seventies (Baily and Chakrabarti 89, Brynjolfsson 93, Drucker 91, Roach 91, Thurow 89). Providing more and better information does not automatically result in better organizational performance (Sol 92). Not surprisingly, Drucker (1991) states that 'the single greatest challenge facing managers in the developed countries is to raise the productivity of knowledge and service workers in organizations'. Illustrative in this sense is that substantial gains are made especially by organizations that succeed in integrating services within an entire chain rather than in streamlining subprocesses. Typical examples can be found in transport, health care, financial and telecom services (Davidson 93, Short 92, Smith and McKeen 93, Sol 92).
2.2 Organization and information: where is the rub?

Organizations can be seen as purposeful systems, justified in their existence by the function they perform in their environment (Ackoff and Emery 72). Organizations with structures, processes and technologies that are well suited to their environment are more likely to survive than poorly adapted ones (Burns and Stalker 61, Lawrence and Lorsch 67). The problems of organizations in our globally networked society indicate a mismatch between internal structure and external conditions. In other words, organizations need to transform their business in order to survive.

Business literature abounds with all kinds of solutions for organizational transformation. Similar to consumer products, the concepts offered seem to have a life-cycle of their own: they grow, mature, age and fade away (Vantrappen 92). Likewise, with organizations becoming more and more dependent on information systems, the information systems literature has become manifest as a buzzwords generator in its own right (Sol 85).

At first sight, the concepts of both bodies of literature seem to have some kind of relationship. An adhocracy organization, for example, seems to be best supported by advanced 'Computer Supported Cooperative Work systems', while Electronic Data Interchange 'seems to fit well with 'Just-In-Time-management'. Different organizational structures seem to require different technologies and vice versa. Over time, various relationships between organizations and IT have been suggested, especially in implementation literature, ranging from 'the use of IT leads to increased centralization' to 'the use of IT is dependent on existing power structures'. Empirical evidence for these views is both conflicting and confusing (Attewel and Rule 84, George and King 91, Markus 88, Miller et al. 91, Raymond et al. 93). George and King (1991) therefore suggest a more neutral relationship of mutual reinforcement.

This interpretation explains the differences in empirical evidence and shows why generalizations about possible structural relationships are difficult to obtain by evaluation of the use of IT in organizations. Instead of looking ex-post, it can be more fruitful to look at the use of IT in organizations ex-ante. The question is not whether there exists a synergetic relationship, but rather how such relationship can be established (Sol 92). IT and organizational structures should be examined not from an emergent perspective, but from a design point of view. This perception of the role of IT in organizations has been labelled 'business engineering' (BE).

2.3 What is business engineering?

'Business engineering' (BE) has become popular both in the business and information systems literature to denote organizational transformation. Some remarkable successes with the BE concept are reported in literature (Davenport and Short 90, Grover 93, Hammer 90, Kaplan an Muirdock 91). There is no consensus yet on its definition (King 91, Smith and McKeen 90), or even its name: 'business process engineering' (Hammer 90), 'the new industrial engineering' (Davenport and Short 90), 'core process redesign' (Kaplan and Muirdock 91), and 'business process redesign' (Grover 93) seem to be as many names for what simply can be called 'working smarter' (Drucker 91). A common characteristic of BE literature is the focus on the design of information technology (IT) and organizational processes and structures in an integral way (Davenport and Short 90, Dennis 93, Fried 91,
Grover 93, Hammer 90, Kaplan and Muirdock 91, Vogel et al. 93). Bots and Sol (1988) distinguish three perspectives with respect to the design of organizations and their technical information structures:

- **the macro-perspective**, concentrating on the cooperation between and above different organizations and on supporting and enabling technical infrastructures
- **the meso-perspective**, concentrating on coordinating activities that take place within the boundaries of an organization and on supporting and enabling technical architectures;
- **the micro-perspective**, concentrating on the primary business processes that are performed at the workplace level, typically by an individual or a small group, and on enabling and supporting work stations.

**figure 1 Positioning of Business Engineering**

These design perspectives can be used to better position the fuzzy concept of 'business engineering'. Depending on the perspective, three different forms of organizational transformation can be distinguished (Sol 92): infrastructure engineering (macro), process engineering (meso), and task engineering (micro), see figure 1. *Business engineering* is the combination of these three forms. *Infrastructure engineering* deals with strategic choices about partners, markets and products and has always been part of the job of executive management. *Task engineering* deals with job design and screen lay-out and has been the job of industrial engineers and system analysts for years. In particular, *process engineering* is truly innovative, for a number of reasons.

Organizations are seldom structured with the possibilities of new information technologies in mind (Huber and McDaniel 86, Strassman 88). Most of the business structures nowadays have not been designed at all, but have emerged over time (Hammer 90). IT has mainly been used for automation of the resulting structures and processes, and seldom or
not with new business structures in mind (Applegate et al. 88, Chen and Nunamaker 89, Davenport and Short 90, Robey 83). Infrastructure engineering is ineffective without process engineering (Streng 92); for successful implementation of strategic choices, business processes need to be redesigned and restructured (Davenport 93). Therefore, business engineering hinges on the process dimension. Business engineering focuses on changing both technology and user environment in an integral way instead of holding one constant and changing the other. This makes business engineering a hard to tackle ‘soft’ problem, with both social and technical components.

### 2.4 A design methodology: what to look for?

Business engineering (BE) is characterized by 'radical change' (Heygate and Brebach 92) that is 'cross-functional' (Hammer 90) and 'IT-enabled' (Smith and McKeen 93). Mostly within short time frames, complex design decisions must be made about entire business and technical infrastructures. For successful implementation, personnel, financial, and information policies of an organization need to be aligned. BE can thus have a great effect on the functioning of an organization as a whole (Hammer 90, Kaplan and Muirdock 91), and it is not without risks (Smith and McKeen 93). BE must therefore be carefully managed and planned (Heygate and Brebach 92). What in fact is needed is a 'good-and-sound' design methodology for guiding a BE process.

Before looking into possible design methods, concepts, tools and techniques for BE, the notion of a 'good-and-sound' design methodology must be further defined. A design methodology can intuitively be defined as a coherent set of activities, guidelines and techniques that can structure, guide and improve complex design processes. With respect to design processes, Simon (1973) states that 'one of the most important uses of a computer is to model complex situations and to infer the consequences of alternative decisions to overcome bounded human rationality'. So, modelling and automated support is crucial for a design methodology. The analytical framework of Seligman et al. (Seligmann et al. 89) has therefore been used for sharpening the intuitive definition as it pays explicit attention to these aspects. According to this framework, a design methodology is characterized by a way of thinking, controlling, working, and modelling as presented in figure 2. Preferably, these "ways" are supported by a coherent set of automated tools (a designers' environment or workbench).

A way of thinking determines the underlying philosophy or 'weltanschauung' of a design methodology. It can be seen as a delineation of the problem domain the methodology addresses (here: BE), possibly supported with a design theory (here: a coherent set of concepts and principles for designing organizations). The way of thinking is often overshadowed by, or implicit in, the techniques and methods embedded in the methodology. These hidden assumptions are, however, of great influence on the ultimate appropriateness of methodologies. The way of controlling deals with the managerial aspects of the methodology. It includes planning and feedback and determines in which ways various persons and groups should interact. The way of working consists of the tasks of the methodology which must be performed. Synonyms for tasks in this context are steps, phases, activities or actions. The way of modelling consists of the design products or intermediate results of the methodology. This will be mainly models of the resulting system in terms of conceptual models or specific model formalisms.
The original analytical framework has successfully been used for understanding the structure of a number of IS methodologies (Verhoef 93, Wijers 91). The framework can, however, also be used in a more prescriptive way. Following the configuration hypothesis of Mintzberg (1979a), a 'good and sound' design methodology is a methodology which explicitly addresses all five dimensions of the framework and has a 'good and consistent fit' between them.

![Diagram of the analytical framework: a design methodology and its support](image)

**Figure 2** The analytical framework: a design methodology and its support

### 2.5 A brief literature review into BE and related fields

The normative definition of a 'good-and-sound' design methodology has been used for analyzing and understanding recent BE literature and literature from related fields. A brief literature review has been carried out into design methods, concepts, tools and techniques for supporting BE design processes. This literature review has consisted of four steps. First, four bodies of literature have been selected for further investigation:

- **BE methods:** literature on how to apply BE, often with examples;
- **new organizational forms:** literature on new IT-enabled organizational forms;
- **organization design:** literature on division of labour and coordination;
- **information systems design:** literature on designing IT applications.

The first two bodies of literature can be seen as complementary, since literature on 'new organizational forms' actually describes possible results of the application of a 'BE
method'. The last two bodies of literature can also be seen as complementary, since organization design and information systems design each cover one side of the definition of BE.

Next, for the publications considered relevant, each was condensed into a one-page summary to obtain a general overview. Third, the analytical framework for understanding as described in the previous section has been applied to gain a better understanding. A matrix has been set up with the dimensions of the framework on the X-axis and the different literature categories on the Y-axis. The cells of the matrix contain summary results and key words (the result can be found in the appendix). Finally, the matrix has been used for further analysis to determine (partial) support for a 'good-and-sound' design methodology for BE. Detailed results of the literature analysis are given in (Meel 94), but some general conclusions about the four categories can be made here.

BE methods found in literature roughly follow the pattern of general problem solving, but pay little attention to explaining why this specific method would be more beneficial than other methods. Suggested new organizational forms have often common characteristics: they are flat, project- and team-oriented, and make use of advanced information technology, especially communication technology. Both literature on BE methods and literature on new organizational forms appear to be rather fuzzy and not well elaborated, although both offer intriguing results and promising viewpoints. The literature on organization design offers a rich set of design theories and modelling concepts, but pays less attention to the procedural aspects of design. The literature on information systems design offers a rich set of methods, modelling formalisms, techniques and tools, but pays less attention to design theories. Most striking is the fact that neither organization design literature nor information systems design literature pay much attention to the possibilities of integral organization and information systems design.

It appears that BE obtains little methodological support. However, an optimist can see the field of BE as an integrating research area for organization and information systems design. By using modelling formalisms and support tools from information system design and design theories and modelling concepts from organization design in a problem solving approach, actual business process redesign efforts can be structured, guided and supported. Since it would be hard to integrate these admittedly equivocal aspects in a strictly deductive way, an exploratory case study has been carried out. Before starting, a rough research framework has been defined.

3 Defining a research framework

Explorative research does not release the researcher from formulating an explicit conceptual framework before starting his study. A rough (not rigorous) research framework ('working frame' in Miles 79) needs always to be defined before the field can be entered (Checkland 91, Miles 79, Mintzberg 79b, Yin 89). For the exploratory case study reported in this paper, a 'problem solving perspective' which relies strongly on the use of 'dynamic models' has been chosen.
3.1 Problem solving

Business engineering greatly influences the functioning of organizations. Personnel, financial, and information policies need to be aligned for successful change. The effects of organizational change need to be thoroughly explored in advance to legitimate such drastic changes, regardless of the complexity of the organizations involved. Costs, loss of production and safety risk, however, generally prohibit the experimentation with alternatives in reality. According to Sol (Sol 82), it can be fruitful to use a systematic, model-based problem solving approach for this kind of design problem. Based on the model cycle of Mitroff et al. (1974), Sol (1982) argues that problem solving can be seen as a model cycle, consisting of a number of steps and products, as shown in figure 3.

A problem situation is a situation in which a decision maker, or a group of decision makers, is dissatisfied with the current situation, has alternative courses of action available, can make choices which have a significant effect, and has some doubt as to which alternative to select (Ackoff 81, Dur 92). Note that this definition does not imply that all alternatives are known in advance (Bots 89). The first activity of the problem solving cycle is the creation of a mental model of the current problem situation, which can broadly express the actual problem situation and potential solutions. Once formulated, this model is referred to as 'conceptual model'. The second activity is the more detailed specification of the problem situation, necessary to obtain an empirical model of the problem area. This results in a descriptive empirical model of the problem situation. The third activity is to analyze the problem situation and generate solutions for it, using the empirical model as a substitute for the real problem situation. Solutions are translated in terms of the empirical model, transforming the descriptive empirical 'as is' into a prescriptive empirical model 'as could be' or experimental model. Comparing the effects of the various experimental models results in a set of tentative solutions, which can be implemented in the problem situation. Provided that the empirical model 'as is' corresponds closely to the problem situation, the experimental model provides reasonably
accurate information concerning the effects of a specific solution, should it be implemented. This implies that the descriptive empirical model must resemble the problem area closely and that this correspondence must be tested empirically.

3.2 Dynamic models

Models of organizations resembling real problem situations can be very complex. First, organizations can be modelled from many different perspectives, depending on the viewpoints within the organization. Second, depending on the perspective, a great number of sequences of activities within the organization can be modelled. Third, most of these sequences are dependent on time or status and most of them take place concurrently. Models are needed which can have different levels of abstraction, consisting of parallel, concurrent sequences of activities and which can incorporate an explicit time dimension. Models with these characteristics are defined as 'dynamic models' (Dur and Bots 92).

For these dynamic models, a number of description forms do exist, like (coloured) Petri-nets, Causal Models, Event Network Models and Action Diagrams (Sol and van Hee 91, Sol and Crosslin 92, Sol and Verbraeck 92). However, the basic building blocks of these formalisms do have a high level of abstraction. Application of these formalisms to real life problem situations results in rather abstract representations and tends to be difficult to comprehend for those who are not experts in these techniques. For complex problem situations, the resulting models tend to become very large. It can be very difficult to comprehend such a dynamic model, thus impeding its validation (Dur 92). A dynamic modelling formalism that tries to overcome these problems is task/actor modelling (Bots 89, Dur 92).

Task/actor modelling offers building blocks that do resemble actual phenomena within organizations, and yet still have the appropriate features for modelling dynamic aspects of business processes. The basic modelling constructs of task/actor modelling are 'tasks', 'actors' and 'items'. A task/actor model consists of a number of actors which can perform a number of tasks on distinct items in order to achieve a certain goal. To incorporate dynamic aspects, the notion of an 'agenda mechanism' is used. Each actor has his own agenda which tells him which tasks he needs to perform. When a task occurs, it is put on the agenda of the actor; upon finishing a task, an actor selects a new task from his agenda. Actors and items are represented by object definitions; the sequencing and coordination between tasks is represented by task structures.

Object definitions and task structures can be transformed quite easily into an executable discrete event simulation model. Animation is used to visualize the dynamic aspects of the modelled business process by showing status changes of the model over time (Verbraeck and de Vreede 93, Wierda 91). Within the problem solving cycle of figure 3, the animated discrete event simulation model is equivalent to the empirical model. For the correspondence test between the dynamic simulation model and the problem area, structural and replicative validation can be used (Dur 92). Structural validation is performed to test whether the model behaves in special situations consistently with the expectations of the designer. Replicative validation is performed to test the degree of similarity between the numerical output of the simulation model and the corresponding values in reality (Kleijnen 92).
3.3 The experience

Dynamic modelling in a problem solving context as described in the previous sections has been used and refined in a number of research projects (Sol 92), for example in designing interorganizational information systems for international rail transport of cargo (Wierda 91), in the development of an information system for treasury management and in-house banking (Motshagen 91), in the application of fleet management information systems with mobile data communication (Schrijver and Sol 91), in a project supporting investment decisions on application of EDI in the port of Rotterdam (Streng 92), and in evaluating the effects of introducing new information technology into administrative organizations (Dur 92). Each study strongly focused on modelling of complex organizational processes to evaluate in advance the effects of introducing information technology in organizations. These findings have made clear that dynamic modelling can be of value for supporting business engineering in real life problem situations. Therefore, dynamic modelling is used as a research framework for the study as described in this paper.

4 The case of the Amsterdam Municipal Police Force

As stated in the introduction, crime is becoming a social problem of major importance in the Netherlands. To improve the performance of the Police Force, a large scale reorganization has been set in motion in the late eighties which is still going on today.

4.1 The problem situation

Within the force, lots of ideas do exist for improving its performance. Strategic reports contain discussions about topics such as 'innovation', 'prevention', 'the police as an enterprise', and 'the police manager as entrepreneur'. Keywords in the current situation are 'decentralisation', 'management by objectives', 'project work' and 'information technology'. At the operational level, so called 'neighbourhood teams' are created: small, decentralized units responsible for general police work in small geographical areas. This implies that the teams have to deal with all matters in their neighbourhood themselves, from the first call for service by the public to handing someone over to the public prosecutor. Within the neighbourhood teams, 'management by objectives' has been introduced. The top management of the force annually defines 'force objectives': problem areas on which the activities of the force must be focused. Based on these objectives, the neighbourhood teams must set up a number of 'project plans', in which the teams define how they expect to reach these objectives. Projects are meant to open up opportunities for more preventive action, and should become the main activity of each neighbourhood team.

Information technology is regarded as one of the essential building blocks of the change process. The Amsterdam police force has developed a multi-user transaction processing system to facilitate the paper work within neighbourhood teams. This system is organized around a central database, which can be used at a remote level. The force's former central registration and information department is no longer needed. In the past, a management reporting tool has been created for providing figures about incidents and crime rates of a neighbourhood as whole. This reporting tool has a lot of shortcomings. Most of the data definitions of the tool are out of date. It has no interface to the force's central transaction processing system, so it must be fed with data manually. Since it is not very motivating to
input the same data twice, the reporting system contains a lot of inaccurate and outdated information. However, the main problem is that the system can deal with 'traditional' police data concerning incidents and accidents, but hardly with data concerning projects. No information could be generated about individual projects. According to the users, the reporting tool did not fit in the project-oriented neighbourhood teams anymore.

This problem with relevant, correct and accurate management information for neighbourhood teams was the immediate cause for the Amsterdam Municipal Police Force to authorize a first study. The assignment was to investigate the possibilities of management information systems for supporting project work. To gain insight in the business process of neighbourhood teams, dynamic modelling was used.

4.2 Conceptualization

The first step of the case study was the conceptualization of the problem area under study. The actual problem area has been defined as the daily activities of the neighbourhood teams of the Amsterdam Police Force. The Force has twenty-four neighbourhood teams, each consisting of sixty to eighty officers. Each team has the same organizational structure: each neighbourhood team has its own chief, assisted by a staff officer, and consists of eight crews of six to eight officers and one chief. Crews work in shifts and have their own schedule, defining starts and ends of the crew shifts. There are two types of shifts: during the 'assistance shift', a crew is responsible for car patrol and answering questions from the public at the desk of the neighbourhood team. During the 'prevention shift', a crew performs tasks like paper work and project activities; these tasks are mostly performed in the office. For the documentation of the results of the conceptual phase, object definitions as described by Bots (1989) were used.

4.3 Specification

Based on the results of the conceptualization, a first dynamic simulation model has been developed for the activities of one of the neighbourhood teams. With this model, it could be shown that neighbourhood teams spent just 15% of their total daily capacity on project work. The main activities of a neighbourhood team appeared to be car patrol and handling the resulting paper work. Given this observation, one can wonder whether a new MIS would contribute to a better functioning of the entire organizational chain. The objective of the case study was therefore reformulated from MIS design to looking for ways to increase the time spent on execution of project work. Based on the results of this first prototype dynamic model, a more sophisticated dynamic model has been developed for three of the twenty four neighbourhood teams of the force (for a detailed description of this model, see Meel 94).

A modelled neighbourhood team consists of a number of crews, each with its own schedule. During a scheduled shift, each police officer is responsible for a specific set of tasks. During car patrol the officers respond to requests for assistance; when on desk duty, an officer helps citizens at the desk of the neighbourhood team; when on office duty, an officer performs administrative tasks, project activities, and other tasks, such as

---

Note that this reformulation of problem and goals during the specification phase is quite common for this approach. The empirical model serves well in avoiding type III errors (solving the wrong problem).
investigation, maintenance and looking after the neighbourhood team dog. Tasks within the model are triggered by work supply. Calls for service are generated by citizens and received by the operator in the communication centre. The operator asks the citizen a few questions to decide whether police action is required. If so, he assigns the call a priority ranging from 1 to 3 and sends it to the operator who communicates with the patrol cars. If no patrol car is available, the communication operator can decide, based on the call’s priority, its location, and an additional set of rules, whether it is necessary to interrupt a patrol car, to let the neighbourhood team start a new car patrol team or to ignore the call. The communication operator is therefore able to change the capacity of the number of police officers in the model. A call that has been handled by a car patrol team can result in paper work for the neighbourhood team. Other work supply types in the model are project activities, citizens at the desk of the neighbourhood team and 'secondary', which is an aggregation of activities which are too insignificant to be individually incorporated in the model. A work supply generator is used for modelling that piece of the problem situation which influences the amounts of work supply, but which is not direct relevant according to the problem statement.

To be able to simulate the activities of a neighbourhood team, data such as inter-arrival times, amounts of paper works, and absence figures has been analyzed. The results of these analyses have been added to the model by means of theoretical and empirical random distributions. Much attention has been paid to data with respect to car patrol as this is the main activity of the neighbourhood teams. For want of aggregated, directly usable information, raw data from the registration system of the communication centre has been analyzed. First, the numbers of incoming calls have been analyzed for the three neighbourhood teams involved. Per year, per week and per day, these numbers are rather stable, but they show strong hourly fluctuations, mainly depending on the period of the day. Within the data, seven periods have been distinguished with similar inter-arrival characteristics. The inter-arrival time per period has been modelled by an exponential random distribution. Other random distributions related to car patrol have appeared to depend mainly on the accident type of incoming calls. Accident types are represented by 120 different codes, in the registration system of the communication centre. These codes range from accidents such as 'traffic accidents' and 'noise disturbance' to more exotic ones like 'ghost driver', 'unleashed pitbull' and 'exhibitionist'. By grouping codes with similar characteristics such as service time, inter-arrival times and definition, the 120 codes were aggregated to 33 code groups. Goodness of fit between these exogenous model data and real data has been verified with the Chi-square test and the Kolmochorov-Smirnov test (Meel 94).

4.4 Correspondence check

To test its correspondence with the problem area, the dynamic simulation model was validated by means of structural and replicative validation.

The animation features of SIMAN/Cinema (Pegden et al. 90) were used for the structural validation. In SIMAN/Cinema, graphical animation screens can be defined, representing the internal status of the model. This results in a movie-like animation of the behaviour of the model at run time. An example screen dump of the animation is given in figure 4, showing the business processes of one of the modelled neighbourhood team. The structural validation was carried out with the help of the police officers of the neighbourhood teams.
involved. The developed model, and especially the graphical representation of the model, has turned out to be very recognizable.

Samples from the time registration system of the neighbourhood teams involved were used for the replicative validation. Discrepancies between model and sample results were tested with the t-test for comparison. Both the structural and the replicative validation indicated an acceptable correspondence between the dynamic model and the problem area.

\[
\text{Wijkteam 1} \quad \checkmark
\]

\[
\text{Dag: 1} \\
\text{Tijd: 11 2}
\]

\[
\text{Ovr: 6} \\
\text{Wpl: 0}
\]

\[
\text{Adm: 1} \\
\text{Koffie: 0}
\]

**figure 4** Business process of a modelled neighbourhood team

### 4.5 Solution finding

Several experiments were carried out with the developed dynamic simulation model. The results were analyzed and discussed with members of the police staff.

Based on conversations with police officers, our own findings, the results of a task group of the police force, and an analysis of the dynamic simulation model, twenty-five potential change alternatives were identified. Among them were alternatives such as the employment of more police officers, the development of faster computer systems, and different procedures for car patrol. After the identification phase, the alternatives were translated in terms of the dynamic simulation model of the problem situation.

Contrary to police expectations, some of the obvious alternatives had no consequences for the time spent on projects. Alternatives such as the introduction of new and faster information technology for paper work, or more rigorous procedures within the communication centre, have no structural effects on the time spent on projects. The optimization of subprocesses does not seem to lead to a better functioning of the police
organization. The deployment of more police officers in the model does not lead to a proportional increase in time spent on execution of projects either. This phenomenon appeared to be caused by detachments to other departments or illness. A third of the police officers of neighbourhood teams is structurally not available for daily operations. Rather than employing more police officers, alternatives should aim at decreasing this structural absence. The greatest effect is achieved by the removal of the division of labour between car patrol and office duties. This implies that car patrol officers are added to the officers doing office work while remaining available for requests for assistance. Between service requests, car patrol officers consequently work in the office. This construction does not automatically result in the expected negative effects such as an increased number of ignored calls or interrupted activities of patrol teams.

Based on the above results, a mix of alternatives is suggested for the three neighbourhood teams examined:

- handling of requests for assistance by office based teams;
- reducing the percentage of agents placed in other departments from 7% to 3%;
- reducing the percentage of agents absent by illness or leave from 22% to 17%;
- reducing the percentage of time spent on 'remaining' tasks from 19% to 14%.

These alternatives lead to a combined increase in the time spent on execution projects from 15% to 40%. For the total capacity of 156 police officers, this implies up to 40 extra officers available daily for projects for the three neighbourhood teams involved. Extrapolated to the total capacity of the neighbourhood teams of the force, this increase implies a saving of 8% on the force's annual budget.

4.6 Implementation

When the alternatives are to be implemented, project work will become the main activity of neighbourhood teams. However, this change in focus has severe implications. First, the tasks of the neighbourhood team chief need to change. Presently, most of the time of the team chief is taken up by day-to-day operational problems and human resource management. However, when project work becomes a major part of the daily routine, different skills are needed. The team chief must be able to plan projects in advance, to raise funds for the projects of his team within and outside the force, to manage large, time-consuming projects and to evaluate the impact of the projects on the safety in the neighbourhood.

Second, the problems with the current management reporting tool will become more urgent. At the moment, the Amsterdam Municipal Police Force is developing an automated crime analysis system. This expert system can create graphical and tabular representations of the crime rates in a certain geographical area and can analyze trends. However, the management information tools developed by the teams for their own usage are not sophisticated at all. With the aid of standard software packages, small reporting systems have been developed for answering simple management questions like 'How much personnel capacity is available the coming month?', 'Which projects are currently running and what is their status?', and 'How many incidents have been handled by the neighbourhood team?'. Despite technical and procedural shortcomings, the functionality of the reporting systems seems to be appropriate for the team chiefs and the information is
actually being used by them. What is needed, in fact, for the near future are not sophisticated, narrow focused crime analysis systems but flexible reporting tools which can give information about the business processes as a whole.

4.7 Evaluation

The results of the case study were presented to the management and two task forces of the Amsterdam Municipal Police Force. The low percentage of time spent on project work turned out to be an eye opener. The results showed a gap between the vision of top management and actual daily routines. The presentations formed a basis for a critical discussion about the original reorganization objectives, implementation and current business processes. The alternatives given and arguments provided proved to be acceptable. In particular, the introduction of flexible response-teams and the reduction of the structural absence were seen as valuable improvements. The fact that the results not only indicated these effective directions, but also the many ineffective directions for business engineering, was regarded as one of the major merits of the case study.

5 Reflection

Although the case study results are promising, the question whether the methods and techniques used form a 'good and sound' design methodology can not be answered right away. Therefore, the approach used in the case is analyzed in further detail and combined with the results of the literature survey as described in section 2.

5.1 Way of thinking

The way of thinking consists of a delineation of the problem domain and a design theory. A delineation of the 'soft' problem of BE has been given in section 1.

As a starting point in formulating a design theory, the SocioTechnical design paradigm seems to be quite appropriate for BE because it distinguishes explicitly between a social and a technical subsystem and advocates a joint optimization of them. This paradigm can be closely related to the positioning of BE between organization design and information systems design in section 1. It is also interesting to notice that the SocioTechnical approach is the only school of thought which has components both in organization and IS design literature (cf. the matrix in the appendix). Furthermore, many SocioTechnical design concepts can be found in descriptions of new IT-enabled organizational forms, see f.i. (Lewin and Stephens 93, Ostroff and Smith 92, Stalk 92).

SocioTechnics offers a wide range of design principles (Eijnatten 93). Four of them fit the case results:

- in complex and turbulent environments, the organization as a whole needs to be decomposed in a number of parallel operating subunits which are responsible for one complete business process in the chain from supplier to consumer;
- these subunits ought to operate as semi-autonomous teams and must be able to carry out their daily routines quite independently from other subunits;
management structures should be designed to support general decision making, giving semi-autonomous teams self-control over the budgets, equipment, personnel and information they use;
management information should not give highly specific information about a limited set of aspects, but ought to give generalized and aggregated information about a business process as a whole.

Together, these four basic principles constitute a global (though limited) design theory for flexible, information intensive service organizations in complex, hostile environments.

5.2 Way of controlling

The case study was carried out in close cooperation with police officers from various departments. Dynamic models were developed in an incremental way by means of prototyping. A control strategy which seems to fit most with that of the case study is an adaptive strategy (Keen 80). This strategy sees a design project as an adaptive process of learning for both consultants and stakeholders. Consultants can learn from the stakeholders about current operations, bottlenecks in current functioning and possible improvements. Stakeholders can learn from consultants about design methodologies, techniques and tools. For effective learning, a close cooperation between stakeholders and consultants is needed.

Design projects must be incremental: a design project must aim at quick delivery of an initial model to give stakeholders the opportunity to respond and thus clarify what they really want. The initial model needs to be further developed by means of a number of iterations. At the start of a design project, organizational and informational problems are often fuzzy and ill-structured, and final objectives are rarely concise. The project plan should be very open-ended at the start of a design project, but after each completed step, the project plan needs to be adjusted and discussed.

To reduce complexity, design projects must also be middle out: rather than trying to grasp the complete organization at once, take a single unit or process and start from there.

5.3 Way of working

The way of working in the case study was driven by actual problems in a real world problem situation. As stated in section 1, this problem orientation has been found in several other articles on BE. A problem solving way of working differs from the traditional steps undertaken in IS design because it focuses not only on information but also on organizational problems. Organization design literature offers far less support for a way of working. (Laboratory) experiments as advocated in some Human Relations literature are not always possible. Diagnostic contingency studies are difficult to relate to real business needs. However, it is interesting to notice that although the way of thinking of the Classical School of Management seems to have become obsolete, their 'scientific' way of working, focusing explicitly on real business problems, is currently achieving more recognition (Davenport and Short 90). SocioTechnics advocates a problem oriented way of working as well.
5.4 Way of modelling

In the case study, the neighbourhood team’s business process proved to be very hard to model. Although some aspects of the force are regulated by law, police officers are professionals and develop their own procedures. Reasons for structuring at the operational level are seldom made explicit and most of them have evolved over time. Police officers use their own ‘slang’ to communicate with each other and it takes some time to understand their language. Most business processes are not scheduled, but depend on time and status and are very dynamic.

The vocabulary for describing business processes offered by organization design literature is rather abstract. Relationships between concepts are mostly linear and seldom take interactions into account, see f.i. (Schoonhoven 81). Organization design seems to lack a realistic vocabulary for describing functional, material and dynamic relationships. In information systems design, business processes are expressed by means of modelling formalisms such as entity relationship diagrams and data flow diagrams. However, these modelling techniques can only represent static aspects. A major exception is formed by the object oriented approaches which model business processes with a number of concurrent interacting objects. However, object oriented modelling techniques focus more on the creation of object oriented program structures than on analyzing and diagnosing business processes. Dynamic modelling is a modelling approach which elaborates the object oriented modelling paradigm in a form more suitable for business engineering. In the case study, a special form of dynamic modelling, task/actor modelling has been used (Dur 92).

5.5 Automated support

For the realization of an executable simulation model in the case study, the simulation environment SIMAN/CINEMA was used (Pegden et al. 90). This environment offers a general purpose simulation language and very good real-time animation features. In SIMAN/Cinema, graphical animation screens can be defined, which can represent the internal status of a simulation model. At run time, this results in a movie-like animation of the behaviour of the model.

Organization design literature does not pay much attention to support tools, this in contrast with IS design literature. However, Data Base Management Systems, Data Dictionaries and 4th generation languages can be very useful for implementation, but less for analysis and design. CASE-tools can be very helpful for creating and checking of graphical, static representations of business processes, but are of less help for representing dynamics aspects and experimentations with alternative designs. Discrete event simulation workbenches do not have these deficiencies with respect to analysis and design. However, they are less sophisticated in the model management area.

5.6 Looking for fits

The results of the preceding sections are summarized in figure 5. To be able to speak of a ‘good and sound’ design methodology in terms of section 2, the fit between the different ways must be examined.
The way of working and the way of modelling show a good fit. The problem solving way of working emphasizes the use of models in a design process. The modelling representation techniques (objects, task structures, animated simulation models) differ in level of abstraction but can be translated into each other in a flexible way. The output of one step can be used as input for a following step. The model construction proceeds from low to high complexity, which accounts for the middle-out control strategy. Multiple iterations of (parts of) the problem solving cycle are needed to construct such models, which fits the adaptive, incremental control strategy.

The way of thinking is characterized by an engineering perspective (as opposed to an emergent perspective). This fits with the adaptive and yet controlled problem solving process and the heavy usage of descriptive and prescriptive modelling. The modelling constructs used offer sufficient conceptual freedom to represent both technical and organizational aspects of the problem domain. The way of thinking is also characterized by a SocioTechnical perspective which is made explicit in a number of design principles. These principles can give guidance in the transformation of the descriptive model into prescriptive models during the solution finding phase. Given a descriptive model, the number of options for changing and combining model objects is exponential and the number of possible solutions can easily get out of hand. SocioTechnical design principles can indicate where to start looking for improvements. Besides design principles, the SocioTechnical school of thought also advocates a way of working which fits with the
general problem solving process, and an adaptive way of controlling which emphasizes a close cooperation between consultants and stakeholders.

Discrete event simulation workbenches and languages effectively support the way of working, modelling and controlling. The adaptive way of controlling emphasizes learning, both for consultants and stakeholders; learning implies experimentation. With respect to field experiments, simulation offers optimal control over experimental conditions. Animation offers possibilities to present understandable models to stakeholders who are mostly not familiar with modelling techniques in general and dynamic modelling techniques in particular (Verbraeck and de Vreede 93). With an animated simulation model, the correspondence of the empirical model with the problem domain at hand can be checked both structurally and quantitatively (Dur 92). Problems can be analyzed by means of statistical techniques or by closely tracing animated model behaviour. Solutions can be translated in terms of changes in model parameters and model structure. In fact, simulation methodology itself fits well into the general problem solving cycle (Sol 82). It is also interesting to notice that simulation languages like SIMULA have given an important impulse to the current object oriented approaches. In simulation languages, lots of elegant and sophisticated modelling constructs can be found to represent complex, dynamic entities such as organizations (Bots 89, Dur 92).

Finally, if looked upon from a researcher's, rather than a practitioner’s point of view, the combination of design theories and executable, animated discrete event simulation models can be a very beneficial one. Experimenting with design theories in actual problem settings is often impeded by the high level of abstraction of the theories at hand and seldom feasible. However, a discrete event simulation model, enhanced with animation, can give a concrete form to these design theories. Conversely, simulation makes it possible to test design theories in actual problem situations in an experimental setting.

6 Conclusions and further research

Business Engineering is becoming a popular word for integral design of both organizational structures and processes and information systems. Although interesting and stimulating experiences with BE are reported in literature, the concept of Business Engineering is still immature. There are lots of ideas, partial theories, and design principles, but a solid framework does not exist, let alone a methodology for Business Engineering. Therefore, an exploratory case study has been carried out with BE within the Amsterdam Municipal Police Force. A selection from conceptual work and practical experience served well as a research framework for this case study. The results of the case study have shown that a problem solving approach to BE using dynamic modelling is not only practical, but also congruent with SocioTechnical principles for organization design, and adaptive, incremental design. In other words, the research reported in this paper indicates that partial theories from organization and information systems design literature combined with empirical findings from case studies can be forged into a hard core for a systematic approach to solving ’soft’ BE problems.

Dynamic modelling alone is not sufficient for such types of problems. The focus of existing literature in this area is especially on the modelling dimension. Modelling formalisms and automated support for representing complex and dynamic problem
situations have been elaborated but far less attention is paid to aspects other than modelling. With respect to the way of working, the focus is primarily on the task 'problem conceptualization' and 'model specification'. Other tasks and control aspects receive far less attention. Integration of dynamic modelling with elements from fields such as sociotechnical design, soft systems methodology, and information systems design can compensate for this modelling bias. Further research will focus on elaborating the systematic approach as outlined in section 5 into in more detail. A second case study is currently underway at the Criminal Investigation Department of the Amsterdam Municipal Police Force to sharpen and fine tune the approach (see also Meel 94).

In parallel to case study research, new support tools are being developed to facilitate the use of the design approach. Until now, the way of supporting lacks tools and techniques for experimenting with large and complex dynamic models. As business processes can be organized in many ways, the number of possible alternatives, and therefore, the number of possible experiments is almost infinite. Computers and analysts, however, are scarce resources. Tools and techniques are needed to structure and plan the experimentation phase to get a satisfactory cost/benefit ratio. Meel and Aris (1993a, 1993b) show how support tools for regression based experimental design can lead to an efficient and effective experimenting phase.

The way of supporting also lacks tools and techniques for working with groups and group modelling. During the case study, an appeal was made for the participation of a great number of functionaries of the police force in interviews, validation sessions, and presentations. Although the application of these techniques resulted in valuable information, it also implied a heavy work load, both for researcher and the client organization. Various iterations were needed to obtain proper data, especially when the data from various sources was conflicting. De Vreede (1993) shows how a group decision room can be used to speed up the dynamic modelling process and enhance stakeholder participation.

Combination of the outcomes of all these research efforts should be aimed at creating a set of refutable protective belts around the 'hard core' as described in this paper. Hard core and protective belts form what Lakatos would call a research programme into the development of a structured, systematic approach for solving 'soft' BE-problems. The agenda for this research programme is ambitious, challenging and - ironically - open ended.

References

ATTEWELL, P. AND J. RULE, 'Computing and Organizations: What we know and what we don't know', *Communications of the ACM*, vol. 27, nr. 12, December 1984, p. 1184-1192


BOOCH, G. *Object Oriented Design with applications*, Benjamin Cummings, Redwood City, 1991


EMERY, F.E., AND M. EMERY, *Participative design*, Canberra: Centre for Continuing Education, Australian National University, 1974


FRIED, L., 'A blueprint for Change', in: *Computerworld*, 2 december 1991, p. 94-95


IBM CORPORATION, DATA PROCESSING DIVISION, HIPO - A design Aid and Documentation Technique: IBM Corporation Manual No. GC20-1851-0, White Plains, New York, October 1974
KLEINEN, J.P.C., 'Verification and Validation of Simulation Models', Research Memorandum, Tilburg University, The Netherlands, March 1992
A Hard Core for Soft Problems

MAYO, E., The Human Problems of an Industrial Civilization, Macmillan, New York, 1933


MEEL, J.W. VAN, The Dynamics of Business Reengineering: Reflections on two case studies within the Amsterdam Municipal Police Force, Doctoral dissertation, Delft University of Technology, Delft, the Netherlands, 1994


NIJSEN, G.M., Modeling in Data Base Management Systems, North-Holland, Amsterdam, 1976


SOL, H.G. *Simulation in Information Systems Development*, Doctoral dissertation, University of Groningen, the Netherlands, 1982


VERBRAECK, A., AND G.J. DE VREEDE, 'Animation as a communication vehicle in simulation studies', in: *Proceedings of the European Simulation Multiconference (ESM93)*, Lyon, France, June 7-9, 1993
VERHOEF, *Effective Information Modelling Support*, Doctoral dissertation, Delft University of Technology, Delft, the Netherlands, 1993
VREEDE, G.J. DE, 'Improving Coordination within Organizations through information technology. Handling patients and combating organized crime', *Contribution to the ICIS Doctoral Consortium*, Florida, 1993
WIERDA, F.W. *Developing Interorganizational Information Systems*, Doctoral dissertation, Delft University of Technology, Delft, the Netherlands, 1991
## APPENDIX

<table>
<thead>
<tr>
<th>BE methods</th>
<th>Way of thinking</th>
<th>Way of controlling</th>
<th>Way of working</th>
<th>Way of modelling</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business Process Reengineering (Hammer 90, Hammer &amp; Champy 93)</td>
<td>discontinuous (Hammer 90); inductive (Hammer &amp; Champy 93)</td>
<td>role model: leader, owner, reengineering teams, steering committee, reengineering czar; list of common errors (Hammer &amp; Champy 93)</td>
<td>approach used in examples: benchmark, set rigorous objectives, analyze existing system, generate alternatives, implement (Hammer 90); identify processes, select process, understand process, redesign process, implement change (Hammer &amp; Champy 93)</td>
<td>generic rules and examples (organize around outcomes, not tasks; have those who use the output of the process perform the process; subsume information-processing work into the real work that produces the information; link parallel activities; capture information at the source etc.) (Hammer 90, Hammer and Champy 93)</td>
<td>CASE-tools</td>
</tr>
<tr>
<td>New Industrial Engineering (Davenport &amp; Short 90, Davenport 93)</td>
<td>viewing IT &amp; BPR as having a recursive relation</td>
<td>cross functional design team, visible commitment of senior management, continuous process improvement</td>
<td>5-step-approach: set objectives, identify key processes, understand &amp; measure processes, identify (IT-driven) alternatives, rapid (organizational) prototyping (Davenport &amp; Short 90)</td>
<td></td>
<td>datadictionary</td>
</tr>
<tr>
<td>Business Process Improvement (Harrington 91)</td>
<td>simplify and streamline operations</td>
<td>executive improvement team (EIT), process improvement team (PIT), BPI champion ('czar')</td>
<td>5 phases: organize for improvement, understand the process, streamline, measure and control, continuous improvement</td>
<td></td>
<td>CASE-tools</td>
</tr>
<tr>
<td>Continuous Process Improvement (Robson 91)</td>
<td>KISS: Keep it Straightforward and Simple</td>
<td>the 'natural' team, 'storyboarding'</td>
<td>start up, identify opportunities, select the team, get started, analyze the process, focus on critical process elements, 'listen' to the process, improve the process</td>
<td></td>
<td>CASE-tools</td>
</tr>
<tr>
<td>Business Process Redesign (Fried 91)</td>
<td>holistic process view</td>
<td>multidisciplinary BPR project team, steering committee</td>
<td>7-step-approach: analyze organization, define current process, educate working group, brainstorm about change opportunities, analyze recommended action</td>
<td></td>
<td>CASE-tools</td>
</tr>
<tr>
<td>Enterprise Analysis (Dennis ea 93, Vogel ea 93)</td>
<td>redesign as a multidisciplinary group effort</td>
<td>cross-functional design team</td>
<td>4-step-approach: prepare change process, model enterprise processes, generate redesign proposals by hard &amp; soft thinking, implement (prototyping)</td>
<td></td>
<td>Electronic Meeting System</td>
</tr>
<tr>
<td>Core Process Redesign (Kaplan &amp; Muirdock 91)</td>
<td>company = 3 à 4 cross functional core processes</td>
<td>cross-functional involvement, strong leadership</td>
<td>5-step-approach: Identify processes, define performance requirements, pinpoint problems, develop redesign vision, implement change alternatives</td>
<td></td>
<td></td>
</tr>
<tr>
<td>New org. forms</td>
<td>Way of thinking</td>
<td>Way of controlling</td>
<td>Way of working</td>
<td>Way of modelling</td>
<td>Support</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>---------------------------------------------------------------------------------</td>
<td>---------------------------------------------------</td>
<td>---------------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Cluster Organization (Applegate ea 88)</td>
<td>IT as enabling technology for flat structures</td>
<td>thoughtful planning, responsible management</td>
<td>restructure the organization drastically</td>
<td>characteristics of the Cluster Organization in terms of: structure (flat, flexible, dynamic, project oriented), management (decentral decision making, central control), human resources (autonomous workers, part time management), advanced IT (ES, EIS, group systems, telenetworks)</td>
<td>social architecture, enabling &amp; supporting advanced global communication technology (e-mail, video conferencing etc)</td>
</tr>
<tr>
<td>Organizational Network (Charan 91)</td>
<td>social networks as building blocks for boundaryless organizations</td>
<td></td>
<td>build and sustain; build: identify decision makers, reassign problem executives, focus on business problems; sustain: define network's deliverables, share information freely, evaluate regularly, develop new performance metrics</td>
<td>THE managerial challenge of the future</td>
<td>characteristics of the Information Based Organization in terms of: impact of IT on organizations (decision support, flat structures, task-focused teams), requirements (clear simple objectives, self-control, information responsibility), management problems (reward systems, common vision, management structure, supply of top management people) (Drucker 88) component types of the dynamic network in terms of strategic roles &amp; organizational structure: Prospectors ('first-to-market', flexible divisionalized structure, autonomous workgroups), Defenders (competing on value and/or cost, functional organization structure), Analyzers ('second-in'-strategy, imitate &amp; improve products, mixed structure such as the matrix) design principles: organize around processes, flatten hierarchy, assign ownership, maximize supplier &amp; customer contact, integrate managerial &amp; non-managerial tasks, self-management etc. basic principles: business processes are building blocks of corporate strategy, key processes must consistently provide customer value, make strategic investments in support infrastructure, the CEO must be the champion of a capabilities-based strategy</td>
</tr>
<tr>
<td>Information Based Organization (Drucker 88, 92)</td>
<td>knowledge is the primary organizational resource (Drucker 92)</td>
<td></td>
<td>use for: forecasting of industry's prospects (strategists), examining international competition &amp; their implications for public policy (policymakers) &amp; guiding redesign efforts (organization designers)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dynamic Network (Miles&amp;Snow 86)</td>
<td>an industry is a dynamic network of synergetic components</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Horizontal Organization (Ostof &amp; Smith 92)</td>
<td>structure around processes, organize around teams</td>
<td>make the CEO leader of the transformation</td>
<td>shift to aggressive goals, organize around chosen capability, make progress visible,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Capabilities based organization (Stalk ea 92)</td>
<td>the business processes as competitive weapon</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Organization design</td>
<td>Way of thinking</td>
<td>Way of controlling</td>
<td>Way of working</td>
<td>Way of modelling</td>
<td>Support</td>
</tr>
<tr>
<td>---------------------</td>
<td>----------------</td>
<td>-------------------</td>
<td>----------------</td>
<td>-----------------</td>
<td>---------</td>
</tr>
<tr>
<td>Classical School of Management (Taylor 11, Fayol 16)</td>
<td>horizontal &amp; vertical division of labour</td>
<td>expert-driven (Taylor 11)</td>
<td>time &amp; motion studies, production measurements</td>
<td>organization chart; management principles: importance of standards, control &amp; functional division of labour (Taylor 11); clear authority, unity of command (Fayol 16)</td>
<td></td>
</tr>
<tr>
<td>Human Relations/ Human Resources (Mayo 33, Likert 61, McGregor 60)</td>
<td>satisfied workers are productive workers</td>
<td></td>
<td>(laboratory) experiments (Mayo 33), controlled experiments, structured sample interviews (Likert 61), development of management skills by active learning (McGregor 60)</td>
<td>managerial interpersonal skills (Mayo 33), management styles: exploitative, benevolent, consultative, participative; the linking pin organization (Likert 61); theories on human behaviour: Theory X &amp; Theory Y (McGregor 60)</td>
<td></td>
</tr>
<tr>
<td>Socio-Technical Design (Trist &amp; Bamforth 51, Emery 59, Engelstad al 69, Cherns 76 &amp; 87, Emery &amp; Emery 74)</td>
<td>joint optimization of technical &amp; human subsystem</td>
<td>participative design (Emery &amp; Emery 74)</td>
<td>variance analysis: identify success criteria, draw system layout, list process steps, identify unit operations &amp; variances, construct variance matrix, identify key variances, construct variance control table, suggest technical &amp; social changes (Engelstad et al 69)</td>
<td>composite work group (Trist and Bamforth 51); an organization is a socio-technical system = a social &amp; a technical subsystem (Emery 59); design principles: compatibility of means and ends, minimal critical specification, variance control, etc. (Cherns 76, 87)</td>
<td></td>
</tr>
<tr>
<td>Strategic management (Chandler 62, Ansoff 65)</td>
<td>structure follows strategy</td>
<td></td>
<td>diagnostic study: analyze organizational tasks &amp; environment, determine required degree of differentiation &amp; integration, examine actual attributes of the organization, determine actual degree of differentiation &amp; integration (Lawrence &amp; Lorsch 67)</td>
<td>chapters in history of industrial enterprises (Chandler 62); growth strategies: market penetration, market development, product development, diversification (Ansoff 65)</td>
<td></td>
</tr>
<tr>
<td>Contingency Theory (Burns &amp; Stalker 61, Lawrence &amp; Lorsch 67)</td>
<td>organization design must fit environmental demands &amp; situational factors</td>
<td></td>
<td></td>
<td>dependencies: 'organic' management systems for changing conditions, 'mechanistic' management systems for stable conditions (Burns &amp; Stalker 61); relationships: environmental diversity must fit differentiation, environmental demand for interdependence must fit integration, integration &amp; differentiation have inverse relationship (Lawrence &amp; Lorsch 67)</td>
<td></td>
</tr>
<tr>
<td>Information processing view (Galbraith 77, Daft &amp; Lengel 86)</td>
<td>organizations can be seen as information processing entities</td>
<td></td>
<td></td>
<td>task uncertainty; basic mechanistic model; design strategies: reduce information needs (slack resources, self contained tasks), increase Information capacity (vertical IS, lateral relationships) (Galbraith 77)</td>
<td></td>
</tr>
<tr>
<td>Decision theory (Simon 77, Huber &amp; McDaniel 86)</td>
<td>decision making as central organizational activity</td>
<td></td>
<td></td>
<td>decisions: programmed vs not-programmed; decision makers: administrative vs economic; decision process: intelligence, design, choice; (Simon 77); design guidelines for decision units, sensor &amp; message handling &amp; decision management units (Huber &amp; McDaniel 86)</td>
<td></td>
</tr>
<tr>
<td>Transaction cost approach (Williamson 75, 81)</td>
<td>transaction as basic unit of organizational analysis</td>
<td></td>
<td></td>
<td>the transaction, normative rules: if assets are nonspecific, markets are favourable; if assets are specific, firm structures are favourable (Williamson 81)</td>
<td></td>
</tr>
<tr>
<td>IS design</td>
<td>Way of thinking</td>
<td>Way of controlling</td>
<td>Way of working</td>
<td>Way of modelling</td>
<td>Support</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------------</td>
<td>--------------------</td>
<td>---------------</td>
<td>-----------------</td>
<td>---------</td>
</tr>
<tr>
<td>System Life Cycle (Chaplin 76, Hice ea 74, Benington 66)</td>
<td>the SLC as phasing for IS design projects (Benington 56)</td>
<td>project management, decomposition of project team in functional areas</td>
<td>definition study, preliminary design, detail design, program &amp; human job development, testing, data conversion &amp; implementation, operations &amp; maintenance (Hice ea 74)</td>
<td>documentation standards (Hice ea 74); HIPO-diagramming (IBM 74); flowcharting (Chaplin 70); hierarchical (Tsichritzis 76), network (CODASYL 74), &amp; relational (Codd 70) data model; entity-attribute-relation representation (Chen 80); deep structure elementary sentences, conceptual data model (Nijssen 76); for analysis: Data Flow Diagrams, structured English, decision tables &amp; decision trees (DeMarco 79); for design: structure charts, design heuristics: cohesion, coupling, module size, span of control, scope (Yourdon &amp; Constantine 78)</td>
<td>CASE tools, Data Dictionary</td>
</tr>
<tr>
<td>Data oriented (Tsichritzis 76, CODASYL 74, Codd 70, Chen 80, Nijssen 76)</td>
<td>data is more stable then procedures &amp; processes</td>
<td>expert strategy</td>
<td>requirements collection &amp; analysis, conceptual data modelling, choice of a DBMS, logical data modelling, physical database design, implementation</td>
<td>for analysis: Data Flow Diagrams, structured English, decision tables &amp; decision trees (DeMarco 79); for design: structure charts, design heuristics: cohesion, coupling, module size, span of control, scope (Yourdon &amp; Constantine 78)</td>
<td>CASE tools, Data Dictionary</td>
</tr>
<tr>
<td>'Structured' approaches (Yourdon &amp; Constantine 78, DeMarco 79)</td>
<td>from abstract to detailed modelling to reduce complexity</td>
<td>project management</td>
<td>analysis: study environment, build logical model current system, build logical model new system, build physical model (DeMarco 79); design: transform analysis, transaction analysis (Yourdon and Constantine 78)</td>
<td>CASE tools, Data Dictionary</td>
<td></td>
</tr>
<tr>
<td>Process oriented (Lundeberg 81, Martin 89)</td>
<td>the objective of an IS is to support business processes</td>
<td>user oriented development, prototyping</td>
<td>change analysis, activity study, information analysis, data system design, equipment adaption (Lundeberg 81); information strategy planning, business area analysis, systems design, construction (Martin 89);</td>
<td>CASE tools, Data Dictionary</td>
<td></td>
</tr>
<tr>
<td>Behavioural (Ward&amp;Mellor 85, Hatley &amp; Phirrai 87)</td>
<td>information systems as finite state machines</td>
<td>iterative (Hatley &amp; Phirbai 87)</td>
<td>build essential model, build implementation model, build system (Ward&amp;Mellor 85); development cycle: requirements, design, implementation, testing (Hatley &amp; Phirbai 87)</td>
<td>CASE tools, Data Dictionary</td>
<td></td>
</tr>
<tr>
<td>Object Oriented (Dahl &amp; Nygaard 68, Jackson 83, Booch 91, Coad &amp; Yourdon 90)</td>
<td>problem situations can be modeled naturally by objects</td>
<td>middle-out, incremental</td>
<td>entity action step, entity structure step, initial model step, function step, timing step, implementation step (Jackson 83); identify classes &amp; objects, identify semantics, identify relationships, implement classes &amp; objects (Booch 91)</td>
<td>CASE tools, Data Dictionary</td>
<td></td>
</tr>
<tr>
<td>Participative, (Mumford 83, Eason 88)</td>
<td>end-users ought to participate in analysis- and design- phases</td>
<td>participative design groups (Mumford 83), participative project structures (Eason 88)</td>
<td>diagnose business &amp; social needs, set efficiency &amp; social goals, develop technical &amp; social solutions, combine them to socio-technical solutions, choose best solution (Mumford 83); 'toolbox' (Eason 88)</td>
<td>CASE tools, (Eason 88)</td>
<td></td>
</tr>
<tr>
<td>IS type specific, eg: KBS (Hickman 89), DSS (Keen 80)</td>
<td>different types of IS need different design approaches</td>
<td>adaptive, evolutionary design (Keen 80)</td>
<td>knowledge analysis phase: analyze static domain knowledge, analyze expert tasks, construct conceptual model (Hickman 89)</td>
<td>CASE tools, (Eason 88)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Expert Shells, DSS generators</td>
</tr>
</tbody>
</table>
TRUSTED LINKAGES AND SYSTEM CHAOS:
THE CASE FOR DYNAMIC MODELLING OF ORGANISATIONS

Doug Seeley
InterDynamics Pty. Ltd.
Burns Road
Inglewood SA 5133, Australia
Fax: +61 8 389 2294
e-mail: r.macri@unisa.edu.au

James R. Warren
School of Computer & Information Sci.
University of South Australia
The Levels SA 5095, Australia
Fax: +61 8 302 3381
e-mail: james.warren@unisa.edu.au

Keywords: Alignment, dynamic decision support, implicit models, organisational modelling, SimView, trusted linkages, uniformity assumptions, whole systems perspective.

Abstract

It is commonplace for organisations to experience a lack of “alignment” between the goals of strategic-level management and realities of the operational level. The largely financial performance measures of the boardroom fail to have meaning on the shop floor. We believe that this lack of alignment owes to a flawed implicit model of organisational dynamics that is deeply ingrained in corporate culture. Major flaws in this implicit model include masking of system dynamics under assumptions of uniformity, over-estimation of practical capacity of systems, fragmenting performance measures that encourage sub-optimal local decisions, and, in general, a lack of a “whole systems perspective.” A more faithful model of the organisation can be achieved by representing the discrete dynamics of individual transactions. This allows an integrated systems view that includes the causal connections of the organisation’s component systems. Event-Based Financial Dynamics (EBFD) bases performance measures on the financial impact of individual transactions. The result of using a dynamic model with EBFD is to achieve “trusted linkages” — that is, a set of performance measures that can span the levels of management without loss of meaning. The SimView performance modelling tool is at the heart of a Dynamic Decision Support (DDS) platform to support active decision-making based in a whole systems perspective and trusted linkages. The DDS platform encourages visualisation and understanding of the causal flows in the organisation.

1. Introduction

Most organisations are not harmonious entities in which the strategic perspective of upper-level management is smoothly carried out by operational staff and effectively monitored and

---

1This work is supported in part by an Australia Research Council Grant “Performance Measures for Lean Manufacturing,” in collaboration with Mitsubishi Motors Australia Ltd.
2SimView is a registered trademark of TechSearch, Inc., the commercial arm of the University of South Australia (UniSA). The software, which originated at the UniSA, has been developed and extended by InterDynamics Pty. Ltd.. This paper is produced with the explicit consent of InterDynamics Pty. Ltd.
reported by middle management. In fact, the norm is often one of pent-up frustrations at all
levels wherein management’s role in life seems often to be “fighting fires.” Furthermore,
even when an incisive analysis is done by outside consultants, it often cannot be imple­
mented because of internal politics. In this paper, we take the position that the reasons
behind these difficulties is a flawed approach to decision-making, especially strategic, and at
the basis of these flaws are inappropriate performance measures and an immature systems
view.

One flaw of the decision-making process is that the financial models that form the basis of
strategic decision-making are not made explicit. Even when they are embodied in a network
of tables and formulae in a spreadsheet system, the causal relations are often not present or,
at best, are very difficult to find and follow. Moreover, these financial models regard the
operational system as a kind of black box, and make little attempt to account for the organi­
sation’s dynamics, interdependencies and natural variations. This results in a significant lack
of alignment between financial-based decisions, such as capital investment labour policy, and
the day-to-day operational decisions regarding production and quality. This incongruity is
deepened by a number of assumptions about what should be measured as performance indi­
cators. Strategic management typically oversimplifies the nature of capacity and does not
acknowledge system bottlenecks, the negative effects of fragmentation, or the presence of
discrete chaos in operational behaviour. Even though these assumptions are rarely explicit,
they implicitly constitute a model of the dynamics of the organisation’s business — a model
that leads to severely flawed decision-making.

Exacerbating this situation is the wide variation in what different people think a system (and
hence, what an organisation) actually is. Earlier in this century, the word “system” typically
meant a set of rules to follow, or the static structure of some organisation. This evolved to a
kind of “black box” view, where entities in the organisation and the organisation itself are
thought to respond in a stimulus-response manner to inputs, independent of the rest of the
system. It is this local cause-and-effect thinking which appears most prevalent nowadays,
and which is reinforced by performance measurements focused on efficiency and localised
costs or value addition. However, we take the position that it is only by describing the causal
sequences of interactions that characterise how a system responds to the demands of its envi­
ronment that we achieve understanding of how the system works. This approach also
focuses on the role of feedback loops which structure the dynamics of the system, as has
been emphasised by the systems dynamics work of Forrester (1961, 1971) and Senge (1990).
We will go a stage further and acknowledge the chaos which discrete dynamics can cause,
and which is often smoothed over in the interests of simplicity. It is only this latter perspec­
tive of the discrete dynamics of how the whole organisation works that we will call an inte­
grated systems view; it includes the time-dependencies of interactions, as well as their causal
inter-relationships.

We show how the use of a discrete dynamic model of an organisation can be used to derive
performance measures that link the operational realities of intrinsic variation, constraints and
capacity back to strategic (usually financial) concerns. This state of affairs, where oper­
tional and strategic concerns are in alignment, we term trusted linkages. Having made the
case for an integrated systems view and trusted linkages, we describe a software system that
makes this possible using a methodology termed Event-Based Financial Dynamics (EBFD).
Integral with the integrated systems view is the explicit and graphical description of the
causal flows in a visual manner that builds confidence in its application. Dynamic Decision
Support (DDS) becomes possible when combining the visualisation of the financial modelling with a discrete and dynamic modelling tool. Simview, a commercially available simulation tool, is at the heart of a DDS platform that combines an animated visualisation capability, hands-on ease of model building and the support for analysis with a powerful but accessible financial modelling capability.

2. Conventional High-Level Decision-Making is Significantly Flawed

2.1 Assumptions and Implicit Models

Let us examine the thinking that goes on at the various levels of the decision-making hierarchy. First of all, people are not usually very reflective about assumptions; they rarely examine how they think, let alone examine the effectiveness of how they think. Usually they rely in their decision-making upon guidelines gleaned from the attitudes of their peer group, and by a fear of the conservative views of the boards of directors. It is not so much a matter of individual attitudes, but rather the self-reinforcement which the fixed attitudes of a corporate culture keep in place that prevents one's thinking from being critically reviewed. Conservative reactions in the corporate culture sustain this self-reinforcing set of assumptions. Surely fiscal responsibility and probity are essential to corporate decision-making and planning, but thinking based upon them provides an implicit model of how the organisation works. That is, these hidden assumptions are, by default, describing the manner in which productivity is achieved over time.

An implicit model is acceptable if it is roughly correct; but what if it not? The trouble with an implicit model is that it is an unexamined model. Its assumptions rarely get questioned; and it is difficult to discredit something invisible. The implicit model is invisible because it is not stated anywhere, and because all the other aspects of corporate culture have learned to function as if it were true. Implicit models surround everything done in corporate culture, sometimes acting as invisible barriers preventing us from being open to opportunities for improvement. Are the implicit models used in higher-level decision-making good ones? We believe the answer to be a resounding “No.” There are several reasons for this:

a) The lack of a true “big picture,” a whole systems perspective.

b) The assumption that the actual system works in a smooth (uniform) manner, ignoring the significant consequences of variation and of dynamics, the time-dependencies in operations, and especially the chaos that can result from discrete dynamics.

c) The generation of distrust by fragmenting performance measures that encourage sub-optimal local decisions.

d) Mis-applied efficiency and the lack of recognition of constraints.

e) Major misunderstandings regarding the nature of system capacity.

2.2 Including the Whole Systems Perspective

Organisations are made up of many components interacting in such a way that products and/or services result. Although profit and loss statements give you a snapshot of the inputs and outputs of the organisation for a given period, during that same period there is a lot of
activity going on that can be termed the organisation’s dynamics. Together, the components and their interactions produce the dynamics or behaviour of the system over time. In response to each request or demand (input) from the environment of the organisation is a sequence of interactions leading to a system output. It is these sequences or flows that indicate how the system works. Moreover, interactions which may be separated by some “distance” along the sequence may be causally connected flows.

We are strongly biased to see the world in terms of simple, perhaps naive, cause-and-effect reasoning. It is built into the structure of European languages; for example:

- The assembly line is not working because there are not enough parts available when we need them.
- The trim line in automobile production is being fed with sufficient bodies from the paint line on a daily basis.

We tend to see cause-and-effect on simply local terms. We tend to blame the workcell immediately before us for our own difficulties, and we do not account for the deleterious effects that our variation in supply to a following cell will have on that cell’s ability to smoothly produce at its intrinsic capacity.

Some of the causal connections in typical organisations actually “loop back” to earlier parts of a production sequence, causing what is termed feedback. Feedback relationships within a system are very important as they have the power to keep everything under control, or to cause things to “blow up.” The work of Forrester (1961, 1971) and Senge (1990) of the Sloan School of Management at M.I.T. has pointed out for many years that the lack of recognition of the role of feedback in our complex industrial and commercial systems causes many incorrect decisions and operating policies to occur. Feedback can only be recognised by looking for the threads of activity which interconnect all of the disparate parts of the whole organisation. What appears to be a harmless delay in an interaction in an assembly area might have disastrous consequences in a remote finishing area.

An important feedback loop for any organisation is the impact that the organisation’s products or services have upon its market share (revenue); and how market demand following on from this share returns to the organisation in the form of demand, often after some significant time delays. This we term the environmental loop (illustrated in figure 1). Furthermore, when this environmental feedback is understood, we call it “closing the loop.” Closing loops is an important part of the systems perspective. It means that one is always alert to the presence of feedback.

![Diagram](image)

**Figure 1.** The environmental loop of an organisation
The productive flows in figure 1 are not surprising — these relationships exist for almost all businesses. However, in terms of closing loops, there are two aspects of figure 1 that deserve emphasis. First, although organisations often possess a strong emphasis on marketing, as a function, the linkages between the quality of the goods and services produced by the business and the taking of a market share or contribution to market growth, is often regarded as too hard to model explicitly. Because it is difficult to prove what the relationships are, they are left to “gut feelings” or even ignored. Yet, the relationships and delays experienced by these linkages are crucial to the business’ profit-making ability.

Instead of saying “We don’t know what the relationship between quality factors and market-share is; therefore, we can ignore it,” it is far better to say “We think it could be this” (making explicit what is probably a non-linear relationship, possibly by using a graph). When a relationship is made explicit:

- The relationship can be examined;
- Its consequences can be seen;
- Efforts to verify it can happen;
- Most importantly, everyone in the company can refer to it and coherent management actions can result.

When a relationship is not made explicit, we lose the best basis against which decisions can be assessed. The illusion is that by ignoring a relationship that is difficult to determine, the situation has been handled reasonably. This is grossly misleading. It implicitly imposes the conclusion that there is no relationship; and, because it is implicit, it is rarely examined.

Not acknowledging such important interactions and delays, leads to a view of the organisation which is highly fragmented. This results in an organisation that responds poorly to the marketplace, leaving market share to be lost to smaller, brasher competitors. Making effective, competitive decisions requires an understanding of how the organisation works as a whole system. Local cause-and-effect thinking is not enough. Instead of just focusing upon individual interactions (when an input is received by a component), the focus should be on the threads of activity, the sequences of interdependent interactions that comprise the productive process. We shall call such threads productive flows.

### 2.3 The Uniformity Assumption Ignores Dynamics and Variation

Whatever guidelines and attitudes contribute to decision-making, they are each saying something about the assumed dynamics of the system. Often what they are implying is that production in all components of the system is occurring in a smooth and even fashion. If this were actually the case, and it is quite rare, the production process would be quite predictable. But demand is often highly variable, and production rarely smooth, contributing to an over-estimation of practical capacity. Hence, conventional decision-making rationales are implying that the organisation’s dynamics can be ignored. To not look at the dynamics of an organisation is to ignore a crucial aspect of its essence and uniqueness, and ignores factors that contribute to the bottom line. Productivity, service, profit and return on investment are tied to the rich interactions of an organisation’s dynamics, and cannot be derived from its static structure alone.
Consider, for example, a demand from a winery of 360,000 cartons of wine a year that is translated (via a uniformity assumption) into 1,000 cartons of wine each day. But let us say that the actual demand pattern is as shown in figure 2. The seasonal variations will sometimes be accommodated, but at other times the demand bursts will result in significant lags in delivery. When variation is not accounted for effectively, then chaos can emerge from the discrete dynamics that are actually occurring. The outcome will bear little relation to the tranquil predictions following from the uniformity assumption.

![Monthly Demand for Australian Wine](image)

Figure 2. Non-uniform demand pattern for a winery

The fact that production is ordered into sequences (interactions) is an important consideration in achieving a realistic estimate of system capacity, especially when there is competition for scarce resources. Because many interactions are time-dependent, undergoing various time periods for their processing, the ordering (the linking of the interactions) can easily result in unanticipated delays. At near-capacity, such unnecessary delays can often get out of hand, triggering delays and bottlenecks in other parts of the system. Very quickly, parts of the whole operation can grind to a halt. These dynamics are the inevitable result of attempting to work the organisation at full resource utilisation. Such an outcome is far from the minds of decision-makers who are assuming smoothness, full capacity, and the common but flawed wisdom of full utilisation. The reality of the organisation’s discrete dynamics is far from conventional decision-making assumptions.

### 2.4 Fragmenting Measures Generate Disharmony and Lower Profits

Business culture often takes the competitive nature of the marketplace and imposes it upon individual units internally, stressing and measuring individual performance and excellence. This implies the assumption of independence, that these units contribute separately to throughput and that these contributions can be measured separately. This notion stems from not viewing the operations as integrated networks of production flows with rich interdependence between individual units. The result of this incorrect independence assumption is fragmented decision-making, deterring the establishment of smooth flows of activity throughout the organisation. Typical examples of this are “time and motion” studies that measure utilisations, but ignore the interdependencies amongst the resources necessary for productive flows. The resulting decisions, that are typical for so-called “downsizing” activities, can be very deleterious for the essential operations of the organisation and should be considered misguided at best. Although we are not arguing against the minimisation of expense, when it is personnel that are being made redundant, decision-making based upon such fragmented assumptions is not only misguided but also has a high cost in human anguish.
Armed with the independence assumption and the misapplied ideal of individual performance, management establishes a regimen of fragmented performance measurement based on such notions as:

- The "efficiency" or utilisation of individual units such as a machine, workcell or worker;
- Product costs and profits that are based upon cost allocations of overheads.

A subtle way in which fragmentation occurs is through the organisational structuring mechanism of strictly functional divisions. These supposedly lead to cost savings due to economies of scale; however, by not structuring the organisation along the lines of its network of productive flows, the very essence of the organisations is divided and often strangled. Consider some typical functional division of a manufacturing operation as shown in figure 3. The critical measures of success are not to be seen by measuring any one functional unit. Furthermore, the presence of constraints or bottlenecks and their interdependent consequences, or the fact that the production process is determined by product mix not individual products, are not taken into account. The interdependence of people and workcells remains inadequately unacknowledged. Hence, measures based on functional divisions may obscure the perception of productive flows; and bureaucratic devices and human nature may combine to further clog operations.

![Vertically Partitioned Organisational Functions](image)

**Figure 3.** Performance criteria based on functional units “fragment” the productive flows.

Under a regime of performance measures fragmented by functional units, people are tempted to play "numbers games" with performance measures. Often individuals will unnecessarily compete with each other in a climate that lacks harmony. Moreover, a key fact not sufficiently acknowledged by fragmented management is the way in which the performance of one workcell is affected by the smoothness of the productive flow input by preceding workcells. This sets up a climate of sub-optimisation wherein section and division managers try to maximise "value-added" activities and minimise costs with no regard for the impact on the parts of the system that follow down-the-line in the productive process.

How do we acknowledge interdependence (lateral linkages) with other sub-systems? One way is to reward the supply of smooth flows at appropriate production levels from one sub-system to another. Goldratt (1986) has acknowledged this in his "drum-buffer-rope"
approach to production scheduling. Details of how this can be done are the subject of a separate paper on Wholistic Performance Measurement (Seeley 1994).

2.5 Simplistic Efficiency and the Existence of Constraints

One of the common guidelines for managing operations is that there should be no waste. The “no waste” attitude in the revolutionary Toyota assembly line operations (Ohno 1978, 1986; Johnson 1992), for instance, has supported this approach. The identification and elimination of waste is a key feature of activity based management (Marrow 1992; Turney 1992a, 1992b). Conventional management accounting systems have not focussed specifically on waste but on the efficiency with which resources are used to produce outputs; and performance measures based on so-called “efficiencies” and resource utilisation are still very common. Activity-based management aims to eliminate non-value added activities, that is, waste. Unfortunately, the valid consideration of utilising resources effectively is sometimes applied to everything — the various resources, departments and personnel in an organisation — as if they were completely independent of each other. This emphasises a viewpoint where resources and personnel are isolated units, fragmenting a system that needs to act as a whole. The over-emphasis on efficiency of organisational components may well have negative impacts that outweigh any benefits.

Excessive attention to efficiency can prevent crucial decisions from being made in a truly economic manner. For example, a bottleneck in the production of automobiles might be such that the addition of a second $50,000 hoist could alleviate the problem. However, from the perspective of full resource utilisation, one sees that the second hoist will be used only 20% of the time. Managers with a fixation on “efficiency” (which they naively equate with utilisation) regard this purchase decision to be wasteful and unwarranted. What this fixity prevents them from seeing is that the incremental increase in automobile production would rapidly recover the purchase cost. In other words, profit can sometimes be increased by an expenditure that lowers the local efficiency rating. The value of the expenditure can readily be perceived via any of a variety of models once the blinders of efficiency-obsession are removed.

Eli Goldratt (1986, 1992) has emphasised the importance of bottlenecks or constraints (not only productive constraints, but attitudinal constraints). He has shown how a great deal of wasted effort is created by so-called efficiencies in production paths not governed by a bottleneck. He and his associates have developed a methodology (the Theory of Constraints) and elaborate software to demonstrate these effects to people and to encourage a change in their thinking about the effective scheduling of production. They show how pushing all component processes to capacity will create large over-supplies of non-critical parts and services, and will add frustrations and interpersonal pressures between people in various parts of the organisation. The common notion that all available resource capacity should be used is incorrect because it looks at local utilisation rather than global throughput. Moreover, the global impact of sudden congestion when capacities are approached in the presence of variation is poorly understood in the minds of most managers (see section 3.5). Striving for full resource utilisation is an inappropriate strategy because it causes waste, congestion and unwarranted stress to the organisation.
2.6 The Legacy of Broken Linkages and Poor Alignment

Unwarranted assumptions of smooth operation and inappropriate, fragmenting performance measures are at the heart of what we believe is one of the principal drivers of distrust and tension within organisations, stifling their productive potential. At the heart of this generation of distrust is the poorly understood fact that:

*Organisations forced to operate at full resource utilisations and capacities, in the presence of typical patterns of variation, will inevitably face congestion, bottlenecks and system breakdowns.*

An immediate consequence of this fact (highlighted by Goldratt [1992]) is a climate of so-called “fire fighting” in which middle managers are in an almost constant state of reacting to near-emergencies. This fire-fighting syndrome is so common that many people feel that it is a necessary way to operate. Much of the day-to-day chaos, and belief in Murphy’s Law, stems from inadequate understanding of discrete dynamics. Moreover, this lack of understanding also leads naturally to blame being directed at people rather than at the recognition of the real causes: failing to account for variation and over-estimating capacity. Management blames operations for not delivering the capacity that management’s assumptions of smoothness seem to indicate, whereas operations blames management for not providing enough resources to cope with the bursts in demand that commonly occur. Attempting to operate at full resource utilisation and capacities is asking people to perform the impossible and creates a lack of harmony within the organisation that is far more costly than the perceived benefits of full utilisation.

Hence, the state of affairs in conventional business practice is that performance measures are based on a kind of business folklore, rather than on an explicit dynamic model of the organisation. This folklore creates an implicit model of the organisation’s productive process which cannot be adequately examined and verified in ordinary circumstances, further justifying the folklore. A similar state of affairs existed in the production of software for several decades. The situation has been improved by *Software Quality Assurance* (Dunn 1990), which upgrades the production of quality software from a craft to an engineering process with explicit procedures, methods, tools and metrics. Thus, as problems with software quality have become increasingly obvious, there has been the introduction of significant new formalisms to address the problems.

The increase in telecommunication linkages within organisations, along with the decentralisation of control brought about by microcomputers and computer networking, is providing many more opportunities to regard organisations as dynamic and whole systems composed of interlinked and mutually interacting components. The legacy of the attitudes and thinking of a corporate culture that has effectively denied the essential wholeness and dynamics of business systems themselves can no longer be denied and avoided. When the harm done to both the bottom line and the harmony of the workplace becomes increasingly evident, the situation where implicit corporate modelling diverges so widely from the dynamic realities of the operational levels cannot be tolerated. We believe that there will be an increasing pressure for business practices to mend the broken linkages between operations and corporate financials.
3. Discrete Dynamics and Performance Measurement

It is well understood that performance measures have an important effect upon the running of an organisation and the behaviour of individuals within it (Kaplan 1983; Nanni et al. 1992; Lessner 1992). Even so, it is our observation that the impact of performance measures is greatly underestimated. Certain measures can so strongly establish behaviour and perceptions within an organisation that people do not see the severe constraints and fixed attitudes that result. For example, a railroad can be so fixed on “gross tonnage shipped per year” that they do not notice the enormous savings attainable by matching and smoothing their ongoing demand on a weekly or daily basis. When career advancement depends upon the management of performance measures is when they really become fixed in behavioural “concrete.” Often, “working the numbers” produces deleterious behaviour by reinforcing people to always put local concerns first, and never notice how global harmony will produce a prosperous bottom-line, simply because no one’s career will improve as a result.

Traditional performance measurement techniques are based on the use of financial information. Measuring performance has been achieved primarily by calculating variances between planned and achieved outcomes. At the upper management levels, the primary focus has been on various measures of profitability and return on investment. At lower levels, the emphasis of management accounting systems has been on the measurement and control of costs. Efficiency measures such as material or labour cost per unit of output are also common. These accounting-based measurements have been widely criticised as inhibiting important beneficial changes in manufacturing (Cross and Lynch 1990; Fisher 1992; Johnson 1992; Johnson and Kaplan 1987; Kaplan 1990; Maskell 1989; McNair et al. 1989; Nanni et al. 1992). Table 1 summarised some typical traditional financial measures.

<table>
<thead>
<tr>
<th>Level</th>
<th>Objective</th>
<th>Measurement Period</th>
<th>Sample Measures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Upper</td>
<td>Maximise return-on-investment (ROI)</td>
<td>annual, 5-year plans</td>
<td>Overall expenditures, corporate revenues</td>
</tr>
<tr>
<td>Middle</td>
<td>Minimise budget variations</td>
<td>monthly, quarterly</td>
<td>Departmental costs, efficiencies, quality</td>
</tr>
<tr>
<td>Operational</td>
<td>Maximise throughput, make numbers “look good”</td>
<td>daily, weekly</td>
<td>Production throughput, resource utilisations</td>
</tr>
</tbody>
</table>

Table 1. Hierarchy of traditional, financial-based performance measures

The current emphasis on performance measurement in management accounting focuses on the belief that different types of performance measures are appropriate at different levels in the organisation and that a range of performance measures are needed to monitor business strategies. This has led to the concept of performance measurement hierarchies (Berliner and Brimson 1988; Cross and Lynch 1990; Grady 1991; Green and Flentov 1990; Keegan et al. 1989). Each of the hierarchies is based on the principle of working from the strategic objectives to derive effective measures of performance for the different levels within that organisation. A further development in linking of performance measures is the balanced scorecard method (Beische and Smith 1991; Kaplan and Norton 1992; Maisel 1992), which proceeds
from the strategic priorities, or critical success factors, of senior management. A set of perfor-
ance measures that directly support company strategies can be derived for each level
within the organisation. Each measure must be able to be linked to a critical success factor.
Measures at one level link to the measures at the next level. Such a structure can link finan-
cial measures at the top with the operational measures at the production level, and should
ensure a balanced and multifaceted view of each of four or five key strategic objectives.

However, the literature on performance measurement indicates that linked, integrated hier-
archies of measurement are still not observable in practice (Euske et al. 1993; Johnson 1990;
Kaplan 1990; Taylor and Graham 1992). Numerous operational and physical measures may
exist for activities and processes, but they tend to have come into existence on an ad hoc
basis (Kaplan and Norton 1993), often as a result of managers constructing their own infor-
mation networks to supplement those created by the formal management accounting process
(McKinnon and Bruns 1993). Kaplan and Norton (1993) reported the state of four
companies that had introduced their balanced scorecard methodology, and all appeared to be
in a developmental or experimental stage at the time of the writing.

The important questions to address a system of performance hierarchies are:

- Do the measurements at the operational level accurately reflect the firm’s strategic
  indicators?

- Does the measurement process that delivers decision-making numbers to upper
  management accurately reflect the way the organisation actually works operationally?

The “command and control” mentality, when applied to budget variations and efficiencies,
for example, will often fail to recognise important production issues. The aggregation into
large time periods and the use of averages will often distort important capacity considerations
because the organisation’s dynamics have been ignored.

In this section, we make the case that the discrete dynamics of an organisation’s productive
process must be included in order to produce strategic measures that accurately reflect the
realities of the operational levels. Achieving this state of affairs we term producing trusted
linkages. When trusted linkages are associated with measurements that reflect the dynamic
performance of the organisation as a whole, and with smoothing productive flow, then we
believe that significant improvements in profitability will result.

3.1 Operational Variations Cannot be Ignored

Recent attention on the management policies used by Toyota automobile production (Ohno
1978, 1986; Johnson 1992), and the recommendations of Deming (Deming 1986; Scherken-
back 1991) and Goldratt (1992) have highlighted the deleterious effects of variation in
production flow through an organisation. Variation generates waste in the form of excessive
work-in-progress inventory. One Australian organisation even calls variation the “disease”
of business. It is certainly true that a steady flow of production or service makes the most
effective use of an organisation’s resources. Toyota’s emphasis on detecting and reducing
variation as close to its origination as possible can be very effective advice.

Eradicating variation at an operational level, however, is not always easy. Moreover, there
are often reasons for necessary variation, such as external demands and inputs, or the defects
and stoppages of complex machines, or the inevitable variation in processing that results from changes in product mix. **Just because it is desirable to eliminate variation, does not mean that we can ignore it when it actually does exist, and it often does.**

We are still left with the essence of the management problem:

> When variation exists, if management assumes it out of existence, then the system must behave contrary to management’s plans.

Often, the ignoring of variation is a result of ways of thinking that are too fixated on the static structure of the organisation. Although everyone is aware that the organisation has dynamics (i.e. it is producing goods and services over time) many people do not conceptualise the nature of the events making up those dynamics. Understanding the series of cause-and-effect interactions that lead to these dynamic events is an important part of systems thinking. Typically, there is only an interest in the results, often financial, of what some department, division or worker accomplished over a given period of time. This produces a strong bias on results, while ignoring process. The effect of this bias is made worse by the viewpoint of organisations that perceive themselves as broken up into functional divisions; the fragmentation that this divisional structure produces may even generate additional variation internally.

The most fundamental effect of process time variability is the non-linear manner in which a system component can become highly congested, leading to instability and chaos as the component’s capacity is approached. This non-linearity is very surprising to our common sense because of the dramatic and sudden manner in which the resulting congestion can become extreme. Our observations of performance at relatively low levels of utilisation lulls us into thinking that another increase in the demand to a component can easily be handled. Consider, however, a stereotypical system response graph as shown in figure 4, Murphy’s Curve. For much of Murphy’s Curve, typically for demand which is less than 85% of capacity, there is very little congestion; but once this critical point is passed by the input demand, severe congestion becomes a fact of life.

We believe that much of the wisdom that managers perceive in Murphy’s Law (“Anything that can go wrong, will go wrong”) stems precisely from the interaction of normal variations in real-world processes with conventional queuing dynamics. Consider, for example, the responses of the two classic queuing systems (Molloy 1989) in figure 5 (Poisson [bursty] arrivals and services) and figure 6 (Poisson arrivals and deterministic services). Taken in isolation, a stochastic queuing system is highly predictable over a reasonably long period of observation; and it is not warranted to term it chaotic. However, at high levels of utilisation, it can be seen that the system will respond to a linear increase in demand with a non-linear (supra-linear) increase in congestion.

Baker and Gollub (1990) characterise chaotic systems as exhibiting an error in their predicted behaviour that grows exponentially over time with respect to the error in measurement of the system’s initial conditions. In real life there are numerous sources of minor variation in utilisation of a system — small changes in rate of job arrival or small variations in the rate of servicing jobs. In the “region of instability” (above about 85% utilisation) these small, commonplace variations lead to unexpected, and non-intuitive, major backlogs. While the system is somewhat different from a chaotic system in the formal sense, it shares a funda-
mental behavioural similarity: a small measurement error can lead to a wildly inaccurate perception of the resulting performance. In the region of instability, it is entirely appropriate to term the situation for management as "chaos" is the conventional meaning of the word.

The tendency of a queuing system to race toward congestion can be alleviated somewhat by reducing "bustiness" in the arrival and/or service rates of jobs. Figure 6 illustrates that deterministic (zero variance) service rates can greatly reduce system congestion, allowing higher utilizations to be pursued without ensuing chaos. However, as long as there is some variation in arrivals and/or services, the same problems emerge if 100% capacity is approached too closely.

Unfortunately, there is a tendency to assume smooth production rates based upon time-aggregated (e.g., monthly) data. When highly variable demand is "overviewed" in this manner, disaster is almost guaranteed. Not only is this true for external demands, it can be true for internal demands, as well.
3.2 The Performance Cover-Up

The major reason that actual variation (and hence actual operational behaviour) is covered-up in decision-making channels is that, for reasons of overview and the importance of financial measures, the decision-makers assume that everything in the organisation works in a uniform manner. That is, management assumes that productive flow and provision of service occur at constant rates. This assumption of uniformity arises from the following business practices when analysing the organisation, when reporting on results and when collecting supporting data:

Averages. Thinking that average values are sufficient to enable us to describe operational behaviour is common, but incorrect thinking.

Allocations. Apportioning overhead and operational costs over various resource usage, based upon pro rata allocations of costs, completely ignores the non-linear nature of cost accrual, and the overall system mix of resources required to make the next increment of improved performance.

Uniform rates. Financial calculations are usually based upon uniform rates of production which are based upon averages over long periods of time, completely ignoring bursts and variations in dynamics and consequently ensuring operational instability.

Consider the highly-uneven production curve for a Paint Shop shown in figure 7. Applying averaging to the system will lead one to believe that the peaks and valley in production will be evened out over time, and thus the total daily throughput is an adequate summary of the shop’s performance. Thinking that finished-goods inventory, internal buffers and delivery schedules will automatically even things out ignores what the impact of variation is within the organisation’s internal operations. At the minimum, there must be a great deal of work-in-progress inventory available to fill the internal buffers. Presumably, with a large enough buffer between the shops, the Trim Line will never “run dry” and need to stop because it has no input. However, if the buffer ever runs dry, the Trim Line will never attain its theoretical maximum level of production. Similarly, if there are any physical limits to the size of the buffer between the shops, then the Paint Shop may become blocked and have to stop until the Trim Line reduces the size of the buffer. If this happens, then the Paint Shop must fail to reach its theoretical maximum production levels. The local dependency upon such a buffer obscures the requirement for smooth production flows. It is difficult for such systems to handle bursts of demand, and they often require long periods of time in order to “recover.” When the recovery times are greater than the periods between bursts, the system will collapse. Major bottlenecks, and other breakdowns in productive flow can result when there is high variability in systems wherein a theoretical capacity balances an averaged demand.

Common and well-established business practice has reinforced the use of such uniform descriptions of the business dynamics. It is the simple and natural first approach in the analysis of business needs. The uniformity assumption works well, only if the bursts in demand do not overstrain the actual available capacity and buffers. This means that if a burst is temporarily greater than capacity, then there must be sufficient time and space buffering for the system to recover. The reason that the assumption does not often hold is management’s bias to obtaining 100 percent efficiency from all resources. This efficiency attitude ensures that various parts of the business will invalidate the uniformity assumption—
components lack the excess capacity that is necessary to recover. No wonder management is always “fighting fires,” and as a side effect creating enormous stress and anxiety for both operational and management personnel.

![Figure 7. Bursts in delivery of vehicles from the Paint Shop to the Trim Line](image)

As we move up the decision-making hierarchy, the uniformity assumption is made more frequently and more comprehensively. Because this ignores the impact of putting various parts of the business into critical operation, it disconnects these levels from what actually happens at the operational end of the business. Hence, decision-making at these levels can often become very “unreal.” (This can be successfully compensated for by the use of very conservative capacities, but at the substantial costs of unnecessary capital investment, and almost guarantees that some labour hired now must be laid off a bit later.) This leads to frustrations at either end: (a) the operational people know that many of the decisions coming down do not make sense for their actual work practice; and (b) the financial people cannot understand why it is so hard for operations to meet their financial objectives.

3.3 Aggregation Distorts Rather Than Simplifies

Aggregation, a process similar to averaging, is used in an attempt to simplify decision-making at higher levels. Aggregation or “lumping,” is the adding together of quantities that measure different things, even if the processing requirements of each have quite different implications for the organisation’s productive flow, either in time or location. Time-based aggregation is simply the application of the uniformity assumption described above. It is generally thought that suitable aggregations simplify decision-making by making the important (financial) distinctions more clear. However, such measurements bear little resemblance to operational realities where specific actions must be taken to improve quality and performance. Moreover, they rarely reflect actual financial realities in that their use along with the uniformity assumption gives very poor predictions of how the organisation should respond to changes. When such aggregations are used in extrapolations or assumed in linear allocations, their unreality is magnified.

Aggregation distorts in the following manner. Mixing measurements of different products or services hides the individual effects that may be indicating immanent system instability, as well as key profit and loss issues. When measurements are lumped over large time periods, they indicate very little about what actually went on during that time period, its dynamics and variations. Often such lumping is the result of financial reporting practices or the convenience of indirect data collection for other reasons. For example, if the processing of mail by a mail centre were to treat small letters, large letters, and small parcels all as simply “pieces of mail,” then this would be aggregation or lumping. Similarly, lumping all traffic
across a toll bridge as simply “vehicles” ignores the product mix realities presented by the processing delays for buses and trucks when compared to commuter cars.

Allowing such gross and distorting simplifications to occur in an age of electronic data collection and computer networks seems very obsolete. Unfortunately, these distortions are supported by the batch processing mentality of many large IT (Information Technology) departments, wherein such poor practice is cast in the digital concrete of virtually un maintainable computer code. Sometimes the simplified views that such aggregations provide are called snapshots; unfortunately, they are not even that. A snapshot would be an accurate data collection at some short interval of time. Aggregations are mostly inaccurate data collection smeared over a large interval of time.

3.4 Discrete Chaos and Murphy’s Law

In the previous sub-sections we described: a) how chaotic behaviour can occur in a subsystem (e.g., a service centre, work-cell, plant) when full theoretical capacity is reached in the presence of significant variation in productive flow; and b) how the phenomenon of “fire-fighting” in organisations is encouraged by inappropriate assumptions that promote this condition. Murphy’s Curve (figure 4) from section 3.1 describes this situation. Although this result follows simply from queuing theory as it applies to a single work-station, what appears to be not fully appreciated is that the same observation applies at different levels of abstraction (e.g., work-cells and departments) within many organisations. What this means is that the conditions for chaotic behaviour can appear at many places and at different levels, especially under a climate of over-efficiency. Once this is understood, the next difficulty arises; that is, once chaos arises in one part of a system it can spread rapidly to other parts because it can put neighbouring parts into the critical region of operation. Once this occurs, the overall system can suffer various levels of dysfunction and breakdown. The lack of appreciation of this flow of events leads to well-planned systems, apparently with sufficient capacity, breaking down.

Eli Goldratt (1992) articulates a similar view of over-efficient systems when he says, “Show me a balanced system, and I will show you disaster waiting to happen.” Balanced systems are ones where capacity of processing just balances the input of demand. In figure 8 below, the workstation on the right is running in a so-called balanced manner while subject to significant variation; hence, it is operating in the critical region. Because unstable behaviour occurs in the critical region, congestion can occur at any time and blow-up enough to fill a protecting buffer. When an intervening buffer is filled, this blocks products from leaving the preceding workstation. The latter workstation is then said to be blocked.

It is a common assumption that this kind of capacity matching of inputs to outputs is an appropriate strategy, especially when the uniformity assumption is used to plan systems operation. Figure 9 below illustrates the spreading of blockages. Even if blockages are relatively rare, the circumstances that lead to them, along with the deleterious effect on system performance they produce, can combine to make the occurrence of more blockages much more probable. This phenomenon has been extensively studied as applied to such phenomenon as store-and-forward deadlock within computer networks with the analytic tool of queuing networks. Once blockages start to spread, they can be just as uncontrollable as a bush-fire.
Of course disaster is a relative phenomenon, and therefore what makes the capacity/reliability nexus even more awkward is that there is a range of disasters that could occur. These extend from: a) simple blockages, where production cannot proceed because there is no available buffer space following it (this as the result of a bottleneck where demand has exceeded capacity); to b) spreading blockages, where a blockage breeds further blockages in neighbouring workcells because the original blockage has disrupted the productive flow; to c) gridlock (see figure 10 below) where the blockages feed back onto themselves, preventing any further production until extra resources are brought in to help. Any of these situations may be what constitutes a breakdown for the organisation. And the situation is not just a matter of the loss of productivity; it also depends upon the recovery costs involved, and these can be substantial.
3.5 Capacity Cannot be Divorced from Reliability

The operating capacity of a complex system subject to significant variation cannot be determined without specifying the reliability of the system’s performance as well. In other words, you cannot indicate what the capacity of the system is without also saying what is the frequency and type of system breakdown that you are willing to tolerate. This is not widely appreciated except in an intuitive manner amongst some operational people who develop a “big picture” of their organisation.

A given organisation can establish its own pragmatic definition of what constitutes a breakdown based both on immediate recovery costs and the loss of future demand from negative market perceptions of the organisation; it will be able to tolerate only a certain frequency of the occurrence of breakdowns. Only when this is clear, can system capacity actually be determined. For complex systems, this determination will only be possible with an accurate model of the discrete detail of the system behaviour, such as a simulation model that is validated against actual system behaviour. Consequently, take extreme caution when told or requested that a system or subsystem run at capacity.

This not a simple situation because of various kinds of breakdown are possible along with their differing costs, and because capacity can often be viewed in a number of different ways. For example in a railroad, capacity can be applied to:

- Only the track with its configuration of loops and depots, independent of sources and sinks of traffic;
- Only the track and the railroad’s rolling stock; that is, a given allocation of locomotives and commodity cars (and the running of the locomotives for coal, say, are restricted to their feasible configurations); or
- The track, the rolling stock and a regime of labour utilisation constraints determined by union contracts and current management policies.

Hence, asking the question, “What is our system capacity?” requires many qualifications before it can be answered effectively.

3.6 Prescription: Account for Discrete Dynamics within Performance Measures

We have painted a picture of the conventional use of financial measures in organisations that is rife with distortions based upon the convenient but flawed assumptions of uniformity and aggregation, distortions with a serious, negative impact on effective decision-making. Moreover, decisions based upon notions of capacity and efficient use of resources have been shown to be seriously flawed. The core of these flaws is the lack of sufficient understanding of the discrete dynamics of the organisation. These observations are made in a climate wherein organisations are under strong pressure to rationalise resources, re-engineer their work processes and to become internationally competitive. How can these flaws be overcome and a climate of trusted linkages established wherein the discrete dynamics of the organisation are taken into account?

An accurate understanding of the organisation is not derived simply in using systems dynamics models of the high-level performance of an organisation. Using continuous simu-
Since we have concluded that accurate estimations of systems capacity and reliability require effective discrete simulation modelling, we will look there for mechanisms to ensure that our performance measures obtain the fidelity between the throughputs of the operational level and the predominantly financial considerations of upper management. Our performance measures will be given the coherence and internal consistency afforded by a single, discrete dynamic model or a coordinated set of models at different levels of abstraction for different levels of management and/or decision-making time horizons. We call this a model-based management approach (illustrated in figure 11, again, under the qualification that we believe that the models must be discrete to be sufficiently accurate).

4. Event-Based Financial Dynamics

The answer to providing much more effective decision-making for on-going improvements lies in keeping the operational realities valid at higher levels of decision-making by ensuring that the financials are faithful to operational behaviours (i.e., trusted linkages). It also means finding measures at the operational level that reflect the strategic considerations of the organisation such as in the “balanced scorecard” approach. And it means avoiding the fragmentation of isolated measurements by being sure to include the flows of causal connections between all of the components of the organisation, in short providing a “big picture” of what is actually going on.

The financial life of the organisation originates with the individual transactions wherein the legal obligations of either an expenditure by the organisation or an income from a client are committed. It is these individual transactions which are the events driving the financial activities of the organisation. Although some of the resource-based commitments (labour usage, energy usage, equipment usage, maintenance, etc.) are built into the system specification, all financial processes flow from these commitments. When a model of the organis-
tion explicitly accounts for these events, we can say that the model is event-driven. When the performance of the organisation is derived from these events, then event-driven measurements create event-driven performance measurement. When such an approach is taken, then congestion, non-linearity and unstable behaviours will be reflected appropriately because the non-linear and unreliable nature of capacity can be accounted. As these measurements are abstracted for the sake of higher-level decision-making, fidelity to these realistic behaviours can be maintained. Moreover, opportunities to alter the mix of resources applied to production, and/or the mix of products and services provided will be accurately represented in these financial and performance measurements.

How can this be done in modelling software? — By building into the model specification stage a capability to measure both transaction-based financials and the time-based financials that reflect on-going resource usage. The process whereby the measurements are taken and reported is normally periodic in time, although providing the measures on an “as needed” basis or in real-time are also possibilities. The causality of the measurement process is usually not directly related to the causal flows in the productive processes of the organisation. However, there are times when on-going operational data is used fairly directly to adjust operating parameters (e.g., put more cashiers on when congestion in the check-out lines reaches a threshold); and there are other situations when more abstracted financial or quality data could affect operations, usually after some delay. For modelling clarity, this suggests that the performance measurement process in the organisation should be available in a separate performance view of the system (figure 12), where both the measurement logic and the measurement connections to the on-going productive process can be visualised, understood and alternatives can emerge.

![Figure 12. A Performance View of the system as separate from production](image)

The prospect of doing real-time performance measurement for management’s use at each level of the organisation now becomes attainable, as was presaged by Stafford Beer (1975) in his Cyberstride vision for the economy of Chile in the early 1970’s. The impact of this capability on the business culture of the organisation will be, in our opinion, immense (and beyond the scope of this paper). The capacity to include, segregate and visualise this event-based measurement capacity in a modelling system, we will term financial dynamics. The visualisation of financial dynamics along with the inclusion of discrete dynamic modelling, we will term Dynamic Decision Support (DDS).
5. Trusted Linkages Methodology

Earlier, we utilised a (rather conventional) 3-level hierarchy for discussion of the management process within an organisation: operational, middle and upper (see table 1). Associated with each of these is a different time-horizon for decision-making: immediate, tactical and strategic. Our methodology for Dynamic Decision Support, based upon our experience with a variety of organisations within Australia, suggests as well a 3-level approach to modelling:

1. **Detailed Modelling.** This is the kind of operational modelling for which discrete simulation systems were originally designed. We embrace a “process view” of modelling which is event-based (i.e., inter-event processing times are used) and describes system causality (how things happen in the system) via flows of intra-system interactions. There can be a variety of degrees of detail, approximation and aggregation within this level.

2. **Intermediate Modelling.** Ideally this level is “hybrid” (a combination of discrete and continuous processes are supported), and as such it provides a bridge from detailed modelling to the high-level modelling below. It should have strong use in parametric and optimisation studies of system performance as it is capable of representing system dynamics. This type of modelling would typically be at the middle management level in aid of system-wide decision support which may often involve repeated components (sub-models).

3. **High-Level Modelling.** This level of modelling is in support of upper management decision-making especially for investment and policy decisions. In particular, it has strong financial components and does not attempt to model system dynamics explicitly (conventionally this is done with spreadsheets); but its consequences must take a “safe” approach to the dynamic concerns of congestion and capacity. The challenge is to save money by delaying investments in physical plant capacity, by providing a more effective way to study capacity dynamically, and to provide a space to consider alternatives effectively.

Table 2 illustrates the mapping of the above modelling approaches to the levels of management and their respective decision-making time horizons. Note that this hierarchy bears some resemblance to the *macro, meso and micro* levels described by Bots and others (Bots and Sol 1988; Bots et al. 1994). However, our strategic level incorporates both the macro (inter-organisational) and meso (organisational logistics) levels. Our middle level provides a bridge from the micro (operational) level to the strategic concerns.

<table>
<thead>
<tr>
<th>Management Level</th>
<th>Time Horizon</th>
<th>Modelling Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operational</td>
<td>Immediate</td>
<td>Detailed</td>
</tr>
<tr>
<td>Middle</td>
<td>Tactical</td>
<td>Intermediate</td>
</tr>
<tr>
<td>Upper</td>
<td>Strategic</td>
<td>High-level</td>
</tr>
</tbody>
</table>

**Table 2. Time horizon and modelling approach by level of management**
Consequently if we are concerned with a dynamic model-based decision support for the entire organisation, then we are may be faced with a collection of dynamic system models. When these models can be coordinated such that their corresponding linkages are consistent and integrity maintained with operational dynamics, then we have an integrated set of dynamic models for management and accounting. Although we have not yet fully tested the feasibility of such an integration, it remains our goal for software development, and most of the pieces have come together. The organisational coherence that such an approach would provide promises to have significant benefit for the organisation, especially when financial measurement is included.

The key issue is how to establish high-level modelling in support of decision-making that has integrity with the actual operational behaviour. To achieve this, we must find trusted ways to link together these different modelling levels. To every high-level model, there must correspond an appropriate intermediate model, and every intermediate model can potentially be verified by a detailed model. This is to ensure that fidelity with operational behaviour of the system is sustained. When this is done, we have ensured that the various levels of decision-making in the organisation are in alignment. In our efforts to achieve integrity, we attempt to preserve:

- Causal relationships,
- Time delays and variation,
- Considerations of product mix,
- The effects of variation upon capacity, and
- The “whole system” interconnection of the parts.

When we move up a level of abstraction and away from some operating detail, only the causal relationships at the new level can be maintained; however, the effects of the more detailed causality can be accounted if we base the descriptions of time variation at the more abstract level upon them. Wherever congestion is known to arise in the discrete dynamics of the detailed level, we then reduce the theoretical capacities of workstations by an amount that would prevent significant congestion from arising. The abstraction of product mix can be approached by increasing the variation associated with the processing of abstract products appropriately. Just as with the causal relationships, whole system interconnections can still be accounted for at higher levels of abstraction. An attempt is being made to put this abstraction process upon a firm theoretical footing, by first doing empirical studies of different levels of modelling which are then used to suggest theoretical observations to be proven.

This integrated model-based management approach to the organisation supplies the full potential of current IT capabilities which is the equivalent of real-time system audits. Also, such models provide an overall coherence to measurement, exploration and decision-support, because the “big picture” of the organisation is maintained. The various interactions between different parts is represented by one entity, the model. However, because there are many types of models possible, including econometric, static financial, and continuous system models, it is important to distinguish a discrete, event-driven model from such alternative approaches. Hence, we term this approach integral systems management, wherein “integral systems” implies that the dynamics are represented by: a) discrete events, b) flows of causal interactions, c) performance measures that are event-driven, and d) linkages between models.
that retain fidelity. Moreover, because the modelling is specifically event-driven, assumptions can be properly examined and verified, leading to a much higher level of quality assurance of business practice.

The task that we are addressing in the Trusted Linkages methodology is to maintain fidelity between our performance measures and their operational reality. The following guidelines are necessary for ensuring that fidelity can be attained:

1. When strategy flows down the decision-making hierarchy from upper management to the operating floor in the form of performance measures, then the strategies must be represented by true, total system measures, and must accurately reflect operational realities by the embedding of the causality producing them in appropriate discrete, dynamic models.

2. When information about operational reality flows up the decision-making hierarchy, averages, aggregations and uniformity assumptions are set aside and replaced by abstractions of dynamic, model-based data (i.e. event-oriented) that account for essential variation in the dynamics.

3. When performance measures are applied to only a part of the total system, then they must take into account the interactions with the productive flows of adjacent subsystems, and the eventual feedback effects from overall operations.

Our business and administrative cultures have fostered a fragmented approach to decision-making and ongoing improvement because of the lack of systems-based conceptual tools for whole systems perception and measurement. This has been supported by the enormous inertia with which static and linear conceptualisations of how the world works have been held by these cultures, and especially how productivity and wealth are measured. Moreover, the fragmented approach to job measurement and advancement, has inhibited any motivation to perceive, understand and act upon the various interaction effects between apparently distant components of a whole system. The three fidelity rules above address such fragmentation and replace it with the wholistic and causally linked approach of Integrated Systems Management. This in turn, addresses the complex, non-linear operating behaviours of real organisations; details of this approach as we have developed it so far, are included in the appendix.

6. A Dynamic Decision Support Platform

For the past 6 years, we have been developing a visual modelling and simulation tool, at first with the University of South Australia's School of Computer & Information Science and lately in a commercial vein with InterDynamics Pty. Ltd. In the past 3 years this tool, called SimView (Seeley and Macri 1991), has been used as the core of a highly visual and animated decision support capability. As such, it is an example of a Visual Interactive Modeller or VIM (Bell 1985; Bright and Johnston 1991). In fact, one of the authors [Seeley] designed and implemented with graduate students one of the earliest VIMs as a queuing network animator and analyser called ANISIM at the University of British Columbia (Alemparte et al. 1975). Interestingly, the development of ANISIM was motivated by a desire to gain insight into the stochastic gridlock which occurred in early computer networks.
Like its forerunner, SimView's basic conceptual approach centred around the notion of the flows of interactions that make the systems work, or produce their products and services. This "flow" notion derives from GPSS's "transaction," and a conceptualisation of its "life-cycle" (Schriber 1991). This was put into a firm, object-oriented basis in SimView when development started in C++ in late 1988. Consequently, SimView's conceptual framework consists of flows of items and activities of agents interacting with system objects or resources. Performance measurement has been done by using measurement items to "read" object attributes and perform the computations necessary for reports on a periodic basis. This capability has now been built automatically into the decision support platform that embeds SimView.

6.1 The SimView Strategy

In our experience, the many techniques of the operations researcher or management scientist (such as mathematical programming and discrete simulation) have not had the impact upon organisations that was expected by their practitioners. Typically, a careful and highly considered model would be built of some operation within an organisation at the behest of a champion, only to see the recommendations of the study simply ignored in any subsequent decision-making. Consequently, at an early stage of the commercialisation of the software, we consciously set out to follow through with our client's use of the software to the organisation's decision-making stage and its implementation. Moreover, with our very first clients we noticed that the highly visual, animated capability of the software had a strong impact upon, and invited "buy-in" by, the various levels of management. We have continued to take these observations into account, and they have been largely responsible for the significant management consulting practice that we have developed alongside the software.

The key principles we have used to bring the potential and application of discrete simulation into the foreground of the organisation are: a) using the visualisation and animation capabilities to inspire confidence that the software's "reality" is the same one that the people in the organisation experience; b) ensuring that accurate financial information that can convince accountants can be derived from the application; and c) always inviting our clients to see how a study of one part of the organisation fits into the overall organisation, the "big picture." The visualisation aspect we find especially important to winning confidence in, and understanding of, the dynamic model (and we are far from the first to recognise the power of animation and graphics in simulation; e.g., Verbraeck and De Vreede 1993). Hence, we follow these five tenets in our development and application of SimView:

1. Model building by easy-to-use visual demonstration
   - visual programming invites participation in the model building process
   - provides visual mediation of how things actually work

2. Inspire staff confidence with schematic animations
   - rapid turnaround of what-if explorations
   - strong support for on-going improvements and total quality

3. Provide practical decision support
   - cultivation of systems thinking
   - cultivate holistic perspectives and sensitivity

4. Provide financial measures that align with operational performance
   - trusted linkages between operations and upper management
• inspires a more harmonious work environment

5. Support corporate objectives
• save significant sums on resource acquisitions
• deploy human resources effectively

6.2 The Decision Support Platform

Following through with client's application of SimView has provided us with a rich trove of insights and requirements that over the last year has transformed the software from merely a visually programmed and animated, discrete simulation system to a full-fledged decision-support platform for a wide range of organisations. We have altered our software development focus from merely providing enhancements to discrete simulation devices to providing practical decision support. That is, what we have developed and used with our clients is a software system providing rapid customisation of Dynamic Decision Support tools for end-users. The key components of this transformation have been: a) software connectivity to spreadsheets, b) traceability for model verification, c) capabilities for visual financial modelling, d) event-based financial measurement providing trusted linkage, and e) support for what-if exploration. In particular, this transformation has enabled us to deliver customised "response panels" that give managers the feeling of "piloting" their aspect of the organisation through their decision-making futures. The animation of the dynamics of the performance measures provides "fast forward" previews into the consequences of making a particular schedule of decisions. Moreover, the convergence of these software capabilities, along with this software's strong impact on the business culture of the organisation (applications to work process analysis and total quality management have not been described), may augur an important new development in the application of the simulation of dynamic systems. Hence, by embedding the discrete simulation and modelling capability within the DDS platform, we have opened up a broader and deeper range of application (figure 12).

Figure 12. A Dynamic Decision Support (DDS) Platform
7. Conclusions

7.1 Recommendations

We have emphasised the value of looking at the whole system, to not fixate on misled implicit models of performance, but to look to measures that reveal the actual dynamics and are tied to the operational realities of the system. This includes the ability to see the organisation in its context with suppliers and customers, and even as part of an inter-organisational system as shown in figure 13.

![Figure 13. Environmental feedback for export coal by rail](image)

While the ideal is to “close the loop” as in figure 13, this may not be attainable in practice. However, simply pushing the boundaries at either end of the demand and supply sides of the system can provide significant results. The real goal is to achieve a broader perspective of what the “system” really is, what drives production. We wish to avoid fragmenting views of “Us against Them” and leave simply “Us” (see figure 14).

![Figure 14. An integrated view of the system, void of fragmenting local performance concerns](image)

There has been some interest in inter-organisation modelling (Streng and Sol 1992; Wierda 1991); and rightly so, because the potential benefits are large. When what should truly be regarded as a single system is divided across organisational boundaries, there is an increased potential for some subsystems (i.e., organisations) to act without concern for their impact on others, and without awareness of the potential benefits of cooperation. However, the principle should be applied equally to the internal view: the organisational mind-set must not be
fragmented into functional units that compete rather than cooperate to achieve productive flows.

Work done by InterDynamics Pty. Ltd. in modelling the impact of increasing demand in the sugar/train system around Townsville, Queensland for Queensland Rail (QR) exemplifies gains from putting aside fragmenting performance measures and adopting a broader view of “the system.” QR had the policy of sending a locomotive out to the area of the four sugar mills it was servicing and having the locomotive pick up loaded wagons from among the four mills until it had a full load of 35 wagons. The locomotive remained out until it had this maximum 35-wagon load since this justified the investment that had been made in new, powerful locomotives. A reply to a query about the rail system’s capacity to handle the doubling of sugar supply from one of the four mills serviced brought the typical response of an order by QR to purchase more rolling stock in the form of sugar wagons.

Operators had long suspected that the system would have a greater capacity if the trains ran on a regular schedule of every 12 hours; but this would result in shorter trains that did not “efficiently” utilise the 35-wagon capacity of the locomotives. A dynamic model demonstrated that the regular schedule would greatly increase the system’s capacity. The trains would run to specific mills with 21 to 28 wagons (enough to handle the peek production of that mill, plus 2 wagons extra). QR accepted the recommendation and implemented the new policy; they were able to deal with the increased demand and actually reduced the rolling stock requirement (no new wagons were needed, and in fact 20% were mothballed). The new policy also gave early warning to over-production problems at the mills, and provided work practices that the unions found highly desirable. More than 10 million dollars was saved — and it all owed to putting aside the fragmenting performance measure of fully-utilising the capacity of the locomotives and looking to a solution which operators had already suspected.

The “trusted systems” methodology proposed herein is based upon effective (discrete) modelling of operations such that fidelity is maintained through all levels of the organisation. The following is a summary of the steps in our modelling process.

- Identify the strategic objectives of the organisation and their associated indicators.
- Choose an appropriate level of modelling detail for the objective of the study.
- Determine a framework of subsystems which describe the organisation at this level of detail.
- Determine the time-based nature of the productive flows between these subsystems (this will describe the burstiness of these flows).
- Determine the causal dynamics of the strategic indicators within the organisation.
- Construct an event-driven model of the organisation at the chosen level of detail which captures this dynamic behaviour and measures the strategic indicators.

7.2 The Future: Trusted Linkages and Complex Systems

The Trusted Systems approach to performance measures advocated in this paper can be seen in the light of parallel developments occurring in the broad, interdisciplinary field of Complex Systems (Warnecke 1993), and the paradigm shift occurring within the fields of
Information Technology (IT) and Artificial Intelligence (AI). An important shift away from centralised, hierarchical computing facilities (typified by the mainframe-based IT department) has been going on for more than a decade, with an inexorable move towards decentralised networks of more autonomous workstations. In AI, the recent successes of behaviour-based robotics, neural networks and genetic algorithms has heralded a swing away from the more traditional top-down approaches to intelligence. The event-based approach may in fact be part of a larger paradigm shift away from top-down knowledge and the authority of intellectual frameworks, and towards a more bottom-up or emergent approach to knowledge and being effective in the world.

The event-driven measurement methodology outlined herein works from transactions, the actual events of expense and income — accurately reflecting non-linearities, showing actual marginal impacts on financial flows and returns on investments. These events are causally linked together through the mechanism of a verifiable, dynamic model. The complex systems perspective takes the approach that actual behaviour and intelligence emerges from the rich interactions of large numbers of interconnected parts. It is a fundamentally bottom-up approach to knowledge and conceptualisation, in contrast to the traditional approach of forcing fixed attitudes or intellectual ideals onto organisations, systems and people.

What this means in practice is that we should pay attention to what is actually happening on a moment to moment basis, and to the complex web of interconnections that actually exists in our organisations. When we pay attention to what people and systems are actually doing on an on-going basis, and derive our management practices at each level or abstraction of the organisation from such observations, then we are following the complex systems paradigm. Being so informed by the reality of the operational praxis (the complex network of work practices), our decisions and improvements will have a powerful impact upon our strategic goals.

Today, through the wide availability of powerful and graphical computers, simulation technology allows us to deal in low level detail without abstracting away (via inappropriate averaging and aggregation) the actual operational complexities that dominate organisational behaviour. The objective ahead is to perfect the decision tools, Dynamic Decision Support (DDS), with simulation at their heart. The tools must provide the right measures, that bind the operational realities to the organisation’s strategic aspirations. To achieve this, the modelling tools must avoid making the system into a “black box” — it must be a “white box.” Animation and ease-of-use in the DDS platform can encourage human beings to develop, refine and test their intuition for the system’s behavioural characteristics by experimenting with the dynamic model. The white-box model becomes an explicit statement of the understanding of the organisation’s behaviour.

References


Trusted Linkages and System Chaos


Acknowledgments

Special thanks are owed to the InterDynamics directors, Bob Bridges, Riccardo Macri and Tony Griffith, for their contribution to the tools and experience at the heart of this paper; to Helen Thorne and Allison Southwick for their help in refining the trusted linkages concept and placing it in context with other theories of performance measurement; and to Gary Gibson for help in identifying relevant papers.
MODELLING ORGANIZATIONAL COORDINATION

Gert-Jan de Vreede and Daniel T.T. van Eijck
Delft University of Technology
School of Systems Engineering, Policy Analysis and Management
P.O. Box 5015, 2600 GA Delft, The Netherlands
Phone: +31.15.78.71.70, 54.16 Fax: +31.15.78.34.29
e-mail: {devreede, vaneijck}@sepa.tudelft.nl

Keywords: Organization (re)design, dynamic modelling, task coordination, process coordination, simulation.

Abstract

Coordination is a key issue in the design of organizational structures and processes. In our research tradition we use a dynamic modelling approach to address organization design problems. In this paper we propose to extend the dynamic modelling approach with a body of concepts that allows for explicitly modelling coordination on three perspectives: the organization perspective, the business process perspective, and the single actor perspective. We use a case study to introduce and illustrate these concepts. Furthermore, we formulate a way of transforming these concepts into a simulation model. We conclude by stating some unsolved problems and directions for further research.

1. Introduction

It is commonly recognized that in the post-industrial society organizations find themselves in a changing, demanding environment, see e.g. (Huber 1984; Lewin and Stephens 1993). Because organizations supposedly behave like purposeful systems, (Ackoff and Emery 1972), they are subject to a constant need to adapt their structures, processes, and technologies in order to meet new demands from their environment, (Burns and Stalker 1961; Lawrence and Lorsch 1967). In response to these demands, organizations are often inclined toward applying Information Technology (IT) to automate or support their current way of working without changing anything in the organization’s business processes. Hammer (1990) argues that it is because of this focus that significant investments in IT have not resulted in substantial performance improvements. He claims that more substantial gains and improvements can be achieved by changing or reorganizing the business processes. There is some evidence which
supports Hammer’s assertion, see e.g. (Hammer 1990; Davenport and Short 1990; Drucker 1991; Dur 1992; Sol 1992; van Meel 1994).

Reorganizing (in fact ‘organizing’) can be pragmatically defined as the process of dividing business processes into tasks and distributing (groups of) tasks and related responsibilities among one or more individuals or groups. In order to guide and monitor the business process as a whole these tasks, often carried out in parallel, have to be coordinated. Hence, a need to reorganize the current way of working implies a need to reorganize the way in which an organization achieves the coordination of its business processes.

However, changing or reorganizing an organization’s way of coordinating its business processes is easier said than done. (Re)designing coordination can be regarded as a very complex activity. There are many possible courses of action and many variables. Moreover, options for organizational change are difficult to test in reality, meaning that the effects of these changes cannot be known in advance. To overcome this problem, we build on the work of Sol (1982, 1992) who describes dynamic modelling as an approach to point out bottlenecks in the business processes of an organization and to analyze the effects of options for organizational change. This approach has been applied successfully in a number of projects, see e.g. Verbraeck (1991), Wierda (1991), Dur (1992), and van Meel (1994).

Dynamic modelling comprises of modelling techniques which facilitate the simulation of business processes over time. The approach follows a process of problem solving in constructing empirical dynamic models of an organization’s business processes. First a model for understanding (a descriptive model) is developed that correctly describes the organization’s current situation at a suitable level of detail. Then, after analysis of this model, one or several models for design (prescriptive models) may be built: derivatives of the model for understanding that depict possible organizational changes. These models are valid (i.e. have predictive value) provided that they do not drastically deviate from the basic structure of the model for understanding. For a more elaborate overview of the approach we refer to (van Meel 1994).

In spite of redesigning coordination being a key issue in reorganization efforts, the dynamic modelling approach so far has offered little support for explicitly modelling coordination. In this paper we propose to extend the dynamic modelling approach with a body of concepts that facilitates modelling organizational coordination during the construction of both models for understanding and models for design. The next section first discusses the notion of coordination within organizations and known approaches to modelling organizational coordination. The concepts we propose for modelling organizational coordination are then discussed in Section 3. These concepts are illustrated in a case example of a service organization. Section 4 describes how the concepts of Section 3 can be translated into simulation models. The paper concludes with a summary of our findings and directions for further research in this area.
2. Organizational coordination

Malone and Crowston (1994 p. 90) state that "coordination ... can occur in many kinds of systems: human, computational, biological and others". The notion of coordination addressed in this paper concerns the structuring and adjustment of operational activities in organizations. We refer to this as organizational coordination.

2.1 A definition of organizational coordination

Organizational coordination is a subject that is widely described in literature on organization design. Simon (1947) relates coordination to the behaviour of individuals within an organization. According to Simon (a group of) individuals need(s) to be coordinated because they are not capable of applying the right strategies for their actions, due to a lack of insight about the strategies of other individuals. Therefore, he defines coordination as providing everybody with insight into everybody else's behaviour, so that every individual is capable of taking the right decisions.

Galbraith (1977) sees coordination as an important ingredient of organization design. He describes organization design as the search for coherence between the goals or purposes for which an organization exists, the people that do the work and the patterns of division of labor and interunit coordination.

Most widely known is probably Mintzberg's description of coordination who states that, "every organized human activity gives rise to two fundamental and opposing requirements: the division of labor into various tasks to be performed and the coordination of these tasks to accomplish the activity," (1979, p. 2). Mintzberg presents six basic coordinating mechanisms that, in his view, describe the way in which organizations can coordinate their activities, (Quinn et al. 1988): Mutual adjustment, direct supervision, standardization of work processes, standardization of work outputs, standardization of worker skills, and standardization of norms.

Thompson et al. (1991) define coordination as the bringing into relationship of otherwise disparate activities. According to them the goal of coordination is to attune the potentially different and conflicting objectives of the actors performing activities in order to avoid possible chaos and inefficiency. They distinguish between three models which describe the way in which coordination can be achieved: markets, which tend to coordinate more or less automatically and unseen, hierarchies, which depict formalized coordination, and networks, which focus on the informal coordinating mechanisms within a group of (relatively independent) social elements.

More recently, Malone and Crowston (1994) define coordination as managing dependencies between activities. They mention actors and (interdependent) activities as key concepts. They mention four basic coordination processes that may occur within or among organizations: managing shared resources, managing producer/consumer relationships, managing simultaneity constraints, and managing task/subtask relationships.
Within the scope of this paper, we restrict to a definition of organizational coordination that focuses on the coordination of operational activities:

*Organizational coordination is the mutual influencing of working processes of two or more organizational actors in order to attain a certain objective.*

This definition is in line with the definition of Malone and Crowston, but as we focus on operational activities in an organization, we refer to working processes instead of activities and mutual influencing with a certain objective instead of managing. Three concepts are of importance here:

- Multiple actors (defined as organizational units that may take decisions).
- The mutual influencing of working processes.
- The attaining of an objective.

We chose for mutual influencing instead of mutual adjustment because the latter term is often used to describe only one specific coordination mechanism, see (Mintzberg 1979). The concept of attaining objectives or goals is important because coordination problems are often caused by the attainment of different goals by actors. By assuming that actors are able to exchange information about their work processes at any moment, they can carry out work in parallel and combine their products in the end. Actors may inform each other about the starting or ending times of their tasks in order to synchronize. Actors may issue instructions to other actors in order to get things done. We submit that the definition as given above is a good starting point for modelling organizational coordination.

2.2 Approaches to modelling organizational coordination

The essence of modelling organizational coordination is not only modelling what activities actors perform, but mostly how they cooperate and influence each other’s activities. In this respect we want to be able to model important organizational phenomena that relate to coordination like: parallel work processes, synchronization of work processes, and control (management) of work processes. We can formulate two requirements that have to be met by models describing organizational coordination:

1. They have to capture time-aspects, i.e. they have to be dynamic models.
2. They have to capture the communication between the actors involved in the coordination process.

In literature several modelling approaches can be found that seem to meet these requirements. The Petri net formalism (Petri 1980; Reisig 1985) is one of the first approaches to modelling concurrency and synchronization. Typical applications areas are communication protocols, computer systems, and distributed systems, see (van der Aalst 1992). Based on the Petri net formalism, a substantive number of modelling approaches have been formulated, see e.g. (Sol and Van Hee 1991, Sol and Crosslin 1992). However, two reasons make Petri nets less suitable for modelling organizational coordination. First, they are poorly communicable to
people that are not trained in reading them. Second, Petri nets tend to grow very large and incomprehensible, if the modeled situation gets slightly complex.

A mechanistic view on organizational coordination is offered by diplans, a formal graphical language for expressing plans, (Holt 1988). The strong characteristic of diplans models is that they explicify the coordination effort in a certain situation. The disadvantages are similar to those of Petri nets: they are poorly communicable and quickly grow very complex and incomprehensible.

Task structures as proposed by (Bots 1989) offer a means of modelling the activities (tasks) and their relationships of individual actors. However, coordination that involves two or more actors at the same time cannot be modeled straightforwardly, although the formalism offers a construct for modelling situations where coordination with other actors is required. Task structures have proven to be well communicable.

Dur (1992) proposes task structures from a different perspective: in his task structures, task can be performed by different actors, but all involve the same item (e.g. a form or product being processed). Because of this focus on items, it is not possible to describe coordination tasks in which no item is being processed. Furthermore, it is not possible to use single task structures to describe the coordination of business processes that involve more than one class of items. Like Bots' task structures, they have proven to be well communicable though.

Up to now we have elaborated on organizational coordination in general and shown different ways of modelling it. In the dynamic modelling approach that we discussed in the introduction, two modelling steps are taken in order to arrive at models for understanding or for design, (Sol 1982). The first step is called conceptualization and results in a conceptual model. A conceptual model offers the concepts in terms of which the modeller describes a situation (Bots 1989). The second step is called specification, resulting in an empirical model (usually a simulation model). The specification activity concerns the transformation of the conceptual structures into a simulation model and the addition of empirical data from the actual situation. We proceed in the next section by introducing a set of concepts for the conceptualization task that meet the requirements described in this section and overcome the disadvantages of the other approaches we discussed. In Section 4 we will illustrate how these concepts can be transformed into simulation code.

3. Concepts

In this section we propose a body of concepts for constructing a conceptual model of organizational coordination. We introduce a set of basic building blocks and perspectives for modelling organizations, business processes and organizational coordination. Before describing the concepts, we introduce a case study that is used as an example to illustrate the concepts.
3.1 An illustrative case study

The case study concerns Leyenburg Hospital, a medium sized Dutch hospital in the city of The Hague. At the present time, health care in the Netherlands finds itself in a very dynamic period. Apart from severe changes in the health assurance system, the average age of the population is increasing whereas the number of people working in the health care field is decreasing. Also, people are becoming more aware of their rights as a patient and tend to have a more critical attitude toward the medical profession. Finally, regulations from the Dutch government prescribe a reduction in the number of hospital beds.

Faced with these increasingly severe economic and social demands, the hospital management at Leyenburg decided to reorganize the way of working at the hospital in order to, (Ramondt and Rhébergen 1991):

1. Provide care for patients in a more economical way.
2. Provide care for patients in a more "patient friendly" way.

Both goals have been translated as improving coordination of the tasks of physicians and staff in such a way that a patient has to spend less time during a visit to the hospital. In order to achieve improved coordination, the hospital has to change its current organizational structure to one based upon the current and expected composition of patient flows through the hospital, (Ramondt and Rhébergen 1991; Vreede 1993). It was, however, not clear in which way the hospital had to change its current structure, nor what form the new structure should take.

In order to find possible bottlenecks in dealing with the patient flows and in order to evaluate alternative ways of coordination, we developed simulation models and animation models of the problem situation. Because of its magnitude only a part of the hospital has been modelled: the policlinic of the Neurology Department, which according to the hospital management can be considered to be a representative department for the whole hospital.

The simulation and animation models included the work procedures of the hospital's employees, like physicians, registration staff, and secretaries. The tasks they performed as well as the coordination between these tasks were determined and incorporated into the model. In order to coordinate the flow of patients, information is needed about when patient need to have a follow-up appointment, what examinations they have to go through, etc. This information is put down on forms, which are given to the patients who themselves have to bring them to the person or supporting department it is destined for. As a result patients are going back and forth through the hospital during a their visits to the Neurology policlinic. Apart from being medically treated, they have evolved into a means by which the physicians, supporting departments and the registration desk coordinate their activities. Using the simulation and animation models we were able to identify several reorganization options from which the actors involved expected that it would enable them to coordinate their activities better, without having to "use" the patients.
Both the hospital’s old and new situation can be modeled with the body of concepts which will be introduced in the next sections. In the remainder of this paper we will use the hospital’s current (as-is) situation to illustrate this. Since the purpose of the case study in this paper is to illustrate the concepts we used to model the coordination in the problem situation, we will not discuss the results of the case study itself in greater detail. They are reported in (Vreede 1993).

3.2 Building blocks: objects

We adopt a systems approach as the basis of our body of concepts. Not just only because is has proven to be a useful approach in organization science (Kramer and de Smit 1977), but merely because it can be used as a frame of reference for developing analysis and modelling concepts (Checkland 1981). We follow Sol (1982) in defining a system: "A system is a nested structure of entities". We, however, do not use the term entity here because it is generally replaced by the term "object" in the literature. Hence we define a system as a nested structure of objects. In addition, we note that, according to Sol (1982), a delineation of a part of the world as a system is a matter of choice and that this part of the world can be described by the attributes and actions of its objects, see also (Holbaek-Hanssen et al. 1975).

In order to work together in an organization, objects need to communicate. In the case study, the physicians need to communicate with the registration staff and indirectly, they communicate with a number of supporting departments. Hence, we see an organization as a network of communicating objects and therefore a system. The description of such a system is given by a description of each of its objects. The properties of an object are described by its attributes and actions. The attribute part contains data elements describing the state of the object. The action part contains the tasks and decisions that reflect the possible behaviour of an object. This view comes close to the object-oriented view, where objects have attributes and methods, see e.g. (Kim and Lochovsky 1989). However, the object-oriented view usually includes more aspects, like encapsulation and polymorphism, see for instance (Booch 1994).

For our concepts we have to describe the relevant objects in organizations, how these objects interact and coordinate their activities, how they are part of the organizations’ processes. These processes can be seen as systems. We define a business process as a sequence of activities carried out by interacting objects, to obtain an operational result. Hence, an organization comprises of a system of linked sub-systems, the sub-systems being business processes.

Within each business process we identify three kinds of objects: Nodes, Links and Items. A node can be either an actor (e.g. a physician) or a repository (e.g. a database). Both repositories and actors inherit the properties of node. When a node is an actor, its action part, apart from inherited actions, is not empty and it can establish state changes in the system. When a node is a repository, its action part is empty, apart from inherited actions, and it contains data that can be examined or changed by actors. Repositories may serve as a coordination mechanism: the job database is the node between the dispatchers and the service
engineers and serves as mechanism that coordinates the work of the engineers and the dispatchers.

An object that has a non empty action part can alter its own attributes. Altering attribute values of other objects must be done by means of communication, i.e. sending an item. An item contains information (in its attribute values) that triggers an action (or more precise: a reaction) in the receiving object; it does not carry out any actions itself. We use the term 'item' instead of 'message' because an item may not only represent the information itself but also its physical carrier. The arrival of a physical carrier at an actor may also trigger actions.

The construct that connects the nodes in such a way that they can exchange items is called a link. A link is specified by its attributes and may carry out actions related to the kind of technology it represents. It routes items from node to node. It may also change attributes in items (e.g. a link may behave erroneously and cause distortion). The configuration of the nodes and the links in the organization corresponds to the perception of the modeller of the analyzed situation. All objects that take part in a modeled system inherit the properties of one of these three basic classes. We state five specializations of the three basic object classes that are convenient in modelling organizations. They inherit from NODE and ITEM respectively. Together, the eight object classes form the basic elements of conceptual models of organizations and the coordination of their business processes. A formal description of the eight object classes is as follows:

<table>
<thead>
<tr>
<th>Object Class</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE</td>
<td>identifying attributes</td>
</tr>
<tr>
<td></td>
<td>actions</td>
</tr>
<tr>
<td></td>
<td>receive item</td>
</tr>
<tr>
<td></td>
<td>process item</td>
</tr>
<tr>
<td></td>
<td>send item</td>
</tr>
<tr>
<td></td>
<td>change attribute value</td>
</tr>
<tr>
<td>ACTOR</td>
<td>identifying attributes</td>
</tr>
<tr>
<td></td>
<td>status attributes</td>
</tr>
<tr>
<td></td>
<td>actions</td>
</tr>
<tr>
<td></td>
<td>work process</td>
</tr>
<tr>
<td>MESSAGE</td>
<td>identifying attributes</td>
</tr>
<tr>
<td></td>
<td>information contents</td>
</tr>
<tr>
<td></td>
<td>actions</td>
</tr>
<tr>
<td>PRODUCT</td>
<td>identifying attributes</td>
</tr>
<tr>
<td></td>
<td>product properties</td>
</tr>
<tr>
<td></td>
<td>actions</td>
</tr>
<tr>
<td>PERSON</td>
<td>identifying attributes</td>
</tr>
<tr>
<td></td>
<td>personal properties</td>
</tr>
<tr>
<td></td>
<td>actions</td>
</tr>
</tbody>
</table>
Using these object classes as basic building blocks, we can define three perspectives on organizations and the way they coordinate their business processes:

1. **The organization perspective.** From this perspective we look at the nodes in the organization and the items they communicate in order to coordinate their activities.
2. **The business process perspective.** From this perspective we focus on items being processed by a sequence of tasks. These tasks are carried out by one or more nodes and have to be coordinated.
3. **The single actor perspective.** From this perspective we look at a single actor and the tasks he/she has to perform and coordinate.

These perspectives are discussed in the next three sections.

### 3.3 The organization perspective

Having defined the building blocks, we now introduce the way of constructing an organization model for analyzing coordination and the way of representing it. In constructing the organization model, the first step is to set the system boundaries. Then, the relevant nodes, items and links have to be identified and described according to the object class definitions as given above. Finally, a graphical representation of the model has to be made. We illustrate this way of working by applying it to our case study.

**Step 1: Set system boundaries and reduce problem**

In our case study, we choose to include the physicians, the registration staff, the secretaries, and the patients. We do not, for example, model the patients' home-physicians that send the patients to the hospital, but we assume that patients arrive with certain intervals at the policlinic's registration desk. Also, only three supporting departments are modelled: the Blood Test Department, the Radiography Department, and the Clinical-Neuro-Physiology Department. Together these departments perform 91 percent of the all the examinations on Neurology patients.

**Step 2: Identify relevant actors, repositories, links, messages, products, and persons**

The entries in table 1 show all objects identified in the case study. There are no products, all the items are persons or messages.

<table>
<thead>
<tr>
<th>Actors</th>
<th>Repositories</th>
<th>Persons</th>
<th>Messages</th>
<th>Links</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physician</td>
<td>Waiting room</td>
<td>Patient</td>
<td>Patient file</td>
<td>Walking route</td>
</tr>
<tr>
<td>Registration employee</td>
<td>Patient file archive</td>
<td></td>
<td>Appointment note</td>
<td>Telephone line</td>
</tr>
<tr>
<td>Secretary</td>
<td></td>
<td>Patient</td>
<td>Examination note</td>
<td></td>
</tr>
<tr>
<td>Support. dept. employee</td>
<td></td>
<td></td>
<td>Admission note</td>
<td></td>
</tr>
</tbody>
</table>

*table 1 Objects in the hospital study.*

**Step 3: Describe objects in object classes that inherit from the predefined ones**

Below, an object definition is given of one example object from each predefined class.
Step 4: Make graphical representation of the organization model
The icons for the graphical representation of the organization model are given in figure 1. Using this graphical representation for the case study results in the organization model depicted in figure 2.

**figure 1 Icons for organization models.**

**figure 2 Organization model for the hospital case.**
3.4 The business process perspective

The organization perspective gives a rather static view on the objects that are concerned in an organization. We introduce the business process perspective to be able to model the dynamics of the organization concerned. We use the term business process to denote a sequence of tasks and decisions, carried out by one or more actors, leading to a certain operational result (e.g. serving a customer, making or selling a product, etc.). Obviously, the model of such a business process is called a business process model.

Bots (1989) also uses tasks and decisions to model organizational activities. However, he attaches the tasks and decisions to a single actor. Dur (1992) did it the other way around and attached actors to tasks, but limited the process to only one item class. We go one step further and allow multiple item classes to flow through the business process and also allow decisions to be carried out by a specific actor. We propose the following object classes:

**object class TASK**
- attributes:
  - identifying attributes
  - preconditions
  - postconditions
  - required items
  - required delay
  - next task(s) or decision(s)
  - subtasks
  - actions
  - execute
  - alter attribute values of items
  - alter attribute values of task’s actor

**object class DECISION**
- attributes:
  - identifying attributes
  - preconditions
  - postconditions
  - required items
  - required delay
  - next task(s) or decision(s)
  - decision rules (including subdecisions)
  - actions
  - execute

Every task and decision can be found in the action part of one of the actors of the organization model. Tasks and decisions may require items in order to be carried out. As mentioned in section 3.2, the actor carrying out the tasks, receives the items from repositories or other actors. When the set of tasks and decisions is determined and described, the business process model is complete. We propose four steps to arrive at the business process model.

**Step 1: Choose system boundaries and identify processes**
The system boundaries of the business process model are based on the system boundaries of the organization model. Depending on the problem that has to be solved, a further reduction can be made by including or excluding certain processes. In our case study we have to identify alternative ways of coordinating the flow of patients through the Neurology policlinic. Therefore, the boundaries of the business process model are set to the entrance of the patients in the policlinic and the departure of the patients through the policlinic’s exit or through the policlinic’s secretary when patients need to be admissioned.

Sometimes it is hard to identify the actual processes in an organization. Starting from the organization model, we propose two methods of identifying processes for the business process model. The first method is to evaluate the flow of one item. In many cases, especially in the logistic and administrative world, a business process can be defined as the routing and
transformations an item undergoes from the moment it enters the system to the moment it exits the system. In fact, the modeller sits on an item, watching the tasks and decisions that are carried out during the item’s flow, see (Streng 1993). The modeller notes the pre- and postconditions and the operational results after activities are carried out. He can also identify the actors that are responsible for decisions and activities. The second method is to watch the activities of each actor. The activities that contribute to a business process should be connected to the activities of other actors. The link objects are good starting points for this kind of analysis. In this way, a chain of activities will appear. This chain will either end or cross a system boundary. Both methods require observations and interviews with actors that take part in the actual situation. Table 2 describes a few examples of tasks and decisions that we identified using the first method.

<table>
<thead>
<tr>
<th>Actors</th>
<th>Task or decision</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physician</td>
<td>IF there is a patient in the waiting room</td>
</tr>
<tr>
<td></td>
<td>THEN call patient</td>
</tr>
<tr>
<td></td>
<td>See patient</td>
</tr>
<tr>
<td></td>
<td>Write notes</td>
</tr>
<tr>
<td></td>
<td>Up-date patient file</td>
</tr>
<tr>
<td>Registration employee</td>
<td>Get patient file</td>
</tr>
<tr>
<td></td>
<td>Receive patient</td>
</tr>
<tr>
<td></td>
<td>Make appointment</td>
</tr>
<tr>
<td></td>
<td>IF patient needs examination appointment and mobility is ok</td>
</tr>
<tr>
<td></td>
<td>THEN direct patient to supporting department</td>
</tr>
<tr>
<td></td>
<td>IF patient needs examination appointment and mobility is not ok</td>
</tr>
<tr>
<td></td>
<td>THEN call supporting department for appointment</td>
</tr>
<tr>
<td>Secretary</td>
<td>Get patient file</td>
</tr>
<tr>
<td></td>
<td>Prepare patient file</td>
</tr>
<tr>
<td></td>
<td>Handle administration</td>
</tr>
<tr>
<td></td>
<td>Direct patient to admission</td>
</tr>
<tr>
<td>Supporting department employee</td>
<td>Make appointment for examination</td>
</tr>
</tbody>
</table>

*table 2 Examples of identified tasks and decisions in the hospital case.*

Step 2: Identify pre- and postconditions and used repositories per task and decision
We argued that organizations can be modeled as networks of communicating objects. We defined coordination as the mutual influencing of work processes of actors. Hence, to coordinate, actors must be able to influence individual tasks or decisions of other actors. Actors communicate through the exchange of items. This means that we need a mechanism of inter task communication. The theory of Petri nets gave us the idea to use pre- and postconditions to establish this. A task’s and a decision’s attributes contain the pre- and postconditions, and the action part contains the transition between pre- and postconditions. Preconditions must be met before a task can be executed, postconditions are met after a task is executed. Fulfilment of the preconditions may require items that are available outside the actor, for example in a repository. The postcondition may require that items are sent to a place outside the actor. The transition may account for the altering of attributes or the transformation of an item. The transition between pre- and postconditions may implicitly model the objective that is attained by executing the task. The construct of pre- and
postconditions and transitions for executing tasks is used to model inter task communication. In this way we are able to model synchronization, parallelism and control, the three important coordination phenomena we put forward in section 2.2.

In the case study, an example of a precondition is that at least one patient should be available in the policlinic's waiting room before a physician can see him or her. The postcondition of seeing a patient is the fact that a physician can up-date the patient's file and send the patient home, to the secretary for admission, or to the registration desk to make further (examination) appointments. Apart from the pre- and postconditions, we have to identify the repositories that are used to communicate and store items. The result of this step for the case study is already visible in table 2.

Step 3: Identify decision rules or probabilities per decision
The result of a decision may be expressed by a probability or a decision rule. The decision rules can be of the form IF condition THEN decision. If no exact rules can be identified or if the decision is based on a probabilistic value, a stochastic variable may be used to model the decision. Examples of decision rules are also presented in table 2.

Step 4: Make a graphical representation of the business process model
We use the symbols as depicted in figure 3 to make a graphical representation of the business process model. The arrows may include probabilities or the outcomes of decision rules, the tasks and decisions may be combined with the repository symbols of figure 1 to denote the location of the items that are used while executing a task or decision. The actor should always be noted in task or decision symbols. Part of the business process in our case study is be depicted by the business process model in figure 4. Note that we added the repository symbols of the organization model to give a better insight.

![Icons for business process models.](image)

We explicitly note that ending up with a business process model that is a valid representation of the real situation in an organization, is not as simple as given above. During the process of designing the business process model, a number of reductive steps have to be done in order to manage and reduce complexity. It is very important to realize that the choice of which processes to incorporate into the business process model should be problem driven.
3.5 The single actor perspective

The third perspective on the conceptual modelling of organizational coordination is the behaviour of the individual actors and their role in the business processes. We use task structures as proposed by Bots (1989) to model the work processes of actors. As an example we take the physician of the case study. We identify two steps.

Step 1: Identify and draw task structure per actor
The work process of a physician is carried out every day he is at work. The physician calls his patients from the waiting room and sees them. After seeing them, the physician either sends the patient home, or writes an admission note, or writes an appointment note with or without an examination note. After letting the patient out, the physician updates the patients file. Then he is ready to see the next patient.

We use the symbols as depicted in figure 5 to make a graphical representation of task structures, see also (Bots 1989). Tasks and decisions may be compound. This is depicted by a small bomb in the upper left corner of the icon, meaning that the task or decision is able to "explode" into one or more subtasks or subdecisions. Using these icons, the task structure of the physician can be represented by the task structure of figure 6.
Step 2: Identify time delay and variable assignments per task
In order to capture the dynamics of the situation, a time delay has to be specified for each task. This time delay may be deterministic or probabilistic. Also for each task, the attribute assignments have to be determined. For example, the task "see patient" has a time delay that is determined by a normal distribution. The task "write examination note" assigns values to an instance of the object examination note, e.g. patient = mrs. Farbrace, examinations = blood test and CT-scan, and period = within two weeks.
4. Simulation

In this section we demonstrate how the modelling concepts of the previous section can be translated into simulation models. Such simulation models can be used to analyze an organization's business processes and the coordination required for these processes. Although outside the scope of this section, we mention that for such a quantitative analysis, empirical data such as process times, capacities of actors, and interarrival times of items, also have to be incorporated into the simulation model. A simulation model is constructed in three steps, according to the three modelling perspectives: the organization perspective, the business process perspective, and the single actor perspective. We start this section with a brief description of the simulation language we used.

4.1 Concepts of the simulation language

In the case study we used SIMAN/Cinema, a process oriented language that allows processes to be executed in parallel, see (Pegden et al. 1990). Strong features of SIMAN/Cinema are its possibility to separate empirical data from the structure of the simulation model, its support for statistical analysis and its animation facilities, see also (Streng 1993). A simulation model written in SIMAN/Cinema can be thought of as a collection of items flowing through a piece of simulation code. Items can be created and disposed of. Items can be sent to other pieces of code by conditional branch statements. Code that is restricted to one physical location can be assigned to a station. Items may be routed between stations. Stations may contain queue/resource combinations that act on items. For the construction of a simulation model from our modelling concepts, we use the SIMAN/Cinema statements given in table 3. There are many more statements but only the ones stated in table 3 are necessary for our purposes. For a more elaborate discussion of the SIMAN/Cinema language, see (Pegden et al. 1990; Systems Modelling 1989).

<table>
<thead>
<tr>
<th>Statement</th>
<th>Operands</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE</td>
<td>Batchsize, Offset, Interval, Maxbatches</td>
<td>Creates items</td>
</tr>
<tr>
<td>DISPOSE</td>
<td>StationID</td>
<td>Disposes an item</td>
</tr>
<tr>
<td>STATION</td>
<td>QueueID, Capacity, Balklabel</td>
<td>Implements a logical (geographical) unit</td>
</tr>
<tr>
<td>ROUTE</td>
<td>Duration, Destination</td>
<td>Transfers an item between stations</td>
</tr>
<tr>
<td>QUEUE</td>
<td>QueueID, Quantity</td>
<td>Queues up items for a resource</td>
</tr>
<tr>
<td>SEIZE</td>
<td>Priority, ResourceName, Quantity</td>
<td>Lets an item capture a resource</td>
</tr>
<tr>
<td>DELAY</td>
<td>Duration, StorageID</td>
<td>For time delays</td>
</tr>
<tr>
<td>RELEASE</td>
<td>ResourceName, Quantity</td>
<td>Releases a resource from an item</td>
</tr>
<tr>
<td>SEARCH</td>
<td>QueueID, StartItem, EndItem, Condition</td>
<td>Searches a queue for an item</td>
</tr>
<tr>
<td>REMOVE</td>
<td>QueueLocation, QueueID, Label</td>
<td>Removes an item from a queue</td>
</tr>
<tr>
<td>BRANCH</td>
<td>Maximum, Randomstream, IF, Condition, Label, Primary, WITH, Probability, Label, Primary, ELSE, Label, Primary</td>
<td>Controls flow of items through code</td>
</tr>
<tr>
<td>ASSIGN</td>
<td>ALWAYS, Label, Primary</td>
<td>Assigns values to variables and attributes</td>
</tr>
</tbody>
</table>

| Variable = Value |

Table 3 SIMAN/Cinema statements.
4.2 The basic structure: the organization model

For each object in the organization model we propose a conversion to the SIMAN/Cinema language. We do not specify the item construct as they can be simply translated into SIMAN entities with attributes.

Actors can be modeled with the STATION statement and a QUEUE/RESOURCE combination. An actor is seized by an incoming product, message or person and performs (part of) its work process. An actor has only limited capacity, so the items will be queued up if the actor is busy. This results in the following code for an actor:

```plaintext
ActornameST STATION, Actorname;
QUEUE, ActornameQ;
SEIZE: ActornameR;
BRANCH, 1: ALWAYS, ActornameWP;
ActornameRA RELEASE: Actorname;
BRANCH, 1: ALWAYS, ActornameLI;
```

In this example, Actorname stands for the name of the actor. This name can be extended with a Q to denote the queue, with an R to denote the resource, with WP to denote the address where the code for the actors work process is located, with RA to denote the address where the item returns after the work process has been accomplished and with LI to denote the address where the link interface is located. This construct is more complex when an actor shows autonomous behaviour, because the actor should then be kept "alive" by a control item, see (van Eijck and de Vreede 1994).

The repository construct can also be modeled by a station. The only task of a repository is to receive and supply items. This can be done by the following code:

```plaintext
RepnameST STATION, Repname;
BRANCH, 1: IF, Supply, RepnameSP IF, Request, RepnameRQ:
RepnameQ: DETACH;
RepnameSP QUEUE, RepnameQ: IF, NQ(RepnameQ) == 0, RepnameLI;
ELSE, RepnameRQA;
RepnameRQA BRANCH, 1: SEARCH, RepnameQ: Searchcondition;
REMOVE, J, RepnameQ, RepnameLI: DISPOSE;
```

Because items are not placed in any order in a queue, the queue must be searched first to find the item that is requested. The RepositorynameLI address points to the place where the link interface is located. After having completed a work process, the actor may send items away to links objects. An actor, however, may have more than one link (one for each actor or repository it is connected to). So in one way or another, the destination of the item should be determined. This can be done anywhere, so the easiest way is to reserve an attribute in the item where its destination is stored. This may be altered during a work process or somewhere else. It may also be stored in a global variable or determined instantly when sending the item. The following code can be used for sending an item to a link object:
The duration is zero, because any time delay in transport is arranged for by the link object. The destination can be determined from the item, from a variable or local expression or can have a standard value. The destination must be a link object.

The link object is modeled by the following code:

```
LinknameST STATION, Linkname;
BRANCH, 1: IF, Active, LinknameGO;
    BRANCH, 1: IF, NOT Active, LinknameNOGO;
LinknameGO ROUTE, LinknameDELAY, LinknameDEST;
LinknameNOGO DELAY, LinknameDELAY2;
    BRANCH, 1: ALWAYS, LinknameST;
```

This construct simulates the possible working or failure of the link object. The LinknameDELAY variable stands for the time it takes to send an item to its destination (note that a link has only one origin and one destination). If a link is inactive, the item waits (LinknameDELAY2) and tries again (this construct can be replaced by other mechanisms, e.g. the item can be disposed if it is not sent). The actual failure of links can be generated by stochastic generators.

With these concepts it is possible to simulate the basic structure of the organization model. We have not yet specified the way in which the tasks of actors are performed. This is done in the following section.

4.3 Adding business dynamics: the business process model

The business process to be modeled is reflected in the tasks that are carried out by actors. This means that the work processes of the relevant actors should be specified. In an actor, the item is sent to the initial task or decision of a work process by the following code:

```
ActornameWP BRANCH, 1: ALWAYS, Initial_task_or_decision;
```

This may also be a conditional branch, determining the initial task from the attribute values of the incoming item. It might even be that items have to be copied to accomplish a work process.

We now need simulation code for the task and the decision objects. Tasks have two properties: they take time and they may influence attribute and variable values. Decisions also take time and they may influence the sequence of tasks to be performed.

A task is modeled by the following piece of simulation code (note that this code always belongs to an actor STATION):
The assignment of attributes may also include a destination attribute that determines the flow of the item after the work process is completed. In case of a task that has no successor tasks, the last line of code in this construct is replaced by:

\[
\text{BRANCH,1: ALWAYS, ActornameRA;}
\]

A decision can be modeled by the following piece of simulation code:

\[
\begin{align*}
\text{Decisionname} & \quad \text{DELAY: DecisionnameDUR; }\nonumber \\
\text{BRANCH, N:} & \quad \text{IF, WITH and/or ALWAYS constructs;}
\end{align*}
\]

In the branch statement, items may jump to other tasks and decisions according to the decision rules or probabilities specified in the decision object. \( N \) represents the maximum number of items that exit the decision for one incoming item; this means that the item can be copied by the branch statement.

The decision rules are specified by a structure of conditional branch statements. Note that tasks and decisions are specified on the level of leaf elements. Compound tasks and decisions are constructs that simplify graphical modelling, but have to be expanded (or exploded) when the simulation model is constructed.

The organization model and business process model describe an implicit form of coordination: the items that flow between actors and repositories. In order to coordinate the tasks and decisions carried out by single actors, explicit coordination in terms of pre- and postconditions has to be defined. The next section will describe how coordination in terms of pre- and postconditions can be implemented in simulation code.

### 4.4 Adding single actor coordination: pre- and postconditions

For establishing coordination that cannot be achieved by item routing, we have to specify the pre- and postconditions for the tasks and decisions in the work processes. Preconditions may be expressions that operate on global variables or attributes. There are two ways of implementing this. The first is by assigning values to attributes and variables that may be tested later on (the conditions are implicitly specified by the values of attributes and variables). In this case, the code for the task and decision object needs to be extended with code for precondition testing by using a simple conditional branch statement like the following (note that this code precedes the specified code for tasks or decisions, so the labels Taskname and Decisionname are replaced by TasknameA and DecisionnameA):
Task/Decname BRANCH, 1: IF, Preconditions met, Task/Dec.nameA:
                        IF, NOT Precond. met, Task/Dec.nameWT;
Task/DecnameWT DELAY: Duration of test cycle;
Task/DecnameA ........ Rest of code for task or decision
                        (see previous section)

The second way of implementing pre- and postconditions is by using the WAIT and SIGNAL construct. The above can be replaced by the following:

Task/Dec.name WAIT: Precondition expression (integer);
Task/Dec.nameA ........ Rest of code for task or decision
                        (see previous section)

Any time a condition is met, the WAIT statements should be signalled by the following statement:

Somewhereincode SIGNAL: Precondition expression (integer);

We prefer the first construct because we don’t need to use integers here and because the code for testing a precondition is only located at one position in the simulation model (while the SIGNAL statement may be far away from the WAIT statement, making the model quite complex and hard to debug).

5. Conclusions

In this paper we have presented a body of concepts to support modelling organizational coordination from three perspectives: the organization perspective, the business process perspective, and the single actor perspective. With these concepts we were able to model the business processes and their coordination in the case study described in the paper. Experiences in other case studies support these findings, see (van Eijck and de Vreede 1994).

In addition, we demonstrated how these concepts can be successfully translated into a simulation model. Similar translations can be done with other simulation languages like Simula (Birtwistle et al. 1984) and DEMOS (Birtwistle 1979), GPSS (Gordon 1981), and Simscript (Markowitz et al. 1963) as these simulation languages use concepts that are similar to the SIMAN/Cinema approach. This means that our modelling concepts are not restricted to the SIMAN/Cinema language.

Some situations are still hard to model and need more powerful concepts. Especially with respect to the single actor perspective we can think of situations that cannot be modeled in a straightforward manner. Two examples:

- **Interruption:** Interrupting an actor, for example when a telephone rings, is still hard to model with our set of concepts. This problem requires adaptation of the actor (or the node) object.
Modelling Organisational Coordination

Parallel task execution by one actor: It is, e.g., impossible to model an actor receiving a telephone call and writing a note at the same time. Using the task structures as proposed by Bots (1989), the actor is restricted to executing one task at a time.

We believe that the concepts can be further extended to a method for modelling business processes. Using the concepts within a design methodology, combined with automated support tools, the modeller can concentrate on modelling a specific situation from the three perspectives described and leave the making of a simulation model to an automatic model generator (that can do this job much faster). The goal of further research is therefore to come up with an integrated approach to address improving organizational coordination. This involves three directions for the coming two years:

• Developing more concepts that solve the above problems.
• Developing a set a of modelling guidelines (a design methodology) for employing the presented body of concepts.
• Designing and developing automated support for translating the concepts into executable simulation code.

References


DYNAMIC PROCESS MODELLING THROUGH MULTI-LEVEL RBNs

Aphrodite TSALGATIDOU, Dimitris GOUSCOS and Constantin HALATSIS
Department of Informatics
University of Athens
TYPA Buildings, Panepistimiopolis, Ilisia 157 71 Athens, GREECE
Phone: ++30-1-7230172,7291885,7217941
Fax: ++30-1-7219561
E-mail: {afrodite.gouscos,halatsis}@uranus.di.uoa.ariadne-t.gr

Keywords: dynamic modelling, Petri nets, control flow, requirements engineering.

Abstract
This paper presents an approach for specifying and validating the dynamic aspects of IS at multiple levels of abstraction, during the requirements specification phase of system development, using a Petri net based model, called Rule-Based Net (RBN). The produced specifications explicitly model the dynamic system behaviour, they are object-oriented, rule-based, executable and can be validated using graphical animation. In this way, we come up with concise, compact, modular, validated and easily modifiable requirements specifications at the very early stages of system development. Furthermore, these specifications can be easily mapped to design and implementation structures, resulting in a system which satisfies user's needs and is flexible enough to incorporate future changes of behaviour.

1. Introduction

Whereas information systems are inherently dynamic, traditional approaches towards Information Systems (IS) development focus on the modelling of static aspects and are not capable of specifying the dynamic properties of these systems. The lack of methodologies and associated tools for modelling the dynamic aspects of information systems has been realised by a growing number of researchers and a few approaches to dynamic modelling of information systems have recently emerged (DYNMOD-I 1990; DYNMOD-II 1991; DYNMOD-III 1992).
Along these lines, this paper presents an approach for specifying the dynamic aspects of information systems at multiple levels of abstraction using a formal Petri-net (Murata 1989) based model, called Rule-Based Net (RBN). As the last abstraction level of the RBN model has been described elsewhere in detail (Tsalgatidou et al. 1993; Tsalgatidou et al. 1994), a more simplified view of the model is presented and the main focus lies on the various abstraction levels of the RBN model and on the mapping from one level to another.

RBN graphical specifications are developed in an object-oriented rule-based framework. These specifications are executable and they model explicitly control flow within an organisation; as far as validation is concerned, the graphical nature of the RBN formalism allows for graphical animation techniques, whereas the theoretical foundation of the model provides a basis for formal validation.

![Diagram of the sequence of models developed during building dynamic object-oriented rule-based requirements specifications in a top-down fashion](image)

**Figure 1.** The sequence of models developed during building dynamic object-oriented rule-based requirements specifications in a top-down fashion

More specifically, the objective of our approach is to explicitly model and validate the dynamic behaviour of an information system at the early stages of system development. This is accomplished by first constructing an object-rule schema, which contains only the static aspects
of the application at hand, and subsequently enhancing the schema with dynamic behaviour. The
latter enhancement takes place according to a stepwise, top-down approach, and an intermediate
RBN model is produced in each enhancement step. Fig. 1 depicts the sequence of models
employed during this process.

The proposed approach is described in the next section, and the various schemas employed
(static Object Schema, dynamic Object-Rule Schema, context, high-level, medium-level, low-level
and detailed RBNs) are briefly defined in the following. Section 5 discusses RBN validation
and the last section concludes the paper.

2. Modelling the Dynamic Behaviour of an Information System

The dynamic behaviour of an information system is captured in a dynamic Object-oriented
Rule-based Schema (dynamic ORS) in terms of behaviour units (BUs) which are defined per
object class. Each BU has a certain behaviour described in terms of dynamic rules. Each rule
has a THEN-part which contains various actions and, optionally, an IF-part containing some
preconditions.

The behaviour of each BU is triggered by a signal of a specific signal type. Signals are produced
either by the external environment or by the execution of the actions of a dynamic rule. When a
signal of a specific type is created (either by the external environment or by the execution of the
actions of a rule) it is directed to the appropriate BU and triggers its behaviour (which is
described in terms of dynamic rules). This means that the rules of that BU which have their IF-
part satisfied (or rules with no IF-part) will execute the actions described in their THEN-part.
The execution of these actions may generate other signals which are sent either to the external
environment or to another BU of the same or of another object class of the dynamic ORS. In this
way, signals constitute the sole means of communication between the various parts of the
information system under study and between the information system and its external
environment. An example of an ORS may be seen in fig. 2.

The procedure that has to be followed in order to construct a dynamic ORS is the following:
First a static object schema (static OS) is constructed. The static OS is a set of object classes of
the application at hand, and contains only their static class and instance properties. Details about
the construction of the static OS may be found in (Tsalgatidou et al. 1994). Having constructed
a static OS, the next step is to develop a context RBN RC. RC is a Petri net that contains one
transition (called “INFORMATION SYSTEM”) standing for the whole information system
under study and other transitions which model the external environment. Places of RC model
signals exchanged between the information system and its external environment, see fig. 3(a).
At the following step, the "INFORMATION SYSTEM" transition of $R_C$ is decomposed into a number of transitions corresponding to the application object classes (that have already been defined in the static OS) and into a number of places which model the signals exchanged between object classes and are inscribed with the signal names. Transitions and places of $R_C$ which model the interaction of the system with the external environment remain the same, unless it is realised, during this decomposition, that the signals exchanged between the system and its external environment need to be modified. The RBN produced at this stage is a high-level RBN $R_H$; fig. 3(b) provides an example of such an RBN.

At the next abstraction level, a medium-level RBN $R_m$ is produced by decomposing each object class-transition of $R_H$ into a number of transitions modelling the behaviour units of the corresponding object class and a number of places modelling the signals exchanged between these behaviour units. Fig. 3(c) presents the medium-level RBN which resulted from the decomposition of the high-level net of fig. 3(b).

From a medium-level RBN $R_m$, a low-level RBN $R_l$ may be produced which depicts the dynamic rules that describe the behaviour of each behaviour unit. Thus, each BU-transition of $R_m$ is decomposed into a number of transitions corresponding to the dynamic rules of this BU. At this stage, transitions are inscribed only with the rule names (see fig 3(d)).

A detailed RBN $R_d$ is produced at the lowest abstraction level. Each transition of $R_d$ is inscribed with the contents, i.e. the IF- and THEN- parts of the represented rule, the BU and the object class where the rule is defined, as well as the object classes that inherit this rule (more details may be found in (Tsalgatidou et al. 1993)).

```plaintext
object class CUSTOMER_ORDER subclass of ORDER
  instance properties
    name ArrivalTime domain TIME
    name Priority domain PRIORITY
    name Customer domain CUSTOMER
    name Invoice domain 0:1 INVOICE
  end instance properties
  class behaviour
    class BU CUSTOMER_ORDER.ProcessOrdersOnHold
      triggered by ProcessOrdersOnHold(Product)
      rule R1
        execute NotifyOrders(signal.Product,OnHold) ;
      end BU CUSTOMER_ORDER.ProcessOrdersOnHold
    end class behaviour
    instance behaviour
      instance BU CUSTOMER_ORDER.ProcessNow
        triggered by ProcessNow()
        rule R1
          if self.Quantity > self.Product.StockQuantity
          then execute SetPriority(OnHold) ;
          NoShipment(self.OrderNo) -> EXT ENV ;
        rule R2
          if self.Quantity <= self.Product.StockQuantity
```

Aphrodite Tsalgatidou, Dimitris Gouscos, Constantin Halatsis
then execute SetPriority(Processed);
  IssueInvoice(self.ObjectId) -> INVOICE;
  DecreaseStock(self.Quantity) -> self.Product;
end BU CUSTOMER_ORDER.ProcessNow
end instance behaviour
end object class CUSTOMER_ORDER

Figure 2. Part of a dynamic ORS

In summary, the specification of the dynamic behaviour of an information system starts with development of a static OS from which a context RBN $R_C$ is constructed. Subsequent refinements of $R_C$ in lower abstraction levels result in a detailed RBN $R_d$ which contains all information necessary for modelling in a graphical manner, and within an object-oriented rule-based framework, the dynamic behaviour of the system under study. During RBN refinements, the static OS is also gradually enhanced with the dynamic behaviour of each object class, thus resulting at a complete dynamic ORS when the RBN model has been refined down to the lowest abstraction level.

All the aforementioned models are formally defined in the following section, while fig. 1 depicts the sequence in which these models are created. It is worth noting that, although the refinement process outlined above proceeds in a top-down fashion, a bottom-up approach is also possible, since the mappings from one model to another are bidirectional.

3. The static Object Schema and the dynamic Object-Rule Schema

3.1. The static Object Schema (static OS)

Modelling of an information system starts by construction of a static Object Schema (static OS) which consists of a set of object classes ($classes(OS) = \{C_1, C_2, ... \}$) and their static class and instance properties. In a static OS there is a unique class OBJECT, and class $C_i$ has a set of subclasses $\{C_{i1}, C_{i2}, ... \} = sub(C_i)$. A static OS supports single inheritance, so that each class $C_i$ other than OBJECT has a unique superclass and it inherits all the properties defined in its unique superclass as well as in all the upper superclasses up to class OBJECT. More details about OS and its construction may be found in (Tsalgatidou et al. 1994).
3.2. The dynamic Object-Rule Schema (dynamic ORS)

A dynamic Object-Rule Schema (dynamic ORS) is an object-oriented schema which consists of a set \( \text{classes}(\text{ORS}) = \{C_1, C_2, ... \} \) of object classes and a set \( \text{sig}(\text{ORS}) = \{S_1, S_2, ... \} \) of signal types. Each object class has a set of class and instance properties and a set of class and instance behaviour units; the latter describe the behaviour of the class and its instances in terms of rules. Each behaviour unit \( B \) consists of a set of rules \( \text{rules}(B) = \{R_1, R_2, ... \} \) with preconditions and actions and is triggered by a signal of a specific type \( S \) (when \( (B) = S \)); each rule \( R_i \) has certain preconditions and actions and, at the time of triggering, the rules of \( B \) which have their preconditions satisfied are fired and execute their actions.

The preconditions \( \text{iff}(R_1) \) of a rule \( R_1 \) of a behaviour unit \( BU_{ij} \) may refer to the properties of the class \( C_i \) where \( BU_{ij} \) is defined (def \( (BU_{ij}) = C_i \)) and to the parameters of the signal type \( S \) which triggers that behaviour (when \( (BU_{ij}) = S \)). The actions then \( (R_1) \) of \( R_1 \) may also refer to the parameters of the triggering signal type \( S \) and the properties of class \( C_i \) and, furthermore, they may also modify the values of the properties of class \( C_i \) and produce signals of specific types which trigger other behaviour units or which are sent to the external environment. In case \( BU_{ij} \) is a class behaviour unit, actions then \( (R_1) \) may create or delete instances of \( C_i \). An example of an object-rule schema may be found in fig. 2.

There are two possible cases for the flow of signals of a signal type \( S \): either (a) they are produced by the external environment \( X \) and directed to an object class or (b) they are produced by (the execution of the actions of a dynamic rule of) an object class and directed either to the external environment or to a behaviour unit of the same or of another object class. Therefore, the producers \( \text{prod}(S) \) as well as the consumers \( \text{cons}(S) \) of a signal type \( S \) constitute two sets made up of the external environment \( X \) and/or of object classes of the dynamic ORS. For any signal type \( S \), \( \text{prod}(S) \) and \( \text{cons}(S) \) are not necessarily disjoint, but there is no signal type \( S \) such that \( \text{prod}(S) = \text{cons}(S) = \{X\} \), i.e. there is no signal which is both produced and consumed solely by the external environment.

As already mentioned, each class \( C_i \) of a dynamic ORS has a set of subclasses which inherit all the static features of \( C_i \) and of its superclasses, and may also define some new ones; the dynamic features of a class, i.e. its class and instance behaviour units and rules, may also be inherited as they are or redefined accordingly (Tsalgatidou et al. 1993), and new dynamic features can be defined as well.
4. The Rule-Based Net (RBN) models

The RBN models are based on Petri nets (Murata 1989) and more specifically on Predicate-Transition nets (PrT-nets) (Genrich and Lautenbach 1981). Petri nets have been used by many researchers for modelling the dynamics of information systems; see for example the work by Conrath et al (Conrath et al. 1991), where Petri nets are used for specifying the work flow of an office automation system or the work by Ang et al. (Ang et al. 1991) where Petri nets are used as an office process specification formalism. Guha et al. (Guha et al. 1991) employ generalised stochastic Petri nets (GSPNs) to model system performance.

In the proposed approach, Petri nets are used within an object-oriented framework as the underlying formalism for modelling the dynamic aspects of an information system in terms of rules. The various RBN models which were mentioned in section 2 and lie at different levels of abstraction are defined in the rest of this section. All these models can be formally mapped to each other (Gouscos 1994), and the mapping process can proceed either top-down towards more detailed models, or bottom-up towards more abstract models. At all levels of abstraction defined below places represent signal types, tokens present in a place model signals of the corresponding signal type which are exchanged within the various parts of the information system and between the information system and its external environment, and each specific marking corresponds to a state of the modelled system.

4.1. Context RBNs

A context RBN $R_C$ is defined as a tuple

$$R_C = <P_C, T_C, F_C, O_C, \text{sig}_C>$$

where $P_C$ is a set of places, $T_C$ is a set of transitions, $F_C$ is a set of arcs connecting places to transitions and transitions to places, $O_C$ is the underlying structure, (which has resulted from the static $OS$ with the addition of some signals from and to the external environment) and $\text{sig}_C$ is a function that maps places to signal types. $T_C$ contains one transition which corresponds to the information system under study and is conventionally named "INFORMATION SYSTEM" and other transitions $X_i$ which model the external environment that produces (consumes) various signals sent to (coming from) the information system. Each place $p$ of $P_C$ is inscribed with a signal type $\text{sig}(p)$ and the parameters of this signal type. Thus, places in a context RBN are inscribed with the signals consumed or produced by the external environment (see fig. 3(a)).

4.2. High-level RBNs

A high-level RBN $R_h$ is defined as a tuple

$$R_h = <P_h, T_h, F_h, O_h, \text{sig}_h, \text{class}_h>.$$
where $\text{class}_h$ is a function mapping transitions to object classes. $O_h$ is the underlying structure of $R_h$ and it has resulted from the corresponding $O_C$ of the context RBN $R_C$ with the addition of the signals exchanged between the various object classes. $P_h, T_h, F_h,$ and $\text{sig}_h$ are similar to the corresponding elements of the context RBN, but have different values. Here, the unique "INFORMATION SYSTEM" transition of $R_C$ has been decomposed into a number of transitions (corresponding to and inscribed with the object classes of the underlying structure) and into a number of places representing and inscribed with the signals exchanged between these object classes (see fig. 3(b)).

4.3. Medium-level RBNs

A medium-level RBN $R_m$ (see fig. 3(c)) lies at one level of abstraction lower than the corresponding high-level RBN and is defined as a tuple

$$R_m = \langle P_m, T_m, F_m, O_m, \text{sig}_m, \text{bu}_m \rangle$$

where $\text{bu}_m$ is a function mapping transitions to behaviour units. $O_m$ is the underlying structure of $R_m$ and it has resulted from the corresponding $O_h$ of $R_h$ with the addition of the behaviour units of each object class. $P_m, T_m, F_m,$ and $\text{sig}_m$ are similar to the respective elements of the corresponding high-level RBN, but have different values: here, each object class-transition of the high-level RBN is decomposed into a number of behaviour unit-transitions and into a number of places corresponding to signals exchanged between behaviour units of the same object class.

4.4. Low-level RBNs

A low-level RBN $R_l$ lies at one level of abstraction lower than $R_m$. More specifically, each behaviour unit-transition of $R_m$ is here decomposed into a number of transitions corresponding to the rules of that behaviour unit. Each transition $t$ in a low-level RBN is inscribed only with the name of the corresponding rule ($\text{rule}(t)$). A low-level RBN $R_l$ is formally defined as a tuple

$$R_l = \langle P_l, T_l, F_l, O_l, N_l, \text{sig}_l, \text{rule}_l \rangle$$

where $O_l$ is the underlying structure of $R_l$ and has resulted from the structure $O_m$ of $R_m$ with the addition of the rules of each behaviour unit. $\text{rule}_l$ is a function that maps each transition $t$ of $R_l$ to a rule of the underlying structure $O_l$. $P_l, T_l, F_l,$ and $\text{sig}_l$ are similar to the corresponding elements of the medium-level RBN, but have different values. $N_l$ is a multiplicity function of arcs, and each arc $f$ of $F_l$ is inscribed with its multiplicity $N_l(f)$, where $N_l(f) = 1$ or $N_l(f) = u$, $u \geq 1$. $u$ represents an a multiplicity which is undefined at modelling time but takes a certain value at run-time (see fig 3(d)). It should be noted that this notational convention does not violate standard Petri net models because, for any net with arcs of undefined multiplicity, one can always construct a second net with arcs of multiplicity 1 such that for any given initial marking $M_0$, the reachability trees of both nets have the same terminal markings (projected to the places of the first net).
Dynamic Process Modelling Through Multi-Level RBNs

(a) context RBN, $R_c$

(b) high-level RBN, $R_h$

(c) medium-level RBN, $R_m$

(d) low-level RBN, $R_l$

Figure 3. RBNs at various abstraction levels
4.5. Detailed RBNs

A detailed RBN $R_d$ is defined as a tuple

$$R_d = \langle P_d, T_d, F_d, O_d, N_d, \text{sig}_d, \text{class}_d, \text{bud}_d, \text{if}_d, \text{then}_d \rangle,$$

where $P_d$ is a set of places, $T_d$ is a set of transitions, $F_d$ is a set of arcs connecting places to transitions and transitions to places, $N_d$ is a multiplicity function of arcs, $O_d$ is the underlying structure of $R_d$ (i.e. the full-detail dynamic ORS which resulted from $O_l$ with the addition of the contents of the various rules), $\text{sig}_d$ is a function that maps places (i.e. elements of $P_d$) to signal types, $\text{class}_d$ is a function which maps transitions to object classes, $\text{bud}_d$ is a function mapping transitions to behaviour units, $\text{if}_d$ is a function mapping transitions to preconditions and $\text{then}_d$ is a function which maps transitions to actions.

In $R_d$ every transition $t$ corresponds to a dynamic rule of the underlying structure and is inscribed with the corresponding preconditions, actions, behaviour unit and class, defined respectively as $\text{if}_d(t)$, $\text{then}_d(t)$, $\text{bud}_d(t)$ and $\text{class}_d(t)$. Each place $p$ of $P_d$ is inscribed with a corresponding signal type $\text{sig}_d(p)$ and the parameters of $\text{sig}_d(p)$. Fig. 4 depicts an example of a detailed RBN $R_d$ which resulted from the gradual decomposition of the context RBN $R_c$ of fig. 3(a); a very small part of its underlying structure is depicted in fig. 2.

5. Validation of RBN specifications

Validation of requirements specifications is a hard process which refers to the production of specifications that are complete and consistent, and include no ambiguous or vague points. One way to validate the dynamic requirements specifications constructed by means of the methodology outlined above is graphical animation and execution of the RBN models, which can be performed at any level of abstraction, from context down to detailed RBNs.

Graphical animation is a valuable tool for reducing errors during requirements specification and for ensuring that the intended system behaviour has been properly captured and modelled. The proposed approach is supported by a set of tools which enable stepwise construction and graphical animation of the RBN models, and the latter can proceed either (a) in a user-driven fashion, where by “user” we mean a systems analyst working together with end-users of the application or (b) automatically. In the first case, the user decides which rules will fire and which will not, and comments on the obtained results, whereas in the second case, and for some given initial marking, the system evaluates all alternative paths (i.e. the preconditions of all triggered rules), executes the corresponding actions and produces various alternative markings. Therefore, it can be checked if and under what conditions certain RBN markings (and, consequently, certain system states) are reachable. Furthermore, missing rules, dead-end rules (i.e. rules which
can never fire), wrong rules (i.e., rules that produce wrong results) and circular rules can also be identified. User feedback can be used to correct any errors discovered, thus helping to design the rule base interactively.

![Diagram]

Figure 4. Detailed RBN $R_d$ resulting from the decomposition of the low-level RBN of fig. 3(d)

It should be noted that, although animation can be carried out at all abstraction levels of RBNs, user-driven animation makes sense only at the lowest abstraction level of detailed RBNs, where the user can easily see the contents of each rule and decide which rules to fire. At all other RBN levels animation is carried out automatically and, for a given initial marking, the user may only observe communication between rules, BUs, object classes and between the system and its external environment through signals.

However, although animation provides remarkable assistance in the case of small systems, larger
applications necessitate tools which can automatically detect contradictions, inconsistencies and redundancies in the model. The formal foundation of RBNs offers a sound basis for the development of such tools. More specifically, the successive mappings from context to detailed RBNs are performed by means of various transformations, formally defined in (Gouscos 1994), that decompose transitions and add some new places in each level, apart from enriching the structure of the resulting RBN with new features. The structural transformations applied during the top-down mapping towards detailed RBNs are in close analogy to the Fusion Parallel Transitions (FPT) and Fusion Series Transitions (FST) defined in (Murata 1989) which, together with some other transformations, have been shown to preserve the properties of liveness, safeness and boundedness of Petri nets. This is an important feature of our mapping, since some of these properties are desirable for RBNs, as well.

It is easy to see that usage of specialised forms of Petri nets facilitates the development of criteria for checking desirable properties. Of all the behavioural (marking-dependent) and structural (marking-independent) Petri net properties found in the literature, only a subset is of interest for RBN validation; this is due to the specific interpretation under which RBNs are used to model the dynamics of information systems. Under this interpretation, RBN places model signal types, RBN tokens model control signals, RBN markings model information system states, RBN transitions model the system dynamics and RBN firings model specific behaviour; therefore, the following properties are desirable for an RBN:

1. **Reachability**: if the information system must reach a state \( S \) corresponding to a marking \( M \), starting from a state \( S_0 \) corresponding to a marking \( M_0 \), then \( M \) must be reachable from \( M_0 \).

2. **Liveness**: if the information system must perform some behaviour \( b \) corresponding to a transition \( t \), starting from a state \( S_0 \) corresponding to a marking \( M_0 \), and this must be repeated for a finite number of times, then transition \( t \) must be L2-live under marking \( M_0 \).

3. **Persistence**: if the information system must perform some behaviour \( b_1 \) corresponding to a transition \( t_1 \) and some behaviour \( b_2 \) corresponding to a transition \( t_2 \) independently of one another and starting from a state \( S_0 \) corresponding to a marking \( M_0 \), then transitions \( t_1 \) and \( t_2 \) must be in a persistence relation under marking \( M_0 \).

4. **Structural liveness**: when there are signals available for all signal types which the information system receives from the external environment, the information system must be able to perform any specified behaviour \( b \), depending of course on values of the signal parameters and of the object instances available in the underlying structure. Therefore, under an initial marking \( M_0 \) corresponding to the state described before, all transitions must be live; this means that the net must be structurally live.
All these properties can be checked by constructing the reachability tree of the RBN under analysis. Furthermore, specific criteria might be formulated for RBNs, taking into account the particular form of these nets. An algebraic approach may be alternatively employed by working with the incidence matrix of the net under analysis.

6. Conclusions

This paper presents an approach for explicitly modelling the dynamic aspects of information systems in terms of dynamic rules within an object-oriented framework. The construction of requirements specifications starts from a static Object Schema which is included as one constituent in an abstract RBN and then gradually refined together with the overall RBN to lower levels of abstraction, in order to include the dynamic aspects too. The outcome of this process is a detailed RBN, whose underlying structure is a full-detail dynamic Object-Rule Schema containing all static and dynamic features of the modelled information system.

Dynamic features are explicitly represented in terms of rules which are grouped in behaviour units; behaviour units are in turn grouped under object classes, and all the rules of each behaviour unit are triggered by signals of a specific signal type. Rules, behaviour units and object classes inscribe the transitions of the various RBN models. RBN places represent signals exchanged between the information system and its external environment and within parts of the information system itself. Thus, RBNs represent control, rather than data flow within a system, as is the case with many traditional approaches to information systems development. Furthermore, RBN specifications can be validated either by formal validation techniques, or by graphical animation offering the advantages of rapid prototyping.

In conclusion, the strength of the approach proposed in this paper lies in the explicit modelling of business policy at various levels of abstraction, in the construction of graphical and executable specifications and in the employment of graphical animation and formal techniques for validation purposes. Furthermore, since the behaviour of each object class is localised in a unique transition of a 'high-level' RBN that is subsequently decomposed into a number of transitions and places corresponding to the BUs, rules and signals of that object class, RBN specifications may be considered compliant to the so-called localisation principle (Rolland 1992) which recommends that each object should be described in isolation from the others.

The proposed approach is supported by a set of tools which constitute the VENUS environment (Tsalgatidou et al. 1994). VENUS is being developed on Sun SPARC workstations running BSD UNIX, X11R5 and OSF/Motif, in ANSI C++ and Prolog. This environment is designed
to provide syntax-directed editors, graphical editors, animators and mapping tools for building entity-relationship models, mapping them to static object-oriented models, enhancing them with dynamic behaviour in terms of behaviour units, signals and dynamic rules, mapping them to RBNs at different abstraction levels, animating them etc. Using VENUS, a systems analyst may gradually construct and animate graphically RBNs which can be totally or partially folded and unfolded; partial folding and unfolding results in multi-level RBNs where the object class, behaviour unit or rule of interest can be pin-pointed. In this way, clear, concise, compact, modular and easily modifiable dynamic specifications can be formulated; the latter provide a basis for development of information systems that satisfy users' needs and are flexible enough to incorporate future changes of behaviour.

We are currently working towards the mapping of requirements specifications to design and implementation structures, so as to provide a complete methodology for information systems development. Further development of formal validation algorithms is also currently under study.

References


