EFFECTIVE INFORMATION
MODELLING SUPPORT
EFFECTIVE INFORMATION MODELLING SUPPORT

Proefschrift

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft
op gezag van de Rector Magnificus,
Prof. drs. P.A. Schenck,
in het openbaar te verdedigen
ten overstaan van een commissie
aangewezen door het College van Dekanen
op maandag 28 juni 1993 te 16.00 uur

door

Theodorus Franciscus Verhoef

informatica ingenieur

geboren op 6 februari 1967 te Amsterdam
Dit proefschrift is goedgekeurd door de promotor:

Prof. dr. H.G. Sol

en door de leden van de promotiecommissie:

Prof. drs. B.K. Brussaard
Prof. dr. ir. J.L.G. Dietz
Prof. dr. W. Gerhardt
Prof. dr. H. Koppelaar
Prof. dr. ir. R. Maes
Je suis condamné à être libre.
Cela signifie qu’on ne saurait trouver à ma liberté
d’autres limites qu’elle-même ou, si l’on préfère,
que nous ne sommes pas libres de cesser d’êtres libres.

Jean-Paul Sartre, 1943
Published and distributed by

T.F. Verhoef
Hennenberg 5
2641 LS Pijnacker
the Netherlands

The author acknowledges support by the Software Engineering Research Centre (SERC).

Cover design by Guido van der Harst.

CIP-DATA KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Verhoef, Theodorus Franciscus

Effective information modelling support / Theodorus Franciscus Verhoef. - [S.l. : s.n.]. - Ill.
Thesis Technische Universiteit Delft.
With index., ref. - With summary in Dutch.
ISBN 90-9006176-2
NUGI 855
Subject headings: information systems development / information modelling / knowledge acquisition.

© 1993 by T.F. Verhoef

All rights reserved. No part of this publication may be reproduced, stored in retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior permission of the author.
## Contents

1 Problem area

1.1 Introduction .................................................................................. 1
1.2 Understanding information systems development ............................ 2
1.3 Information modelling support ...................................................... 10
1.4 Improving information modelling support ..................................... 15
1.5 Research question and outline of the study ................................. 19

2 Approach to representing modelling knowledge ............................... 21

2.1 Introduction .................................................................................. 21
2.2 Information systems development method .................................... 22
2.3 Meta-modelling technique ............................................................. 23
2.4 Approach to acquisition of modelling knowledge ......................... 46
2.5 Conclusions ............................................................................... 54

3 Representing prescribed modelling knowledge ................................ 57

3.1 Way of modelling ......................................................................... 57
3.2 Way of working ........................................................................... 64
3.3 Relationship between a way of modelling and a way of working .... 69
3.4 Conclusions ............................................................................... 72

4 Representing applied modelling knowledge: Case 1 ....................... 77

4.1 Information Engineer A ................................................................. 77
4.2 Information Engineer B ................................................................. 90
4.3 Information Engineer C ................................................................. 100
4.4 Observations .............................................................................. 116
Chapter 1

Problem area

1.1 Introduction

In response to the strong post-war technology push, society has changed and is changing very rapidly. Today’s society is characterised by turbulence and complexity. Information systems have become an integrated and accepted part of our society, being used in widely varying fields such as chemical manufacturing, banking, space travelling, patient monitoring, and telephone switching. Many organisations nowadays rely heavily upon information systems. Information systems development has become a profession in itself.

This profession is, however, still in its infancy. Information systems development projects that exceed their planned time and budget even by as much as 100% are not uncommon. Deliverables that do not meet the users’ expectations, that are unreliable and not easy to use are regular news items.

From the mid-1970s onwards, various methods and techniques have been introduced in this field, all of which aim at improving the productivity of those developing information systems and the quality of the resulting information systems, see [Sol, 1985]. Each of those methods and techniques provides its own guidelines for managing development projects, carrying out development processes, and delivering development products.

A plethora of methods and techniques are available nowadays. Organisations tend to develop their own variants of existing methods, based on more or less successful experiences, see [Necco et al., 1987]. This situation has been referred to as “the methodology jungle” [Avison and Fitzgerald, 1988].

More recently, automated tools have been introduced to support the information systems development life cycle. These are variously referred to as workbenches, support environments or CASE tools (Computer Aided Systems Engineering). The latter term is preferred and used throughout this thesis. As Wijers points out [Wijers, 1991], the major benefits of CASE tools are in the support of documentation and verification tasks, the integration of various representation techniques and the automatic generation of new specifications and code. These benefits are explored in more detail in the literature, see [Butler Cox, 1987].
[Bubenko, 1988], [Martin, 1988], [Vonk, 1988a], [Vonk, 1988b], [Chen et al., 1989], and [McClure, 1989].

The research area addressed in this thesis concerns the support of the information systems development life cycle. In this chapter, we further refine the description of the research area. Starting from a problem solving perspective, we offer a terminological framework for understanding information systems development in section 1.2. This section also defines the notion of “information modelling”. In section 1.3 we show that methods, techniques, and CASE tools deliver “information modelling support” to a limited extent. Section 1.4 presents our view on “effective information modelling support”. This view leads us to formulate our research questions in section 1.5.

1.2 Understanding information systems development

1.2.1 A problem solving perspective

Information systems development can be looked upon as a problem solving process. Sol provides a general outline of problem solving processes [Sol, 1982], based upon [Mitroff, 1974]. He identifies the activities of conceptualisation, problem specification, solution finding, and implementation. The first two activities aim at delivering a model of the problem situation as perceived in reality, first in broad terms (conceptualisation), and then in detail (problem specification). The activity of solution finding departs from a specific model of the problem situation and ends in the choice of the most satisfactory solution. Implementation of the solution chosen influences the problem situation.

Sol characterises information systems development as a complex and ill-structured problem solving process (see [Sol, 1988]). As Bots clarifies (see [Bots, 1989]) this, there are two sides to the “structuredness” of problems. The first relates to the what: models that describe a problem situation. The second relates to the how: strategies for designing a solution to the problem situation. Thus by implication, two views on problem solving are distinguished: a product oriented view and a process oriented view. These two views are closely related, as indicated in [Sol, 1982]. He views a problem solving process as a sequence of transformations of models that describe a problem situation. Throughout this thesis we will use the following definitions to refer to the products of problem solving processes and the problem solving processes themselves.

**Definition 1.1** A model is a simplified, stylized representation of a system, abstracting the essence of the system’s problem studied.

**Definition 1.2** Modelling is the problem solving process of a problem as perceived in a given system.

These definitions have been adopted from [Wijers, 1991] and are applicable to all kinds of problems though here they are applied only to the problem of information systems
development. We therefore refer to problem solving processes in this context as *information modelling*, and to their (intermediate) results as *information models*.

Note that the definition of a model does not restrict the language used while representing the system at hand. Hence, the designation “model” covers a wide range of system representations, including mental representations, notes, diagrammatic representations, and executable specifications. From this point on we refer to languages used to represent a system as *model types* or *modelling techniques*, these being synonyms.

### 1.2.2 A framework for understanding

Sol puts great emphasis on understanding a problem situation before solving it (see [Sol, 1982]). We adopt a problem solving perspective on information systems development and because of this, we need to choose a vehicle for communication. To this end, we present a set of terms, a *framework*, for understanding information systems development as such, see figure 1.1. This framework is regarded as a useful means in solving methodical problems. The framework distinguishes a set of aspects that characterise information systems development processes: a way of modelling, working, controlling, thinking, and supporting. Information systems development methods (in short, methods) comprise only a way of modelling, working, controlling, and thinking.

![Figure 1.1: The framework for understanding information systems development](image)

A *way of modelling* provides means for structuring problems by distinguishing between
types of models required for problem specification and solution finding. A way of working provides means for structuring problems by distinguishing between types of tasks to be performed for problem solving. A way of modelling and a way of working relate to the product oriented view and the process oriented view of problem solving processes. Together, a way of modelling and a way of working involve the tasks to be performed and the model types to be used during information systems development processes. These tasks are referred to as operational tasks. In addition, managerial tasks are required to control the operational ones. These tasks constitute a way of controlling. Ways of modelling, working, and controlling vary between methods. Characteristics of these ways relate to an underlying philosophy regarding organisations and information systems. A way of thinking refers to such a philosophy. Finally, the collection of tools that support information systems development is referred to as a way of supporting.

The framework for understanding information systems development has been reported on in [Seligmann et al., 1989] and [Wijers, 1991]. These publications differ in a number of small ways in their presentation of the framework. Our presentation below is based upon, but differs slightly from, the latter.

1.2.3 Way of modelling

A way of modelling refers to the model types that are required to structure problem specification and solution finding. During information systems development, various aspects are of interest at different levels of detail. Therefore, several modelling techniques are usually required. We present a way of modelling as an interrelated network of concepts and relationships, verification rules, and external representations.

A large number of modelling techniques can be used in information systems development. The expressive power of a modelling technique is determined by the concepts it offers, and the possible relationships between these concepts. The application of various modelling techniques usually leads to a number of information models which may have several interdependencies. For example, data stores in a data flow diagram can relate to relationship types in an entity-relationship diagram. Therefore, relationships between concepts of different modelling techniques may exist.

Each modelling technique includes a set of verification rules to be satisfied. Some verification rules are invariant, i.e. they hold irrespective of the modelling stage in which the modelling technique is applied. Invariant verification rules are typically restrictions with respect to an upper bound to the number of instances of a specific concept or a specific relationship between concepts. For example, functions in a function decomposition hierarchy are never allowed to have more than one name, data flow diagrams are never allowed to be the decomposition of more than one process, and processes in data flow diagrams are never allowed both to be decomposed into a new data flow diagram and to be specified in a process specification.

Usually, concepts and relationships between concepts have an external representation. Information models that are expressed in a modelling technique are often represented by
diagrams, matrices or lists. Concepts are usually represented by graphic symbols, such as circles, arrows, and boxes. It should be noted that one concept can have more external representations, depending on its context. An entity for instance can be represented as a rectangle in a data model, and also as an item in an entity list. It is also possible that one external representation represents different concepts. A typical example can be found in SADT [Marca and McGowan, 1987]. Here, determining the concept that is represented by an arrow, depends on whether the arrow head is connected to the top or the bottom of a box. In the former case, the arrow represents a “control flow”, in the latter case it represents a “mechanism”.

To reiterate, a way of modelling defines an interrelated set of concepts, external representations, and invariant verification rules. It therewith constrains the types of information models that can be constructed in information systems development processes. Applying a way of modelling results in a number of (partial) information models. Information models that are expressed in a modelling technique are regarded to be specific instances of a way of modelling.

1.2.4 Way of working

A way of working refers to the process-oriented view of information systems development. It structures a problem solving strategy. This implies discerning several tasks and alternative orders of performing tasks. Choices between alternative orders are often determined by characteristics of the problem situation at hand.

Discussing a way of working includes discussing indications of how a specific task should be performed. Indications may vary from providing informal guidelines and suggestions to providing more structure by decomposing a task into subtasks, each covering a subproblem of the problem to be solved by the main task.

In specific situations, specific tasks are actually performed and specific decisions are actually made. Tasks and decisions that are part of an actual information systems development process are regarded as specific instances of a way of working.

Summarising, a way of working defines a set of tasks that can be performed, decompositions of tasks into subtasks and decisions, possible orders of tasks and decisions, and directives and heuristics on how to perform tasks and how to make decisions.

1.2.5 Relationship between way of modelling and way of working

Information systems development involves several task types. In [Olle et al., 1988a], five step categories are distinguished: abstraction, checking, form conversion, review, and decision. We distinguish between two types of tasks: modelling tasks and non-modelling tasks.

Modelling tasks are concerned with model manipulation, e.g. construction, transformation, verification, validation, and refinement. They may concern manipulation of instances of
concepts, for example removing a data store from a data flow diagram. Examples of non-modelling tasks are explanation and training tasks.

We use the notion of modelling task when discussing the relationship between a way of modelling and a way of working. Specific relationships between a way of modelling and a way of working concern the scope of modelling tasks, the consistency characteristic of modelling tasks, and information dependencies between modelling tasks.

The scope of a modelling task denotes the type of concepts that may be manipulated in that specific modelling task. Some tasks involve organisational schemata whereas others concern events to be handled by the information system to be developed. Tasks that are part of a way of working all have their own scope consisting of modelling concepts that are part of a way of modelling.

The consistency characteristic of a modelling task involves the set of constraints which must be satisfied after the specific modelling task has been performed. These consistency requirements are specified by certain verification rules, which are referred to as the post-conditions of that task. An example of a postcondition in state-transition diagrams is that each state is involved in a state transition. Various tasks that are part of a way of working all have their own postconditions which are expressed using modelling concepts that are part of a way of modelling.

Information interdependencies between modelling tasks concern the specific constraints between the resulting products of a modelling task and the resulting products of its successors. For instance, when performing the modelling task “Construct a data flow diagram” based upon the modelling task “Construct a context diagram”, all stimuli and responses that are part of the context diagram must be included in the data flow diagram. The result of the current task is thus constrained by the results of previous modelling tasks. Interdependencies between modelling tasks are specified in terms of specific modelling concepts. Again, a relationship between a way of modelling and a way of working is observed.

Information interdependencies also exist between modelling tasks and decisions. Decisions to modify existing models or to follow a certain strategy are heavily influenced by the so-called modelling context, that is by knowledge about the information modelling process history and by the models constructed so far. Decision making is guided by decision rules that refer to the modelling context. An example of such a decision rule is that if an event list is available a process oriented strategy should be preferred. Decision rules that guide the making of a decision that is part of a way of working refer to a modelling context containing existing models which are constructed using a specific way of modelling.

To reiterate, a way of modelling and a way of working are closely related, see figure 1.1. Specific relationships between a way of modelling and a way of working concern the scope of modelling tasks, the consistency characteristic of modelling tasks, and information dependencies between modelling tasks and decisions.
1.2.6 Three views on a way of modelling and a way of working

We have focused on a way of modelling, a way of working, and the relationship between these ways. Together, a way of modelling and a way of working involve operational tasks to be performed and modelling techniques to be used during information systems development processes. Choosing a way of modelling and a way of working in an information systems development project leads to an (often implicit) agreement on a set of conventions for the execution of information systems development processes and the structure of products resulting from these processes. Choices for specific conventions that constitute a way of modelling and a way of working can be discussed at three levels, referred to as the product view, the model view, and the component view, see figure 1.2. Distinguishing between these views enables us to refer to ways of modelling and ways of working more precisely.

<table>
<thead>
<tr>
<th>Product view</th>
<th>Way of modelling</th>
<th>Way of working</th>
</tr>
</thead>
<tbody>
<tr>
<td>products</td>
<td>stages</td>
<td></td>
</tr>
<tr>
<td>Model view</td>
<td>model types</td>
<td>modelling tasks</td>
</tr>
<tr>
<td>Component view</td>
<td>model components</td>
<td>model manipulations</td>
</tr>
<tr>
<td></td>
<td>process</td>
<td></td>
</tr>
<tr>
<td></td>
<td>flow</td>
<td></td>
</tr>
<tr>
<td></td>
<td>datastore</td>
<td></td>
</tr>
</tbody>
</table>

Figure 1.2: Three views on information modelling

The product view refers to milestone documents and deliverables, and to corresponding
stages in a development life cycle. Examples are a problem analysis document and a functional design stage. Project managers are usually interested in a way of modelling and a way of working from a product view.

The model view refers to model types that are used to construct models of various types as part of milestone documents, and to the order in which corresponding modelling tasks are performed. Examples are NIAM schemata, Bachman diagrams, and Jackson's entity structures, with regard to a way of modelling from the model view, and the corresponding modelling tasks, with regard to a way of working from the model view.

The component view makes explicit which model components have to be used to deliver models of a certain model type, which invariant verification rules have to be satisfied, and which graphic conventions are applicable. Moreover, it refers to the specific modelling tasks to be performed in order to construct a model of one model type, and their feasible order.

Directives and guidelines offered by methods often refer to only one or two views. Project management methods (the term is borrowed from [Olle, 1991]), refer to the product view. SDM [Turner et al., 1988] is a well-known example. Technique oriented methods (again, see [Olle, 1991]) focus on a way of modelling from model and component view. NIAM [Wintraecken, 1990] is an exponent of this category.

The three views are considerably interrelated. Discussing a way of modelling and a way of working from a certain view equals being interested in a certain level of detail. Choices for specific ways of modelling and working at higher levels constrain choices at lower levels. For example, if deliverables have been specified (product view), model types to be used have to conform to the report contents that have been determined (model view). In this example, the product view constrains the model view. Prescribing the use of Bachman diagrams implies the use of certain model components. Here, the model view influences the component view. The model view does not determine the component view however. For example, five different NIAM variants are distinguished in [Oosterom, 1990].

CASE tool developers follow a bottom-up approach while implementing knowledge about modelling techniques to be used and modelling processes to be performed. Model components have to be defined rigorously in order to implement repository structures that subsequently serve as a basis for editors which focus on specific views of the repository. CASE tools require precise knowledge of a way of modelling from the component view.

1.2.7 Way of controlling

A way of controlling constitutes a management perspective of information systems development processes. Project management concerns considerations of time, means (both manpower and facilities), and quality, see [Kensing, 1984] and [Sol, 1988]. A way of controlling, therefore, includes directives and guidelines on progress control, resource allocation, and quality management and control.

It should be clear that an effective way of controlling relates to a way of working and a way of modelling. A way of working provides a view on activities that are to be performed.
Resources are normally allocated to manageable units, usually stages in a systems development life cycle. Progress control implies attaching checkpoints and milestone documents to stages. A way of modelling indicates which products should be delivered. Quality management relates to deliverables. For example, models that result from information modelling processes are subject to a number of syntactic requirements that need to be fulfilled.

A close interaction between a way of controlling on the one hand and a way of working and a way of modelling on the other hand is part of our view on information systems development. In figure 1.1 this interaction is represented using the control paradigm, see [Brusseard and Tas, 1980].

Summarising, a way of controlling includes a set of directives and guidelines on information systems development project management, more specifically, on management of time, means, and quality aspects.

1.2.8 Way of thinking

Ways of modelling, working, and controlling differ among methods. Differences can be traced back to an underlying perspective [Mathiassen, 1981], Weltanschauung [Sol, 1983] or philosophy [Avison and Fitzgerald, 1988]. This underlying philosophy consists of a set of (often implicit) key assumptions which together constitute a specific view on information systems development.

A way of thinking is looked upon differently, see [Mathiassen, 1981], [Sol, 1983], [Bubenko, 1986], [Avison and Fitzgerald, 1988], and [Wijers, 1991]. In essence, a way of thinking constitutes assumptions with respect to problem solvers, problem domains, and problem solutions.

Regarding problem solvers, Bubenko refers to assumptions "about the skills and capabilities of the professional designers" (see [Bubenko, 1986]). Sol distinguishes between substantive rationality and bounded rationality as two different perspectives to describe problem solvers' behaviour [Sol, 1982]. In the former perspective, it is assumed that problem solvers have the disposal of perfect knowledge and unlimited processing power, in the latter perspective, problem solvers seek a satisfactory solution, not necessarily an optimal one.

In many publications, assumptions with respect to problem domains (variously referred to as domains, application areas, situations, problem classes, and target classes of applications) are considered to be part of a way of thinking, see [Mathiassen, 1981], [Floyd, 1986], [Kensing, 1984], [Bubenko, 1986], and [Avison and Fitzgerald, 1988].

In [Wijers, 1991], a way of thinking is related to assumptions about problem solutions, in particular about information systems. He mentions assumptions on information systems in relation to their environment, assumptions on components of information systems, and assumptions on characteristics of such components. Examples of such assumptions are: "information systems consist purely of computers", "information systems involve people", "information systems serve as a means for organisations", and "information systems include manual systems".
Summarising, a way of thinking relates to a set of underpinning assumptions on the problem solver, the problem domain, and the problem solution. A way of thinking consequently influences a way of modelling, a way of working, and a way of controlling, see figure 1.1.

1.2.9 Way of supporting

An understanding of information systems development can be achieved by distinguishing between a way of modelling, a way of working, and a way of controlling, embedded in a way of thinking. Tools, whether they are automated or not, support information systems development. A way of supporting refers to the collection of tools that offers support.

Examples of simple tools are pens, paper, whiteboards, overhead projectors, and other office facilities. More sophisticated aids are editors, compilers, debuggers, word processors, and CASE tools. These mostly support a way of modelling in different stages of the development life cycle. A way of working is supported by CASE tools that support model transformation, report generation, and method navigation. Project management tools are dedicated to support a way of controlling.

Several conferences, e.g. CRIS’88 [Olle et al., 1988b], stress the growing importance of an automated way of supporting. A way of supporting should fit neatly into ways of modelling, working, and controlling in order to offer effective support as depicted in figure 1.1, see [Wijers, 1991].

1.3 Information modelling support

We have presented a framework for understanding information systems development. The framework guides us to restrict the problem area. In this thesis we address the support of the information systems development life cycle, as far as a way of modelling and a way of working are concerned.

Three views have been presented from which a way of modelling and a way of working can be studied. From the product view, a way of working refers to stages in a development life cycle. Although any breakdown of a development life cycle into stages is considered to be arbitrary, we adopt the terminology of [Olle et al., 1988a]. They propose a life cycle which consists of twelve stages. We refer to the stages of information systems planning, business analysis, and system design as the early stages. Using this terminology, we further restrict the problem area of this thesis to the support of the early stages of the information systems development life cycle, as far as a way of modelling and a way of working are concerned. We restrict our use of the terms “information modelling”, “information model”, and “CASE tool” analogously.

The early stages are considered to be particularly important. Dunn indicates among other things that the most expensive errors to correct in development projects are those made in the early stages, not those made during implementation (see [Dunn, 1984]).

In spite of the introduction of methods, techniques, and CASE tools, the early stages are still poorly understood (cf. [Guindon and Curtis, 1988]). Activities in these stages are
characterised by incompleteness and vagueness [Belady, 1985]. They are concerned with integrating ambiguous, incomplete, and sometimes even conflicting views on the problem area (see [Parnas and Clements, 1986] and [Swartout and Balzer, 1982]).

In [Vitalari and Dickson, 1983], these activities are characterised as

- identifying important cues amidst a sea of extraneous information, setting goals.
- generating and testing hypotheses concerning the system attributes, processing dis-similar information from multiple sources, distinguishing between relevant and irrelevant information, reformulating the problem into a common nomenclature for technical and non-technical audiences, and finally specifying a consensus-based, error-free set of system requirements.

Methods, techniques, and CASE tools aim at improving productivity and quality of information modelling processes and resulting products. This section covers the question as to what extent these claims are justified, as far as the early stages are concerned. Section 1.3.1 focuses on methods and techniques. Section 1.3.2 deals with information modelling support by CASE tools.

1.3.1 Information modelling support: methods and techniques

The field of methods and techniques has become a research area in its own right. Bubenko observes that the research area of methods and techniques lacks a sound conceptual foundation (see [Bubenko, 1986]). Research has focused on methods and techniques as such (see section 1.3.1.1), and on the way in which information modelling processes as such are actually performed in practice, the extent to which methods and techniques are used to support information modelling processes in practice, and the way in which methods and techniques are applied in practice (see section 1.3.1.2). In the remainder of this thesis we distinguish between prescribed modelling knowledge, i.e. knowledge about a way of modelling and a way of working that is part of a method handbook on the one hand, and applied modelling knowledge, i.e. knowledge about the actual application of a way of modelling and a way of working in practice on the other hand.

1.3.1.1 Prescribed modelling knowledge

We present an overview of research approaches in the area of methods and techniques, similar to the historical outline of [Welke et al., 1991]. In the early 1980s, research focused on method comparisons. Sol points out that several distinct approaches can be used in evaluating and comparing methods [Sol, 1983]. Most comparisons have been normative in nature, i.e. they have used a list of criteria to be met by the methods that were subject to comparison, see among others [Olle et al., 1982], [Olle et al., 1983], [Shomenta et al., 1983], and [Colter, 1984]. These comparisons resulted in the insight that there is no one method that delivers the desired levels of productivity and quality. This situation has been referred to as the mirage of universal methods in systems design [Malouin and Landry, 1983], the search for the philosopher's stone [Parnas and Clements, 1986], or seeking a silver bullet [Brooks Jr, 1987].
Different design methods are suited to different classes of problems [Fairley, 1985]. In [Mathiassen, 1981] methods are actually characterised by their application area. Several publications have been based on this insight. These suggest a so-called toolkit approach, see [Wood-Harper et al., 1985] and [Benyon and Skidmore, 1987]. The essence of this approach is that a method should be selected or constructed from an available set of methods, based on contingency factors (cf. [Davis, 1982]). However, researchers proposing a toolkit approach do not give any guidelines for filling the toolkit with appropriate methods and techniques, and for performing an adequate selection from the toolkit given a specific problem situation, as [Welke et al., 1991] indicate.

The current generation of methods and techniques are subject to a new trend, i.e. paying attention to different strategies for development of new information systems, modification of existing information systems, and package selection. Examples of methods that include so-called scenarios are Information Engineering [Martin and Palmer, 1991] and Method/1 [Flaatten et al., 1989]. This trend can be seen as a variant of the toolkit approach. A toolkit is offered consisting of predetermined scenarios and guidelines for scenario selection.

The current generation of methods and techniques suffers from a number of deficiencies, see [Sol, 1985], [Bubenko, 1986], and [Essink, 1986]. Brinkkemper observes that this holds in particular for the early stages of information systems development [Brinkkemper, 1990]. Deficiencies relate to a way of modelling (fuzzy and artificial concepts), to a way of working (few guidelines, incompleteness), and to a way of controlling (lack of quality control), while the amount of support realised up to now, does not meet expectations with regard to increases in productivity and quality.

### 1.3.1.2 Applied modelling knowledge

Several recent publications deal with the extent to which methods and techniques are used in practice (cf. [Mathiassen and Munk-Madsen, 1985], [Connor, 1985], [Case Jr, 1985], [Yourdon, 1986], and [Chikofsky, 1988]). They all report a very limited use of methods and techniques by systems developers. According to [Connor, 1985],

the process of designing software systems is still an art (or craft) characterized by considerable folklore, black magic, and bursts of inspiration.

Scientific reports on practical experiences in the use of methods and techniques, dealing with their applicability, are rarely heard of. A notable exception is [Wijers, 1991]. Wijers describes, in a detailed fashion, the information modelling knowledge applied by three experienced information engineers, focusing on their way of modelling and way of working. He does, however, not compare these applications to the application as suggested by the respective method handbooks. In [Bansler and Bodker, 1993], a comparison between handbooks and application can be found, however, at a very global level and with a strong emphasis on the organisational context of the use of methods and techniques.

The way in which information engineers actually perform information modelling processes in practice, has been subject to several research studies. Most of these studies have focused on experienced subjects. Results focus both on knowledge used by information engineers
and on the information modelling process itself. Again, a product oriented and a process oriented view on information modelling are noted. We explore these views one by one.

Vitalari explores what he calls “the content of the analyst’s knowledge base” [Vitalari, 1985]. His study shows that the behaviour of experienced information engineers is supported by different knowledge bases: organisation specific knowledge, functional domain knowledge, application domain knowledge, and method knowledge. Guindon considers the last category to be crucial, as it provides problem independent suggestions [Guindon, 1990b]. Wijers concludes with respect to a way of modelling that expert information engineers use refined modelling concepts in diagrams, such as concrete versus conceptual entities, and primary process flows versus secondary process flows. He observes that they also explore non-diagramming concepts throughout information modelling processes. These concepts mainly cover organisational aspects, examples being “objective”, “problem”, and “primary process” [Wijers, 1991].

Information modelling processes as performed in practice tend to refine or even to deviate from ways of working as suggested by methods. Guindon summarises her observations as follows: (i) design appears to be a collection of interleaved, iterative, loosely-ordered processes under opportunistic control, (ii) top-down balanced development appears to be a special case, and (iii) good designers work at multiple levels of abstraction and detail simultaneously (see [Guindon, 1990a]). Several authors refine the notion of opportunistic control, see [Ballay, 1987], [Stomph-Blessing, 1988], and [Wijers, 1991]. They note that intermediate models constructed in an information modelling process influence the course of subsequent information modelling processes and the nature of subsequent models. Ballay observes that information engineers tend to highlight different aspects by choosing for various modelling techniques while focusing on one subproblem, or to switch between subproblems while focusing on one aspect [Ballay, 1987]. Summarising, the above authors all conclude that choices for a specific way of working are strongly dependent on the modelling context.

Information modelling processes are heuristic in nature. Guidelines are applied frequently. Guindon gives several examples of guidelines: “consider a simpler problem and later expand the solution”, “simulate scenarios in the problem domain to acquire more information about the problem structure”, and “divide the system into loosely coupled subsystems and conquer” [Guindon, 1990b].

Wijers points out, with regard to a way of working, that information engineers apply individual strategies with preferences for specific modelling techniques in order to solve methodical shortcomings of the methods and techniques applied, and that they are well aware of their own strategies [Wijers, 1991]. Another observation he presents is that consistency characteristics gradually change during information modelling processes.

We conclude that methods and techniques support information modelling processes to a limited extent only, with regard to a way of modelling as well as a way of working. Modelling knowledge as applied by experienced information engineers turns out to deviate from prescribed modelling knowledge, regarding both a way of modelling and a way of working. Bomans formulates it in a poetic way: “In the realm of spirit, methods may be compared to a crutch. Real thinkers walk freely.” [Bomans, 1962].
1.3.2 Information modelling support: CASE tools

Introduction of CASE tools in organisations involves a considerable amount of investment. Grochow estimates that the equipping of a 150-person IS organisation with CASE technology will cost three million dollars [Grochow, 1988], nevertheless, CASE tools have gained a certain popularity, which is reflected in their sales worldwide.

According to [McClure, 1987] over a hundred vendors have introduced CASE products into the market, some of these are project management tools, others cover parts of the development life cycle. Generally, the latter are divided into two categories. CASE tools supporting the former stages in the life cycle are referred to as upper CASE tools or front-end tools. Lower CASE tools or back-end tools support the later stages in the life cycle. A small number of CASE tools claim both to deliver integrated support for the entire life cycle and to cover project management aspects. These tools are called IPSEs (Integrated Project Support Environments).

Since we have restricted the problem area to the support of the early stages of the information systems development life cycle, we focus on upper CASE tools when dealing with a way of supporting. For convenience's sake, we refer to upper CASE tools as CASE tools in this thesis.

The current generation of CASE tools does not confirm productivity and quality claims. On the contrary, as a consequence of the introduction and use of CASE tools, productivity decreases have been reported, see [Norman and Nunamaker, 1989] and [Chen et al., 1989]. Reports on actual contributions to quality increase have not been found, although some of the organisations that have introduced CASE tools have perceived a systems quality increase [Everest and Alanis, 1992].

In [Voelcker, 1988] it is mentioned that especially upper CASE tools remain an area of concern:

> The required level of knowledge turns out to be difficult to imbed in adequate computer aids for information engineers in the early stages of systems development.

Studies on the use of CASE tools in the Netherlands (see [NGGO, 1988] and [Kusters and Wijers, 1992]) provide more specific results. It is shown that currently CASE tools are mainly employed for the support of documentation and verification activities. Guidelines on why and how to perform modelling tasks are not part of automated tools. These studies confirm that CASE tools are product oriented. They focus on a way of modelling while neglecting the support of a way of working.

CASE tools are found to provide rigid and unflexible support (e.g. [Smolander et al., 1991]). A view of information modelling processes to be supported is not explicitly available in these tools. Instead, modelling knowledge has been hard-coded as part of the repository of CASE tools. It is not possible to tailor or customise CASE tools in such a manner that modelling knowledge based on information engineers' practical experiences can be included. Therefore, information engineers have to adapt their way of working to CASE tools instead of vice versa.
It can be concluded that CASE tools support information modelling processes to a limited extent only. Modelling knowledge as applied by information engineers does not correspond to modelling knowledge supported by CASE tools. Information engineers have to adapt their way of working. As a consequence, employment of CASE tools is mainly reduced to documentation and verification activities.

1.4 Improving information modelling support

Methods, techniques, and CASE tools all support information modelling processes to a limited extent only, see section 1.3. Several research approaches that focus on improving and comparing methods and techniques as such have not turned out to be successful. Modelling knowledge as applied in practice tends to deviate from modelling knowledge offered by method handbooks, see section 1.3.1. Section 1.3.2 has shown that information engineers have to adapt their working styles to the current generation of CASE tools instead of the other way around. CASE tools are considered to be too rigid and inflexible to provide adequate support. In this section, the focus is on improving information modelling support by addressing the issue of flexibility. Section 1.4.1 introduces a meta-modelling approach to incorporating flexibility in CASE tools. This approach explores the notion of “CASE shell”, to adapt a CASE tool to the way of working of individual information engineers. Section 1.4.2 points out, however, that the meta-modelling approach has some serious drawbacks.

1.4.1 Information modelling support and flexibility

We discuss the availability of explicit modelling knowledge consisting of both a way of modelling and a way of working as a means to improve information modelling support. Preferably, information engineers should be able to specify, to adapt, to refine, and to extend modelling knowledge as part of CASE tools, according to their own practical experience. The key assumption is that the availability and the use of explicit knowledge about the way modelling processes are performed result in more effective support of information systems development processes (e.g. [Smolander et al., 1991] and [Wijers, 1991]). Information modelling support environments that dispose of explicit modelling knowledge can be customised to individual ways of modelling and ways of working. Such flexible support environments allow information engineers to preserve their own working style. We share this key assumption, as it fits our ideas on effective support.

In order to structure the discussion of flexible information modelling support environments, three levels of abstraction are distinguished. Several authors distinguish between these levels of abstraction, see [Brinkkemper and Falkenberg, 1991], [Imber, 1991], [Smolander et al., 1991], [Wijers, 1991], [Araujo and Carapuça, 1992], [Brough, 1992], and [Heym and Österle, 1992]). Figure 1.3, taken from [Wijers, 1991], shows these levels of abstraction. In this hierarchy, each level can be seen as the result of an instantiation process of the next higher level. At each level, models constitute a population of the next higher level. Note
that we use the term “instantiation” to refer to the process, and the term “population” to refer to the corresponding product.

At application level, we are concerned with the development of a specific information system. As discussed before, we distinguish between actual information models of the application domain constructed according to a specific way of modelling, and the actual information modelling processes performed according to a specific way of working. At this level, information engineers apply modelling knowledge to do their job.

The next higher abstraction level, the so-called meta-level, concerns the development of specific modelling knowledge. Again, we discern a product oriented view and a process oriented view. The product oriented view involves actual models that represent modelling knowledge. These models are so-called meta-models. They prescribe a way of modelling and a way of working to be applied by information engineers at application level. The process oriented view concerns the process of specifying modelling knowledge, the meta-modelling process.

Analogously, the theory level is concerned with defining the meta-level, i.e. it defines how meta-modelling processes should be performed, and it prescribes a meta-modelling technique to be applied by so-called method engineers at meta-level. The theory level is also referred to as the “meta-meta-level” (see [Imber, 1991], [Smolander et al., 1991], and [Araújo and Carapuça, 1992]). Brough calls it the “method definition level” [Brough, 1992]. The term “methodology level” is also used [Heym and Österle, 1992].

1The prefix meta indicates a higher level of abstraction.
We want to stress the difference between the views in figure 1.2 and the hierarchy of levels of abstraction in figure 1.3. Models at different levels of abstraction concern different domains. A model at application level represents an application domain, for instance bookkeeping, a model at meta-level represents modelling knowledge, for instance the use of functional hierarchy diagrams. Models from different views are more or less detailed representations of one and the same domain, i.e. modelling knowledge, and are therefore all placed at meta-level.

Having discussed the three levels of abstraction, we are able to discuss adaptive information modelling support environments in more detail. Such support environments are receiving a growing interest. They incorporate modelling knowledge used (meta-level) and application models constructed (application level). These environments are modelling knowledge independent and are implemented as such as so-called CASE shells. A CASE shell is itself without knowledge of any specific information modelling process, but requires modelling knowledge to be offered in a specific format. By incorporating this knowledge in the format required, the CASE shell is transformed into an information modelling support environment for that specific modelling process.

One of the earliest CASE shells reported was SEM (System's Encyclopedia Manager) [Teichroew et al., 1980]. Similar tools are MetaView, MetaPlex, and RAMATIC, see [Sorenson et al., 1988], [Chen and Nunamaker Jr, 1989], and [Bergsten et al., 1989] respectively. These CASE shells all restrict themselves to the support of a way of modelling, although they emphasise different aspects. MetaView and MetaPlex focus on support of the network of concepts and their relationships, while in RAMATIC the importance of support of external representations is stressed.

Wijers proposes a generic architecture for information modelling support environments [Wijers, 1991]. This proposal pays attention to adaptive support of a way of working as well. We present this architecture in figure 1.4.

![Diagram](image)

Figure 1.4: A generic architecture for information modelling support environments

According to this architecture, information modelling support environments deliver auto-
mated support to information engineers. At application level, an *interpretation mechanism* interprets explicit modelling knowledge including both a way of modelling and a way of working. Information engineers perform actual modelling processes according to that way of working, and deliver application models according to that way of modelling. An "instance of" relationship exists between the resulting application models and the actual information modelling process history on the one hand and the modelling knowledge used on the other hand. Application models and modelling process history are stored in an application knowledge base. The *meta-model editor* enables method engineers to specify and adapt modelling knowledge. Meta-models are stored in a *meta-knowledge base* at meta-level.

To summarise, we share the assumption that the explicit availability of modelling knowledge consisting of both a way of modelling and a way of working contributes to improving information modelling support. In [Wijers, 1991], it is shown that it is feasible to develop an information modelling support environment that support information modelling processes based on explicitly available modelling knowledge.

### 1.4.2 Information modelling support and effectivity

The discussion so far has focused on improving information modelling support for *individual* information engineers. We emphasise that this is a partial discussion. This section discusses two drawbacks of the meta-modelling approach.

*One*, as it is stated in [Wijers, 1991]:

> Although individual and adaptable modelling support is effective to the *individual* information engineer, it is not necessarily so for the *organization* for which this person is working. Support tuned to the individual may contradict the organizational need for standardization. Preferably the results of one information engineer are transferable to another. [...] some parts of the modelling knowledge used by an information engineer need to be standardized, other parts are free or only recommended. Future research should find mechanisms for this problem of managing the "modelling knowledge base" used within an organization.

A *balance* is apparently required between two conflicting requirements, flexible and personalised support of modelling tasks for individual information engineers on the one hand, and coordination of individual modelling tasks on the other hand. Coordination in the domain of information modelling mainly amounts to *transferability* of results.

*Two*, it is, to say the least, not clear how to respond to the economic question, whether it is worthwhile to specify a detailed modelling knowledge base for each information engineer individually. Experiences from [Wijers, 1991] are, however, not in favour of a positive response.

The former discussion leads us to formulate our view on effective information modelling support. We consider information modelling support to be effective, if it satisfies the following requirements:
facilities are available to standardise parts of a way of modelling and a way of working (coordination);

facilities are available to tune and extend organisational standards towards individual modelling support, with regard to a way of modelling and a way of working (flexibility).

In [Sol, 1991], experiences are presented that are in line with the above. These experiences apply not only to information engineers, but to information workers in general. It is stated that effective support of information workers can be improved to a considerable amount by the exploitation of explicit knowledge about the actual performance of modelling tasks. According to studies in several problem domains, such as the public sector [Dur and Bots, 1992], insurances [Dur, 1991], banking [Motshagen, 1991], and maintenance [Weelderen, 1991], it is essential to understand the dynamics of the primary process both from micro-perspective and from meso-perspective. In our case, the primary process refers to information modelling itself.

1.5 Research question and outline of the study

We have focused our attention on effective information modelling support. We have restricted the notion of information modelling to a way of modelling and a way of working during the early stages of the systems development life cycle. We have tentatively described the notion of “effective information modelling support” as information modelling support, where the property of flexibility is considered to be necessary. This implies that modelling knowledge (covering both a way of modelling and a way of working) is explicitly available. Finally, we have shown some pitfalls when dealing with flexibility as such, leading to a specific view on the notion of “effective information modelling support”.

In order to achieve a more effective information modelling support, insight is required into the extent to which and the manner in which modelling knowledge by individual information engineers deviates from modelling knowledge as prescribed by e.g. a method handbook. This insight may provide clues as to possible ways of tuning generic and standardised modelling knowledge towards individual modelling support. As stated before, little attention has been given in the literature to the applicability of modelling knowledge. Furthermore, insight is required into ways in which standardised or prescribed method knowledge may be adapted towards individual needs with regard to a way of working and a way of modelling.

We formulate our main research questions as follows:

1. Is it possible to acquire a detailed insight into differences and similarities between prescribed modelling knowledge and applied modelling knowledge?

2. Is it possible to develop mechanisms that support the adaptation of generic modelling knowledge towards individual modelling support?
This thesis is structured as follows. To get a detailed insight, both in prescribed and applied modelling knowledge, especially in similarities and differences between prescribed and applied modelling knowledge (research question 1), we perform an extensive investigation. In chapter 2, we present the approach we use to acquire and represent both prescribed and applied modelling knowledge. The results of applying our approach are presented in the three subsequent chapters. Chapter 3 deals with prescribed modelling knowledge, while chapters 4 and 5 are involved with representing applied modelling knowledge. Chapter 6 discusses conclusions, both on similarities and differences between prescribed and applied modelling knowledge. Based on these conclusions, we propose a model for effective information modelling support in chapter 7 (research question 2). This chapter also shows some applications. In chapter 8, the findings are evaluated.
Chapter 2

Approach to representing modelling knowledge

2.1 Introduction

In chapter 1, we have motivated our interest in acquiring a detailed insight in both prescribed and applied modelling knowledge. Therefore, we have to be able to represent modelling knowledge. Chapter 2 concerns our approach to representing modelling knowledge.

It is important to recall from chapter 1 that representing modelling knowledge involves two different levels of abstraction: the theory level and the meta-level (see figure 1.3, page 16). Developing an approach to representing modelling knowledge is a theory level activity; note that developing an approach includes choosing a meta-modelling technique. Applying the approach developed is a meta-level activity, the next lower level of abstraction; note that this results in specific meta-models, which represent both prescribed and applied modelling knowledge.

In sections 2.2 and 2.3 we choose a specific information systems development method as well as a meta-modelling technique. These choices are essential to the representation of both prescribed and applied modelling knowledge. With regard to the representation of applied modelling knowledge, we also have to develop an approach to acquiring modelling knowledge. Section 2.4 presents the approach to acquiring modelling knowledge.

Both the development and the application of our approach to representing modelling knowledge are summarised in figure 2.1. This figure presents an outline of chapter 2 in relation to the three following chapters. In this figure, we use the task structure diagramming technique which is introduced in section 2.3.3.
2.2 Information systems development method

We first present our choice of an information systems development method. The information systems development method to be chosen has to satisfy three requirements. One, it should cover the early stages of the systems development life cycle since our problem area concerns these stages. Two, it should be a well-established method, since practical experience must be available (specifically, in the Netherlands). Three, literature giving an in-depth description of the method should be available. Note that the second requirement is necessary to represent applied modelling knowledge, whereas the third requirement is necessary to represent prescribed modelling knowledge.

The Yourdon method covers the early stages of the systems development life cycle. According to [Holloway, 1992], Information Engineering and Yourdon are among the most widely used methods. In the Netherlands in particular, the Yourdon method is often used, hence, Yourdon also fits the second requirement. Literature on Yourdon is readily available. It is even better practice to refer to the Yourdon method as a method variant group, since several authors have published their own variants, see [DeMarco, 1978], [Gane and Sarson, 1979], [McMenamin and Palmer, 1984], [Ward and Mellor, 1985b], [Ward and Mellor, 1985a], [Ward and Mellor, 1986], [Hatley and Pirbhai, 1987], and [Yourdon, 1989]. Two major drawbacks of the several variants are the poor support of the later stages of the systems development life cycle and the exclusion of a way of controlling. As these drawbacks do not conflict with the three requirements, and as the Yourdon method satisfies the three requirements, we decide to choose the Yourdon method as our information systems development method.

The several variants put emphasis on different aspects, new variants have introduced new
insights, especially on the importance of analysing the current situation, on approaches to be followed, on notations to be used, and on application domains. With regard to the importance of the current situation, it is argued, see [DeMarco, 1978], that it is necessary to develop four subsequent system models: the current physical model, the current logical model, the new logical model, and the new physical model. In [Gane and Sarson, 1979], however, another opinion is found, the current physical model is abandoned. According to [Yourdon, 1989], the current situation must not usually be paid attention to, since “the process of developing a model of the current system may require so much time and effort that the user will become frustrated and impatient, and ultimately cancel the project”. The early variants suggested a top-down approach. In [McMenamin and Palmer, 1984] the event partitioning approach has been introduced, which has received growing attention. Different variants employ different notations for the same concepts. Examples of differences between the variants of [DeMarco, 1978] and [Gane and Sarson, 1979] can be found in [Gane, 1990]. A new application domain has emerged: real-time systems. This domain has been addressed extensively in [Ward and Mellor, 1985b], [Ward and Mellor, 1985a], [Ward and Mellor, 1986], and [Hatley and Pirbhai, 1987].

All variants have in common that they employ the data flow diagramming technique and the entity-relationship diagramming technique. These techniques are wide-spread, see e.g. [NGGO, 1990].

We base our representation in this chapter on the most recent version, i.e. [Yourdon, 1989].

2.3 Meta-modelling technique

In order to represent the modelling knowledge of [Yourdon, 1989], a meta-modelling technique has to be used that helps in representing both a way of working and a way of modelling. Several modelling techniques have been employed as meta-modelling techniques. Most of them only pay attention to representing a way of modelling, see for example the CDIF meta-model [Imber, 1991] and the Object-Property-Role-Relationship model [Smolander et al., 1991]. These meta-modelling techniques are variants of the entity-relationship diagramming technique [Chen, 1976]. In [Brinkkemper, 1990] the NIAM variant of [Nijssen and Halpin, 1989] is used in order to represent concepts and relationships between concepts, complemented by first order predicate calculus to express verification rules.

Very few authors cover a way of working as well. One of the first attempts is found in [Brinkkemper, 1990]. He uses natural language to formulate so-called modelling procedures in an informal way. In [Wijers et al., 1990], a meta-modelling technique is presented which has been developed intentionally to represent a way of modelling and a way of working, as well as their interrelationships. The meta-modelling technique has two main components: a concept structure diagramming technique, which is used to represent a way of modelling, and a task structure diagramming technique, which is used to represent a way of working.

The present meta-modelling techniques are, however, incapable of coping with the complex domain of modelling knowledge in a natural way. Welke gives several examples of concepts in modelling knowledge that correspond to complex structures [Welke, 1988]:
DFDs [Data Flow Diagrams] beget (lower-level) DFDs, the predominantly hierarchical structures of a Structure Chart are broken off at arbitrary levels and spawn, in turn, more detailed (lower level) diagrams, E-R diagrams are done at various levels of specificity (Subject, Class, Individual Entity).

The current generation of meta-modelling techniques "flatten" these complex structures, which leads to overspecification [Hofstede et al., 1992a]. In [Wijers, 1991] it is concluded that the introduction of the notion of complex concept to reduce overspecification would be a useful extension to meta-modelling techniques.

The meta-modelling technique to be chosen has to satisfy two requirements. In the first place, it should be possible to represent a way of modelling and a way of working, as well as their interrelationships. Second, it should be possible to represent complex structures and constraints as part of a way of modelling.

The area of semantic data models (see [Hull and King, 1987], [Peckham and Maryanski, 1988], and [Bekke, 1991]) provide various constructs for relating objects. One of these constructs, often referred to as object grouping or association, involves the construction of new object instances, these being finite sets of existing object instances. This construct is appropriate for representing complex structures. As a consequence, object grouping is a useful construct when dealing with the complex domain of modelling knowledge. The area of semantic data models, however, fails to pay adequate attention to a way of working, see our first requirement.

A data modelling technique which also includes such grouping constructs is presented in [Hofstede and Weide, 1993]. This technique is called the Predicator Set Model (PSM). Using PSM leads to the specification of a so-called information structure. A language that can be used to specify verification rules and updates over PSM information structures is presented in [Hofstede et al., 1992b]. This language is called Language for Information Structure and Access Descriptions (LISA-D). The task structure diagramming technique presented in [Hofstede and Nieuwland, 1993] can be used to specify tasks to be performed as part of an information systems development process, and their interrelationships.

In [Hofstede, 1993], considerable attention is paid to the expressive power of PSM from a theoretical point of view, resulting in three particular conclusions. One, PSM is powerful enough to model its own syntax. Two, the relevant axioms of set theory (see for example [Crossley et al., 1972]) can be compared to the main concepts in PSM. This comparison also motivates the necessity of a powerful constraint language (LISA-D). Three, it is demonstrated that context free grammars can be translated to PSM schemata.

Application of the data modelling technique, the data manipulation language, and the task structure diagramming technique to the domain of modelling knowledge leads to a metamodeling technique, which satisfies our requirements. PSM is primarily used to represent a way of modelling. The task structure diagramming technique is used to represent a way of working. LISA-D serves for the representation of interrelationships between a way of modelling and a way of working. LISA-D can also be used for the specification of verification
rules. We use a combination of PSM, the task structure diagramming technique, and LISA-D as our meta-modelling technique.

<table>
<thead>
<tr>
<th>Modelling knowledge</th>
<th>Meta-modelling technique</th>
</tr>
</thead>
<tbody>
<tr>
<td>structure of concepts</td>
<td>2.3.1. Object structures</td>
</tr>
<tr>
<td>invariant verification rules</td>
<td>2.3.2. Constraints</td>
</tr>
<tr>
<td>external representations</td>
<td>2.3.3. Task structures</td>
</tr>
<tr>
<td>structure of tasks and decisions</td>
<td>2.3.4. Relation between Object structures and Task structures</td>
</tr>
<tr>
<td>scopes of tasks</td>
<td>2.3.2. Constraints</td>
</tr>
<tr>
<td>information interdependencies between tasks</td>
<td></td>
</tr>
<tr>
<td>task specific verification rules</td>
<td></td>
</tr>
</tbody>
</table>

Figure 2.2: Modelling knowledge in relation to the meta-modelling technique

This meta-modelling technique is discussed in four sections, see also figure 2.2. We first introduce the data modelling technique PSM. In the context of meta-modelling, we refer to the result of the use of PSM as an Object structure. Object structures represent complex structures of concepts as part of a way of modelling, see section 2.3.1. Section 2.3.2 pays specific attention to constraints. In section 2.3.3 we present the task structure diagramming technique. Use of this technique results in a Task structure. Task structures represent ways of working. Finally, we present the representation of the relationships between Object structures and Task structures in section 2.3.4.

It is clear from figure 2.2 that the external representations of concepts are not dealt with in this thesis. We give an in-depth view of this issue in [Hofstede et al., 1992d] and [Hofstede et al., 1992e].

### 2.3.1 Object structures

PSM is used to represent a way of modelling, more specifically, to represent important concepts in information systems development methods and their interrelationships. Such a representation is referred to as an Object structure.

In this section the main concepts in Object structures are introduced informally. This section is based on [Hofstede and Weide, 1993].
2.3.1.1 Object types

One of the key concepts in Object structures is that of Object type. Many synonyms have been used to refer to this data modelling concept, such as type, object class, object category, concept, and entity type.

Special kinds of Object types are Entity types and Label types. In contrast to Entities, Labels can be represented on a communication medium.

2.3.1.2 Relationships between object types

Relationships relate an arbitrary number of Object types, and consist of a number of Roles. A Role denotes the way an Object type participates in a Relationship.

A Relationship may be treated as an Object type itself (objectification), and can therefore play a Role in other Relationships.

![Figure 2.3: A simplified Object structure of State Transition Diagrams](image)

Figure 2.3 illustrates the concepts of Object type and Relationship. This figure shows an Object structure representing State Transition Diagrams (STDs) from [Yourdon, 1989]. According to Yourdon, valid state changes in STDs are shown by connecting relevant pairs of states with arrows. In figure 2.3, the Object type State is represented by a circle. A Relationship between states is discerned. This Relationship consists of two Roles which are represented by boxes: from and to. This Relationship is objectified into a new Object type: State transition. As a consequence of this objectification, state transitions can have a Relationship with other Object types, in this case with the Object types Action and Condition. These Relationships consist of two roles: triggers and is-triggered-by, and is-caused-by and causes, respectively.

For completeness' sake, we have provided the Object types Action, Condition, and State with their identifying Label types: A-name, C-name, and S-name. From now on, we omit Label types in the graphical representation of Object structures for simplicity. At the same time, we assume that Object instances are uniquely identified by Label instances.
In the case of \( n \)-ary Relationships \((n > 2)\), we do not name the involved Roles individually. Instead, we assume the set of Roles to be ordered, and provide a sentence which describes how the \( n \) Object types are related. In the example of figure 2.4, a ternary Relationship relates the Object types "Arrow", "Box", and "Box". The sentence "... relates ... to ..." is based on this ordering.

**2.3.1.3 Object abstraction**

Object abstraction results in the construction of a new Object type based on one or more existing Object types, provided that instances of the new Object type occur in the population of the existing Object types. Two mechanisms are distinguished for object abstraction: specialisation and generalisation. In semantic data modelling, the "is..a" relation is typically used to refer to these mechanisms.

Specialisation, also referred to as subtyping, is a mechanism for constructing one or more Object types based on one existing Object type. The former Object types are referred to as Subtypes, the latter is referred to as the corresponding Supertype.

A specialisation relation between a Subtype and a Supertype implies that the instances in the population of the Supertype may also be instances in the population of the Subtype,
whereas all instances in the population of the Subtype occur in the population of the Supertype. The population of the Subtype is a subset of the population of the Supertype. If a Supertype is related to several Subtypes, these Subtypes are allowed to share instances. Specialisation is a non-strict, non-total partitioning mechanism.

Subtypes are required to be defined in terms of their Supertype. For each of the subtypes, a constraint has to be formulated (referred to as the Subtype defining rule, see also [Bommel et al., 1991]), which is to be used as a criterion to decide whether instances of a Supertype belong to the Subtype involved.

Object types that participate in a specialisation relation as a Subtype may be a Supertype in other specialisation relations. Specialisation relations are organised in specialisation hierarchies. A specialisation hierarchy is an acyclic directed graph with a unique top. The top of a specialisation hierarchy is referred to as the pater familias [Troyer et al., 1988]. Identification of Subtypes is derived from their Supertypes, as Object types inherit all properties from their ancestors in specialisation hierarchies.

Specialisation is illustrated in figure 2.5, which shows a specialisation hierarchy. Each specialisation relation is represented as an arrow. The Object type Flow is the pater familias of this specialisation hierarchy. The example shows that Subtypes are allowed to share instances. Periodic flows for instance are either internal or external. In this example, the Subtype defining rule may specify that External flows are those Flows which have a Process of type “External agent” as source or destination.

Generalisation is a mechanism that allows for the creation of one new Object type by uniting existing Object types. The former Object type is referred to as the Generalised object type, the latter are the corresponding Specifiers. The population of the Generalised object type is required to be equal to the union of the populations of the Specifiers.

Analogous to specialisation, generalisation relations are organised in hierarchies. Both identifications and properties of Object types in a generalisation hierarchy are derived from their descendants, i.e. from the leaf Object types in the hierarchy.

![Figure 2.6: Example of a generalisation hierarchy](image-url)
A generalisation hierarchy is illustrated in figure 2.6. This example shows a generalisation hierarchy which represents boxes in Jackson's entity structure diagrams (see [Jackson, 1983]).

### 2.3.1.4 Object composition

Three mechanisms are distinguished for object composition, resulting in the creation of a new Object type based on one of more existing Object types. The existing Object types are referred to as Element types, the constructed Object types are called Set type, Sequence type, or Schema type, depending on the mechanism. In semantic data modelling, the is_member_of relation is a typical analogon.

The concept of Set type can be compared to power sets in conventional set theory [Levy, 1979]. An instance of a Set type is a set of instances of its Element type. Such an instance is identified by these Element instances.

![Diagram](Figure 2.7: A simple example of a Set type)

A simple example is depicted in figure 2.7. In this figure the Object type Entity group is a Set type, having Entity as Element type. As a result, each Entity group instance is a set of Entity instances. Entity groups are identified by their constituent Entities, whereas Entities can be identified by their Name.

![Diagram](Figure 2.8: A simple example of a Sequence type)

Sequence types represent sequences of Element types. Sequence types are ordered Set types. Their instances are tuples of Element instances of arbitrary length. Figure 2.8 shows an example of Entity lists, where ordering of entities is considered to be important.

Schema types define (part of) a schema as a new Object type. An instance of a Schema type is a population of the associated schema.
As an example of a Schema type, the Object structure which represents STDs from [Yourdon, 1989] is shown in figure 2.9. Here, STD is a Schema type, containing States that can be further decomposed in STDs.

### 2.3.2 Constraints

Verification rules explicitly exclude certain populations of an Object structure. This section presents constraints which are used to specify verification rules. Note that constraints are found at the theory level, and verification rules at the meta-level, see figure 1.3, page 16.

Some types of verification rules occur frequently. In order to facilitate the specification of such verification rules, some constraints have been assigned a graphical convention. We refer to these constraints as graphical constraints. Section 2.3.2.1 presents the various graphical constraints.

We use the data manipulation language LISA-D to specify non-graphical verification rules. Therefore, we introduce the concept of relationship descriptor in section 2.3.2.2. This basic concept is employed in section 2.3.2.3 to specify verification rules that cannot be specified graphically.

#### 2.3.2.1 Graphical constraints

This section covers the specification of graphical verification rules. Graphical verification rules are easy to specify. They are associated with some graphical representation. These graphical representations are adopted mostly from NIAM (see e.g. [Wintraecken, 1990]).
We refer to [Bommel et al., 1991] and [Hofstede and Weide, 1993] for the formal definition of the graphical constraints. The former define constraints for Object types and Relationships (see sections 2.3.1.1 and 2.3.1.2), while the latter add constraints with reference to Set types (see section 2.3.1.4).

![Graphical constraints table]

Figure 2.10: Overview of graphical conventions for graphical constraints

Figure 2.10 provides an overview of the graphical conventions used when expressing graphical verification rules. The corresponding graphical constraints are explained below.

Figure 2.11 is used throughout this section to illustrate some of the constraints by the use of an extended example. This example concerns a simple Object structure of Process models. Process models are assumed to consist of Processes and Data flows only. The Object type Data flow is related to the Object type Process in two ways, as input Data flow and as output Data flow. Processes may be decomposed into new Process models. In this case, relations between Data flows at the two levels of decomposition are also registered. Note that dotted lines are used to indicate the Roles involved in graphical constraints.

**Enumeration constraint**

The *enumeration constraint* specifies that instances of a Label type may assume a value from a specific enumerated domain only.
Figure 2.11: Simple Object structure of process models, with several sample verification rules

**Total role constraint**

The *total role constraint* is used to specify that *all* instances of an Object type must be involved in a particular set of Roles. The black dots in figure 2.11 represent examples of the total role constraint: each Data flow is input of some Process, and each Data flow is output of some Process. The black dot in the circle represents the use of the total role constraint when several Relationships are involved: each Process is connected to a Dataflow, be it an input or an output Data flow.

**Uniqueness constraint**

The *uniqueness constraint* specifies that instances of Object types may play a particular combination of Roles at most once. The arrows in figure 2.11 represent examples of the uniqueness constraint: each Data flow is input of at most one Process, and output of at most one Process. Uniqueness constraints over several Relationships are represented by the character “u” in a circle.

**Occurrence frequency constraint**

The *occurrence frequency constraint* is more general than the uniqueness constraint. It involves not only a particular set of Roles, but also two natural numbers, say n and m. It is used to specify that a particular combination of Role instances occurs at least n and at most m times. In the case of the uniqueness constraint, n = m = 1.

In the example of figure 2.11, Processes may be decomposed into Process models. Data flows which are input for those Processes correspond to Data flows which are input for the decomposed Process model. Since Data flows connect two Processes, and since these Processes both may be decomposed into new Process models, it is clear that each Data flow may correspond to at most two other Data flows. This is an example of an occurrence frequency constraint.

**Role set comparison constraints**

*Role set comparison constraints* involve two sets of Roles. Given two sets of Roles, and
given a bijection between these Role sets, Role set comparison constraints specify how the populations of the sets of Roles depend on each other.

Four types of Role set comparison constraints are distinguished. *Equality constraints* specify that all sets of instances that instantiate the former set of Roles also instantiate the latter set of Roles and vice versa. *Subset constraints* specify that all sets of instances that instantiate the former set of Roles also instantiate the latter set of Roles. *Exclusion constraints* exclude any set of instances to be a subset of the population of both sets of Roles. *Membership constraints* apply to Relationships between Object types and Set types, these specify that all instances of the Object type which populate the Relationship should be element of instances of the Set type they are related to. In figure 2.11, the cross symbol between the two Relationships between the Object type Process and the Object type Data flow represents an exclusion constraint: if a Data flow is input for a specific Process, it should not be output of that Process, and vice versa.

**Specialisation constraints**

*Specialisation constraints* involve Object types in a specialisation hierarchy. There are two specialisation constraints. *Subtype total constraints* specify that the union of the populations of the involved Object types should be equal to the population of their lowest common ancestor Object type. *Subtype exclusion constraints* specify that the populations of the involved Object types are disjunct.

**Set type constraints**

*Set type constraints* restrict the possible populations of a Set type in relation to its constituting Object type. Three types of set type constraints are distinguished here. *Set type exclusion constraints* express that instances of an Object type may occur at most once in an instance of the Set type. *Set type cover constraints* express that every instance of a certain Object type has to occur in at least one instance of the Set type. *Set type cardinality constraints* express that instances of a Set type consist of at least a minimum and at most a maximum number of instances of the Object type involved. Set type constraints are illustrated in figure 2.12.

### 2.3.2.2 Relationship descriptors

Graphical constraints have been presented in the previous section. This section presents the concept of *relationship descriptor*, which is used in the next section as a basic concept for the specification of non-graphical verification rules.

The concept of relationship descriptor forms the core of the data manipulation language LISA-D. We introduce this concept informally. For a formal and complete treatment of relationship descriptors see [Hofstede et al., 1992b].

Intuitively, relationship descriptors specify a binary relation which connects the populations of two Object types in an Object structure. Paths in an Object structure are employed in order to specify such a binary relation. In the most elementary form, these paths consist of Object type names and Role names.
An example is given to clarify the construction of such paths. This example uses the Object structure in figure 2.13, which is a simplified version of the Object structure presented in figure 2.11. This Object structure represents Processes connected by Data flows. Figure 2.14 shows a sample population of this Object structure. The sample population concerns a functional view of a compiler (the figure has been taken from [Sommerville, 1989]). Table 2.1 shows how this population relates to the Object structure.

Atomic relationship descriptors consist either of an Object type name or of a Role name. An example of a relationship descriptor consisting of an Object type name is Process. This relationship descriptor results in an identity relation between Process instances, see table 2.2. An example of a relationship descriptor consisting of a Role name is produces. This relation descriptor results in two binary relations: a binary relation between Process and Data flow, see table 2.3, and a binary relation between Process and Outgoing flow, see table 2.4.

Relationship descriptors can be concatenated. Concatenation is considered to be a powerful mechanism, since arbitrary Object type populations in an Object structure can be connected. A characteristic example in figure 2.13 is the relationship descriptor Process
Figure 2.14: Sample population of figure 2.13, concerning a functional view of a compiler

<table>
<thead>
<tr>
<th>Object type</th>
<th>Population</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process:</td>
<td>$p_1$ (lexical analysis), $p_2$ (syntax analysis), $p_3$ (build symbol table), $p_4$ (semantic analysis), $p_5$ (code generation), $p_6$ (environment)</td>
</tr>
<tr>
<td>Data flow:</td>
<td>$d_1$ (input program), $d_2$ (symbolic program), $d_3$ (program symbols), $d_4$ (symbol table), $d_5$ (syntax tree), $d_6$ (analysed program), $d_7$ (object code)</td>
</tr>
<tr>
<td>Incoming flow:</td>
<td>$(p_1, d_1), (p_2, d_2), (p_3, d_3), (p_4, d_4), (p_4, d_5), (p_5, d_6), (p_6, d_7)$</td>
</tr>
<tr>
<td>Outgoing flow:</td>
<td>$(p_1, d_2), (p_1, d_3), (p_2, d_5), (p_3, d_4), (p_4, d_6), (p_5, d_7), (p_6, d_1)$</td>
</tr>
</tbody>
</table>

Table 2.1: Population of the Object structure of figure 2.13 according to the sample of figure 2.14

produces Data flow is-input-for Process. The determination of the corresponding binary

<table>
<thead>
<tr>
<th>Process</th>
<th>Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1$</td>
<td>$p_1$</td>
</tr>
<tr>
<td>$p_2$</td>
<td>$p_2$</td>
</tr>
<tr>
<td>$p_3$</td>
<td>$p_3$</td>
</tr>
<tr>
<td>$p_4$</td>
<td>$p_4$</td>
</tr>
<tr>
<td>$p_5$</td>
<td>$p_5$</td>
</tr>
<tr>
<td>$p_6$</td>
<td>$p_6$</td>
</tr>
</tbody>
</table>

Table 2.2: The relation descriptor Process
Table 2.3: The relation descriptor *produces*, interpreted as a binary relation between Process and Data flow

<table>
<thead>
<tr>
<th>Process</th>
<th>Data flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1$</td>
<td>$d_2$</td>
</tr>
<tr>
<td>$p_1$</td>
<td>$d_3$</td>
</tr>
<tr>
<td>$p_2$</td>
<td>$d_5$</td>
</tr>
<tr>
<td>$p_3$</td>
<td>$d_4$</td>
</tr>
<tr>
<td>$p_4$</td>
<td>$d_6$</td>
</tr>
<tr>
<td>$p_5$</td>
<td>$d_7$</td>
</tr>
<tr>
<td>$p_6$</td>
<td>$d_1$</td>
</tr>
</tbody>
</table>

Table 2.4: The relation descriptor *produces*, interpreted as a binary relation between Process and Outgoing flow

<table>
<thead>
<tr>
<th>Process</th>
<th>Outgoing flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1$</td>
<td>$(p_1, d_2)$</td>
</tr>
<tr>
<td>$p_1$</td>
<td>$(p_1, d_3)$</td>
</tr>
<tr>
<td>$p_2$</td>
<td>$(p_2, d_5)$</td>
</tr>
<tr>
<td>$p_3$</td>
<td>$(p_3, d_4)$</td>
</tr>
<tr>
<td>$p_4$</td>
<td>$(p_4, d_6)$</td>
</tr>
<tr>
<td>$p_5$</td>
<td>$(p_5, d_7)$</td>
</tr>
<tr>
<td>$p_6$</td>
<td>$(p_6, d_1)$</td>
</tr>
</tbody>
</table>

relation consists of the concatenation of the binary relations resulting from the atomic relationship descriptors *Process*, *produces*, *Data flow*, *is-input-for*, and *Process*, see table 2.5. This relationship descriptor delivers all Processes which are connected to each other by a Data flow.

Note that the explicit concatenation of *produces* to *Data flow* excludes the interpretation of *produces* as a binary relation between Process and Outgoing flow. Concatenation of

<table>
<thead>
<tr>
<th>Process</th>
<th>Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1$</td>
<td>$p_2$</td>
</tr>
<tr>
<td>$p_1$</td>
<td>$p_3$</td>
</tr>
<tr>
<td>$p_2$</td>
<td>$p_4$</td>
</tr>
<tr>
<td>$p_3$</td>
<td>$p_4$</td>
</tr>
<tr>
<td>$p_4$</td>
<td>$p_5$</td>
</tr>
<tr>
<td>$p_5$</td>
<td>$p_6$</td>
</tr>
<tr>
<td>$p_6$</td>
<td>$p_1$</td>
</tr>
</tbody>
</table>

Table 2.5: The relation descriptor *Process produces Data flow is-input-for Process*
relationship descriptors provides the context by which the number of possible interpretations (in other words, the number of possibly resulting binary relations) is decreased.

Now that we have discussed Object type names and Role names as part of paths, it is clear that relationship descriptors relate instances of the begin Object type and the end Object type only (or, in the case of more possible interpretations, the possible begin Object types and the possible end Object types). Intermediate Object instances are necessary for the determination of the binary relation, but they are invisible in its final result.

![Diagram of relationship descriptors]

Figure 2.15: Keywords

Paths do not always consist only of Object type names and Role names. They may employ predefined keywords for connections and operations. The most relevant ones are summarised in figure 2.15. These are discussed below. They mostly use the sample Object structure of figure 2.9.

The keywords IS-NUMBER-OF and WITH are defined to handle Labels. The keyword WITH relates Object types to Label types. IS-NUMBER-OF relates Label types to Object types. Examples of relation descriptors using these keywords are: Entity WITH Name, and Name IS-NUMBER-OF Entity.

The keywords INVOLVED-IN and OF are used in connection with Relationships. The relationship descriptor Entity INVOLVED-IN Identification relates Entities to the instances of Relationship Identification in which they occur. The begin Object type is therefore Entity and the end Object type Identification. The precise reverse is achieved by formulating Identification OF Entity.
The keyword ASSOCIATED-WITH serves as a useful abbreviation in cases where several relationships relate two given Object types. Using the combination of the keywords OF and INVOLVED-IN results in all Relationships that connect the two given Object types. In the example given in figure 2.3, the relationship descriptor Action INVOLVED-IN OF State transition would result in all combinations of Action instances and State transition instances. This keyword combination may be abbreviated to ASSOCIATED-WITH, hence: Action ASSOCIATED-WITH State transition.

The keywords IN and CONTAINING are used in relation to Set types and Sequence types. The relationship descriptor Entity list CONTAINING Entity relates Entity lists to their constituting Entities, while Entity IN Entity list gives precisely the inverse.

The keywords PART-OF and COMPRISING are used in relation to Schema types. COMPRISING relates instances of Schema types to instances of Object types in those Schema types. PART-OF relates Element types to Schema types, in reverse.

Additional keywords exist for the specification of more complex relationship descriptors. New relationship descriptors may be formed by logical connections between existing relationship descriptors. From the relationship descriptors R and R' the following relationship descriptors can be constructed: R AND-ALSO R', R OR-ELSE R', R BUT-NOT R', and NOT R. The three binary keywords correspond to the elementary binary set operators intersection, union, and division, and are applied on the begin Objects of the involved relationship descriptors R and R'. The latter keyword corresponds to the elementary unary set operator negation. In figure 2.3, the relationship descriptor State transition is-caused-by Condition WITH C-name 'At floor' AND-ALSO State transition triggers Action WITH A-name 'Signal floor reached' contains state transitions fulfilling both conditions only.

Recursion can be expressed by means of the transitive closure of relationship descriptors. The keyword ANY-REPETITION-OF has been introduced to this end. In the example of STDs (see figure 2.9), States may be decomposed in STDs which contain States. These States may be further decomposed, and so on. The binary relation between States and their direct or indirect Substates is obtained by the relationship descriptor ANY-REPETITION-OF (State decomposed-into STD COMPRISING State).

The keyword THAT has been introduced to express correlation between specific Object instances. For example, to find the States that appear in their own decomposition, the binary relation between States and their direct or indirect Substates has to be intersected with the binary identity relation between States. Illegal decomposition cycles are specified by the following relationship descriptor: ANY-REPETITION-OF (State decomposed-into STD COMPRISING State) THAT State.

Type coercions convert a population of a relationship descriptor into a single value. Three functions are available to this end. The function NUMBER-OF counts the number of elements occurring in a relationship descriptor. The functions MIN and MAX calculate the minimal and the maximal element in a relationship descriptor, respectively. These two functions require the Label type of the begin Object type of the relationship descriptor involved to be of an ordered domain.
We have presented four different ways of constructing more complex relationship descriptors. A convenient mechanism to reduce the complexity of relationship descriptors is the assignment of subrelationship descriptors to variables. In figure 2.3, an example of an assignment is: LET Floor state transitions BE State transition is-caused-by Condition WITH C-name 'At floor'. This defines the variable Floor state transitions.

2.3.2.3 Non-graphical constraints

Relationship descriptors specify a binary relation (or, in the case of more possible interpretations, several binary relations) which connects the populations of two Object types in an Object structure. Relationship descriptors serve as building blocks for (non-graphical) constraints. Predicates have been introduced in LISA-D to specify (non-graphical) verification rules. The following provides an overview.

Verification rules are expressed by predicates. These evaluate to a boolean value which states whether the population of an Object structure satisfies the specified verification rule or not. Relationship descriptors are the most basic predicates. The predicate R evaluates to "true" if and only if the population of relationship descriptor R is not the empty set.

New predicates are constructed from basic predicates. Given two predicates, P and P', the use of logical operators leads to the predicates: P AND P', P OR P', and NOT P. The use of quantification leads to the predicate: FOR-EACH x IN R HOLDS P. Note that, in the last case, the elementary predicate P must contain variable x. As an example, the verification rule in figure 2.13 that each Data flow should be output of a Process is formulated as follows: FOR-EACH d IN Data flow HOLDS d is-output-of Process.

Two notational conventions for predicates dealing with set comparison are: R INCLUDES R', which is defined as NOT (R' BUT-NOT R), and R EQUALS R', which is defined as (R INCLUDES R') AND (R' INCLUDES R).

2.3.3 Task structures

In this section, we discuss the part of the meta-modelling technique that enables us to represent a way of working, more specifically, to represent tasks to be performed as part of an information systems development process, and their interrelationships. We refer to a representation of a way of working as a Task structure.

The main concepts in Task structures are introduced informally in this section, based on the Task structure diagramming technique presented in [Hofstede and Nieuwland, 1993]. This technique is an extension of the Task structure diagramming technique presented in [Wijers, 1991]. External representation conventions for concepts occurring in Task structures are adopted mostly from [Bots, 1989].

2.3.3.1 Task objects and their ordering

Tasks are considered to be actions taken to achieve a certain goal (see [Bots, 1989]). Tasks are often defined recursively in terms of subtasks, concerned with certain subgoals. This
implies that tasks can be decomposed into a hierarchy of subtasks until a desired level of detail has been reached. If performing a task involves alternative strategies, i.e. choices between subtasks, *decisions* represent these moments of choice. Decisions coordinate the sequence of performing tasks, and are either terminating or non-terminating. Terminating decisions include an extra choice option, namely termination of the execution path of that decision. *Synchronisers* represent moments of synchronisation in performing parallel tasks. *Triggers* model sequential order of tasks, and are represented graphically as arrows, see figure 2.16. *Initial items* are those tasks or decisions that have to be performed first as part of a decomposed task. Initial items of decomposed tasks must be defined since it may not always be clear which tasks or decisions are initial, e.g. in iterative structures.
Finally, an important rule in Task structures is that tasks with the same name have the same decomposition (e.g. the tasks with name B in figure 2.16).

### 2.3.4 Relation between Object structures and Task structures

In section 1.2.5 three relationships between a way of modelling and a way of working have been discussed. We recapitulate them here, and reformulate them in terms of representing modelling knowledge, that is, in terms of Object structures and Task structures. The relationships concern the scope and the consistency characteristic of modelling tasks, and information interdependencies between modelling tasks:

- the *scope* of a modelling task denotes the concepts of which instances may be manipulated in that specific modelling task. In other words, the scope of a modelling task, which occurs in a Task structure, expresses the set of Object types, that occur in an Object structure, of which the population may be manipulated by that specific modelling task;

- the *consistency characteristic* of a modelling task involves the set of verification rules which must be satisfied after having performed the specific modelling task. In other words, modelling tasks, that occur in a Task structure, have their own verification rules which are expressed in terms of Object types in an Object structure;

- *information interdependencies* between modelling tasks relate to the specific constraints between the resulting products of one modelling task and the resulting products of its successors, that is, information interdependencies between modelling tasks, which occur in a Task structure, involve constraints which, again, are expressed in terms of Object types in an Object structure.

The second relationship refers to verification rules which hold for some specific set of modelling tasks only, as opposed to the task independent verification rules which are part of a way of modelling. Modelling constructs for expressing task independent verification rules and for expressing task specific verification rules do not differ. We therefore refer to section 2.3.2 where verification rules are presented in general.

In this section, we pay specific attention to the first and the third relationship between a way of modelling and a way of working, that is, to specifying scopes of modelling tasks and specifying information interdependencies between modelling tasks. To this end, the concept of relationship descriptor is used. Furthermore, we use the concept of Schema type, which enables us to represent scopes and information interdependencies in a very compact way.

#### 2.3.4.1 Scopes

We defined Task structures as rooted decomposition hierarchies. Tasks are decomposed into a hierarchy of subtasks until a desired level of detail has been reached.

In a decomposition hierarchy, tasks are either leaf tasks or non-leaf tasks. Non-leaf tasks serve to define the flow of control, i.e. to define in which order their subtasks may be
performed. Leaf tasks are those tasks in which information engineers actually construct or change their application models. Therefore, leaf tasks lead to a change in the population of Object structures. Since leaf tasks are not decomposed further, their execution is assumed to take place instantaneously. Leaf tasks are transactions, which comprise a sequence of update statements.

To specify update statements which add or delete Object instances to populations, the keywords ADD and DELETE are used. The keyword READ is used to return the value of an existing or a new Object instance. An example of the specification of an update statement, again based on figure 2.8, is: ADD Entity e IN Entity list el. This update statement specifies the addition of one arbitrary entity to an arbitrary entity list. Note the use of variables to bind the specified Object types to the specific Object instances that are related when executing the update statement. In general, update statements consist of a keyword and a relationship descriptor in which variables are used for binding.

We use the notion of update statement to clarify how the scope of a task is determined. The scope of a leaf task is the set of Object types, of which the population may be manipulated by its constituting update statements. The scope of a non-leaf task is the union of the scopes of its constituting leaf tasks. Consequently, the scope of the main task should be the whole Object structure. In the case of the sample leaf task which consists of the above update statement only, its scope comprises two Object types: Entity and Entity list.

2.3.4.2 Information interdependencies

Information interdependencies between modelling tasks concern the specific relationships between the products resulting from one task and the products resulting from its successors. Information interdependencies are expressed in terms of these product relationships, and hence in terms of Object types.

The introduction of relationship descriptors facilitates the expression of information interdependencies between modelling tasks to a considerable extent. The ease of use can be expressed by an example. Suppose that, as part of a process modelling task, a Context diagram has to be created before a Process can be added to that diagram. This information interdependency is expressed by the following update statements:

ADD Context-diagramd;
ADD ProcesspPART-OF Context-diagramd

The repeated use of variable d expresses the existing information interdependency.

Expressing information interdependencies between complex modelling tasks, however, will probably require a large number of update statements. Note that we used the term “complex” in the previous sentence to refer to tasks that manipulate information models as a whole. In terms of figure 1.2 on page 7, complex modelling tasks are tasks from the model view. To express information interdependencies between complex modelling tasks in a more elegant way, we present some notational shorthands.

Complex modelling tasks can be decomposed into a hierarchy of subtasks until a desired level of detail is reached (see section 2.3.3). When describing a way of working in precise
relation to a way of modelling, the desired level of detail is that level at which instances of 
Object types are added to or deleted from an information model. If we, however, wish to 
represent the decomposition of complex modelling tasks, we should present an iterative task 
structure in which instances of all Object types in the scope of this leaf task are allowed to 
be added or deleted in an arbitrary order. To avoid such an extensive listing of all possible 
update statements for each modelling task when representing its decomposition, three types 
of modelling tasks are distinguished, and notational shorthands are provided for them. 

The first modelling task type is model construction “from scratch”. An example of a task 
of this type is “Create a new data model”. In order to be able to denote modelling tasks 
of this type in a short way, we need to create a Schema type, say “Initial data model”, 
and an instance of this Schema type, say “dm₁”. The Schema type “Initial data model” 
should comprise all Object types which are relevant in this particular task. Examples of 
such Object types are “Subtype” and “Relationship”. We introduce the keyword create to 
specify modelling tasks which construct models “from scratch”. In the above example, the 
task “Create a new data model” is specified as: create Initial data model dm₁. 

The second modelling task type is model construction “by modification”. Modelling tasks 
of this type extend an existing application model, leading to a new application model. 
An example of a task of this type is “Extend the existing data model”. This task may 
involves adding or deleting instances of Subtypes and Relationships, but also adding or 
deleting instances of new Object types, e.g. of “Attributes”. Thus, modification refers to 
a changing population of the existing Object types, and it may refer to the instantiation of 
some new Object types. In the latter case, it is necessary to define a new Schema 
type. This Schema type should include the Object types of the initial Schema type. The 
initial Schema type and the new Schema type are referred to as source Schema type and 
destination Schema type. We introduce the keyword modifying to specify modelling tasks 
which construct models “by modification”. In the example of entity-relationship diagram 
extension, the task “Extend the existing data model” is specified as: create Data model 
dm₂ modifying Initial data model dm₁, where the Schema type “Data model” may refer to 
the larger set of Object types which includes “Attribute”. 

The third modelling task type is model construction “by use”. Modelling tasks of this type 
lead to the construction of a new application model based upon an existing application 
model. An example of a task of this type is “Construct a process model based upon the 
data model”. To specify such modelling tasks, the Schema types “Process model” and 
“Data model” are required as well as a Schema type which defines how the concepts of 
both model types interrelate. For example, suppose that each attribute in the data model 
leads to the creation of one or more data flows in the process model. In this case, the 
Schema type “Process model” will contain an Object type “Data flow”, the Schema type 
“Data model” will contain an Object type “Attribute”, and the new Schema type “Data 
model to Process model” will contain the Relationship between the Object types “Data 
flow” and “Attribute”. We introduce the keyword using to specify modelling tasks which 
construct models “by use”. In the above example, the task “Construct a process model 
based upon the data model” is specified as: create Process model pm using Data model 
to Process model dp. The Schema type “Data model to Process model” is referred to as
auxiliary Schema type. An auxiliary Schema type relates the source and the destination Schema type.

Note that distinguishing between three types of model creation captures not only scopes of modelling tasks (create) but also information interdependencies between modelling tasks (modifying and using).

![Diagram](image)

Figure 2.17: Relationship between DFDs and STDs

The use of the concept of Schema type and the distinction between three types of model creation allow the scope of complex modelling tasks and the information interdependencies between such tasks to be represented in a very compact way. This is illustrated by an extended example, involving two model types, STDs and DFDs. STDs often represent a process specification for a control process in a DFD. This is illustrated in figure 2.17, which is taken from [Yourdon, 1989]. In this figure, the dashed circle represents the control process, whereas the dashed arrows represent the control flows. In the STD, conditions and arrows are shown as labels next to the arrow connecting two states. Note that the conditions in the STD correspond to the incoming control flows in the DFD, and that the actions in the STD correspond to the outgoing control flows in the STD.

To define the existing information interdependencies, three auxiliary activities must be performed: defining the Object structure, defining the Task structure, and defining the necessary Schema types.

In this example, it is assumed that the Task structure consists of three subtasks: "Construct an initial STD", "Construct an initial DFD", and "Relate the initial STD and the initial DFD to each other", which are referred to by the characters a, b, and c, respectively. Furthermore, it is assumed that three Schema types have been defined: "STD", "DFD",
and “DFD-to-STD”. The Schema type “STD” contains at least the Object types “Action” and “Condition”, the Schema type “DFD” contains at least the Object types “Control process” and “Control flow”, and the Schema type “DFD-to-STD” contains at least the Relationship between Condition and Control flow, and the Relationship between Action and Control flow. The existing information interdependencies between the three modelling tasks are clear from the specification of the tasks, see table 2.6.

<table>
<thead>
<tr>
<th>Task</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>create STD $s_1$</td>
</tr>
<tr>
<td>b</td>
<td>create DFD $d_1$</td>
</tr>
<tr>
<td>c</td>
<td>create STD $s_2$ modifying STD $s_1$ using DFD-to-STD $d_{s_1}$</td>
</tr>
</tbody>
</table>

Table 2.6: Specification of tasks $a$, $b$, and $c$ using the defined Schema types.

![Diagram](image)

Figure 2.18: Existing information interdependencies holding for the construction of STDs based on existing DFDs.

Figure 2.18 visualises the existing information interdependencies between these three tasks. In this figure, we use what we call the information interdependencies diagramming technique. The circles represent Schema types. If a Schema type is a subset of another Schema type, the corresponding circles are drawn within each other. Arrows connect Schema types, and may be labelled either “create”, “use”, or “modify”, depending on the modelling task type. Arrows which are labelled “create” are connected only to the created Schema type. Arrows which are labelled “use” connect source Schema types to destination Schema types via auxiliary Schema types. Arrows which are labelled “modify” connect source Schema types to destination Schema types. Arrow labels also refer to the modelling tasks involved in instantiating the Schema types. The name of a modelling task is put in parentheses. The use of dashed arrows indicates that the modelling task involved is an optional one, the use of plain arrows indicates that the modelling task involved will always be carried out, according to the Task structure.
2.4 Approach to acquisition of modelling knowledge

This section addresses our approach to the acquisition of modelling knowledge. We consider two factors of particular interest in choosing a knowledge acquisition approach. One, the domain of expertise is clearly restricted to information modelling. Two, experienced practitioners in this domain of expertise are involved (for convenience, they are called experts).

We distinguish two main stages in the process of acquiring experts' modelling knowledge: one, modelling knowledge is elicited, two, the elicited modelling knowledge is represented, resulting in a conceptual meta-model. The word "conceptual" emphasises our preference for understanding over solving, see [Sol, 1982]. We share the view that knowledge acquisition processes should not include any technological considerations of the implementation of knowledge bases in knowledge based systems, see [Hart, 1989] and [Wijers, 1991].

Figure 2.19: Approach to acquisition of modelling knowledge

In [Wijers, 1991] a useful approach is presented, aiming specifically at the acquisition of experts' information modelling knowledge. This approach consists of four tasks: preparation, elicitation, interpretation, and conceptualisation, see figure 2.19. Elicitation results in verbal data. Interpretation leads to a text based model. Conceptualisation delivers a conceptual meta-model.
This approach fits our two requirements perfectly because it is aimed specifically at experienced information engineers. The four tasks are discussed in more detail below.

Our approach distinguishes itself from the one presented in [Wijers, 1991] mainly in the organisation of the elicitation task. The elicitation task requires the availability of an actual information modelling task to be performed by an experienced information engineer. In [Wijers, 1991], the role of user is performed by an arbitrary person who has learned the characteristics of the organisation involved by heart. Since "protocol analysis requires a high fidelity task that represents a real world problem with sufficient reality" [Vitalari, 1985], our approach includes a real-life case.

2.4.1 Preparation

The preparation task includes the selection of experts using four criteria. The experts should have more than five years of professional experience in the field of information modelling, and be proficient at the Yourdon approach. Their managers should consider them to be experienced and competent, and they should also be experienced in a variety of application domains.

Furthermore, preparation should include the selection of cases to be given to the experts, using the following criteria. The case should be representative in the sense that specifications are informal and therefore ambiguous and incomplete. It should be a realistic case, i.e. the organisation involved or some departments in the organisation involved should have a non-trivial problem that requires solving. Finally, the problem owners should be able to be available whilst the information modelling task is being dealt with.

Preparation will also include familiarising the selected information engineers with the knowledge elicitation task and arranging technical and organisational equipment for the elicitation sessions.

2.4.2 Elicitation

The elicitation task aims at determining and transcribing experienced information engineers' modelling knowledge. The elicitation task is treated in more detail in this section, as the task of eliciting expert knowledge is known to be a major bottleneck in knowledge engineering. Expert knowledge is often "private to the expert, not because he is unwilling to share publically how he performs, but because he is unable to" [Feigenbaum, 1979]: expert knowledge is additional to knowledge contained in textbooks. Many researchers in the area of knowledge engineering point out the difficulties of acquiring expert knowledge (e.g. [Hayes-Roth et al., 1983], [Duda and Shortcliffe, 1983], and [Wilkins et al., 1984]). Several techniques that have proven successful in acquiring expert knowledge, are available. An overview can be found in [Wright and Aytom, 1987] and [Grabowski, 1988].

The choice of a knowledge elicitation technique is a major issue in knowledge acquisition processes. There are two commonly used knowledge elicitation techniques, interviewing and think aloud protocol analysis (in short, protocol analysis). We consider interviewing to be
inappropriate for our purpose since experts' knowledge is known to be poorly accessible. Interviews may encourage the expert to speculate and theorise about their cognitive processes [Wright and Ayton, 1987].

In *think aloud protocols* (in short, protocols), expert knowledge is *cued* by performing modelling tasks. The expert is asked to solve an information modelling problem. The protocol is called concurrent if the expert is asked to verbalise every thought and action *while performing* the information modelling task.

Protocol analysis has been employed in many research areas, such as trust investment [Clarkson, 1962], cryptarithmetic [Newell and Simon, 1972], computer programming [Brooks, 1980], [Tromp, 1989], physics [Larkin et al., 1980], financial analysis [Bouwman, 1983], medical diagnosis [Myers et al., 1983], and media planning [Mitchell, 1987], and has proven useful in gathering a detailed set of data on the subject's solution process [Vitalari and Dickson, 1983].

This approach becomes particularly useful in domains [Batra and Davis, 1989]

> in which the task is relatively complex and open-ended, since the verbal data enables us to study in greater detail the intermediate stages of such processes.

Vitalari even states that protocol analysis requires a high fidelity task that represents a real world problem with sufficient reality [Vitalari, 1985].

In [Ericsson and Simon, 1980] it is argued that protocols will not change task performance, although the speed of task performance will be slowed down. Verbalisation has no effect on the *effectiveness* of task performance. Asking an expert to *explain* what he is doing will, however, disturb task performance, as they are required to access additional knowledge. The information engineer is gently encouraged to think aloud when he falls silent but care is taken not to influence the course of the information modelling task, see [Wright and Ayton, 1987]. Suitable examples of direct probes are “Please keep talking” and “What are you thinking about?” [Ericsson and Simon, 1984].

Based upon the above, we choose a concurrent protocol analysis as our knowledge elicitation technique.

It is important to stress the limitations of this technique. Protocol analysis is known to be labour intensive. The time and effort required to collect and code verbal protocols for analysis is considerable [Vitalari, 1985]. Due to the labour intensivity of protocol analysis, only a small number of subjects can be studied and the ability to generalise and extrapolate to the greater population of information engineers is considerably restricted. The general applicability of the results is also reduced because the spectrum of variables can not be controlled (such as the application domain and the type of organisation involved in the study, the subjects' previous educational background and cognitive style). However in spite of this, protocol analysis is considered to be the only feasible knowledge acquisition technique due to the inappropriateness of interviewing, and due to the preliminary and explorative nature of our investigation. See [Wright and Ayton, 1987]:

> In general it would appear that a protocol is potentially useful for what it contains rather than what it omits.
Figure 2.20 shows how our protocol setting was designed. This figure has been adopted from [Wijers, 1991]. After [Wijers, 1991], simulation by teletype is used in the concurrent protocol setting. Wijers shows that the expert’s think aloud processes interfere with the expert’s communication processes with the users’ community. This interference is explained by the availability of only one communication medium, speech. For this reason, the protocol environment includes various communication media. Obviously, think aloud processes can only be performed by speech. Therefore, the communication processes between expert and users are restricted to other media. Personal computers are employed to simulate verbal communication. Diagrams are exchanged using whiteboards, video, and television, see figure 2.20.

Five roles are distinguished in the protocol environment: the information engineer, the user, the knowledge engineer, the technical support, and the typists. In the expert’s room, the knowledge engineer encourages the expert to think aloud, and he operates the video camera and the tape recorder. The technical support performs the same operating tasks.
in the users' room. The typists support the communication process between expert and users by typing messages. The personal computers are provided with velotype facilities to minimise slowing down the communication process. The teletype connection supports concurrent typing, so that messages can be interrupted mutually, as in "real" dialogues. The tape recorder registers the think aloud process. The whiteboard, the video recorder, and the television are used for the exhibition of diagrams and notes.

Experts are given a warm up session to practise thinking aloud. They are given paper and pencil to work their solutions and to make notes, when appropriate. They are not provided with any information on the case beforehand. Experiments are held at the users' regular work environment. Completion of their information modelling task is constrained to a maximum of four days. During these four days, users are available to have a dialogue with. Experts are also allowed to work for themselves, that is, without any users, in order to analyse and process the information gathered so far and to prepare themselves for future dialogues with users. As a consequence, each experiment consists of a number of dialogue sessions and analysis sessions.

<table>
<thead>
<tr>
<th>In my opinion, we have a clear need for a powerful planning tool, which can handle the complex mass of 360 exams, growing towards 700 exams in 1992, which are divided all into more detailed activities.</th>
<th>Yeah, from 360 to 720 exams, that is an enormous increase, obviously. In 1992, oh dear, the notorious year, they are going to have European students only, aren't they? So, the purpose is clear. But: what is the starting point, what are the boundaries of the system? Let's take an example.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Is, for instance, everything planned by the examiner, is that part of the system? And does the system include the exam department's activities?</td>
<td>First a correction: there is only one exam department. The exam department's activities are within the system. As regards the examiners, ...</td>
</tr>
<tr>
<td>Again a correction? How do you call those things then? Those local institutes? Oh, I referred to those study centres, as a matter of fact. Used the wrong name, stupid! Yeah, but as such, it is correct. I thought so! As regards the examiners, yes, ...</td>
<td></td>
</tr>
</tbody>
</table>

Figure 2.21: Example of a partial protocol transcript

Audio recordings are transcribed after the experiments. The transcription process yields two textual transcripts: the information engineer's think aloud protocol, and the communication between the expert and the user. Transcription also includes the numbering of diagrams...
drawn and notes made by the experts. In the textual transcripts, references to diagrams and notes are made.

Subsequently, the two resulting transcripts are synchronised and integrated into one protocol transcript, constituting the verbal data. A part of a protocol transcript is shown in figure 2.21. The expert's questions during a dialogue session are displayed over the full width. The users' answers are presented in the left column. The right column displays the expert's synchronous think aloud process.

To reiterate, the elicitation step consists of three steps: (i) performing knowledge elicitation session, (ii) transcribing recordings, diagrams, and notes, and (iii) synchronising transcripts.

2.4.3 Interpretation

The transcript consists of a continuous string of words, exclamations, and incomplete sentences that appear to be rather disorganised. Interpretation transforms a transcript into a more accessible representation, a text-based model. The task of interpreting focuses on finding the tasks performed and the concepts applied by the expert, the two parallel activities are called task oriented text analysis and concept oriented text analysis, see [Wijers, 1991].

To support the interpretation task, or more specifically, to support the marking and labelling of text units in a transcript as tasks and concepts, we have guided the development of an interpretation support environment, the Fragment Toolkit. For a detailed description of the support delivered by the Fragment Toolkit, see [Harst, 1991a] and [Harst and Herlé, 1991].

2.4.3.1 Task oriented text analysis

In the task oriented text analysis stage, we construct a task oriented text-based model. There are two different phases in this stage: (i) task recognition and (ii) task structuring.

In the task recognition phase, we attempt to distinguish small, coherent text units in the protocol transcript which represent the performance of a task. Such units are usually referred to as chunks, segments or fragments, see for instance [Ledderhof, 1989] and [Heijes, 1990]. We prefer the term fragment.

Three different heuristics are applied when dividing the protocol transcript into fragments. The first heuristic involves a search for specific phrases which indicate the beginning or the end of a task. Typical examples are "I know enough for the moment" and "This implies I now have to focus on ...", which clearly mark a shift of attention. The second heuristic applies, in particular, to dialogue sessions. These sessions consist of a sequence of expert questions and user answers. Expert questions either focus on a specific aspect of the user answer given or cover a new topic. In the first case, the expert question belongs to the current task, in the second case, a new task has started. The third heuristic characterises parts of a protocol transcript according to the classification of activity types given in table 2.7 (see [Harst and Herlé, 1991]). The activity types "Gather", "Analyse", "Conclude", "Validate", "Construct", and "Explain" all concern the application domain, whereas the
activity types “Reflect” and “Plan” concern the approach that has been performed or is going to be performed. Classifying transcript parts according to table 2.7 is not a unequivocal activity, however, in spite of this, the heuristic is a very useful one. Activities classified “Reflect” or “Plan” often contain very useful clues about the information engineer’s approach. Activities classified “Construct” often become fragments. If several activities of type “Construct” are performed sequentially, it is important to investigate the modelling technique(s) used. If different modelling techniques are used, all fragments remain independent fragments. If only one modelling technique is used, then either the same application domain may be modelled at different levels of detail or different partial application domains may be modelled.

<table>
<thead>
<tr>
<th>Activity type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gather</td>
<td>information engineer gathers information on a subject</td>
</tr>
<tr>
<td>Analyse</td>
<td>information engineer analyses information gathered</td>
</tr>
<tr>
<td>Conclude</td>
<td>information engineer draws a conclusion</td>
</tr>
<tr>
<td>Validate</td>
<td>information engineer validates information, analyses or conclusions with user</td>
</tr>
<tr>
<td>Construct</td>
<td>information engineer constructs a model</td>
</tr>
<tr>
<td>Explain</td>
<td>information engineer explains a question, an idea, a diagram, etcetera to user</td>
</tr>
<tr>
<td>Reflect</td>
<td>information engineer reflects on his own information modelling process</td>
</tr>
<tr>
<td>Plan</td>
<td>information engineer plans his future operational information modelling tasks</td>
</tr>
</tbody>
</table>

Table 2.7: Classification of activity types used during the task oriented text analysis stage

Task recognition ends with a protocol transcript that has been fragmented into small text units which represent the actual performance of a task or the actual making of a decision. It is of particular interest for each task or decision, whether the protocol transcript covers any rationale: which arguments triggered the expert to end a task, or which arguments triggered the expert to make a decision with this outcome?

Task structuring involves imposing structure on the text-based model developed so far. Precedence relationships between tasks are registered explicitly, furthermore, an attempt is made to group connected tasks into larger tasks and to structure tasks by splitting them into smaller tasks. Task structuring ends with a task oriented text-based model, in which decomposition relations and precedence relations between tasks and decisions are indicated.

2.4.3.2 Concept oriented text analysis

At the concept oriented text analysis stage, a concept oriented text-based model is constructed. Interest centers on the nouns and phrases that denote concepts applied by the expert.
A knowledge engineer should be aware that concepts applied by the expert are to be found at meta-level (see figure 1.3, page 16), however, in dialogues with users, typically instances of these concepts will occur, that are at application level. For example, the application of the concept “Process” will lead to a dialogue about specific instances of the concept “Process”.

Relationships between concepts typically occur in the protocol transcripts as short phrases between two nouns, for example: “Employee E handles Activity A”, “Activities are divided into Subactivities”, “Management Activity MA is related to Operational Activity OA”.

A concept oriented text-based model consists of related nouns and phrases.

Summarising, the interpretation task results in a text-based model. This model consists of a number of fragments, which are labelled (task, decision, concept, relationship) and interrelated.

2.4.4 Conceptualisation

The conceptualisation task involves the transformation of the text-based model to a conceptual meta-model. The meta-modelling technique given in section 2.3 is applied in this transformation.

Three different stages can be distinguished within this task, these stages can be performed iteratively in an arbitrary order. The construction of the Object structure and the Task structure are discussed followed by a discussion on the integration of Object structure and Task structure.

2.4.4.1 Construction of the Object structure

An Object structure is constructed based upon the concepts and the relationships marked in the text-based model. Concepts in the text-based model transform into Object types in the Object structure. The various relationships in the text-based model are of particular interest. Examining these relationships may lead to the introduction of Relationships between Object types in the Object structure, but may also lead to the introduction of Generalisation hierarchies or Specialisation hierarchies, or to the introduction of Set objects, Sequence objects, or Schema objects.

2.4.4.2 Construction of the Task structure

A Task structure is constructed on the basis of the tasks and decisions that have been marked in the text-based model. “Precedence” relationships between tasks in the text-based model should be examined for the occurrence of the iterative performances of certain tasks and decisions. In such cases, an iterative structure over these tasks will be found, and a decision should be added to the Task structure which indicates when the iteration ends. “Decomposition” relationships between tasks in the text-based model will lead to decomposition of Tasks in the Task structure.
2.4.4.3 Integration of the Object structure and the Task structure

The Object structure and the Task structure are linked into one conceptual meta-model by the determination of task scopes, verification rules, and information interdependencies between tasks.

Task scopes are defined for all Tasks in the Task structure. Part of the Object structure constitutes a Task scope. The scope of a Task can easily be derived from the text-based model, by analysing the fragments which have been labelled “concept” or “relationship” and which occur in fragments which have been labelled “task”.

Verification rules are rarely available in a text-based model. They can be determined from the rationale marked in the text-based model. Remaining verification rules have to be validated by interviews with the information engineer. In particular, a priori verification rules can not be derived from protocol transcripts, since these are rules which have not been violated by the information engineer during his information modelling tasks.

Information interdependencies are of interest when existing information models turn out to have been manipulated or extended.

2.5 Conclusions

This chapter presented our approach to representing modelling knowledge. We have made choices with regard to a specific information systems development method, a specific meta-modelling technique, and a specific modelling knowledge acquisition approach. The latter choice applies to the representation of applied modelling knowledge only, whereas both of the former also apply to the representation of prescribed modelling knowledge.

With reference to the information systems development method, we have decided to restrict ourselves to the method presented in [Yourdon, 1989], see section 2.2. This method covers the early stages of the information systems development life cycle, it is a well-established method, and literature is available.

We have presented the meta-modelling technique in section 2.3. Use of the meta-modelling technique results in Object structures, constraints, Task structures, and relations between Object structures and Task structures. It allows for the representation of both a way of modelling, a way of working, and their interrelationships. In particular, it is possible to represent complex structures and constraints as they occur in the complex domain of representing modelling knowledge as part of a way of modelling.

Section 2.4 presents the knowledge acquisition approach. This approach consists essentially of four stages: preparation, elicitation, interpretation, and conceptualisation. Concurrent think aloud protocols using simulation by teletype is employed as the knowledge elicitation technique during the elicitation stage.

The preparation stage resulted in the involvement of three experienced information engineers. The subjects are highly educated and have an average of seven years of professional experience in information modelling. They are well familiar with the Yourdon approach,
and have modelling capabilities over a range of applications. From this point on, they are referred to as information engineer A, B, and C. The preparation stage also resulted in two cases being used. These cases are described in more detail in the appendices B and C. The elicitation stage resulted in six protocol transcripts, see [Harst, 1991b], [Herlé, 1990], [Verhoef, 1990], [Verhoef, 1991a], [Verhoef, 1991b], and [Verhoef, 1991c]. They are not presented in any detail here, as they served as auxiliary intermediate results. The interpretation stage resulted in six text-based models, again not presented here for the same reason. The conceptualisation stage resulted in six meta-models of applied modelling knowledge. These are presented in chapters 4 and 5 of this thesis.
Chapter 3

Representing prescribed modelling knowledge

This chapter deals with the representation of prescribed modelling knowledge. Prescribed modelling knowledge is taken from [Yourdon, 1989], as argued in section 2.2. Application of the meta-modelling technique of section 2.3 to Yourdon’s development method results in the actual meta-models of sections 3.1, 3.2, and 3.3. These sections treat the representation of a way of modelling, a way of working, and their interrelationships, respectively. Section 3.4 presents some conclusions.

3.1 Way of modelling

This section represents the way of modelling from [Yourdon, 1989]. It can be recalled from chapter 1 that a way of modelling defines an interrelated set of concepts, external representations and invariant verification rules. Since we consider external representations to be a subject of minor importance, compared to the concepts and the verification rules that are part of a way of modelling, we exclude their representation from this chapter. The essentials can be found in [Hofstede et al., 1992d] and [Hofstede et al., 1992e]. This section is based mainly on [Verhoef et al., 1991] and [Verhoef, 1991d]. The verification rules that are part of Yourdon’s way of modelling are discussed in full detail in appendix A.

The Object structure diagramming technique discussed in section 2.3 is used to represent Yourdon’s interrelated set of concepts. We restrict ourselves to the two main diagramming techniques: the entity-relationship diagramming technique (see section 3.1.1) and the data flow diagramming technique (see section 3.1.2). The relationships between these two diagramming techniques are discussed in section 3.1.3.

3.1.1 Entity-Relationship Diagrams

According to [Yourdon, 1989], an “Entity-Relationship Diagram (ERD) is a network model that describes the stored data layout of a system at a high level of abstraction”. Figure 3.1
shows an example of an ERD.

![ERD Diagram](image)

**Figure 3.1: Example of an ERD**

In this diagram, "Product", "Sale", "Supplier", and "Customer" are Object types. Note that Object types are used here as part of a specific ERD, i.e. at application level, as opposed to Object types that are part of a specific Object structure at meta-level. In [Yourdon, 1989] Object types are described:

An object type represents a collection or set of objects (things) in the real world whose individual members (or instances) have the following characteristics:

- each can be identified uniquely in some fashion;
- each plays a necessary role in the system we are building;
- each can be described by one or more data elements. [...] Many database textbooks describe this as attributing data elements to an object type.

[...] Note that in all the examples of an object, we have used the singular form of a noun. This is not required, but it is a useful convention.

In this example, only the Object types "Customer" and "Supplier" are provided with Attributes: "Name", "Address", and "City". In both cases, the Data element "Name", which is underlined, identifies the Object type involved.
As figure 3.1 shows, Object types may be related. According to Yourdon:

Objects are connected to one another by relationships. […] Note also that there can be more than one relationship between two objects. […] A more common situation is to see multiple relationships between multiple objects. […] Note that, in some cases, we can have relationships between different instances of the same object type.

The Object type "Customer" has as subtypes "Regular Customer" and "Incidental Customer". Yourdon explains these subtypes:

The subtype/supertype object types consist of an object type and one or more subcategories, connected by a relationship. […] Note the subtypes are connected to the supertype via an unnamed relationship.

The Object type "Sale" is an associative object type indicator of the Relationship between "Customer" and "Product". Yourdon clarifies this notion as follows:

A special notation on the E-R diagram is the associative object type indicator; this represents something that functions both as an object and a relationship. […] The associative object type indicator is also the name of the relationship.

After this informal introduction of the main notions within Yourdon’s ERDs, the corresponding Object structure is presented, see figure 3.2. It is necessary that Object-type, Relationship, and Data-element are contained in the Object structure, as should be clear from the previous discussion. They are generalised into ERD-Object because they all have a Name.

Object types can participate in Relationships. An Object type can even participate several times in one Relationship. Each participation is characterised by a distinct Role. In some cases, it is convenient to treat a Relationship as an Object type itself. Such a Relationship is called an associative object type indicator.

Each Object type is described by a number of Data elements, some of which constitute the identification of the Object type involved. Each Object type can have subtypes, which are themselves Object types.

3.1.2 Data Flow Diagrams

According to [Yourdon, 1989], a Data Flow Diagram (DFD) "pictures a system as a network of functional processes". A DFD consists basically of Processes, Data stores, Terminators, and Data flows. A sample DFD is shown in figure 3.3.

In this example, "Receive order", "Ship books", and "Collect payments" are Processes. From Yourdon:

The first component of a DFD is known as a process. […] The process shows a part of the system that transforms inputs into outputs. The process is represented graphically as a circle. […] Note that the process is named or described with a single word, phrase, or simple sentence. For most of the DFD models that we will discuss in this book, the process name will describe what the process does.
"Orders", "Customers", and "Invoices" are examples of Data stores in figure 3.3. According to Yourdon:

The store is used to model a collection of data packets at rest. [...] Typically, the name chosen to identify the store is the plural of the name of the packets that are carried by flows into and out of the store. [...] Stores are connected by flows to processes.

Terminators are another relevant concept in DFDs. Yourdon states:

Terminators represent external entities with which the system communicates. [...] The flows connecting the terminators to various processes (or stores) in our system represent the interface between our system and the outside world. [...] Any relationship that exists between terminators will not be shown in the DFD model.

Processes, Data stores, and Terminators are connected by Data flows. In figure 3.3, they are represented as labelled arrows. Yourdon states:

A flow is represented graphically by an arrow into or out of a process. [...] The flow is described to describe the movement of chunks, or packets of information from one
Figure 3.3: A sample Data Flow Diagram, based on [Yourdon, 1989]

part of the system to another part. [...] For many complex real-world systems, the DFD will show the flow of materials and data. [...] Note that flows are named. The name represents the meaning of the packet that moves along the flow. [...] Note also that the flows show direction: an arrowhead at either direction (or possibly at both ends) indicates whether data (or material) are moving into or out of a process (or doing both).

Figure 3.4: An abstract DFD

There are many types of Data flows, see figure 3.4. Usually, Data flows are unidirectional flows between Processes, Data Stores, and Terminators (we refer to these three Object types
which may be connected by Data flows as DFD Objects). A Dialogue flow is a bidirectional flow between two DFD Objects. Since Yourdon has introduced the notion of Dialogue flow as a notational convention for the sake of convenience, we do not consider this notion to be an elementary concept in Yourdon's way of modelling. We consider a Dialogue flow to consist of two Data flows between the same pair of DFD Objects.

There are also more complex Data flows, that either diverge or converge. A diverging Data flow is a Data flow that splits into a Data flow group. A converging Data flow is a Data flow group that joins into one Data flow. The abstract example of figure 3.4 shows a diverging Data flow.

This informal presentation of DFDs has so far focused on the basic concepts: Processes, Data stores, Terminators, and Data flows. The two remaining issues are decomposition of Processes into a whole DFD, and extensions to DFDs for modelling real-time systems.

Processes may be decomposed into a new DFD, which itself consists of Processes. A decomposition hierarchy of DFDs and Processes is discerned. The root of the decomposition hierarchy is a DFD which usually consists of only one Process. This DFD is referred to as the Context diagram. Each Process has a Number, and the DFD which is its decomposition has the same Number. Processes which are not decomposed any further are described in a Process specification.

The data flow diagramming technique is extended to facilitate the modelling of real-time systems. Yourdon presents these extensions as follows:

> For another class of systems, the real-time systems, we need a way of modeling control flows (i.e., signals or interrupts). And we need a way to show control processes. [...]

A control process may be thought of as a supervisor or executive bubble whose job is to coordinate the activities of the other bubbles in the DFD; its inputs and outputs consist only of control flows. [...]

There is typically only one such control process in a single DFD. [...] The inside of a control process is modeled with a state-transition diagram, which shows the various states that the entire system can be in and the circumstances that lead to a state of change.

After the above informal introduction of the main notions to be found within Yourdon's DFDs, we present the corresponding Object structure of DFDs, see figure 3.5.

The notions of Process, Data-store, Terminator, Data-flow, Control-process, and Control-flow have to be distinguished. The notions of DFD, Process-Specification, STD, D-Name, and D-Number are also self-evident.

Since Processes, Data stores, Terminators, and Control processes are all connected by Data flows and Control flows, they are generalised to DFD-Objects. Since Control flows, DFD Objects, and Data flows share the property of having a Name, they are generalised into DFD-Elements.

The Object type Data-flow-group is introduced to express diverging or converging Data flows. These complexly structured Data flows are modelled as the combination of one Data flow and a set of Data flows.
To express two particular verification rules, the resulting Object structure is extended with two Relationships. The first verification rule states that the Data flows that are input for or output of a Process are also input for or output of the DFD at the next lower level of decomposition. It is therefore necessary not only to distinguish a decomposition relation between Processes and DFDs, but also to relate the Data flows involved.

For Data stores, the verification rule holds that the Name of a Data store should be the
plural of the Name of one of the connected Data flows. To express this rule, one should be able to relate the D-Name of Data stores and Data flows.

3.1.3 Relationship between ERDs and DFDs

ERDs and DFDs are related in various ways.

Object types and Data stores are related. Object types in an ERD correspond to Data stores in a DFD. This relationship is considered to be bidirectional: each Data store in a DFD suggests a corresponding Object type in an ERD and vice versa.

Yourdon's description of what he calls his event partitioning approach clarifies another relationship between ERDs and DFDs. The Event list turns out to be used for constructing both ERDs and DFDs.

The occurrence of nouns in the description of Events may lead to the introduction of Object types in an ERD. Each Event may, at the same time, lead to the introduction of Processes in a DFD which handle the Event at hand.

Events are also related to Data flows and Control flows in DFDs. To express this relationship, Events are modelled as a generalisation hierarchy. Three types of Events exist: Flow-oriented events, Temporal events, and Control events. Data flows in the Context diagram are either an indication that a Flow-oriented event has occurred, or that they are required for processing the Event, or that they are a response to an Event. Control events are associated with a Control flow in the Context diagram.

Figure 3.6 presents the relationships between ERDs and DFDs. In this figure, only those Object types in ERDs and DFDs are shown that are relevant for the description of the relationships between ERDs and DFDs.

3.2 Way of working

In this section, modelling knowledge from [Yourdon, 1989] is represented, as far as a way of working is concerned. The task structure diagramming technique discussed in section 2.3 is used to this end. This section is based mainly on [Verhoef et al., 1991] and [Verhoef, 1991d].

According to [Yourdon, 1989], the analysis process consists of two main stages, namely "Construct the essential model" and "Construct the user implementation model". These two stages are strictly separated. The first stage results in a model of what the system to be developed must do in order to satisfy the users' requirements, whereas the second stage results in a model of how this system will be implemented. Given our focus on the early stages of information systems development processes, this section is restricted to "Construct the essential model". This task is the main task of the task hierarchy.

This task consists of two subsequent subtasks: "Construct the environmental model" and "Construct the behavioral model". According to [Yourdon, 1989], the environmental model
defines the boundary between the system and the rest of the world (i.e., the environment in which the system exists). The behavioral model describes the required behaviour of the insides of the system necessary to interact successfully with the environment.

This discussion leads to the trivial partial task structure of figure 3.7. In the remainder of this section, the tasks "Construct the environmental model" (section 3.2.1) and "Construct the behavioral model" (section 3.2.2) are treated one by one.

### 3.2.1 Constructing the environmental model

Yourdon presents the construction of the environmental model as a straightforward task. The environmental model consists of three major components: a statement of purpose, a context diagram and an event list.

A statement of purpose of the system to be built, should be formulated first. With respect to the order in which the two other tasks should be performed, Yourdon provides the
Figure 3.7: Task structure representing the construction of the essential model

following guidelines:

You can begin with either the event list or the context diagram. It really doesn’t matter, as long as you eventually produce both components of the environmental model and check to see that they are consistent with each other.

You may also find yourself talking with people who are aware of all the things that come into the system and go out of the system; some users may be able to provide you with this information, or the maintenance programmers in charge of maintaining a current version of the system might be knowledgeable in this area. This will provide you with the pieces of the context diagram as a starting point. You can then discuss the transactions that the users send to the system and the responses they expect the system to make. This allows you to create the event list from the context diagram.

However, you may find yourself in a situation where the context diagram is not available. This is particularly common at the beginning of some systems development projects: it may not be easy to immediately identify the terminators and the various flows into and out of the system. In this case, it is often more practical to begin with an ERD diagram that shows the objects and relationships. Candidate events can then be found by looking for activities or operations that cause instances of a relationship to be created and deleted. Creation of the event list can then lead to the development of the context diagram.

Figure 3.8 presents the task structure representing the construction of the environmental model, based on the previous discussion.

3.2.2 Constructing the behavioral model

The task “Construct the behavioral model” results, according to [Yourdon, 1989], in “a well-organized process model and data model for presentation to the end user”. Within this task, two main subtasks are distinguished: “Construct the preliminary behavioral model” and “Construct the final behavioral model”.

66
3.2.2.1 Constructing the preliminary behavioral model

The preliminary behavioral model consists of a preliminary DFD and an initial ERD. No dependencies are distinguished between these two models. The construction of the two may take place in parallel. When these models are constructed, an explicit cross-checking task has to be performed.

The preliminary DFD is constructed using the event list. For each event in the event list, a process is added to the preliminary DFD. This process is named by describing the response that the system should make to the associated event. Two special cases may occur: (i) a single event may cause multiple responses, (ii) multiple events may cause the same response. Subsequently, “appropriate inputs and outputs are drawn” so that the processes in the preliminary DFD are able to make their required responses, and data stores are added, as appropriate, for communication between the processes. When the preliminary DFD has been finished, it should be verified against the context diagram. Input flows that are part of the context diagram should be associated with input flows on one of the processes in the preliminary DFD. Output flows produced by processes in the preliminary DFD should be sent to external stores or terminators in the context diagram.

Very few guidelines are given with respect to the development of the initial ERD. Yourdon explains:

You may find that the event list is useful for creating the initial ERD. Nouns in the
event list will often turn out to be objects in the ERD.

The construction of the preliminary behavioral model ends with a cross-checking task. From [Yourdon, 1989]:

As the ERD and the DFD are being developed in parallel, they can be used to cross-check each other. Thus, stores that have been tentatively defined in the preliminary DFD can be used to suggest objects in the preliminary ERD; and objects that have been tentatively identified in the preliminary ERD can be used to help choose appropriate stores in the preliminary DFD. Neither model should be considered to be the dominant model that controls the other; each is on equal footing and can provide invaluable assistance to the other.

3.2.2.2 Constructing the final behavioral model

Yourdon distinguishes three subtasks as part of the task “Construct the final behavioral model”, namely “Finish DFD”, “Finish ERD”, and “Finish STD”. The name of the latter task suggests that this task is the successor of another task. It is not clear which task this predecessor should be. In this section, “Finish STD” will not be dealt with because of our restriction of a way of modelling to DFDs and ERDs. The two remaining subtasks may be performed in parallel.
The task "Finish DFD" involves a levelling process, having the preliminary DFD as its input. Since the preliminary DFD may consist of dozens or hundreds of processes, distinguishing levels of decomposition is considered to be necessary. As a result of this process, several DFDs come into being. These DFDs are related to each other by decomposition relations.

Yourdon provides three guidelines for upward levelling:

1. Each grouping of processes should involve closely related responses.
2. If you see a group of processes in the preliminary DFD that refer to a common store, and no other process in the preliminary DFD refer to that store, then you can create a higher level bubble that hides the store.
3. You should create aggregates or groups from the preliminary DFD consisting of roughly 7 plus or minus 2 chunks of information, where a process (and its related flows) or a store could be considered a chunk.

Downward levelling is recommended if some processes in the preliminary DFD exist which are no primitive processes. For downward levelling, the following informal guideline is supplied by Yourdon:

In some cases, a pure functional decomposition approach is appropriate. [...] In other cases, the bubble's incoming dataflows and outgoing dataflows will provide the best guidance for downward levelling.

The task "Finish ERD" is presented very briefly. Yourdon only states:

Much of this improvement can take place simply by assigning or attributing data elements to the appropriate object types; this will usually help us identify new object types or unnecessary object types.

Figure 3.9 shows a task structure representing the construction of the behavioral model.

### 3.3 Relationship between a way of modelling and a way of working

Several relationships between a way of modelling and a way of working have been discussed before, see section 1.2.5. They are repeated here for convenience: (i) modelling tasks have a scope, (ii) information interdependencies exist between modelling tasks, and (iii) the resulting products of modelling tasks have to obey verification rules.

The verification rules which are related both to Yourdon's way of modelling and his way of working are discussed in appendix A. This section discusses the other two relationships for Yourdon's way of modelling and way of working.

Yourdon's way of modelling has been presented in section 3.1. The Object structure which represents Yourdon's way of modelling has been presented in figures 3.2, 3.5, and 3.6.
Representing prescribed modelling knowledge

<table>
<thead>
<tr>
<th>Character</th>
<th>Leaf modelling task</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>Formulate statement of purpose</td>
</tr>
<tr>
<td>b</td>
<td>Construct context diagram</td>
</tr>
<tr>
<td>c</td>
<td>Construct event list, context diagram based</td>
</tr>
<tr>
<td>d</td>
<td>Construct ERD</td>
</tr>
<tr>
<td>e</td>
<td>Construct event list, ERD based</td>
</tr>
<tr>
<td>f</td>
<td>Construct context diagram, event list based</td>
</tr>
<tr>
<td>g</td>
<td>Construct preliminary DFD</td>
</tr>
<tr>
<td>h</td>
<td>Check preliminary DFD against context diagram</td>
</tr>
<tr>
<td>i</td>
<td>Construct initial ERD</td>
</tr>
<tr>
<td>j</td>
<td>Cross-check preliminary DFD against initial ERD</td>
</tr>
<tr>
<td>k</td>
<td>Finish DFD</td>
</tr>
<tr>
<td>l</td>
<td>Finish ERD</td>
</tr>
</tbody>
</table>

Table 3.1: Leaf modelling tasks in Yourdon's way of working

Yourdon's way of working has been presented in section 3.2. The Task structure which represents Yourdon's way of working has been presented in figures 3.7, 3.8, and 3.9. In this section we discuss the leaf modelling tasks from this Task structure. Table 3.1 shows the characters which are used to refer to the leaf modelling tasks.

The leaf modelling tasks represent Yourdon’s way of working only from a model view (see section 1.2.6): most leaf modelling tasks lead to the creation of a complete model, e.g. "Construct preliminary DFD". In other words, the scope of these leaf modelling tasks consists of a large number of Object types which are part of the Object structure that represents Yourdon’s way of modelling.

The availability of the concept of Schema type and the distinction between three types of model creation, see section 2.3.4.2, enable us to represent the scopes of the leaf modelling tasks and the information interdependencies between the leaf modelling tasks in a very compact way.

In the Object structure that represents Yourdon’s way of modelling, we define several Schema types by enumerating their element Object types and by uniting Schema types which have been defined before, see table 3.2. Note that the element Object types include Relationships between Object types. To improve readability, Relationships have not been named in the Object structures given in figures 3.2, 3.5, and 3.6. Instead, the Relationship names can be found in appendix A.

From table 3.3 the scopes of the leaf modelling tasks and the information interdependencies between the leaf modelling tasks are easily derived. This table employs the Schema types defined before. The characters in the table refer to the leaf modelling tasks in the Task structure which represents Yourdon's way of working.

For each leaf modelling task, its scope contains the Object types that are element of the Schema types which are created, used, or modified. In other words, the scope of each leaf
<table>
<thead>
<tr>
<th>Schema type</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statement-of-purpose</td>
<td>{Statement-of-purpose}</td>
</tr>
<tr>
<td>Event-list</td>
<td>{Control-event, Event, Event-list, Flow-oriented-event, Temporal-event}</td>
</tr>
<tr>
<td>CD-to-EL</td>
<td>{Control-event-indication, Event-handling, Response-production, Stimulus-recognition}</td>
</tr>
<tr>
<td>Simple-ERD</td>
<td>{E-Name, ERD, ERD-Object, ERD-Object-name, Object-type, Participation, Relationship, Role}</td>
</tr>
<tr>
<td>ERD-to-EL</td>
<td>{Relationship-manipulation}</td>
</tr>
<tr>
<td>EL-to-CD</td>
<td>CD-to-EL</td>
</tr>
<tr>
<td>Preliminary-DFD</td>
<td>Context-diagram ∪ {Complex-flow, Data-flow-group, D-Number, Process-decomposition, Process-number}</td>
</tr>
<tr>
<td>CD-to-DFD</td>
<td>{Data-flow-correspondence, DFD-Name}</td>
</tr>
<tr>
<td>EL-to-DFD</td>
<td>{Event-handling}</td>
</tr>
<tr>
<td>Initial-ERD</td>
<td>Simple-ERD ∪ {Associative-object-type-indicator, Supertype/subtype-indicator}</td>
</tr>
<tr>
<td>EL-to-ERD</td>
<td>{Object-type-derivation}</td>
</tr>
<tr>
<td>DFD-vs-ERD</td>
<td>{ERD-DFD-Coherence}</td>
</tr>
<tr>
<td>Completed-DFD</td>
<td>Preliminary-DFD ∪ CD-to-DFD ∪ {Control-process-decomposition, DFD-Number, Process-specification, Process-spec, STD}</td>
</tr>
<tr>
<td>Completed-ERD</td>
<td>Initial-ERD ∪ {Data-element, Identification, Object-type-description}</td>
</tr>
</tbody>
</table>

Table 3.2: Schema types discerned in Yourdon’s way of modelling.

modelling task is the set of Object types which are element of the Schema types mentioned in table 3.3. Information interdependencies between leaf modelling tasks exist if certain modelling tasks use or modify the population of a Schema type which in its turn has been manipulated before by another leaf modelling task.

Figure 3.10 visualises the existing information interdependencies between leaf modelling tasks, using the information interdependency diagramming technique introduced in section 2.3.4.2. Several information interdependencies are represented. It is clear from this figure that Yourdon’s Structured Analysis results in the following final products: a statement of purpose, an event list, a data flow diagram, and an entity-relationship diagram.
Representing prescribed modelling knowledge

<table>
<thead>
<tr>
<th>Leaf modelling task</th>
<th>Schema type manipulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>create Statement-of-purpose</td>
</tr>
<tr>
<td>b</td>
<td>create Context-d</td>
</tr>
<tr>
<td>c</td>
<td>create Event-list using CD-to-EL</td>
</tr>
<tr>
<td>d</td>
<td>create Simple-ERD</td>
</tr>
<tr>
<td>e</td>
<td>create Event-list using ERD-to-EL</td>
</tr>
<tr>
<td>f</td>
<td>create Context-diagram using EL-to-CD</td>
</tr>
<tr>
<td>g</td>
<td>create Preliminary-DFD modifying Context-diagram using EL-to-DFD</td>
</tr>
<tr>
<td>i</td>
<td>create Initial-ERD using EL-to-ERD</td>
</tr>
<tr>
<td>j</td>
<td>create Preliminary-DFD modifying Preliminary-DFD using DFD-vs-ERD</td>
</tr>
<tr>
<td>j</td>
<td>create Initial-ERD modifying Initial-ERD using DFD-vs-ERD</td>
</tr>
<tr>
<td>k</td>
<td>create Completed-DFD modifying Preliminary-DFD</td>
</tr>
<tr>
<td>l</td>
<td>create Completed-ERD modifying Initial-ERD</td>
</tr>
</tbody>
</table>

Table 3.3: Schema types manipulated by the leaf modelling tasks.

The development of data flow diagrams and entity-relationship diagrams traverses several stages. Subsequent stages turn out to employ larger sets of Object types. Furthermore, it is clear that the construction of the simple ERD as part of the construction of the environmental model is not related to the construction of ERDs during the construction of the behavioral model, but serves only as an auxiliary model.

Finally, as a side remark, it should be noted from table 3.2 that the Object type Statement-of-purpose has been added to the Object structure which represents Yourdon's way of modelling. This addition is necessary because of the presence of the leaf modelling task "Formulate statement of purpose" in the Task structure.

### 3.4 Conclusions

This chapter gives a representation of prescribed modelling knowledge, based on [Yourdon, 1989]. The representation is restricted to a part of the information modelling process (the construction of the essential model) and to the two most well-known modelling techniques (ERDs and DFDs).

An extensive listing of the definitions of all verification rules can be found in appendix A.

Since some assumptions have to be made during the process of representing prescribed modelling knowledge, we present some specific conclusions with respect to the meta-modelling process in section 3.4.1. Furthermore, some conclusions are given on Yourdon's Modern Structured Analysis, see section 3.4.2.
Figure 3.10: Information interdependencies in [Yourdon, 1989]

3.4.1 Meta-modelling process

Given the restrictions to the construction of the essential model and to the ERDs and DFDs, it has turned out to be possible to represent Modern Structured Analysis by the
meta-modelling technique given in chapter 2. In describing the way of modelling, the way of working and their interrelationships in Modern Structured Analysis (see sections 3.1, 3.2 and 3.3), the combination of PSM, the task structure diagramming technique and LISA-D proved to be an adequate meta-modelling technique.

In this section we discuss the design decisions that were made during the representation of Yourdon’s Modern Structured Analysis.

During the construction of the Object structure it became apparent that several modelling constructs which are well known from other information systems development methods, are not discussed in [Yourdon, 1989]. An example of such a modelling construct is the notion of partitioning subtypes. However, such constructs have not explicitly been excluded from the textbook. In this chapter, attention has only been paid to those modelling constructs that are explicitly mentioned in Yourdon’s textbook.

Several “non-Yourdon” Object types have been added to the Object structure. The main reason underlying this decision is that some Object types had Relationships in common, which necessitated the decision to model them as a specifier of a new Object type. ER-element is an example of such an introduced non-Yourdon Object type.

During the integration of the Object structure and the Task structure (see section 3.3), two problems had to be tackled. One, the Task structure contained manipulations of Object types which had not been distinguished before in the Object structure, for example Statement of purpose and Event list. Hence, the Object structure had to be extended with such Object types. Two, the leaf modelling tasks distinguished in the way of working were high-level tasks when compared to the Object types distinguished in the way of modelling. Yourdon’s way of working was apparently presented from model view, whereas his way of modelling was presented from component view, see figure 1.2 on page 7. The notion of Schema type was employed as an elegant way to deal with this discrepancy.

### 3.4.2 Modelling knowledge

Sections 3.1, 3.2 and 3.3 represent modelling knowledge prescribed by Yourdon. It is not the purpose of this study to achieve comparative statements or value judgements. We therefore restrict ourselves to some general observations regarding Modern Structured Analysis:

- **Unclear coherence between model types**
  
The coherence between models produced by performing the (high-level) leaf modelling tasks distinguished in the Task structure is unclear. For instance, it has not been made explicit whether the ERDs delivered by the tasks “Construct ERD”, “Construct the initial ERD”, and “Finish the ERD” have any relations.

- **Linear way of working**
  
  A paradoxical contrast can be observed between the allowed flexibility of project management at managerial level and the signalled rigidity at operational level. With respect to the managerial level, one general remark is made in [Yourdon, 1989]. We quote:
In the most extreme situation, all the activities in the structured project life cycle could be taking place simultaneously. At the other extreme, the project manager could decide to adopt the sequential approach, finishing all of one activity before commencing on the next.
It is convenient to have some terminology to help talk about these extremes, as well as about compromises between the two extremes. A radical approach to the structured project life cycle is one in which activities 1 through 9 take place in parallel from the very beginning of the project, and the survey and the analysis continue until the last day of the project. By contrast, in a conservative approach to the structured project life cycle, all of activity N is performed before activity N+1 begins.

The subject of this chapter is the representation of information modelling activities, which we regard to take place at the operational level, see chapter 1. From chapter 3 we conclude that alternative strategies as part of Yourdon’s way of working do not occur often. This can be deduced from the low number of decisions in the Task structures. Obviously, a linear way of working, in which complete and correct models are delivered in one go, is more or less prescribed.

- **Increasing structuredness**
  As figure 3.10 clearly shows, several modelling stages are discerned in which the use of modelling techniques gradually changes. More modelling concepts are used in the course of Yourdon’s Structured Analysis, and a growing set of verification rules is satisfied.

- **Different levels of detail, way of modelling and way of working**
  The way of modelling has been described in a far more detailed way than the way of working. This discrepancy is most striking in the case of data modelling, where the most detailed tasks are still of a very global level, e.g. “Construct the initial ERD”. Yourdon has paid significantly more attention to the construction of DFDs by providing guidelines and heuristics. This becomes clear from the number of verification rules on DFDs.

- **Several imperfections**
  Some imperfections can be found in the textbook. Three examples of minor imperfections are given here. One, it is not clear whether one Object type can have several subtype/supertype indicators. The phrase “consist of an object type and one or more subcategories, connected by a relationship” (see [Yourdon, 1989], p. 242) suggests that the idea of a partition is not present, however, partitions have not been excluded explicitly. Two, Yourdon is not consistent with respect to the direction of the arrow that represents an associated object type indicator in ERDs (see [Yourdon, 1989], p. 241). Three, the extensions to DFDs for real-time systems (see [Yourdon, 1989], pp. 172-173) have not been dealt with in his description of the analysis process.
An imperfection of far greater importance concerns the diffuse relationship between the way of modelling and the way of working. Yourdon's description of a way of modelling is relatively detailed. Various modelling constructs are introduced as part of the two diagramming techniques. Examples of these modelling constructs are complex flows in DFDs and subtypes in ERDs. Yourdon's description of a way of working presents several data modelling tasks and process modelling tasks in the course of an information systems development project. From this description, however, it is not clear in which data modelling task subtypes may be employed first, or in which process modelling task complex flows are considered to be useful. We have assumed that these specific constructs may be used in the last data modelling task or process modelling task, respectively, since the description of former data or process modelling task does not involve the use of these specific modelling constructs.

Summarising, we conclude that the Yourdon textbook provides the most important concepts only by example. The most common situations are covered by these examples, but the question remains of how concepts should be employed in exceptional situations. Yourdon's way of modelling is presented vaguely and incompletely. The relationship between the way of modelling and the way of working is diffuse, in particular because these ways are presented at different levels of detail. Furthermore, ad-hoc extensions are part of Yourdon's modelling knowledge (e.g. the extension to be able to deal with real-time systems, which are not dealt with in Yourdon's description of a way of working at all!). Information engineers are left with the problem how to deal with these incoherences and how to fill the gaps in the modelling knowledge provided.
Chapter 4

Representing applied modelling knowledge: Case 1

Chapters 4 and 5 concern the representation of applied modelling knowledge. The metamodelling technique given in section 2.3 is used with one exception, which is that verification rules that are part of applied modelling knowledge, are not represented, in order to avoid extensive listings.

This chapter presents the approaches followed by three experienced information engineers when solving case 1. The case concerns a planning problem at the Dutch Open University. The Open University is referred to by its acronym OU in what follows. The case is described in more detail in appendix B. Modelling knowledge acquisition results are presented in sections 4.1, 4.2, and 4.3. Each of these sections pays attention to the information engineer's way of working and his way of modelling. Section 4.4 presents some concluding observations.

4.1 Information Engineer A

This section describes the approach followed by information engineer A. This description is derived from the protocol transcript (see [Verhoef, 1990]) using the approach described in sections 2.4.3 and 2.4.4. The way of working, the way of modelling, and relationships between the way of working and the way of modelling are represented sequentially. We have restricted the representation of relationships between the way of working and the way of modelling to the representation of information interdependencies. Observations conclude this section.

4.1.1 Way of working

When solving case 1, information engineer A followed an approach consisting of four main tasks: problem orientation, problem analysis, system design, and finally validation of findings. These tasks are presented in four sections. The overall task is shown in figure 4.1.
4.1.1.1 Perform problem orientation

The task "Perform problem orientation" (see figure 4.2) consists of three subtasks. The first one is a very broad task, leading to a rough insight into the problem area. After this, the objectives to be achieved in the remainder of the session are defined and validated.

The first subtask, "Perform initial problem investigation", is in itself a complex task. Its decomposition is shown in figure 4.3. This task focuses on two issues in parallel. The first issue deals with the problem type. In this particular case, information engineer A concluded that the problem at hand could be classified as a scheduling problem. The second issue regards the organisation as a whole, resulting in a business model. A business model consists of an organisation structure diagram, an initial object model, and an initial business activity model. The organisation structure diagram and the initial business activity model are related in the sense that business activities are performed in organisation units.

4.1.1.2 Perform problem analysis

The task "Perform problem analysis", see figure 4.4, deals with demarcating the problem area, with constructing models of the problem area, and with refining the objectives formulated in the previous task.

Problem area demarcation is the first subtask. It is an important heuristic to demarcate as small a problem area as possible. The demarcation of the problem area, see figure 4.5, takes place by means of a context diagram, in which the environmental factors are defined, as well as the relationships between the problem area and the environmental factors.
Figure 4.2: Performing a problem orientation

Figure 4.3: Performing an initial problem investigation

Three scenarios are available to achieve a suitable problem area demarcation. The choice of a scenario depends on the characteristics of the problem area. If a global insight into the problem environment exists, the context diagram is constructed immediately. The context diagram is constructed in an output driven way: for each environmental factor, the output flows, which are sent to the environmental factor, are investigated before the input flows.
If no insight into the problem environment exists, then the construction of an event list precedes the construction of a context diagram in most of the cases. In these cases, the way in which the context diagram is constructed depends on the event list. This scenario is not followed if the problem area includes administrative organisation. In these cases, the construction of an initial entity model precedes the construction of the context diagram.

Problem area demarcation is succeeded by the construction of an initial model of the problem area. This task starts with the construction of a process list and an object model, showing the main processes in the real system, and the main objects and their interrela-
tionships in the real system, respectively, see figure 4.6. The object model is then used to create an initial entity model. All objects in the object model appear as entity types in the initial entity model. The process list is used to verify the initial entity model in an intuitive way: for each process on the process list, an investigation is made of which entity types and relationships may be necessary.

The process list is also used to construct a DFD. For each process on the process list, the first investigation determines which data flows should be consumed, the second which data flows should be produced. Process modelling is clearly an input driven task at this stage. The construction of a DFD may involve physical flows in addition to data flows.

The DFD is verified in two ways. One, it is checked against the context diagram in order to verify whether all data flows on the context diagram correspond to data flows on the DFD. This is not a one-to-one correspondence: many data flows on the context diagram may correspond to one data flow on the DFD. For example, the context diagram produced when information engineer A was solving case 1, contained many output data flows which were named “response” and which differed only in their destination. The second verification of the DFD involves reference models. In practice, the verification of the DFD was based on a general scheduling reference model, containing at least two processes: “activity scheduling” and “resource scheduling”. If these verifications lead to adapting the DFD, the process list is adapted accordingly. DFD verification ends the construction of the initial model of the problem area, which consists of the context diagram, the process list, the DFD, and the initial entity model.

Refinement of the objectives deals with refining the level of detail at which they should be realised. For example, information engineer A investigated whether insight into work load was required at employee level or at organisation unit level.

After this refinement, the initial model of the problem area is adapted, see figure 4.7. The adaptation task is performed in several iterations, until two conditions are fulfilled: the model of the problem area has the required level of detail, and the model of the problem area is consistent. Adaptations take place in a random order: the current entity model may lead to adapting the process model. The adapted version of the process model may in turn involve a further refinement of the entity model, and so on.

The second condition, consistency of the model of the problem area, corresponds to a set of verification rules. This set includes the following five verification rules: (i) each process on the process list should also be on the DFD, and vice versa, (ii) each data flow in the context diagram should correspond to a data flow in the DFD, (iii) each data store in the DFD should correspond to an entity type or a relationship in the entity model, (iv) each data flow should relate to an entity type in the entity model, and (v) environmental factors in the context diagram may be associated to entity types in the entity model.

4.1.1.3 Perform system design

The task “Perform system design”, see figure 4.8, transforms the model of the problem area into a model of the information system to be designed. The information needs are defined, and the entity model and the process model are adapted accordingly.
Performing a system design starts with defining and refining information needs. In reality, the refining of information needs took place using a number of problem specific criteria, such as exception handling for special exam categories. Subsequently, the list of information needs is extensively validated with respect to completeness.

After defining, refining, and validating the information needs, a model of the information
system to be designed is constructed, i.e. both the entity model and the process model are adapted and refined. Adapting the entity model amounts mostly to describing the entity types by attributes, and to adding quantitative information to entity types and relationships. The latter adaptation serves as a preparation for subsequent stages, although it also supports communication with the users. The context diagram is adapted in such a way, that the output data flows are explicitly related to the defined information needs. The process list and the DFD are adapted accordingly. This is followed by an investigation of the frequency of the output data flows on the context diagram. This serves as a preparation for subsequent stages.

Finally, the resulting model is adapted analogously to the problem analysis task by several iterations, see figure 4.9, until the above mentioned set of verification rules is fulfilled.

4.1.1.4 Validate findings

The results achieved during the three main tasks are validated in a linear order. First, the statement of purpose, which consists of several objectives, is validated, then the context diagram is presented. Each flow in the context diagram is explicitly related to one of the objectives. Next, a detailed entity model is presented and validated. Both cardinality and optionality constraints are part of the detailed entity model. The process list and the data flow diagram are then shown. Processes in the DFD correspond to processes on the process
list. All data flows in the context diagram correspond to data flows in the DFD. In the following task, the detailed entity model is extended by a data definition: each entity type is described by a number of attributes. Key attributes are indicated. The classifying entity types are provided with sample values. Classifying entity types are those entity types which categorise the instances of another entity type. In the OU case, “exam type” is a classifying entity type. Finally, attention shifts to the DFD again to indicate the processes that can be automated and those that can not.

4.1.2 Way of modelling

We focus on information engineer A’s way of modelling, in particular on the various types of entity models and process models he constructed when solving case 1.

4.1.2.1 Data modelling

From our description of the approach which information engineer A followed when solving case 1, see section 4.1.1, we recapitulate that several data modelling tasks are performed. These data modelling tasks are “Construct initial object model”, “Construct object model”, “Construct initial entity model”, “Adapt initial entity model”, “Extend entity model”, and “Adapt detailed entity model”.

We distinguish between object models, initial entity models, detailed entity models, and extended entity models, for information engineer A’s way of modelling. The object model distinguishes itself from the entity models in that it focuses on objects in a real system, whereas the entity models involve entity types in an information system. The construction of object models is not restricted by many verification rules. Objects may be related in different ways. The main notions in object models are referred to as Object and O-Relationship, respectively. In the case that one and the same Object participates twice in an O-Relationship, these are distinguished by adding a Role. Objects may be provided with so-called generating attributes, which justify the existence of the Object at hand. Figure 4.11 gives a partial meta-model which describes Object models.
The entity models in information engineer A’s way of modelling are a binary variant of the entity-relationship diagramming technique. The notions used when constructing entity models are given in figure 4.12. The main notions are Entity type, Relationship, and Data element. Entity types play a Role in a Relationship with a certain Cardinality. Entity types are described by some Data elements and are identified by a subset of these Data elements. Some Entity types are also partitioned into Subtypes. Both Entity types and Relationships may be provided with quantitative information. The only difference between initial entity models and detailed entity models is that initial entity models do not involve Data elements, Subtypes, or Quantities. Extended entity models also involve Key attributes and Sample values.

Figure 4.12 also shows that Entity model objects in Entity models may be related to Object model objects in Object models.

### 4.1.2.2 Process modelling

It can be recalled from section 4.1.1 that several process modelling tasks are part of the approach followed by information engineer A, when solving case 1: “Construct business ac-
tivity model”, “Construct context diagram”, “Construct context diagram, event list based”, “Construct process list”, “Construct DFD, process list based”, “Verify DFD against reference model”, “Verify DFD against context diagram”, “Adapt process list”, “Adapt DFD”, and “Relate DFD to information needs”.

These tasks all manipulate the same process modelling concepts. The tasks differ in the way in which they relate process models to each other, or in the way in which they relate process models to other models. In other words, information interdependencies turn out to play an eminent role.

Figure 4.13 presents the main process modelling concepts that are part of the approach followed by information engineer A. Process models contain Processes, Data stores, and Environmental factors. These three Process model objects may be connected by Flows. Flows are either Data flows or Physical flows. Note, that the term “Process model” is used here to refer to Context diagrams and DFDs.

Processes are performed at Organisation units. Processes may be decomposed into new Process models themselves. In the case of a decomposition, Data flows which are connected to the Process correspond to Data flows which are input for or output of the new Process model. Processes which are part of a Process model are also listed on a Process list. This Process list serves as an intermediate result only, since it can be derived from the Process model.

Data flows are related explicitly to the system output: they either meet an Objective directly or they are needed for another Data flow. These relationships reflect an output
driven way of working. Data flows may also be related to Events. This relationship reflects an input driven way of working.

Figure 4.13 also shows that Process models and Entity models are related to each other, in several ways. Each Data store can be related to one or more Entity types. Each Data flow can be related to one or more Attributes. Furthermore, Environmental factors may be related to Entity types.
4.1.3 Relationship between a way of modelling and a way of working

Three stages can be discerned in the construction of entity models, resulting in initial entity models, detailed entity models, and extended entity models. Gradually, more verification rules and modelling concepts are used over the entity modelling stages. The construction of the initial entity model is preceeded by the construction of a so-called object model.

The construction of process models starts with the construction of a context diagram. This task may use an initial entity model or an event list, depending on the problem situation at hand. The context diagram is decomposed into a DFD, or a hierarchy of DFDs. DFDs are summarised using Process lists.
The construction of process models and entity models are highly intertwined. The context diagram, the DFDs, and the process lists are used when constructing an initial entity model or a detailed entity model, and vice versa.

Figure 4.14 clearly illustrates the several stages of entity modelling, and the relationships between process modelling and entity modelling.

Figure 4.14: Information interdependency diagram, representing relationships between the way of modelling and the way of working of information engineer A when solving case 1

### 4.1.4 Observations

In the approach followed by information engineer A, extensive attention is paid to the construction of process models and entity models. Process modelling tasks and data modelling
tasks are performed alternately. Extending or refining a data model often leads to adapting a process model, and vice versa. Information engineer A refers to this equilibrium metaphorically as the "zip approach", suggesting frequent shifts in attention between process modelling tasks and data modelling tasks.

Both process models and entity models go through several stages, which are characterised by increasing level of detail, reflected by an increasing number of concepts and verification rules. Throughout these stages, coherence of the model of the problem area is a very important criterion. Several information interdependencies exist between the various model types.

Two different strategies are available to construct process models. Precedence analysis is an output driven approach, in which each data flow is related to those data flows that are needed to produce the former one. The opposite approach involves the construction of an event list, and is characterised as input driven.

An important heuristic in the problem area demarcation stage, is to choose a problem area that is as small as possible. Finally, as an integral part of his approach, information engineer A pays considerable attention to validating his impressions and intermediate models. In particular, completeness of results is often validated.

### 4.2 Information Engineer B

This section presents a description of the approach followed by information engineer B.

![Figure 4.15: The overall task](image)

This description is derived from the protocol transcript (see [Harst, 1991b]) using the ap-
proach described in sections 2.4.3 and 2.4.4. The way of working, the way of modelling, and relationships between the way of working and the way of modelling are represented sequentially. We have restricted the representation of relationships between the way of working and the way of modelling to the representation of information interdependencies. Observations conclude this section.

4.2.1 Way of working

When solving case 1, information engineer B followed an approach consisting of four main subtasks: problem orientation, investigation of information needs, solution finding, and validation of the findings. The overall task is shown in figure 4.15. These subtasks are treated in the following four sections.

![Diagram](image)

Figure 4.16: Performing a problem orientation
### 4.2.1.1 Perform problem orientation

The task "Perform problem orientation", see figure 4.16, results in an insight into the problem area, and, more specifically, into bottlenecks in that area. The first task determines which operating processes constitute the problem area, and in particular, whether the problem area could be demarcated to specific aspects, e.g. to finances or logistics. When performing this task, information engineer B concluded that management and control processes constituted the problem area, rather than operating processes. This conclusion strongly influenced the rest of his approach to the problem, as he decided not to pay any further attention to operating processes, apart from acquiring a brief overview of the main operating processes.

Two tasks are then performed in parallel, namely an investigation of specific bottlenecks and an investigation of the organisation structure.

![Diagram of Investigating specific bottlenecks](image)

**Figure 4.17: Investigating specific bottlenecks**

The investigation of specific bottlenecks, see figure 4.17, leads first to a brief overview of the main operating processes, from input to output. Then, specific bottlenecks with regard to management and control of these operating processes are debated. At this stage, each bottleneck is classified as either an organisational bottleneck or an information bottleneck, and the underlying causes of the bottlenecks are investigated. The sincerity of each of the causes is judged to find out whether the problem area is worth the effort of finding solutions. Subsequently, the problem owner's expected solutions are discussed. In particular, the
benefits to be expected from an automated information system are investigated. Attention is also paid to the possibility of connecting the new information system to existing information systems within the organisation.

The investigation of the organisation structure results in an overview of the main organisation units and their interrelations. The main tasks and responsibilities of each organisation unit distinguished are inquired.

The problem orientation is finished by formulating and validating the problem statement.

4.2.1.2 Investigate information needs

The task “Investigate information needs”, see figure 4.18, starts with the listing of information needs. What output should be delivered by the information system to be designed, is of interest, as is why this output should be delivered. In other words: the information needs that are formulated in this task are related to the bottlenecks listed in the previous task. Three subtasks are then executed in parallel. A list of candidate functions is made up, an entity list is constructed (see figure 4.19), and a list of external parties is made up. To ease communication, some of the candidate entity types are provided with some sample values.

![Figure 4.18: Investigating information needs](image)

Entity types are selected from the list of candidate entity types, using two criteria: is it relevant to store information on these entity types, and is it possible to derive this information from other entity types, or is it elementary information. It is investigated for each of the selected entity types, what attributes constitute the information need involved. The investigation of information needs is finished by a general discussion with the problem owner about information needs.
4.2.1.3 Find solution

Using the candidate entity types and the candidate functions listed in the previous task, the solution is described as a combination of an entity model and a function model, see figure 4.20.

The construction of the function model results in a decomposition hierarchy of function lists, see figure 4.21. A function list contains several functions which are roughly of an equal size (the balancing criterion). Functions may be assigned a priority level. Each function has a list of requirements associated with it. Requirements result from information needs. The function model is verified against the entity model by checking whether leaf functions in the function decomposition hierarchy can be regarded as a manipulation or a set of manipulations of one or more of the entity types and/or relationships in the entity model. In addition to these function specific requirements, requirements are formulated for the information system to be designed as a whole.
4.2.1.4 Validate findings

The task "Validate findings", see figure 4.22, results in preparing and giving a presentation of the findings. The results achieved are briefly discussed. Validation concerns the problem area definition, the list of bottlenecks, the problem justification, the information needs and the function model, respectively. Finally, some recommendations are presented with regard to future development tracts. In the validation stage, the function model should be presented and discussed before the entity model, as the problem owner needs to have an insight into the functionality offered instead of in the issue whether the data structure provides a basis for realisation of the functionality. Often, the entity model should not be presented at all because of its high level of abstraction.

4.2.2 Way of modelling

We focus on information engineer B’s way of modelling, in particular on the various types of data models and process models he constructed when solving case 1. These model types played a role of limited importance. Key notions related to the problem solving process as such. We deal with these concepts before we relate them to the process modelling concepts and the data modelling concepts used, see figure 4.23.

Bottleneck, information need, and requirement are important concepts related to the problem solving process. Bottlenecks disturb one or more operating processes. For operating processes, as well as for managerial processes, a responsible organisation unit is discerned. Organisation units form a decomposition hierarchy, within which each unit in itself has a certain task.

Bottlenecks are classified as either information bottlenecks or organisation bottlenecks. It is important to know the underlying causes of a bottleneck, and to find out whether the underlying causes are important or not. These findings determine the problem solving process.
In order to solve a bottleneck, information needs are formulated, which in turn result in specific requirements.

### 4.2.2.1 Data modelling

Data modelling results in entity models. Figure 4.24 shows the main notions relevant to entity models.

Entity models are considered to be *abstract* models of reality: it is difficult to understand them at first sight. For this reason, Entity types are often illustrated by a Sample value. Entity types are related to one another by Relationships. Each Entity type occurs in a Relationship with a specific Cardinality. Entity types are described by Attributes.

It should be checked whether the information needs which have been formulated can be realised by the Entity model. Information needs can be realised by the presence of Entity types in the Entity model, but may also be realised by the presence of Attributes or Relationships. For this reason, Entity types, Relationships, and Attributes are generalised into Entity model objects.

### 4.2.2.2 Process modelling

Process models are rarely constructed as part of the approach followed. They are used mainly to describe and to communicate the functionality of the information system to be developed. For this reason, they are referred to as Function models. Figure 4.25 presents the main notions in Function models, as they are employed in the approach followed by information engineer B.
Functions can be described at different levels of detail. It is therefore convenient to describe a complex Function as a Function model. Hence, a Function model decomposition hierarchy can be discerned.

Each Function may be subject to several requirements. Furthermore, each Function has a certain Size and a certain Priority. Function models do not only comprise Functions. Data stores and External parties are also part of Function models. As Functions, Data stores, and External parties share the property that they may be connected by Data flows, they are generalised into so-called Function model objects.

If a Function is decomposed into a Function model, it is important to check whether all Data flows connected to that Function correspond to Data flows that are input for or output of the corresponding Function model. To enable this check, it is necessary to register corresponding Data flows within Function model hierarchies.

Entity models are related to Function models. Functions in a Function model are either decomposed into a new Function model themselves, or they perform a specific manipulation of an Entity model object. This manipulation is of a particular Manipulation type, and may take the values "Create", "Read", "Use", or "Delete".
4.2.3 Relationship between a way of modelling and a way of working

Both function modelling and entity modelling clearly go through two stages. The first stage focuses on the identification of entities, functions, and external parties, whereas the second stage involves the actual construction of the entity model and the function model, relating the identified elements.

Both models should be verified against each other with respect to the aspect of completeness. Figure 4.26 shows the relationships between information engineer B’s way of working and his way of modelling.

4.2.4 Observations

The approach followed by information engineer B is characterised as user oriented and data driven. This follows from a number of observations which we will discuss in this section.

User participation is an important aspect of the approach followed, independent of the stage.
Results are often validated. During communication with users, an attempt is made to adopt the users' terminology as much as possible. Emphasis is given to understandability of the (intermediate) results. For this reason, information engineer B decided not to present the data model during the final validation stage, because it was considered to be too abstract.

Three main tasks are defined in information engineer B's approach. In the first task, bottlenecks in the problem area are distinguished. It is important to define the problem
area at an early stage, in order to perform the succeeding tasks as effective as possible, i.e. in order not to investigate irrelevant areas. Furthermore, it is important to judge whether the bottlenecks at hand are worth being solved at all. The second task focuses on the information needs to be fulfilled, whereas the third task leads to a model of the solution proposed.

Data modelling is highly dominant to process modelling. This observation holds for all four main tasks, but in particular for the investigation of information needs. The construction of process models is driven by existing data models: it should be possible to express leaf processes as manipulations on entity types, relationships or attributes. The construction of process models is characterised as output driven: given a systems output covering the information needs, process models are constructed from output to input.

### 4.3 Information Engineer C

This section presents a description of the approach followed by information engineer C. This description is derived from the protocol transcript (see [Herlé, 1990]) using the approach described in sections 2.4.3 and 2.4.4. The way of working, the way of modelling, and relationships between the way of working and the way of modelling are represented sequentially. We have restricted the representation of relationships between the way of working and the way of modelling to the representation of information interdependencies. Observations conclude this section.
4.3.1 Way of working

The approach followed by information engineer C consists of four main subtasks: problem understanding, solution finding, solution evaluation, and solution refinement. The corresponding overall task is presented in figure 4.27. The following four sections treat these subtasks separately.

![Diagram](image)

Figure 4.27: The overall task

4.3.1.1 Understand problem

The task "Understand problem" results in insight into the problem area in terms of the problems, causes and consequences of the problems, and people, organisation units, and processes involved. It consists of three main subtasks, initial problem orientation, detailed problem orientation, and problem analysis, see figure 4.28.

*Initial problem orientation*, see figure 4.29, starts with the acquisition of insight into the organisation structure. First, the organisation structure is sketched in terms of a decomposition hierarchy of organisation units. The problem owner, usually an organisation unit, is positioned in the organisation structure. Subsequently, the problem owning organisation unit is examined more closely by investigating its main operating processes. For each of these processes, it has to be clear what its main product is and by whom it is performed, be it a organisation sub-unit or an employee. The orientation on the problem owning organisation is finished by investigating the organisation units distinguished so far in terms of size (the number of people employed). For each of the organisation units, the name of the manager is also registered.
A rough insight into the problem situation at hand is required after this by listing the problems as perceived by the problem owning organisation unit. To comprehend the urgency
of the problems listed, the consequences of each problem are investigated.

The third subtask within the initial problem orientation formulates a statement of purpose. The statement of purpose involves the type of results to be delivered by the information engineer (e.g. an investigation, solution proposals), and the organisation units and business activities which are considered to be part of the problem area.

Subsequently, an initial data model is constructed. The initial data model is a rough attempt to structure the main objects in the problem area and includes an extensive list of candidate entities, i.e. terms which can not be situated precisely yet.

Finally, the problem orientation results are validated. These results include the organisation structure in relation to the processes distinguished, an initial data model, the list of problems and consequences, and the statement of purpose.

![Figure 4.30: Performing an detailed problem orientation](image)

*Detailed problem orientation*, see figure 4.30, starts with the acquisition of more insight into the various problem causes, resulting in an initial list of problems and problem causes.

In the second subtask, a detailed list of processes performed by the problem owning organisation unit is constructed. The construction of such a process list focuses on three different types of processes, operating processes, planning processes, and checking processes. The construction of this list is started by exploring the operating processes. It should be determined how operating processes trigger each other, how they are decomposed, and by whom they are performed. If some operating processes turn out to be problematic, then it should be clarified which problems are related to these operating processes, how often they occur, with what intensity they occur, and whether their volume is expected to grow in the near future. Asking for quantitative data increases the insight into the problem situation. The order in which operating processes are explored is preferred to be from input to output.
Subsequently, the planning processes are explored. At this stage, it is investigated for each planning process by whom it is performed and which criteria are used in order to achieve a schedule. Finally, the checking processes are explored in more detail. During this investigation, it is of interest to know which operating processes are involved, by whom the checking is performed, how often the checking is performed, and what checking aspects are involved. Checking aspects are timeliness, correctness, and completeness.

Figure 4.31: Performing a problem analysis

The problem understanding task ends with a problem analysis which amounts to the construction of a model of the problem area, see figure 4.31. This model consists of a detailed problem list and an initial process model. During the construction of the detailed problem list, the problems discerned so far are listed. For each of these problems, the problem owner is indicated, possible causes are listed and the tendency is registered. All problems on the detailed problem list are characterised, i.e. they are assigned a so-called COPAFIT value. COPAFIT values are: Commercial, Organisation, Personnel, Administration, Finances, Information, and Technology. Given this problem list, problems are related to each other, based on the registration of the problem owner and possible problem causes on the one hand, and on the type of the processes involved on the other hand. The notion of “process type” refers to operating, planning, and checking processes here.
Sec. 4.3

The initial process model shows a clear difference between operating and controlling processes. Furthermore, it serves as a means to find out whether all main products of these processes are well-known. The model consists of main processes, flows, and external parties. When solving case 1, information engineer C used the initial process model to restrict the problem area to the controlling processes within the problem owning organisation unit, stating that operating processes could be of interest only as far as efficiency improvement was concerned.

4.3.1.2 Find solutions

The finding of solutions, see figure 4.32, is traced back to acquiring a deep insight into the current situation. The initial process model is first refined, including the addition of data stores. Again, a clear distinction has to be made between controlling and operating processes. Subsequently, an initial data model is constructed. The initial data model is verified against the process model. Verification is performed using the rule that each flow connected to an operating process should correspond to an entity or a relationship in the initial data model.

Depending on the existence of an expected solution within the organisation and information engineer C’s current insights, a choice is made between a “refining approach”, an “offensive approach”, and an “innovative approach”. If the problem owner does not have expectations with regard to a specific solution, the innovative approach is chosen. This leads to the construction of a detailed data model, based on the initial process model. Otherwise, it is investigated whether the expected solution is satisfactory. If the information engineer has confidence in the expected solution, the solution is further refined in the refining approach. If not, focus will be on drawbacks of the expected solution, resulting in the offensive approach. Since information engineer C decided to adopt the innovative approach when solving case 1, we focus on this.

The innovative approach, see figure 4.33, leads to the construction of a detailed data model, in a close cooperation with the problem owner. First, all structural entities, i.e. entities which are not related to one specific operating process, are added to the data model, and interrelationships are created between these entities. The detailed data model is extended then by exploring each of the main operating processes individually for its connected data flows, and hence for related entities. The order in which the main operating processes have to be selected is from input to output. The construction of the detailed data model is apparently a process-driven task. When creating the detailed data model one should analyse the behaviour of entities in time and the possibilities of categorising entities into subentities.

The detailed data model, in particular the relationships in it, is used to communicate specific problems in the area of planning and control. Relations in the data model, in particular those relations connected to external entities, may be related to processes in the initial process model. Such relations are investigated with regard to the way they are registered now: manually, automated, or not at all. These relations constitute clues for improving the current problem situation, since their being registered enables checking for
timeliness, completeness, and correctness. Another way of communicating specific problems is to debate, for each entity, how related processes are planned and how they are checked.

A number of solution proposals are listed. These proposals should follow from the insights acquired in the previous tasks, in particular during the innovative solution finding task. The proposed solutions were communicated in short with the main problem owner, in particular to find out whether they were considered to be feasible and desirable.

4.3.1.3 Evaluate solutions

The task “Evaluate solutions”, see figure 4.34, results in preparing and giving a presentation of the results of the analysis task. The preparation starts with an extensive process modelling task. A list of problems and their causes is created in parallel. This list is an updated version of the detailed problem list made in the problem understanding task. After this, two additional tasks are performed simultaneously as part of the preparation. The proposed solutions are listed (in information engineer C’s modelling knowledge acquisition session, these solutions included the possible application of order planning and control con-
cepts), and a detailed data model covering the controlling processes in the process model is constructed. It is checked whether the detailed data model covers all data flows which are part of the detailed process model. The products which constitute the contents of the data flows should be related somehow to the entities which are part of the detailed data model. Finally, it is checked whether the proposed solutions really solve all problem causes listed before.

The detailed process modelling task is presented in figure 4.35. Detailed process modelling starts with the construction of a context diagram which shows all external parties involved. This context diagram is decomposed into a first-level process model. If the first-level process model is too complex, process clustering is applied in order to reduce the number of data flows. Process clustering takes place using the criterium that processes which are related by many data flows are likely to be clustered: "highly coherent, loosely coupled". Each of the processes in the process model is then examined in some more detail, distinguishing - again - between planning, operating, and checking processes. Operating processes are preferably examined in a specific order, which is from output to input. When solving case 1, information engineer C realised at this stage that the case could be compared to a production environment, so that concepts from the area of order planning and control fruitfully could be applied.

4.3.1.4 Refine solutions

The task "Refine solutions", see figure 4.36, focuses on a further refinement of the detailed data model and the detailed process model, which have been constructed in the previous task, "Evaluate solutions".

Refinement of the detailed data model consists of several tasks to be performed. Each entity type is defined, using natural language. Furthermore, entity types are described by
Figure 4.34: Proposing solutions

attributes. Some of the attributes are key attributes. Together, the key attributes identify the entity type.

Refinement of the detailed process model results for each process in the process model either in a process specification or in a decomposition into a new, lower level process model. If constructing a process specification is expected to be a time intensive task, the alternative of process decomposition is chosen. If a considerable difference in the frequency value associated with the connected data flows is discerned, this provides a clue for further process decomposition. When decomposing, it has to be checked whether data flows in the lower level process model correspond to data flows connected to the process which has been decomposed.

4.3.2 Way of modelling

In this section we present the product oriented aspects of the approach followed by information engineer C when solving case 1. We focus on data modelling and process modelling
in the following sections.

4.3.2.1 Data modelling

The main notion in data models, see figure 4.37, is that of an Entity type. Entity types are defined using natural language. For each Entity type, its Volume, that is the number of specific Entity instances of that type, is investigated. Entity types may be related by Relationships. Since one and the same Entity type may participate more than once in one Relationship, the notion of Role is introduced to be able to distinguish between these participations. The Cardinality represents how Entity types are related to one another. Entity types and Relationships are generalised into Data model objects. Data model objects should have a Name.

4.3.2.2 Process modelling

Process modelling is considered to be of great importance throughout the approach followed. This holds in particular for the problem understanding stage. The concepts used when constructing Process models are closely related to organisation concepts and pro-
Figure 4.37: Object structure representing data models

problem solving concepts. For this reason, we include the latter concept categories in our presentation of process modelling concepts.

Figure 4.38 presents the Object structure related to the notion of "problem". Problems have one or more causes and one or more consequences. A problem has a certain frequency of occurrence, and an intensity. Furthermore, a tendency is associated with a problem, i.e. its frequency or its intensity is expected to increase or decrease in the near future.

Problems are solved by solutions. One solution may attribute to the solving of one or more problems. Both problems and solutions are characterised by a copafit value.

In figure 4.39, it is shown which organisational aspects are studied. A hierarchy of organisation units is discerned. For each of the organisation units, the size is registered in terms of the number of people employed. One of these employees is the manager of the organisation unit. Employees and organisation units are generalised into organisation areas.

As figure 4.40 shows, both the concept of "organisation area" and the concept of "problem" are related to the concept of "Process". Organisation units may own one or more problems. Problems concern one or more Processes. Each Process has one organisation area which is responsible for it, while more organisation units may be involved in the actual performance of a Process. The way in which an organisation area performs a Process may differ, according to its associated Process organisation type (e.g. in parallel, in teams). Each organisation area is expected to have one specific Process as its purpose.
Processes have a Frequency of occurrence and a Duration, see figure 4.41. A Product is the
represented applied modelling knowledge: Case 1

Figure 4.40: Object structure representing relationships between “organisations”, “problems” and “processes”

The primary output of a Process. Three types of Processes exist: planning processes, operating processes, and checking processes. Checking processes are related to checking aspects: correctness, completeness, and timeliness. Planning processes involve a planning criterion. Operating processes are further specialised into preparation processes, transformation processes, and finishing processes.

Processes are either specified in a Process specification or decomposed into a Process model, see figure 4.42. A Process model contains not only Processes, but also Data stores and External parties. These three are generalised into Process model objects. Two Process model objects may be related by one or more Data flows. Each Data flow has a Frequency associated with it. The contents of a Data flow are represented as a Product, be it a Physical product or an Information product. To verify whether Data flows connected to a decomposed Process correspond to the Data flows part of the involved Process model, Data flow decomposition has to be represented separately from Process decomposition.

Several interrelationships exist between Process models and Data models, see figure 4.42. Information products and External parties in a Process model may reflect themselves as
Entity types in the Data model. Another interrelationship exists between Relationships in a Data model and Processes in a Process model: each Relationship should be manipulated by a Process.

4.3.3 Relationship between a way of modelling and a way of working

Both data modelling and process modelling go through several stages. These stages are characterised by a gradually changing consistency characteristic, i.e. by an increasing number of Object types and verification rules.

Process modelling is highly dominant to data modelling: the construction of the detailed data model is completely based upon the existing process model.

Remarkably, intermediate results are often discarded. New process models are often constructed "from scratch", in other words, existing process models are often not extended.

The relationships between information engineer C's way of modelling and his way of working are illustrated in figure 4.43.
Figure 4.42: Object structure representing process models, and the way they are related to data models

4.3.4 Observations

The approach followed by information engineer C gives great emphasis to comprehension of the problem situation. To put it in information engineer C’s words: “a good problem
Figure 4.43: Information interdependency diagram, representing relationships between the way of modelling and the way of working of information engineer C when solving case 1.

definition constitutes half the solution”.

The emphasis on comprehending a problem situation results in a number of observations. First of all, process modelling is highly dominant to data modelling. Process modelling does appeal more to the problem owners, since process models are far more easily recognised than data models. During the problem understanding stage, emphasis is on process modelling except in those situations in which a process model is available and clear to all persons involved. The discussion of processes is structured from input to output when discussing the current situation. In cases in which a process model is delivered with respect to a proposed situation, the discussion of processes is preferred to be structured from output to input, since a specific result, the proposed situation, is the starting point for the discussion. The construction of process models always starts with a functional decomposition into processes.
The addition of flows between these processes takes place afterwards. Process modelling is heavily influenced by the distinction between controlling and operating processes. A discussion of operating processes should precede a discussion of controlling processes. It is assumed that this order contributes mostly to acquiring a good comprehension.

The fact that process modelling dominates data modelling reflects itself in the way data models are constructed. The data modelling task is characterised as process driven. The main operating processes in the process model are explored successively for their connecting data flows. These data flows provide for clues with regard to the issue of how to extend the data model.

Achieving a consistent data model is not considered to be an important task at this stage.

The second observation concerns the importance to have a precise insight into the problems and the underlying causes. This insight results in an intensive questioning of specific consequences of problems. Insight into the organisation and in the problem area dominates the first information modelling tasks. During these tasks, little attention is paid to the delivery of specific process models or data models. Instead, organisation structure diagrams and problem lists are delivered. As information engineer C puts it: "Employment of Yourdon techniques may happen only in the last hours of this session. First, I intend to understand the why of the situation rather than oppressing a number of symptoms." This insight also influences the way data models and process models are constructed. Processes are investigated for their frequency and duration, and entities are investigated for their volume, i.e. their number of instances.

### 4.4 Observations

In this chapter, we have presented the approach that three experienced information engineers followed when solving case 1. We present some observations which apply to the approaches in this section. These observations are divided into observations with respect to a way of working and observations with respect to a way of modelling.

#### 4.4.1 Way of working

In the three previous sections, a series of Task structures have been presented which describe the individual approaches followed, see figures 4.1 up to and including 4.10, 4.15 up to and including 4.22, and 4.27 up to and including 4.36.

At a high level of consideration, the approaches followed are similar: they all contain a problem analysis task which precedes a tentative system design task. The information modelling tasks which are part of the problem analysis task are dominated by a small number of objectives. The problem analysis should lead to an insight into the organisation, in its structure and in its main operating processes. The problem analysis should also lead to a problem area demarcation, be it in terms of operating processes, organisation units, or environmental factors, and within the problem area, a problem identification should take
place. The problems identified are investigated in more detail, from different points of view: it is judged whether it is worthwhile to solve them at all, problems are provided with quantitative information (frequency, volume), and causes and consequences are analysed. In this problem analysis stage, some preliminary data models and process models are constructed, in a very shallow and intuitive way. Formulating a statement of purpose is a similar activity performed as part of the problem analysis stage.

The system design task gives emphasis to data modelling and process modelling. Process models and data models clearly go through several stages of structuredness, employing a larger set of concepts and fulfilling a larger set of verification rules throughout these stages.

It is remarkable to observe, that the three information engineers have a different view on the dominance of data models and process models. In information engineer B’s approach, the construction of process models is driven by existing data models. The opposite is essential in information engineer C’s approach: the construction of data models is determined by existing process models. Information engineer A gives equal emphasis to both model types, using existing process models in order to adapt data models and vice versa in an iterative way. He follows what he calls a “zip approach”.

Various information interdependencies between data models and process models are observed. Data models and process models are also related to other model types, e.g. object models and event lists.

Three process modelling strategies are observed. We refer to these strategies as input driven process modelling, output driven process modelling, and data driven process modelling. From an input driven point of view, processes handle events, and lead to other processes. From an output driven point of view, processes satisfy information needs, and other processes are necessary to deliver the input for these processes. From a data driven point of view, processes manipulate data, i.e., create, read, use, and delete instances of entity types, relationships, and attributes.

Three data modelling strategies have been used. We refer to these strategies as noun driven data modelling, object driven data modelling, and process driven data modelling. In the noun driven strategy, each noun in the description of the problem area is considered to be a candidate entity. In the object driven strategy, objects in the real world are related to each other. Each object is questioned for the necessity of storing information on it. The process driven strategy investigates each operating process for entity types, and integrates the resulting partial data models.

The construction of models is primarily driven by the underlying statement of purpose and by the problem area demarcation. Models are constructed only in order to gain additional insight into certain aspects or parts of the problem area, or to communicate aspects or parts of the problem area to users. It is not possible to point out a similar order of model construction over the three information engineers. This order is highly opportunistic, depending on the information engineer’s need for more insight into particular aspects or parts of the problem area. As a consequence, intermediate short-term plans were often drawn up during each of the four days’ sessions.
From the approaches followed, it is clear that validation of impressions, opinions, and (intermediate) results is considered to be important. Validation not only concerns correctness, but also (and in particular) completeness of results. The role of users in general is considered to be very important. This is illustrated by information engineer B, who decided not to discuss the data model as part of the final presentation session because of its abstract character.

The sessions were characterised also by extensive note making. Note making allows the information engineer to follow his approach as closely as possible without overlooking important concerns which are not at the current level of detail. In order to use them at the appropriate time, notes are often consulted.

### 4.4.2 Way of modelling

In the three previous sections, a series of Object structures have been presented which describe the individual notions used, see figures 4.11 up to and including 4.13, 4.23 up to and including 4.25, and 4.37 up to and including 4.42.

Data modelling concepts and process modelling concepts are used at a considerable level of detail, leading to the use of refined subconcepts. Apart from these refined subconcepts, the Object structures which have been used to represent the data modelling concepts and the process modelling concepts show large similarities.

Both data models and process models are used when modelling real systems, but also when modelling information systems. Information engineer A shows remarkable differences between modelling real systems and modelling information systems: he includes physical flows in his process models, and he employs so-called object models instead of data models.

Although we have not focused on other concepts but those used in data modelling tasks and process modelling tasks, we observe that information engineers show interest in organisation structure diagrams, and that they show interest in aspects of the problem area which can not be modelled in diagrams explicitly. Concepts such as “information need” and “bottleneck” are typical examples.
Chapter 5

Representing applied modelling knowledge: Case 2

This chapter presents the approach followed by three experienced information engineers when solving case 2. The case concerns a complex database design problem with a large number of organisational constraints at the Dutch Industrial Insurance Administration Office. We refer to this organisation by its Dutch acronym GAK (Gemeenschappelijk AdministratieKantoor). The case is described in more detail in appendix C. The modelling knowledge acquisition results are presented in sections 5.1, 5.2, and 5.3. Each of these sections pays attention to the information engineer's way of working and his way of modelling. Particular features or interesting details are highlighted. Section 5.4 presents some concluding observations.

5.1 Information Engineer A

This section presents a description of the approach followed by information engineer A. This description is derived from the protocol transcript (see [Verhoef, 1991b]) using the approach described in sections 2.4.3 and 2.4.4. The way of working, the way of modelling, and relationships between the way of working and the way of modelling are represented successively. We have restricted the representation of relationships between the way of working and the way of modelling to the representation of information interdependencies. Observations conclude this section.

5.1.1 Way of working

The approach followed consists of three main subtasks: understanding the problem situation, developing alternative solutions, and choosing one solution. The corresponding overall task is presented in figure 5.1. The following three sections treat these subtasks separately.
5.1.1.1 Understand problem situation

The task "Understand problem situation" consists of three subtasks. A statement of purpose is first formulated. The problem at hand is then investigated in some detail. Finally, a thorough problem analysis is carried out in which the problem is investigated in relation to its context in more detail. Figure 5.2 presents the task structure which represents this task.

The first subtask, "Formulate statement of purpose", see figure 5.3, was initiated by a problem sketch in the form of a presentation by the problem owner, including problem context, problem statement, and an impression of the preferred solution, as presented in sections C.1 through C.4. This presentation was followed by achieving a first comprehension of the pro-
blem as it was presented. Questions should cover uncertainties, both terminological and factual, with regard to the problem and its context. Questions are preferably asked in the order in which they arose during the presentation of the problem sketch. Finally, an agreement was achieved with the problem owner on the statement of purpose. When solving case 2, information engineer A agreed to develop a solution for the problem sketched, without paying particular attention to the preferred solution beforehand.

Figure 5.3: Formulating the statement of purpose

The subtask “Perform problem orientation”, see figure 5.4, succeeds to formulating the statement of purpose. Based on a first understanding of the problem situation at hand, subproblems are distinguished and investigated in more detail. To get insight into the structure of the problem area, a preliminary logical data model may be constructed. The first subtask within the problem orientation task concerns the distinction of subproblems. It is an important heuristic in this subtask that subproblems should be loosely coupled. Subsequently, each distinguished subproblem is investigated separately in more detail. The main issue in this investigation is to detect problem triggers and to find out how the subproblems relate to the main entities in the problem area. If the problem area is characterised to be data oriented, the problem orientation task is finished by constructing a preliminary logical data model of the problem area. Entities may be provided with attributes to point out the subproblems distinguished. The preliminary data model serves mainly to get more insight into the structure of the problem area.

The subtask “Perform problem analysis”, see figure 5.5, is the final subtask in understanding the problem situation. The problem analysis task results in several models of the problem area and the problem context, delivered in a strict order. This order is based on the insight that the problem area and its context could be modelled in terms of different RS-IS combinations, therewith applying the recursion principle of the information paradigm.

The problem context is first modelled. The resulting model consists of two parts: an object model and a process list. The object model depicts the relevant objects and their
interrelationships. The process list lists the main processes in the problem context.

The problem area itself is modelled then. As a first subtask, a logical data model of the problem area is constructed. The problem area is related to the problem context in a specific way: it controls the problem context in the sense of the information paradigm of [Brussaard and Tas, 1980]. Due to this relationship, the logical data model of the problem area is derived from the model of the problem context. The logical data model is constructed using the entity-relationship diagramming technique.

Several rules govern the transformation from the model of the problem context to the logical data model. All objects in the object model appear as concrete entities in the logical data model. Relationships between objects in the object model are transformed either into relationships between entities or into associative entities themselves. Furthermore, conceptual entities are added to the logical data model by investigating whether it covers all information output by the main processes on the process list.

Subsequently, the logical data model is related to the current physical situation by pointing out for each entity and relationship in which physical files they are implemented.

A preliminary physical process model of the problem area is constructed afterwards. Physical files are modelled as terminators or as data stores. The preliminary physical process model also contains processes and data flows. Data flows which are input for data stores are labelled, depending on the way in which they manipulate the contents of the related data store (mutation or addition). The subproblems distinguished are pointed out explicitly in terms of key record fields in the physical files.

Finally, the statement of purpose is reformulated. To formulate the statement of purpose concisely, the recursion principle of the information paradigm is used to relate the problem
context (RS), the problem area (IS), and the system that in its turn controls the problem area (IS').

The results of the problem analysis task are validated. The validation of the data model is guided by the information engineer, formulating specific questions. Most of these questions are based on consistency rules which should hold for data models.

5.1.1.2 Develop alternative solutions

The subtask "Develop alternative solutions" consists of two main subtasks. First, a list of possible solutions is made. These solutions are then developed in more detail and evaluated against a variety of evaluation criteria. Figure 5.6 presents the corresponding Task structure.

In the first subtask, "List alternative solutions and validate their plausibility", a set of possible solution scenarios is listed. In the case of case 1, this first set included not solving the problem at all, solving it in the problem area (IS), and tackling its consequences in IS' in various ways. The first set of possible solution scenarios is validated to find out whether they are considered to be realistic at first glance. During the modelling knowledge
acquisition session some solutions were considered to be out of scope due to organisation policy constraints.

The second subtask, "Develop and evaluate alternative solutions", see figure 5.7, aims at reducing the number of alternative solutions in the most effective way. To this end, a number of functional requirements to be met by any solution scenario have to be formulated. These functional requirements are necessary to further develop and evaluate the alternative solution scenarios.

Development and evaluation of the alternative solutions then take place in an iterative way, i.e. each alternative solution is explored in some more detail and evaluated against one of the functional requirements. The evaluation might lead to the rejection of some of the alternatives. This sequence of activities has to be repeated for each of the functional
requirements. The further development of the non-rejected alternative solutions as such involves a construction of a preliminary physical process model for each of them. A preliminary physical process model contains physical files, and data flows between them. Data flows are labelled to indicate the type of manipulation: select, add, or mutate. In case the type of manipulation turns out to be “select” or “mutate”, particular record fields are indicated. Each of these physical process models is being extended with physical files and data flows by testing it against the requirements formulated until all requirements are met by each alternative solution. The Task structure in figure 5.8 represents this evaluation of alternative solutions against functional requirements.

![Diagram of alternative solution evaluation process](image)

Figure 5.8: Developing and evaluating alternative solutions against functional requirements

The alternative solutions are evaluated against some preferred criteria, see figure 5.9. During information engineer A’s modelling knowledge acquisition session, the feasibility of alternative solutions in terms of amount of file I/O turned out to be the preferred evaluation criterion. To perform the evaluation, a fictitious but representative case in the problem domain is developed. Subsequently, the feasibility of each of the alternative solutions is determined in terms of the number of file manipulations (both read and write manipulations) necessary to handle the fictive case, using the preliminary physical process models.

### 5.1.1.3 Choose solution

The subtask “Choose solution” consists of the actual choice of a satisfactory solution and the further development of the solution chosen, see figure 5.10.

Further detailing the solution chosen is a complex subtask itself, see again figure 5.10. It starts with the acquisition of additional (detailed) requirements. The preliminary physical process model is detailed, refinements being focused on fulfilling the additional requirements. A detailed physical process model of the solution chosen, contains the physical files of the preliminary physical process model. Between the physical files, flows are added as well as processes. Processes are seen as mere transformers. All flows are labelled with the relevant record fields. If timing aspects are considered to be relevant, control processes and control flows are added to the physical process model. Control flows relate control processes to data processing processes. Control flows either trigger a data processing process
Figure 5.9: Developing and evaluating alternative solutions against preferred criteria

Figure 5.10: Choosing a solution

to start or trigger the control process that the data processing process has finished. The resulting detailed physical process model is validated, in particular against the additional requirements. Finally, issues are listed which will be relevant in future stages.
5.1.2 Way of modelling

We focus on the various types of data models and process models information engineer A constructed during the modelling knowledge acquisition session.

5.1.2.1 Data modelling

From our description of the approach which information engineer A followed when solving case 2 (see section 5.1.1), we recapitulate that several data modelling activities are part of this approach. Emphasis to data modelling is in the first subtask, “Understand problem situation”. Three data modelling activities are to be distinguished here, namely “Construct preliminary logical data model”, “Construct object model”, and “Construct logical data model”.

The object model distinguishes itself from the (logical) data models in that it models objects in a real system, whereas the data models model entity types in an information system. Object models are not bound by many verification rules. It should be possible to express all kinds of relationships between objects in a real system, though informally. Figure 5.11 presents the Object structure which describes the main notions in object models.

![Diagram](image)

Figure 5.11: Object models

The logical data models are a binary variant of the entity-relationship diagramming technique. In each relationship, two entity types participate. At least one of these entity types has to have a cardinality of “1”, in other words, n:m relationships must be avoided.

The difference between the preliminary logical data model and the logical data model is that the latter distinguishes a Specialisation hierarchy of *Entity type*. Due to this hierarchy,
it is possible to partially derive the logical data model from the object model. Concrete entities in the logical data model are derived from objects in the object model. Relationships between objects in the object model are transformed either into relationships between entity types or into entity types. The latter entity types are referred to as associative entity types. Conceptual entity types are not derived from the object model. They may be added to the logical data model based on the process list.

The main notions used in data models are presented in figure 5.12. The relationships which exist between object models and data models are shown in figure 5.13.

![Diagram](image)

Figure 5.12: Entity models

5.1.2.2 Process modelling

From section 5.1.1 we recapitulate that several process modelling activities have been part of the approach followed. Emphasis on process modelling activities is in the subtasks “Develop solutions” and “Choose solution”.

When performing a problem orientation, a process list of RS Processes is constructed. After the problem orientation, process models contain only IS Processes. The main process models are preliminary physical process models, which are used during the problem analysis and during the development of alternative solutions. Detailed physical process models are
used while the chosen solution is developed in more detail. Control aspects may be part of detailed physical process models.

Physical process models consist of processes, data stores, and terminators, all connected by data flows. Data stores should correspond to an Entity type in the logical data model. Data flows which go to a data store are labelled according to the type of manipulation, i.e. “mutate” or “add”. The contents of a data flow are labelled in terms of one or more attributes from the logical data model.

Control processes are part of detailed physical process models. In a detailed physical process model, control flows are either sent by a control process to another process in order to trigger the latter, or received by a control process from another process in order to signal that the latter has ended.

Figure 5.14 shows the Object structure which represents the main concepts used in physical process models. This figure shows also which relationships exist between physical process models and data models.

5.1.3 Relationship between a way of modelling and a way of working

Process modelling starts with the initial construction of a process list. After that, and independent of that, some process models are constructed, and extended to a detailed process model.

Both the process list and the detailed process model are used when constructing data models. The first data model is based upon an object model. Independent of the object
model and the data model, a preliminary data model is constructed in the very beginning of information engineer A’s approach.

Figure 5.15 shows the relationships between information engineer A’s way of modelling and his way of working.

5.1.4 Observations

The approach which has been followed by information engineer A when solving case 2 consists essentially of three stages: understanding the problem situation, developing alternative solutions, and choosing a solution.

During the understanding stage, data modelling is dominant to process modelling. The construction of a logical data model is preceded by the construction of an object model. Object models model objects in a real system, data models model entity types in an information system. To understand the problem situation, it is considered important to distinguish loosely coupled subproblems. Explicit attention is paid to the (re)formulation of the statement of purpose.
The two subsequent stages are characterised by more process modelling activities. Process models may include control aspects. In particular during the development of alternative solutions, it is important to formulate functional requirements and to use other criteria to reduce the number of alternative solutions in the most effective way possible.

5.2 Information Engineer B

This section presents a description of the approach followed by information engineer B. This description is derived from the protocol transcript (see [Verhoef, 1991a]) using the approach described in sections 2.4.3 and 2.4.4. The way of working, the way of modelling, and relationships between the way of working and the way of modelling are represented successively. We have restricted the representation of relationships between the way of working and the way of modelling to the representation of information interdependencies. Observations conclude this section.
5.2.1 Way of working

The approach followed consists of three main subtasks: understanding the problem situation, developing alternative solutions, and choosing a solution. The corresponding overall task is presented in figure 5.16. The following three sections treat these subtasks separately.

![Figure 5.16: The overall task](image.png)

5.2.1.1 Understand problem situation

The task “Understand problem situation” consists of three subtasks. First, a statement of purpose is formulated. This task is followed by an orientation subtask and an analysis subtask. Figure 5.17 presents the task structure which represents the task of understanding a problem situation.

The first subtask, “Formulate statement of purpose”, see figure 5.18, was initiated by a problem sketch in the form of a presentation by the problem owner, including both problem context and problem statement, as presented in sections C.1 through C.3. This presentation was followed by achieving a first comprehension of the problem as it has been presented. Questions cover uncertainties, both terminological and factual, with regard to the problem and its context. Finally, an agreement was achieved with the problem owner on the statement of purpose. When solving case 2, information engineer B agreed to evaluate the problem owner’s preferred solution.

The subtask “Perform an orientation”, see figure 5.19, succeeds to formulating the statement of purpose. The orientation starts with acquiring a rough insight into the preferred solution, in the form of a presentation by the problem owner. This acquisition is followed by a data modelling task. Data modelling starts with constructing an initial physical data model. A physical data model consists of files. Relationships between files are determined by investigating keys and foreign keys. An initial logical data model is derived from the
physical one. A logical data model consists of entities that are related to each other by relationships. Files in the physical data model correspond to entities in the logical data model. Subsequently, an initial understanding has to be achieved of the preferred solution. Preferably, this is achieved by simulating a scenario in the problem domain and studying its effects on the preferred solution. The orientation is finished by the development of an approach for the subsequent subtask, the analysis task. When solving case 2, information engineer B decided to focus his analysis task on the development of a realistic problem domain scenario, on a summary of the problem situation, on the construction of a logical data model, and on evaluation criteria in order to evaluate the preferred solution.

The third and final subtask in understanding a problem situation is “Perform an analysis”. It consists essentially of four subtasks. First, the problem at hand is formulated together
with its causes. Then, a preliminary solution evaluation is performed. As a third subtask, a logical data model of the problem area is constructed. Finally, the analysis is validated with the problem owner. The Task structure of the analysis task is presented in figure 5.20.

Both the preliminary solution evaluation and the analysis validation are time-intensive tasks. They are discussed below in more detail.

The task “Perform a preliminary solution evaluation” contains six subtasks, which are performed in a linear order. A list of functional requirements which have to be met by any solution, is first formulated. For each of the functional requirements, subsequently one or more evaluation criteria are formulated. The importance of each functional requirement and of each evaluation criterion is indicated. Knowledge of relative importances may support future decisions on the requirements or criteria to be investigated first. As a next activity, an extended problem domain scenario is constructed, consisting of a sequence of characteristic events to be handled by a solution. The preferred solution is evaluated against the main evaluation criteria, by simulating its behaviour in relation to the problem domain scenario. As a consequence of this preliminary evaluation of the preferred solution, a list of drawbacks of the preferred solution is made up. For case 2, the preferred solution turned out not to focus on the problem causes, but on the problems which occurred as a consequence of these causes. The Task structure of performing a preliminary solution evaluation is presented in
Figure 5.20: Performing an analysis

The task "Validate analysis" validates most of the outcomes of the analysis task, in a strict order. First, the problem statement, the functional requirements, the reality of the problem domain scenario, and the logical data model are validated. Validation of these outcomes may lead to adaptations. The evaluation criteria are then validated. Finally the disadvantages of the preferred solution are communicated. Figure 5.22 shows the related Task structure.

5.2.1.2 Develop alternative solutions

The development of alternative solutions, see figure 5.23, is performed in two passes. First, alternative solutions are listed based on the insights gained so far. The feasibility of these alternatives is then validated.

The second pass focuses on achieving specific understanding of the problem area and the preferred solution to give more alternative solutions. Specific understanding may be achieved by again constructing a logical data model and relating it to a physical data model of the preferred solution, by constructing a physical process model of the preferred solution and by evaluating the preferred solution against a set of preferred criteria. Again, feasibility of the alternatives has to be validated.

During information engineer B's modelling session, the development of alternatives in the first pass was based on the insight that the disadvantages of the preferred solution were related to the fact that development of the preferred solution had not focused on solving the problem causes but on the symptoms. All alternatives deriving from this insight were
cancelled by the problem owner because of organisational policy constraints. When solving case 2, information engineer B developed new alternative solutions in the second pass based on new insights gained by constructing the logical data model and the physical process model, and based on preferred criteria, such as “optimisation should be avoided”, “sorting processes should be avoided”, and “file I/O should be minimal”.

5.2.1.3 Choose a solution

This task results in the actual choice of one of the alternative solutions. This solution is checked against the less important evaluation criteria. New and more detailed functional requirements are formulated to further detail the solution chosen. Finally, an approach is developed for subsequent stages. See figure 5.24 for the corresponding Task structure.

5.2.2 Way of modelling

Information engineer B’s way of modelling is discussed in this section. We focus on process modelling and data modelling to be consistent with chapter 3, although information
engineer B's way of modelling also uses several notions from the area of problem solving as such. Notions such as *problem cause*, *functional requirement*, and *evaluation criterion* may illustrate this.
5.2.2.1 Data modelling

Several data modelling activities are part of the approach followed by information engineer B (see section 5.2.1). Data modelling is performed in particular during the task “Understand the problem situation”. In the orientation phase, a logical data model is constructed based on a physical data model. A logical data model is also constructed as part of the analysis task. The task “Develop alternative solutions” comprises a data modelling activity in which physical aspects are taken into account based on a logical data model.

The physical data model consists of files. Based on key fields in files, relationships between files are modelled. Files can be related in two different ways. Files may be derived from each other, by some manipulation or aggregation. Files may also be related by a so-called father-son-construction, implying a key dependency. Figure 5.25 shows the corresponding Object structure.

The logical data model consists of entity types. Entity types may be related to each other by relationships. If two entity types are related to each other, cardinality constraints have to be added to the Relationship. Entity types may be described by a group of attributes. Entity types and attributes may be provided with sample values to provide more clarity.

The physical data model and the logical data model are related to each other. Files and fields in the physical data model may correspond to entity types and attributes in the logical data model. Figure 5.26 shows the corresponding Object structure which represents the notions occurring in logical data models. Furthermore, it shows the relationship between logical data models and physical data models.
5.2.2.2 Process modelling

Process modelling does not play an eminent role in the approach followed by information engineer B. A physical process model is constructed as part of the task “Develop alternative solutions”. Few clues are provided with reference to the notions that are part of a process modelling activity.

Physical process models contain processes, files, and data flows. Different types of processes are distinguished, such as mutating processes and sorting processes. Physical process models are related to physical data models in the sense that each file in the physical process model should also be part of a physical data model. See figure 5.27 for the corresponding Object structure.

5.2.3 Relationship between a way of modelling and a way of working

Both process modelling tasks and data modelling tasks played a role of minor importance in information engineer B’s approach. Discussing these tasks, we should make a distinction between logical and physical models. A physical process model was made, and used when constructing a physical data model. A physical data model was constructed twice. The results were used when constructing an initial logical data model. This initial logical data model was changed several times.

The information interdependencies between the Schema types which represent the model types used, are represented in figure 5.28.
5.2.4 Observations

The approach followed by information engineer B consists essentially of three stages: understanding the problem domain, developing alternative solutions, and choosing a solution.

In particular, during the problem understanding stage, attention is paid explicitly to the approach to be followed itself, probably due to the novelty of case 2. Explicit attention is also paid to the formulation of a statement of purpose. During this stage, data modelling tasks are dominant to process modelling tasks. Data modelling tasks are complemented by tasks, in which realistic scenarios in the problem domain are simulated. Functional requirements are formulated in this stage in order to validate alternative solutions.
The development of alternative solutions involves both process modelling and data modelling tasks. These tasks are performed with the specific purpose of acquiring a more detailed insight into the problem domain and the preferred solution, to develop new alternatives. Alternatives are validated against the functional requirements, and against a number of preferred criteria.

Information engineer B's approach is characterised as user oriented. Users' terminology is adopted, to ease communication.

5.3 Information Engineer C

This section presents a description of the approach followed by information engineer C. This description is derived from the protocol transcript (see [Verhoef, 1991c]) using the approach described in sections 2.4.3 and 2.4.4. The way of working, the way of modelling, and relationships between the way of working and the way of modelling are represented successively. We have restricted the representation of relationships between the way of working and the way of modelling to the representation of information interdependencies. Observations conclude this section.
Figure 5.28: Information interdependency diagram, representing relationships between the way of modelling and the way of working of information engineer B when solving case 2

5.3.1 Way of working

The approach followed consists of two main subtasks: understanding the problem situation and evaluating the preferred solution. The corresponding overall task is presented in figure 5.29. The following two sections treat these subtasks separately.
5.3.1.1 Understand problem situation

The task “Understand problem situation” consists of three subtasks. A statement of purpose is first formulated. This task is followed by an initial orientation task and subsequently by an analysis task. Figure 5.30 presents the task structure which represents this task.

![Diagram of task structure](image)

Figure 5.30: Understanding the problem situation

The first subtask, “Formulate statement of purpose”, see figure 5.31, was initiated by a problem sketch in the form of a presentation by the problem owner, including problem context, problem statement, and an impression of the preferred solution, as presented in sections C.1 through C.4. This presentation was followed by achieving a first comprehension of the organisation, the problem owner, and of the problem itself. Questions should cover uncertainties, both terminological and factual, with regard to the problem and its context. Questions are preferably asked in the order in which they arose during the presentation of the problem sketch. After this an agreement is achieved with the problem owner on the statement of purpose. When solving case 2, information engineer C agreed to evaluate the problem owner’s preferred solution and to develop alternative solutions. Finally, attention is very explicitly paid to developing an approach to be followed in the subsequent stages. When solving case 2, information engineer C decided to perform the following activities to fulfil the statement of purpose: (i) focus on the problem at hand, on its causes and its effects, (ii) focus on the preferred solution, (iii) evaluate the preferred solution, and (iv) develop alternative solutions, if any.

The subtask “Perform an orientation” succeeds to formulating the statement of purpose. The orientation itself consists of three main subtasks: “Perform overall problem orientation”, “Structure the problem area”, and “Perform overall solution orientation”, as shown in figure 5.32.

The overall orientation on the problem area results in more insight into the problem area as such, and, additionally, in insight into the issue of how to manage the problem area.
To achieve these goals, one should find out first what the bare essentials of the problem at hand are, by investigating the effects of not solving it. Then, the problem at hand has to be split into a number of subproblems. Subsequently, the list of effects and the list of subproblems are used to deal with the problem area management issue, applying the divide and rule principle. Effects and subproblems are related to each other to find out if there are any isolated relationships. In the second subtask within performing an orientation, each distinguished subproblem is investigated in more detail by relating it to the main entities in the problem area. An initial data model is constructed to document the insights in a more or less structured way. In the third subtask which is part of performing an orientation, the preferred solution is explored at a high level. At this level, one should acquire enough information to decide whether the solution removes the effects registered. If it does not, it will certainly be necessary to develop alternative solutions in future stages.

The subtask "Perform an analysis" succeeds to performing an orientation, and is the final subtask in understanding a problem situation. The analysis task is itself split up into three subtasks: "Perform problem analysis", "Perform solution analysis", and "Validate analyses", see figure 5.33.

The first subtask, "Perform problem analysis", see figure 5.34, is characterised as an opportunistic task in which subtasks are highly interleaving. The problem analysis task starts with reformulating the problem at hand, the underlying problem causes and the statement of purpose. After that, an initial data model is again constructed. A list of functional requirements is created at the same time. These two subtasks influence each other to a high extent. Both the initial data model and the functional requirements are validated against the list of problem causes. They are also validated against each other.
Figure 5.32: Performing orientation

The second subtask, "Perform solution analysis", see again figure 5.33, results in an informal
Figure 5.33: Performing an analysis

sketch of the preferred solution. The preferred solution is validated against the functional requirements. This validation may lead to the initial development of alternative solutions. In the third subtask, "Validate analyses" (again, see figure 5.33), the results of both the problem analysis and the solution analysis are validated in cooperation with the problem owner, to check the correctness and completeness of the list of functional requirements and the initial data model, and to find out whether the alternative solutions proposed are plausible. In case 2, all alternative solutions were considered to be out of scope because of organisational policy constraints.

5.3.1.2 Evaluate preferred solution

The task "Evaluate preferred solution", shown as the second subtask in figure 5.29, focuses on the preferred solution, given the conclusion that alternative solutions are out of scope. This task consists of three subtasks, "Acquire detailed understanding solution", "Evaluate main functionality preferred solution", and "Evaluate other functional requirements preferred solution", see figure 5.35.

Acquiring a detailed understanding of the preferred solution, which is the first subtask, fo-
Figure 5.34: Performing a problem analysis

Figure 5.35: Evaluating preferred solution

cuses in particular on the structure of the preferred solution in terms of files, keys and other
fields in records, and relationships between fields in different files. Again, see figure 5.35 for the decomposition of this subtask. The structure of the preferred solution is documented in a physical data model.

The second subtask, the evaluation of the preferred solution, see figure 5.36, is restricted to the most important functional requirements in this subtask. For all components of the preferred solution, i.e. files, keys, attribute fields, and field relationships, it is investigated whether they constitute an essential part of the preferred solution, i.e. whether their omission would not lead to fulfilling the main functional requirements. These investigations might lead to some adaptations to the physical data model of the preferred solution. Subsequently, the structure of the preferred solution is validated with the problem owner by presenting and discussing the physical data model of the preferred solution.

The next subtask concerns the construction of a physical process model of the preferred solution. The physical process model is checked against the physical data model to verify whether the physical data model contains the entity types and the attributes used in the physical process model. The physical process model is also validated with the problem owner.

![Figure 5.36: Evaluating main functionality preferred solution](image-url)
In the third subtask, see figure 5.37, the logical data model and the physical process model are finally validated with regard to the remaining functional requirements. This validation may lead to adaptations of the models involved.

Figure 5.37: Evaluating other functional requirements preferred solution

5.3.2 Way of modelling

For the sake of coherence, we focus on the various types of data models and process models information engineer C constructed during the modelling knowledge acquisition session. These model types were, however, of limited importance when solving case 2. Important notions related to the process of problem solving as such. Typical examples are functional requirement and problem cause.

5.3.2.1 Data modelling

From our description of the approach that information engineer C followed when solving case 2 (see section 5.3.1), we recapitulate that a few data modelling activities are part of this approach. Emphasis on data modelling is in the first subtask, “Understand problem situation”. Three data modelling activities take place as part of the approach. The first one is part of the problem orientation task, and serves to structure the problem area. The second one is part of the problem analysis task. As opposed to the others, the third one is a physical data model. It serves to acquire a detailed understanding of the preferred solution.

Logical data models are constructed using a binary variant of the entity-relationship diagramming technique. A logical data model consists of Entity types that are connected by Relationships. Exactly two Object types participate in a Relationship. Since the data modelling activities have a preliminary character, Relationships are assigned a Name only in cases where misunderstandings might occur. When relating Entity types, cardinalities are assigned. Both 1:1, 1:n, and n:m relationships are allowed. Object types may be assigned attributes, some of which are part of the identification of those Object types. Figure 5.38 shows the Object structure that represents logical data models.

In a physical data model, the relevant files are indicated with their main attribute fields. Relationships between files exist because of key dependencies. Therefore, they are pre-
represented by directly specifying the key attributes in the two files involved. In figure 5.39, the underlying structure of physical data models is presented. This figure also presents how logical and physical data models are related to each other.

5.3.2.2 Process modelling

Process modelling is only applied in the subtask “Evaluate preferred solution”, to find out whether the preferred solution covers the main functional requirements.

Physical process models consist of processes, files, and terminators, all connected by data flows. Data flows which write in a file are labelled according to the type of manipulation, i.e. “mutate” or “add”. The contents of a data flow are labelled in terms of one or more attributes from the logical data model.

Figure 5.40 presents the Object structure which contains the process modelling concepts applied. This figure shows also which relationships exist between physical process models and data models.
5.3.3 Relationship between a way of modelling and a way of working

Both process modelling tasks and data modelling tasks, part of information engineer C's approach, are characterised as shallow. They do not play a role of major importance. Process modelling was performed only to construct *physical* process models. The physical process model was used when constructing a physical data model.

Some initial logical data models were constructed independent of the construction of physical models.

The distinguished relationships between information engineer C's way of modelling and his way of working are represented in figure 5.41.

5.3.4 Observations

The approach followed by information engineer C consists essentially of two stages: understanding the problem situation and evaluating the preferred solution.

In the understanding stage, the approach to be followed was itself a study object, probably due to the novelty of case 2. Explicit attention was paid to the formulation of a statement of purpose. To acquire a sufficient insight into the problem area, subproblems were distinguished, and causes and effects of subproblems were studied and related. Furthermore,
effects of the zero-solution were investigated. Functional requirements were formulated to evaluate the preferred solution. During this stage, data modelling tasks were dominant to process modelling tasks. Process modelling tasks were not performed.

During evaluation of the preferred solution, both process modelling tasks and data modelling tasks were performed. The resulting models are characterised as physical models.

5.4 Observations

This chapter has presented the approaches that three experienced information engineers followed when solving case 2. Some general observations apply to the approaches followed. We present these observations in this section.

5.4.1 Way of working

The approaches followed have been described in this chapter, resulting in a series of Task structures, see figures 5.1 up to and including 5.10, 5.16 up to and including 5.24, and 5.29
Figure 5.41: Information interdependency diagram, representing relationships between the way of modelling and the way of working of information engineer C when solving case 2 up to and including 5.37. At a high level of consideration, the approaches are similar and can be easily compared. Three main tasks are distinguished: acquiring a sufficient understanding of the problem situation, developing alternative solutions, and choosing a satisfactory solution.

The first task turned out to be a time intensive one. This is probably due to the fact that case 2 involved novelty. The three experts all stated this fact explicitly during their modelling sessions ("It is an appealing and intriguing problem. But how can I structure it. I do not see any clue at the moment!"). The novelty and time intensity led to explicit considerations about the approach to be followed, which is observed particularly in the initial stages of the ways of working of information engineers B and C. Extensive validation activities and an explicit formulation of a statement of purpose are activities which all the experts performed as part of the understanding task. The use of information models was reduced to preliminary data models, due to the characteristics of case 2. This fact was also affirmed by the three experts when acquiring insight into the problem situation ("There is no reason at all to develop a process model in this stage. Its contents are a mystery to me!"). Instead of developing process models, other activities were performed to get adequate insight. Developing realistic problem domain scenarios, distinguishing subproblems, relating subproblems and problem effects, and listing functional requirements were among these activities.

The development of alternative solutions involved developing a set of well-defined criteria to evaluate solutions. Criteria were derived from functional requirements. Furthermore, self-preferred criteria were added to the list of criteria. Rejections of alternative solutions were made very rapidly, after shallow development. The list of criteria was used to effectively reduce the complexity of the solution development task. The solution development task and the choice task also involved process models. The choice of a solution included further
development of this solution. Process modelling was performed more frequently than data modelling at this stage, probably due to the characteristics of the problem domain. The problem domain has a non-trivial structure, the problem was how to deal with it.

Summarising, emphasis was gradually shifted from data modelling in the understanding stage, to get an adequate insight into the complexly structured problem area, to process modelling in the design stages, to get an adequate insight into the functionality of possible solutions.

5.4.2 Way of modelling

We have restricted the presentation of a way of modelling to concepts which relate to data modelling and process modelling. Representing the ways of modelling resulted in a number of Object structures, see figures 5.11 up to and including 5.14, 5.25 up to and including 5.27, and 5.38 up to and including 5.40.

Both data models and process models are used in different ways during the information modelling approaches followed. Differences in their usage depend on the stage of the information modelling process. Two differences in usage can be seen. Data models and process models may focus on different systems: the real system or the information system. Furthermore, they may pay attention to different aspects of a system. For instance, they may be used to model an information systems, including implementation aspects.

Data modelling activities and process modelling activities were performed mainly to achieve insight, be it into the problem situation or into an alternative solution. These modelling activities were observed to have an ad hoc character. When enough insight was acquired, resulting intermediate models were discarded. Explicit use of data models or process models in subsequent or future modelling activities was rarely observed. Due to the fact that modelling activities are characterised as ad hoc and shallow, few concepts were used in creating process models and data models. These main concepts are similar for the three experts. The Object structures which represent the concepts employed in data modelling or process modelling can thus be easily compared.

Although we have not focused on concepts other than those relating to data models and process models, we observed that the experts were interested in aspects of the problem area which can not be modelled explicitly in diagrams. Concepts such as "problem cause", "problem effect", and "functional requirement" are typical examples. We assume that the employment of few concepts related to data models and process models, and the employment of many non-diagramming concepts can be ascribed to the novelty of case 2, resulting in a time intensive understanding stage.

The three ways of modelling differed with regard to the choice of graphical conventions used to represent concepts employed. The same applied data modelling concepts were mapped to various graphical notations.
Chapter 6

Effective information modelling support revisited

In chapter 3, we represented prescribed information modelling knowledge from [Yourdon, 1989]. Chapters 4 and 5 have focused on applied modelling knowledge, using two different information modelling cases.

In this chapter we seek to summarise the main insights we gained in these chapters. Section 6.1 presents differences and similarities in ways of modelling. Section 6.2 covers the various ways of working. Although it has not been our purpose to test the approach we have followed to acquire and represent information modelling knowledge, we pay some attention to its evaluation in section 6.3. Section 6.4 presents our overall conclusions, and relates them to our research questions.

6.1 Way of modelling

We first focus on the various ways of modelling. Our main findings are listed in table 6.1. In this section, we discuss these findings in more detail.

<table>
<thead>
<tr>
<th>Ways of modelling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Similar model types</td>
</tr>
<tr>
<td>Model type variants</td>
</tr>
<tr>
<td>Additional modelling concepts</td>
</tr>
<tr>
<td>Different graphical notations</td>
</tr>
</tbody>
</table>

Table 6.1: Insights in ways of modelling

Similar model types

Both Yourdon's way of modelling as it has been prescribed, and the way of modelling as it has been applied in the knowledge acquisition sessions include similar model types,
namely data flow diagrams and entity-relationship diagrams. This is not too unexpected an observation, since the information engineers were asked beforehand to use at least these two modelling techniques as part of their approach, if possible and appropriate.

**Model type variants**

For each of these two model types, we have represented the modelling concepts and their interrelationships, prescribed in chapter 3, and applied in chapters 4 and 5. Focusing on the modelling concepts as part of one of these model types, we observe some similarities and some differences in chapters 3 through 5. We use the term of *model type variant* to refer to one specific set of modelling concepts which has been prescribed or applied. In the case of ERDs and DFDs, we can distinguish seven model type variants: one prescribed in chapter 3, and six applied in the chapters 4 and 5, see table 6.2. In this section we discuss the similarities and differences between the model type variants.

<table>
<thead>
<tr>
<th></th>
<th><strong>ERD variants</strong></th>
<th><strong>DFD variants</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>prescribed</td>
<td>figure 3.2</td>
<td>figure 3.5</td>
</tr>
<tr>
<td>applied</td>
<td>4.12</td>
<td>4.13</td>
</tr>
<tr>
<td>(case 1)</td>
<td>4.24</td>
<td>4.25</td>
</tr>
<tr>
<td></td>
<td>4.37</td>
<td>4.42</td>
</tr>
<tr>
<td>applied</td>
<td>5.12</td>
<td>5.14</td>
</tr>
<tr>
<td>(case 2)</td>
<td>5.26</td>
<td>5.27</td>
</tr>
<tr>
<td></td>
<td>5.38</td>
<td>5.40</td>
</tr>
</tbody>
</table>

Table 6.2: Figures of ERD variants and DFD variants

To structure such a discussion, we must first find a *way of comparing* the Object structures. To this end, we list the main Object types in each model type variant. In order to give a concrete form to the slippery term “main” in the previous sentence, we use a number of conventions when listing the “main” Object types of a model type variant:

1. Label types, such as “Name” and “Number”, are not listed.
2. Relationships between Object types are not listed.
3. Object types which serve only to describe a characteristic of one other Object type, are itemised as properties beneath the other Object type. Examples of properties are “Frequency” and “Size”.
4. Generalised object types (which usually have been introduced in cases where several Object types have a Relationship in common) are not listed.
5. The existence of specialisation hierarchies is shown.
6. Relationships between Object types which express the notion of decomposition are shown only by the name of the Object type involved together with the word "decomposition".

Applying this approach to the Object structures of chapters 3 through 5 leads to a number of Object type lists. Table 6.3 shows the ERD variant as prescribed. Table 6.4 shows the ERD variants as applied. Table 6.5 shows the DFD variant as prescribed. Table 6.6 shows the DFD variants as applied. The characters in the column headers of the tables 6.4 and 6.6 refer to the information engineers. The numbers in the rows of these tables refer to the cases. The notation convention is used to show specialisation hierarchies by denoting the pater familias with an asterisk.

<table>
<thead>
<tr>
<th>prescribed</th>
<th>object type</th>
<th>relationship</th>
<th>role</th>
<th>subtype</th>
<th>associative object type</th>
</tr>
</thead>
</table>

Table 6.3: Prescribed ERD variant

<table>
<thead>
<tr>
<th>applied</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>entity type</td>
<td>entity type</td>
<td>entity type</td>
</tr>
<tr>
<td></td>
<td>- volume</td>
<td>- sample value</td>
<td>- volume</td>
</tr>
<tr>
<td>1</td>
<td>relationship</td>
<td>relationship</td>
<td>- definition</td>
</tr>
<tr>
<td></td>
<td>- volume</td>
<td></td>
<td>relationship</td>
</tr>
<tr>
<td></td>
<td>role</td>
<td>role</td>
<td>role</td>
</tr>
<tr>
<td></td>
<td>- cardinality</td>
<td>- cardinality</td>
<td>- cardinality</td>
</tr>
<tr>
<td></td>
<td>data element</td>
<td>attribute</td>
<td>attribute</td>
</tr>
<tr>
<td></td>
<td>subtype</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>entity type (*)</td>
<td>entity type</td>
<td>entity type</td>
</tr>
<tr>
<td></td>
<td>relationship</td>
<td>relationship</td>
<td>relationship</td>
</tr>
<tr>
<td></td>
<td>- sample value</td>
<td>- sample value</td>
<td>- sample value</td>
</tr>
<tr>
<td></td>
<td>role</td>
<td>role</td>
<td>role</td>
</tr>
<tr>
<td></td>
<td>- cardinality</td>
<td>- cardinality</td>
<td>- cardinality</td>
</tr>
<tr>
<td></td>
<td>data element</td>
<td>data element</td>
<td>data element</td>
</tr>
</tbody>
</table>

Table 6.4: Applied ERD variants

These tables show that the main modelling concepts are similar over the model type variants. ERDs always consist of Entity types and Relationships. Furthermore, the concept of
Table 6.5: Prescribed DFD variant

Table 6.6: Applied DFD variants

Role is always present, albeit mostly implicitly. This concept is necessary to handle cases in which one and the same Entity type participates more than once in one Relationship. DFDs always consist of processes and data stores, with flows between them.

Although the main modelling concepts are similar, we observe that at the same time each model type variant has its own modelling concepts. Differences between the model type variants are clear from tables 6.3 up to and including 6.6. Differences between model type variants are explained by the use of different additional modelling concepts and the use of more refined modelling concepts.

Comparing the prescribed model type variants to the applied ones, we observe that the prescribed modelling concepts of “complex data flow” and “associative object type” are not
used at all. At the same time, the occurrence of specialisation hierarchies in the applied model type variants show that the experienced information engineers use more refined modelling concepts. Finally, the applied model type variants contain more Object types which serve communication purposes (e.g. "Sample value") or which provide quantitative information (e.g. "Frequency" and "Volume").

So far, comparing model type variants has focused on the modelling concepts in the meta-models. Model type variants may, however, also differ due to different sets of constraints. An illustrative example is the use of ERDs. The meta-models given in figures 4.12, 5.12, and 5.38 (on pages 87, 128, and 150), show the use of the binary variant of ERDs, whereas the meta-models given in figures 4.24, 4.37, and 5.26 (on pages 98, 110, and 140), show the use of the n-ary variant. The difference is expressed by means of an occurrence frequency constraint, which restricts the number of Entity types that participate in a Relationship to two in the binary variant.

<table>
<thead>
<tr>
<th>Model type variants</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main modelling concepts are similar</td>
</tr>
<tr>
<td>Different additional modelling concepts are used</td>
</tr>
<tr>
<td>Different refinements of modelling concepts occur</td>
</tr>
<tr>
<td>Different constraints apply</td>
</tr>
</tbody>
</table>

Table 6.7: Insights in model type variants

Summarising, the major observations concerning model type variants are itemised in table 6.7.

**Additional modelling concepts**

In addition to ERDs and DFDs, several other modelling concepts are used, in particular to create a (sometimes only mental) model of organisational aspects during the problem analysis stage. These non-diagramming concepts are found only in the applied ways of modelling. Some typical examples are: "problem", "problem cause", "organisation unit", "information need", and "requirement".

**Different graphical notations**

Although a comparison of the graphical notations prescribed, and applied by the three experienced information engineers, has not been our primary concern, we observe that several different graphical notations are used to denote one modelling concept. Figure 6.1 shows an example of the external representation of the modelling concept "Relationship", which is used as part of ERDs. Information engineer C even used two different graphical notations during one knowledge acquisition session. Clearly, the choice of a fixed set of graphical notations is not considered to be a matter of relevance during the problem analysis stage.
6.2 Way of working

Now that we have discussed the insights gained in a way of modelling, we turn our attention towards the various ways of working. The main results are listed in table 6.8. This table also includes the findings on relationships between a way of modelling and a way of working, since they are closely related. In this section, we discuss these findings in more detail.

<table>
<thead>
<tr>
<th>Ways of working</th>
</tr>
</thead>
<tbody>
<tr>
<td>Similar modelling tasks</td>
</tr>
<tr>
<td>Prescribed linear versus applied opportunistic</td>
</tr>
<tr>
<td>Problem domain dependent</td>
</tr>
<tr>
<td>Information engineer dependent strategies</td>
</tr>
<tr>
<td>Different process modelling strategies</td>
</tr>
<tr>
<td>Different data modelling strategies</td>
</tr>
<tr>
<td>Dependent on way of modelling</td>
</tr>
<tr>
<td>Strong user involvement</td>
</tr>
<tr>
<td>Increasing structuredness</td>
</tr>
</tbody>
</table>

Table 6.8: Insights in ways of working

Similar modelling tasks

Yourdon’s way of working both as it is prescribed and as it was applied in the knowledge acquisition sessions include similar modelling tasks, namely data modelling tasks and process modelling tasks. This is not too surprising a result, since the information engineers
were required explicitly to use at least the entity-relationship diagramming and the data flow diagramming techniques.

**Prescribed linear versus applied opportunistic**

The *order* in which the various modelling tasks are performed differs to a large extent. A clear distinction exists between prescribed modelling knowledge and applied modelling knowledge, in this respect. Whereas an almost strictly *linear* order of performing modelling tasks is prescribed, a very *opportunistic* order is actually used. This order seems to be determined by at least two essentially different factors: the problem domain and the information engineer. Both factors are discussed below.

**Problem domain dependent**

The information engineers reformulated their approach several times during the course of the knowledge acquisition sessions. In some cases, they even scheduled a number of tasks to be performed in advance. In most cases, however, they only stated that they preferred to pay attention to a specific part of the problem domain, usually to fill clear lacunae in their insights into the problem domain. Their momentary needs strongly influenced the order in which the several modelling techniques were used. Modelling techniques were used as a means to increase insight or to communicate insights, be it in the problem domain itself or in a specific solution scenario.

The characteristics of case 2 influenced the applied modelling knowledge to a larger extent than those of case 1. In case 2, all the information engineers emphasised data modelling tasks to gain insight into the problem domain, and process modelling tasks to propose a solution. In case 1, each information engineer showed his own preferred information modelling strategies.

**Information engineer dependent strategies**

When focusing on the emphasis put on either data modelling or process modelling, we observe three different strategies among the information engineers, when solving case 1. Information engineer B clearly considers data modelling to be dominant to process modelling. In information engineer C's approach, process modelling is emphasised. Information engineer A attempts to give equal emphasis to both: intermediate process modelling results and data modelling results have a mutual influence in his approach.

**Different process modelling strategies**

Chapter 4 showed three different process modelling strategies: *input driven* process modelling, *output driven* process modelling, and *data driven* process modelling. From an input driven point of view, processes handle events, and lead to other processes. From an output driven point of view, processes result in information needs, and other processes are necessary to deliver the input for these processes. From a data driven point of view, processes manipulate data, i.e., create, read, use, and delete instances of entity types, relationships, and attributes.
Different data modelling strategies

Chapter 4 showed three different data modelling strategies as well: noun driven data modelling, object driven data modelling, and process driven data modelling. In the noun driven strategy, each noun in the description of the problem area is considered to be a candidate entity. In the object driven strategy, objects in the real world are related to each other. Each object is questioned for the necessity of storing information on it. The process driven strategy investigates each operating process for entity types, and integrates the resulting partial data models.

Dependent on way of modelling

One should realise that such different strategies in a way of working can be essentially traced back to different information interdependencies. Hence, different strategies may be based on different ways of modelling. Examples of information interdependencies between data models and process models are: Data flows are related to Data elements, External parties are related to Entity types, Data stores are related to Entity types, and Processes manipulate Data model objects (be they Entity types, Relationships, or Data elements). Each of these information interdependencies can be represented in terms of an Object structure, by means of a Relationship which associates the two Object types involved. Additional verification rules then represent which strategy is followed. The following two sample verification rules indicate a process driven strategy and a data driven strategy, respectively:

FOR-EACH p IN Process HOLDS p manipulates Data model object

and:

FOR-EACH d IN Data model object HOLDS d is manipulated by Process.

Strong user involvement

The information engineers all aim at strong user involvement. This can be concluded from several observations. They often validate their results with respect to correctness and completeness. They focus on comprehensibility of intermediate information models, by adding sample values or quantitative data. Information engineer B decided not to present an abstract data model for this reason, when solving case 1.

Increasing structuredness

The information interdependency diagrams of figures 3.10, 4.14, 4.26, 4.43, 5.15, 5.28, and 5.41 (on pages 73, 89, 100, 115, 131, 142, and 153) represent relationships between a way of modelling and a way of working, both in prescribed modelling knowledge and in applied modelling knowledge. They clearly show several modelling stages in which the use of modelling techniques gradually changes. More modelling concepts are used in the course of information systems development processes, and a growing set of verification rules is satisfied. The nature of modelling tasks changes from free towards structured.
6.3 Evaluation of the approach to representing modelling knowledge

This section is concerned with an evaluation of the approach to representing modelling knowledge of chapter 2. We briefly review the meta-modelling technique, which has been our way of modelling as a knowledge engineer, at method level (see figure 1.3, page 16). Then we evaluate the process of acquiring and representing modelling knowledge, which has been our way of working as a knowledge engineer.

Way of modelling

We first recapitulate our criteria used while choosing an adequate meta-modelling technique. The meta-modelling technique should enable the representation of both a way of modelling and a way of working, as well as their interrelationships. Furthermore, the meta-modelling technique should allow for the representation of complex structures and constraints as part of a way of modelling. Both criteria clearly related to the expressive power of a meta-modelling technique.

We have decided to use the combination of the object structure diagramming technique PSM, the task structure diagramming technique, and the constraint language LISA-D as our meta-modelling technique. Furthermore, we have introduced a diagramming technique to represent information interdependencies. The results of its use, see chapters 3, 4, and 5, show that this combination has actually enabled the representation of a way of modelling, a way of working, and their interrelationships. The object composition mechanisms in PSM (see section 2.3.1.4) and the constraint language LISA-D have been used to represent complex structures and constraints in a way of modelling. Hence, we conclude the two above criteria to be satisfied.

Although comprehensibility was not formulated as an explicit criterion for the choice of a meta-modelling technique beforehand, we can conclude from the results of the use of our meta-modelling technique in chapters 3, 4, and 5, and also particularly from the meta-model validation sessions with the information engineers, that the meta-models are relatively easily understood. Four factors have contributed to the comprehensibility: (i) it is possible to represent both Object structures and Task structures graphically, (ii) we have presented the Object structure not as a whole, but split up into a number of relatively small Object structures which rarely contained more than twenty Object types, (iii) the notion of decomposition in Task structures allowed for the presentation of small Task structures, and (iv) use of the constraint language LISA-D results in semi-natural sentences which are relatively easily understood, provided of course that the chosen Object type names and Role names are meaningful.

Way of working

In this section, our way of working while representing modelling knowledge is evaluated. We cope with the representation of prescribed modelling knowledge and applied modelling
knowledge separately, as the approach to representing applied modelling knowledge includes the acquisition of modelling knowledge as such.

The process of representing prescribed modelling knowledge has given rise to a meta-model, see chapter 3, by which a detailed understanding of the prescribed modelling knowledge of [Yourdon, 1989] is achieved. The representation process as such is characterised as smooth: a few design decisions had to be taken explicitly, due to the nature of the method handbook of [Yourdon, 1989]. Some modelling concepts and verification rules are introduced in the handbook very implicitly, sometimes only in the exercises. Other modelling concepts and verification rules are not dealt with explicitly, but are not excluded from the handbook either. We have decided to focus our representation process on explicitly presented modelling knowledge only.

The process of representing applied modelling knowledge has led to a series of meta-models, see chapters 4 and 5, by which a detailed understanding of applied modelling knowledge is achieved (see also sections 6.1 and 6.2). The use of think aloud protocols as a knowledge acquisition technique improved the information engineers’ think aloud process, and resulted in explicit reporting on their modelling processes. These results, however, were gained at considerable cost. We will briefly evaluate the costs of this approach to acquiring applied information modelling knowledge (the details can be found in [Harst and Herlé, 1991]).

We structure the discussion by distinguishing between the four stages of our knowledge acquisition approach: preparation, elicitation, interpretation, and conceptualisation (see also figure 2.19, page 46).

In the preparation task, it is important to familiarise the experts with the knowledge elicitation task, as thinking aloud while solving problems is not a natural thing to do. The actual knowledge acquisition sessions were preceded by some short exercise sessions in which thinking aloud while solving a small problem was practised.

The elicitation task is rather fatiguing for the experts. The terminal connection as part of the experimental setting (see figure 2.20, page 49) enables verbal communication, but at a lower speed. Paralinguistic and non-verbal communication were excluded by the experimental setting. The think aloud process together with non-verbal communication lead to a tiring process for the information engineers.

The six elicitation sessions led to voluminous text protocols, each including about 150 pages of text and about 30 diagrams, some of which went through several stages.

The interpretation task resulted in a text-based model, containing text units labelled as tasks or concepts. This task is a time-intensive task for the knowledge engineer, due to the bulky text protocols which have to be interpreted. For example, the text-based model deduced from the protocol transcript of [Verhoef, 1990] includes hundreds of text units labelled as “task” and thousands of text units labelled as “concept” (see [Harst and Herlé, 1991]), which clearly provides an indication of the time-intensive nature of the interpretation task.

During interpretation, it is not always clear whether a text unit refers to concepts at meta-level or at application level (see figure 1.3, page 16). Most fragments clearly describe a
concept used by the expert, or refer to a specific concept instance. A rigid boundary between the two levels can not always been drawn in practice. For instance, task units such as “client” or “primary process” can be seen as subtypes of the Object types “external party” and “process”, or as specific instances of these Object types.

Interpretation and conceptualisation influence each other highly, and are performed iteratively several times. The interpretation of dialogues between the information engineer and the problem owner may change during these rounds.

Additional interviews with the information engineers are inevitable in the interpretation and conceptualisation stages in order to fill in gaps in the achieved understanding of the information engineers’ modelling knowledge. Furthermore the interviews are necessary to validate intermediate interpretations and meta-models.

In the conceptualisation stage, large parts of protocol could sometimes be reduced to small task structures, due to the intention to remove domain dependent elements. For example, case 1 involved a close and time-consuming investigation of “scheduling activities” for all “exam types”. Since “exam type” is clearly a domain dependent concept, the task structure consists of a simple iteration of one single task.

Summarising, the approach to representing modelling knowledge of chapter 2 has greatly contributed towards achieving a detailed understanding of both prescribed and applied information modelling knowledge, but at the price of a time-intensive approach. Similar experiences are reported in [Wijers, 1991].

## 6.4 Conclusions

In the previous sections, we have evaluated our approach to representing information modelling knowledge, and we have summarised our insights in prescribed and applied modelling knowledge. We have presented differences and similarities, both in ways of modelling and in ways of working. Sections 6.1 and 6.2 have shown a variety of differences between prescribed and applied modelling knowledge, and even between individual information engineers working in particular problem domains. From section 6.3 it has become clear that our approach has led to a detailed understanding of modelling knowledge, but at the expense of a time-intensive approach.

We relate our findings to our discussion of “effective information modelling support” in section 1.4, where we have presented two disadvantages of the meta-modelling approach: (i) it is not clear whether it is worthwhile to specify a detailed modelling knowledge base for each information engineer individually, and (ii) support tuned to the individual might contradict the organisational need for standardisation, depending on the differences between applied and prescribed modelling knowledge. As such, our findings strongly confirm these two drawbacks.

As a consequence, the relevance of our second research question has been confirmed, that is, we feel the need to investigate possibilities for the development of specific mechanisms which support the adaptation of standardised modelling knowledge towards individual modelling.
support. The findings in this chapter lead us to refine the formulation of the second research question. It has become clear that the ways of modelling show a large number of differences over the individual experts. The same observation holds for the various ways of working. We have, however, argued that the key differences in the ways of working (concerning different strategies followed by information engineers) can partially be traced back to differences in ways of modelling. Therefore, we focus our discussion in the next chapter on mechanisms which support adaptation of standardised modelling knowledge towards individual modelling support, as far as a way of modelling is concerned.
Chapter 7

Towards effective information modelling support

7.1 Introduction

The objective of this chapter is to formulate a theory about the use of modelling knowledge to achieve more effective information modelling support. First, recall from chapter 1 that we have adopted a specific view on what we consider to be “effective information modelling support”. In section 1.4.1, we have pointed out that information engineers should preferably be able to specify, to adapt, and to refine modelling knowledge as part of CASE tools, according to their own practical experience and their own preferences. Information modelling support environments that dispose of explicit modelling knowledge can be customised to individual ways of modelling and ways of working. In section 1.4.2, however, we have presented two drawbacks of the meta-modelling approach. We have reservations regarding the economic question, whether it is worthwhile to specify a detailed modelling knowledge base for each information engineer individually. Furthermore, we have pointed out that support tuned to the individual may contradict the organisational need for standardisation. From these two drawbacks we considered it desirable to investigate whether mechanisms can be developed that support the adaptation of standardised and generic modelling knowledge towards more individualised modelling knowledge.

In chapter 6 we have summarised the explorative insight we have gained, into the extent to which the individual ways in which modelling knowledge is applied, deviate from prescribed modelling knowledge. We have shown a number of specific examples of deviations in ways of modelling. Furthermore, we have shown several different strategies while considering ways of working. We have pointed out, however, that the main differences in ways of working can be traced back to underlying differences in ways of modelling. Therefore, we can safely focus ourselves on a way of modelling while discussing mechanisms for the adaptation of generic modelling knowledge towards more individualised modelling knowledge.

Intuitively, a meta-model is more generic - or, less individualised - if more choices with regard to the conceptualisation and interpretation (see section 2.4) of a way of modelling
are left open. To put it the other way around, taking a specific point of view in the specification of a way of modelling (e.g. choosing for being supported in the binary variant of entity-relationship modelling instead of the n-ary variant) results in a more individualised meta-model.

We relate the notion of genericity to the set of all possible populations of a meta-model. We refer to meta-model \( \mathcal{MM}_1 \) as being more generic than meta-model \( \mathcal{MM}_2 \), if each population of meta-model \( \mathcal{MM}_2 \) can be transformed in some way or another to a population of meta-model \( \mathcal{MM}_1 \). Two meta-models \( \mathcal{MM}_1 \) and \( \mathcal{MM}_2 \) are equivalent if meta-model \( \mathcal{MM}_1 \) is generic compared to meta-model \( \mathcal{MM}_2 \) and vice versa.

Our approach to the individualisation of generic modelling knowledge consists of a set of specific transformations which transform meta-models. We define a set of basic transformations. More complex, and more suitable, transformations are composed by concatenation of our basic meta-model transformations. The approach is characterised as specific and formal. We clarify these two properties below.

Transformations of populations of meta-models have been discussed by [Hofstede et al., 1992c]. They show a very general approach, in which each population is coded to a finite bitstring. Their approach is too general for our purpose, since each meta-model is considered to be equivalent to each other in this approach, due to the complexity of the coding algorithm. We therefore present an approach which consists of a set of specific transformations, transforming populations into new populations rather than into bitstrings. The impact of the transformations on meta-models is then more easily surveyed and controlled.

The set of specific transformations used to individualise a given meta-model has a formal basis. We adopt a formal approach to avoid deficiencies such as ambiguity and incompleteness. This choice has been extensively and convincingly motivated in the literature, see within the field of information systems e.g. [Cohen, 1989], [Engels et al., 1992], [Hofstede and Weide, 1992], [Sernadas et al., 1989], and [Tse and Pong, 1989].

This chapter is structured as follows. We first define basic transformations in section 7.4. These transformations are referred to as basic, since they add one specific element to a meta-model component, or delete one specific element from a meta-model component. We subsequently present several more suitable transformations from the literature on schema transformations, see section 7.5. These transformations turn out to be composed ones: they can be expressed by concatenating our basic transformations. We discuss for each transformation whether it leads to a more individualised meta-model, to a more generic meta-model, or to an equivalent meta-model, see section 7.6. Finally, some applications of meta-model transformations are given in section 7.7. Section 7.8 concludes this chapter.

To define meta-model transformations, we first have to define formally what we consider to be a meta-model. Section 7.2 is dedicated to this issue. Finally, we define a relational algebra in section 7.3, as a primitive language upon which our language of meta-model transformations is based. The mathematical notations used in this chapter are explained in appendix D.
7.2 Fundamentals

To describe meta-model transformations in a formal way, we first have to define meta-models as such in a formal way. The definition is taken entirely from [Hofstede, 1993].

We use the word meta-model in this chapter in a restricted way, namely to refer to the way of modelling in a meta-model. To be precise, we view a meta-model as an Object structure and a set of graphical constraints. Therewith, we do not consider non-graphical constraints, and external representations. This is a minor restriction only, since non-graphical constraints are exceptional ones. We have presented an informal introduction to Object structures and graphical constraints in the sections 2.3.1 and 2.3.2.1. We recall that an Object structure defines the set of possible application models (according to a specific way of modelling). An application model thus can be considered to be a population of a meta-model. Graphical constraints restrict the set of possible populations of an Object structure.

We pay attention to the formal definition of meta-models as such (section 7.2.1) and populations of meta-models (section 7.2.2). Section 7.2.3 discusses the notion of dependencies in meta-models. This notion is relevant to describe specific meta-model transformations, which delete one specific element from a meta-model component, in which case dependent elements have to be deleted as well.

7.2.1 Meta-models

Formally, a meta-model is a 2-tuple $\mathcal{M} = (\mathcal{OS}, \mathcal{R})$, where $\mathcal{OS}$ is an Object structure and $\mathcal{R}$ is a set of constraints. This section defines the syntax of an Object structure $\mathcal{OS}$ and the syntax of the various types of constraints in $\mathcal{R}$. To further clarify the definitions, an extended example of a meta-model is given.

An Object structure $\mathcal{OS}$ is a 15-tuple consisting of the following components:

1. A non-empty set $\mathcal{O}$ of Object types.

2. A set $\mathcal{L}$ of Label types. Label types are Object types: $\mathcal{L} \subseteq \mathcal{O}$.

3. A set $\mathcal{E}$ of Entity types. Entity types are Object types: $\mathcal{E} \subseteq \mathcal{O}$.

4. A set $\mathcal{P}$ of Roles.

5. A partition $\mathcal{F}$ of the set $\mathcal{P}$. The elements of $\mathcal{F}$ are called Relationships. Relationships are Object types: $\mathcal{F} \subseteq \mathcal{O}$.

6. A set $\mathcal{G}$ of Set types. Set types are Object types: $\mathcal{G} \subseteq \mathcal{O}$.

7. A set $\mathcal{S}$ of Sequence types. Sequence types are Object types: $\mathcal{S} \subseteq \mathcal{O}$.

8. A set $\mathcal{C}$ of Schema types. Schema types are Object types: $\mathcal{C} \subseteq \mathcal{O}$.

9. A function $\text{Base} : \mathcal{P} \rightarrow \mathcal{O}$. The base of a Role is the related Object type.
10. A function \( \text{El} : \mathcal{G} \cup \mathcal{S} \rightarrow \mathcal{O} \). This function yields the Element of Set types and Sequence types.

11. A relation \( \text{Schema} \subseteq \mathcal{C} \times \mathcal{O} \). This relation describes the Elements in Schema types.

12. A relation \( \text{Spec} \subseteq \mathcal{E} \times (\mathcal{O} \setminus \mathcal{L}) \) on Object types, expressing specialisation.

13. A relation \( \text{Gen} \subseteq \mathcal{E} \times (\mathcal{O} \setminus \mathcal{L}) \) on Object types, expressing generalisation.

14. A set \( D \) of concrete domains used to instantiate Label types (e.g. string, natural number).

15. A function \( \text{Dom} : \mathcal{L} \rightarrow D \). This function yields the domain of the Label types.

Note that the set \( \mathcal{O} \) is an auxiliary one: it is used as a shorthand for \( \mathcal{L} \cup \mathcal{E} \cup \mathcal{F} \cup \mathcal{G} \cup \mathcal{S} \cup \mathcal{C} \). All sets are assumed to be finite and disjoint, unless explicitly stated otherwise.

The auxiliary function \( \text{Rel} : \mathcal{P} \rightarrow \mathcal{F} \) yields the Relationship in which a given Role is contained, and is defined by: \( \text{Rel}(p) = f \iff p \in f \). The auxiliary function \( \text{Rel}(\mathcal{P}) \rightarrow \mathcal{F}(\mathcal{F}) \) returns the set of Relationships containing one or more of the Roles specified, and is defined by \( \text{Rel}(\tau) = \{ f \mid \exists p \in \tau \text{Rel}(p) = f \} \).

From this point on, \( \text{spec}(a) \) is used as an abbreviation for \( \exists x \in \mathcal{O} [a \text{Spec } x] \) and \( \text{gen}(a) \) is used as an abbreviation for \( \exists x \in \mathcal{O} [a \text{Gen } x] \). Object type \( b \) is the \text{pater familias} of Object type \( a \) \( (b = \text{PaterFam}(a)) \) if it satisfies \( a \text{Spec}^* b \land \neg \text{spec}(b) \).

<table>
<thead>
<tr>
<th>Type of constraint</th>
<th>Syntax of constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>enumeration</td>
<td>enumeration(l, d)</td>
</tr>
<tr>
<td>total role</td>
<td>total(( \tau ))</td>
</tr>
<tr>
<td>uniqueness</td>
<td>unique(( \tau ))</td>
</tr>
<tr>
<td>occurrence frequency</td>
<td>frequency(( \tau, n, m ))</td>
</tr>
<tr>
<td>equality</td>
<td>equal(( \xi ))</td>
</tr>
<tr>
<td>subset</td>
<td>subset(( \xi ))</td>
</tr>
<tr>
<td>exclusion</td>
<td>exclusion(( \xi ))</td>
</tr>
<tr>
<td>membership</td>
<td>member(p, q)</td>
</tr>
<tr>
<td>subtype total</td>
<td>sub_total(( \chi ))</td>
</tr>
<tr>
<td>subtype exclusion</td>
<td>sub_exclusion(( \chi ))</td>
</tr>
<tr>
<td>set type exclusion</td>
<td>set_exclusion(( \gamma ))</td>
</tr>
<tr>
<td>set type cover</td>
<td>set_cover(( \gamma ))</td>
</tr>
<tr>
<td>set type cardinality</td>
<td>set_card(( \gamma, n, m ))</td>
</tr>
</tbody>
</table>

Table 7.1: Syntax of constraints

Now that the syntax of Object structures has been presented, we focus on graphical constraints, being the second component of a meta-model. \( \mathcal{R} \) consists of constraints of various
types (see section 2.3.2.1). These constraints are denoted syntactically as shown in table 7.1. Several parameters are used in this table: \( l \) is a Label type \((l \in \mathcal{L})\), \( d \) is a set of values \((d \in D)\), \( n \) and \( m \) are natural numbers \((n, m \in \mathbb{N})\), \( p \) and \( q \) are Roles \((p, q \in \mathcal{P})\), \( \tau \) is a non-empty set of Roles \((\tau \subseteq \mathcal{P} \land \tau \neq \emptyset)\), \( \xi \) is a bijection between Roles \((\xi : \mathcal{P} \rightarrow \mathcal{P})\), \( \chi \) is a family of Entity types, i.e. \( \chi \) is a non-empty set of Entity types with a common ancestor in a specialisation hierarchy \((\chi \subseteq \mathcal{E} \land \chi \neq \emptyset \land \forall x, y \in x [\text{PaterFam}(x) = \text{PaterFam}(y)])\), and \( \gamma \) is a Set type (or, if \( \gamma \) participates in a specialisation hierarchy, its pater familias is a Set type) \((\text{PaterFam}(\gamma) \in \mathcal{G})\). It should be clear from section 2.3.2.1 how these parameters are related to the constraints.

![Diagram](image-url)

Figure 7.1: Abstract example of a meta-model

In figure 7.1, we present an abstract example of a meta-model, consisting of an Object structure and a number of graphical constraints. The Object structure in this example is
defined by:

\[ \mathcal{L} = \{K\} \quad \mathcal{G} = \{J\} \]
\[ \mathcal{E} = \{A, B, C, D, E, F, G, H, I\} \quad \mathcal{S} = \emptyset \]
\[ \mathcal{P} = \{j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z\} \quad \mathcal{C} = \emptyset \]
\[ \mathcal{F} = \{a, b, c, d, e, f, g, h\} \]

where Relationship \(a = \{y, z\}, b = \{v, w, x\}, c = \{t, u\}\), and so on. The Roles are related to the Object types: \(\text{Base}(j) = C\), \(\text{Base}(k) = A\), \(\text{Base}(l) = J\), and so on. Finally, \(\text{Elt}(J) = A\), \(\text{Elt}(D) = E\), \(\text{Elt}(F) = F\), \(\text{Spec}(E) = G\), \(\text{Spec}(C) = D\), \(\text{Gen}(D) = C\), \(\text{Gen}(H)\), and \(\text{Dom}(K) = \text{"string"}\).

The graphical constraints are defined as follows in this sample meta-model:

\[
\begin{align*}
\text{total}\{\{t\}\} & \quad \text{sub_total}\{\{B, F\}\} & \quad \text{enumeration}(K, \{'in\', 'out'\}) \\
\text{total}\{o, q\} & \quad \text{sub.exclusion}\{\{F, G\}\} & \quad \text{equal}\{\{p : r, q : s\}\} \\
\text{unique}\{\{t\}\} & \quad \text{set.exclusion}(J) & \quad \text{subset}(\{p : n, q : o\}) \\
\text{unique}\{\{u\}\} & \quad \text{set.cover}(J) & \quad \text{subset}(\{j : x, k : z\}) \\
\text{unique}\{\{w, x\}\} & \quad \text{set.card}(J, 3, 5) & \quad \text{membership}(m, l) \\
\text{frequency}\{\{v\}, 2, 2\} & 
\end{align*}
\]

7.2.2 Populations of meta-models

This section concerns populations of meta-models. The set of possible populations of a meta-model is restricted both by the Object types in its Object structure and by its set of constraints.

We first consider populations of Object structures. A population \(\text{Pop}\) of an Object structure

\[ \mathcal{O} = \langle \mathcal{O}, \mathcal{L}, \mathcal{E}, \mathcal{P}, \mathcal{F}, \mathcal{G}, \mathcal{S}, \mathcal{C}, \text{Base}, \text{Elt}, \text{Schema}, \text{Spec}, \text{Gen}, D, \text{Dom} \rangle \]

is a value assignment of sets of instances to the object types in \(\mathcal{O}\). The expression \(\text{IsPop}(\mathcal{O}, \mathcal{O})\) denotes that \(\text{Pop}\) is a population of the Object structure \(\mathcal{O}\). \(\text{Pop}\) is a mapping \(\text{Pop} : \mathcal{O} \rightarrow \mathcal{O}^{\text{fin}}(\Omega)\), where \(\Omega\) is the universe of instances. Before this universe of instances is defined, specific attention is paid to Label types, as these are the only Object types that can be represented directly. An Object structure can only be instantiated if a link is established between Label types and concrete domains. The instances of Label types then are taken from their associated concrete domains. We recall that the function \(\text{Dom}\) serves this purpose.

Given the universe of Object types \(\mathcal{O}\), the universe of Roles \(\mathcal{P}\), the universe of constraints \(\mathcal{R}\), and the universe of concrete domains \(\mathcal{D}\), the universe of instances \(\Omega\) is inductively defined as the smallest set which allows for the instantiation of Label types, Entity types, Relationships, Set types, Sequence types, and Schema types:

1. \(\mathcal{D} \subseteq \Omega\). Instances from the sorts in the many sorted algebra are elements of the universe of instances. This condition allows for the instantiation of Label types.
2. $\Theta \subseteq \Omega$, where $\Theta$ is an abstract (countable) domain of (unstructured) values that may occur in the population of Entity types.

3. $x_1, \ldots, x_n \in \Omega \land p_1, \ldots, p_n \in \mathcal{P} \Rightarrow \{p_1 : x_1, \ldots, p_n : x_n\} \in \Omega$. In this case, the set $\{p_1 : x_1, \ldots, p_n : x_n\}$ denotes a mapping, assigning $x_i$ to each Role $p_i$. These mappings are intended for the instantiation of Relationships.

4. $x_1, \ldots, x_n \in \Omega \Rightarrow \{x_1, \ldots, x_n\} \in \Omega$. Sets of instances may occur as instances of Set types.

5. $x_1, \ldots, x_n \in \Omega \Rightarrow \langle x_1, \ldots, x_n \rangle \in \Omega$. Sequences of instances are used as instances of Sequence types.

6. $X_1, \ldots, X_n \subseteq \Omega \land O_1, \ldots, O_n \in \mathcal{O} \Rightarrow \{O_1 : X_1, \ldots, O_n : X_n\} \in \Omega$. Assignments of sets of instances to Object types are also valid instances. They are intended for populations of Schema types.

Populations of Object types are bound to a number of formal rules which are defined in [Hofstede, 1993]. We present only two of them, since we will need them later on in this chapter. These rules are to be satisfied by populations in generalisation hierarchies and specialisation hierarchies. The Specialisation Rule states:

$$x \text{ Spec } y \Rightarrow \text{Pop}(x) \subseteq \text{Pop}(y)$$

The Generalisation Rule states:

$$\text{gen}(x) \Rightarrow \text{Pop}(x) = \bigcup_{y \in \text{Gen } x} \text{Pop}(y)$$

Now that the population of an Object structure has been defined, the set of constraints $\mathcal{R}$ is taken into account to define the set of possible populations of a meta-model $\mathcal{M}, \mathcal{M} = \langle \mathcal{O} S, \mathcal{R} \rangle$. A population of a meta-model should be a population of its Object structure and should satisfy its constraints:

$$\text{IsPop}(\mathcal{M}, \mathcal{M}, \text{Pop}) \equiv \text{IsPop}(\mathcal{O} S, \text{Pop}) \land \forall r \in \mathcal{R} [\text{Pop} \models r]$$

An example of a population of the Label types, the Entity types, and the Set types in the meta-model of figure 7.1 is:

- $\text{Pop}(A) = \{a_1, a_2, a_3, a_4, a_5, a_6, a_7\}$
- $\text{Pop}(B) = \{d_1, d_2\}$
- $\text{Pop}(C) = \{d_1, d_2, d_3, d_4, d_5, d_6, h_1, h_2\}$
- $\text{Pop}(D) = \{d_1, d_2, d_3, d_4, d_5, d_6\}$
- $\text{Pop}(E) = \{d_2, d_3, d_4, d_5, d_6\}$
- $\text{Pop}(F) = \{d_3, d_4, d_5, d_6\}$
- $\text{Pop}(G) = \{d_2\}$
- $\text{Pop}(H) = \{h_1, h_2\}$
- $\text{Pop}(I) = \{i_1\}$
- $\text{Pop}(J) = \{\{a_1, a_2, a_3\}, \{a_4, a_5, a_6, a_7\}\}$
- $\text{Pop}(K) = \{\text{in'}, \text{ out'}\}$
Here, the set \{‘in’, ‘out’\} is a subset of the concrete domain “string”. The instances of the Object types \(A, D, H,\) and \(I\) are taken from the abstract domain \(\Theta\).

The populations of some of the Relationships in figure 7.1 are denoted formally below. Figure 7.2 shows a visualisation of these populations.

\[
\begin{align*}
\text{Pop}(a) &= \{y : d_1, z : a_1\}, \{y : d_1, z : a_4\}, \{y : d_2, z : a_1\} \\
\text{Pop}(b) &= \{v : \{y : d_1, z : a_1\}, w : d_6, x : h_1\}, \{v : \{y : d_1, z : a_1\}, w : d_4, x : d_2\} \\
\text{Pop}(c) &= \{u : d_2, t : ‘in’\}, \{u : h_1, t : ‘out’\}
\end{align*}
\]

### 7.2.3 Dependencies in meta-models

This section introduces the notion of dependencies in meta-models. Two Object types \(o_1\) and \(o_2\) are said to depend on each other in cases in which Object type \(o_1\) can only be instantiated if Object type \(o_2\) can be instantiated. Examples of dependencies in figure 7.1 are: (i) Set type \(J\) depends on Object type \(A\), since each instance of Set type \(J\) is a set of instances of Object type \(A\), and (ii) Relationship \(b\) depends on the Object types \(C, D,\) and \(a\), since each Role in Relationship \(b\) takes its value from its base Object type. Note that Relationship \(b\) in the second example depends on the Object types \(A\) and \(B\) indirectly, since Relationship \(a\) depends on them.

Formally, an Object type \(o_1\) depends on an Object type \(o_2\), \(o_1 \triangleright o_2\), if and only if this can
be proven from the following derivation rules:

\[
\begin{align*}
\vdash & \text{Rel}(p) \triangleright \text{Base}(p) \\
\vdash & \text{Elt}(o_1) \\
\vdash & \text{Schema}(o_2) \\
\vdash & o_1 \triangleright o_2 \\
\vdash & \text{Spec}(o_2) \\
\vdash & o_1 \triangleright o_2 \\
o_2 \in \text{Objects(SubRule}(o_1)) & \vdash o_1 \triangleright o_2 \\
o_1 \triangleright o_1 & \vdash \text{error}(o_1) \\
o_2 \triangleright o_1 \wedge \text{error}(o_2) & \vdash \text{error}(o_1) \\
gen(o_1) \wedge \forall o_2, o_1 gen_{o_2} [o_2 \triangleright o_3 \vee o_2 = o_3 \vee \text{error}(o_2) \vee o_2 \triangleright o_1] & \vdash o_1 \triangleright o_3 \\
o_1 \triangleright o_3 \wedge o_3 \triangleright o_2 & \vdash o_1 \triangleright o_2
\end{align*}
\]

We refer to [Hofstede, 1993] for the formal definition of Objects(SubRule(o_1)). Informally, this expression refers to the set of Object types involved in a subtype defining rule.

We use this dependency relationship to define for each Object type o_1 the set of Object types Dep.objects(o_1), the set of Roles Dep.roles(o_1), and the set of constraints Dep.rules(o_1) which depend on that Object type o_1. The set of dependent Object types and the set of dependent Roles are defined in table 7.2.

<table>
<thead>
<tr>
<th>Dep.objects(o_1) =</th>
<th>{ o \in O \mid o \triangleright o_1 } \cup { o_1 }</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dep.roles(o_1) =</td>
<td>{ p \in P \mid \text{Base}(p) \in \text{Dep.objects}(o_1) }</td>
</tr>
</tbody>
</table>

Table 7.2: Definition of dependent Object types and dependent Roles

To define the set of dependent constraints, we have to distinguish auxiliary sets of dependent constraints, one for each type of constraint. The set of dependent constraints Dep.rules(o_1) is easily defined then as the union of the auxiliary sets, see table 7.3.

### 7.3 A relational algebra

This section defines a relational algebra for the meta-modelling technique defined in the previous section. We use the relational algebra in section 7.4 to define how populations of Relationships change as a result of basic transformations. The definition of the relational algebra is based upon [Hofstede, 1993] to a large extent (that is, we have redefined one relational operator and we have added one relational operator).

The relational algebra supports the description of so-called derived Relationships. Relational expressions are used to describe a derived Relationship. A relational expression r is defined by: (i) its schema Schema(r), which is the set of Roles of the derived Relationship, and (ii) its population Val(r) (Pop), which is the population of the derived Relationship.

A relational expression is either a Relationship, or the result of applying one or more relational operators on a Relationship. Eight relational operators are distinguished:
Towards effective information modelling support

<table>
<thead>
<tr>
<th>Dep.enumeration((o_1)) = { enumeration(l, d) \in \mathcal{R} \mid l \in \text{Dep.objects}(o_1) }</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dep.total((o_1)) = { total((\tau)) \in \mathcal{R} \mid \exists_{p \in \mathcal{R}} [p \in \text{Dep.roles}(o_1)] }</td>
</tr>
<tr>
<td>Dep.unique((o_1)) = { unique((\tau)) \in \mathcal{R} \mid \exists_{p \in \mathcal{R}} [p \in \text{Dep.roles}(o_1)] }</td>
</tr>
<tr>
<td>Dep.frequency((o_1)) = { frequency((\tau, n, m)) \in \mathcal{R} \mid \exists_{p \in \mathcal{R}} [p \in \text{Dep.roles}(o_1)] }</td>
</tr>
<tr>
<td>Dep.equal((o_1)) = { equal((\xi)) \in \mathcal{R} \mid \exists_{p \in \text{dom}((\xi)) \cup \text{ran}((\xi))} [p \in \text{Dep.roles}(o_1)] }</td>
</tr>
<tr>
<td>Dep.subset((o_1)) = { subset((\xi)) \in \mathcal{R} \mid \exists_{p \in \text{dom}((\xi)) \cup \text{ran}((\xi))} [p \in \text{Dep.roles}(o_1)] }</td>
</tr>
<tr>
<td>Dep.exclusion((o_1)) = { exclusion((\xi)) \in \mathcal{R} \mid \exists_{p \in \text{dom}((\xi)) \cup \text{ran}((\xi))} [p \in \text{Dep.roles}(o_1)] }</td>
</tr>
<tr>
<td>Dep.member((o_1)) = { member(p, q) \in \mathcal{R} \mid p \in \text{Dep.roles}(o_1) \land q \in \text{Dep.roles}(o_1) }</td>
</tr>
<tr>
<td>Dep.subtotal((o_1)) = { sub.total((\chi)) \in \mathcal{R} \mid \exists_{\chi \in \chi} [e \in \text{Dep.objects}(o_1)] }</td>
</tr>
<tr>
<td>Dep.subexclusion((o_1)) = { sub.exclusion((\chi)) \in \mathcal{R} \mid \exists_{\chi \in \chi} [e \in \text{Dep.objects}(o_1)] }</td>
</tr>
<tr>
<td>Dep.setexclusion((o_1)) = { set.exclusion((\gamma)) \in \mathcal{R} \mid \gamma \in \text{Dep.objects}(o_1) }</td>
</tr>
<tr>
<td>Dep.setcover((o_1)) = { set.cover((\gamma)) \in \mathcal{R} \mid \gamma \in \text{Dep.objects}(o_1) }</td>
</tr>
<tr>
<td>Dep.setcard((o_1)) = { set.card((\gamma, n, m)) \in \mathcal{R} \mid \gamma \in \text{Dep.objects}(o_1) }</td>
</tr>
<tr>
<td>Dep.rules((o_1) = Dep.enumeration((o_1)) \cup Dep.total((o_1)) \cup Dep.unique((o_1)) \cup Dep.frequency((o_1)) \cup Dep.equal((o_1)) \cup Dep.subset((o_1)) \cup Dep.exclusion((o_1)) \cup Dep.member((o_1)) \cup Dep.subtotal((o_1)) \cup Dep.subexclusion((o_1)) \cup Dep.setexclusion((o_1)) \cup Dep.setcover((o_1)) \cup Dep.setcard((o_1))</td>
</tr>
</tbody>
</table>

Table 7.3: Definition of dependent Rules

- related to the schema of a relational expression: projection, extension, join;
- related to the population of a relational expression: union, difference, selection;
- related to the nesting of Relationships: unnest, nest.

We define in this section for each of the relational operators the schema and the population of the resulting relational expression. Since (derived) Relationships are part of a meta-model, relational operators change this meta-model: a new, derived Relationship is added, resulting in an extension of the set of Roles \(\mathcal{P}\) and the set of Relationships \(\mathcal{F}\). If a relational operator results in more changes of the meta-model, we define these changes.

If a relational expression is a Relationship \(f\), then its schema is the set of Roles in \(f\): \(\text{Schema}(f) = f\). Naturally, its population is the population of \(f\): \(\text{Val}(f)(\text{Pop}) = \text{Pop}(f)\).

In general, the schema of a relational expression is derived from the schemata of its constituting relational expressions. The population of a relational expression \(r\) is formally defined as a set of functions. Each function assigns a set of values from \(\Omega\) to one of the Roles in \(\text{Schema}(r)\), determined in the context of the population \(\text{Pop}\) of an Object structure:

\[ t \in \text{Val}(r)(\text{Pop}) \Rightarrow t \in \Omega^{\text{Schema}(r)} \]

176
For the sake of shortness of representation, we refer to the population of a relational expression as $\text{Pop}(r)$ instead of as $\text{Val}[r](\text{Pop})$, therewith overloading the keyword $\text{Pop}$.

Throughout this chapter, we use a specific instance of a meta-model, $\mathcal{M}_1$, which consists of an Object structure $\mathcal{O}_1$ and a set of constraints $\mathcal{R}_1$. Its Object structure is:

$$\langle \mathcal{O}_1, \mathcal{L}_1, \mathcal{E}_1, \mathcal{P}_1, \mathcal{F}_1, \mathcal{G}_1, \mathcal{S}_1, \mathcal{C}_1, \text{Base}_1, \text{Elt}_1, \text{Schema}_1, \text{Spec}_1, \text{Gen}_1, \mathcal{D}_1, \text{Dom}_1 \rangle$$

Furthermore, we use the functions $\text{Fresh.objs}$, $\text{Fresh.roles}$, and $\text{Fresh.rules}$ below. These functions yield the set of Object types, the set of Roles, and the set of constraints, respectively, which are not part of the Object structure of a certain meta-model $\mathcal{M}_1$. Given the universe of Object types $\mathcal{O}$, the universe of Roles $\mathcal{R}$, and the universe of constraints $\mathcal{R}$, these functions are defined as:

$$\text{Fresh.objs}(\mathcal{M}_1) = \mathcal{O} \setminus \mathcal{O}_1$$
$$\text{Fresh.roles}(\mathcal{M}_1) = \mathcal{R} \setminus \mathcal{R}_1$$

Now that we have provided an introduction to the relational algebra, we define each of the relational operators individually.

**Union**

The relational operator $\text{union}$ unites the populations of two relational expressions. To put it formally: suppose that $r$ and $s$ are two compatible relational expressions ($\text{Schema}(r) = \text{Schema}(s)$), then $r \cup s$ is a relational expression defined by:

- $\text{Schema}(r \cup s) = \text{Schema}(r)$
- $\text{Pop}(r \cup s) = \text{Pop}(r) \cup \text{Pop}(s)$

**Difference**

The relational operator $\text{difference}$ excludes the instances in a certain predefined relational expression which also occur in the population of another predefined relational expression. To put it formally: suppose that $r$ and $s$ are two compatible relational expressions, then $r \setminus s$ is a relational expression defined by:

- $\text{Schema}(r \setminus s) = \text{Schema}(r)$
- $\text{Pop}(r \setminus s) = \text{Pop}(r) \setminus \text{Pop}(s)$

**Selection**

The relational operator $\text{selection}$ applies a selection formula on a relational expression. The selection results in excluding those instances from the population of a relational expression which do not satisfy the selection formula.
We first define what we consider to be a selection formula. Subsequently, we present the definition of the selection. Finally, we present an example.

Selection formulas are taken from the domain of Role expressions $\mathcal{R}_{xp}$. This set is inductively defined as the smallest set of Role expressions satisfying the following rules:

1. A comparison of Roles is a Role expression, $p, q \in \mathcal{P} \Rightarrow p \theta q \in \mathcal{R}_{xp}$, where comparison operator $\theta \in \{<,=\}$

2. A comparison of a Role to a value is a Role expression, $p \in \mathcal{P}, c \in \cup \mathcal{D} \Rightarrow p \theta c \in \mathcal{R}_{xp}$.

3. If $x_1$ and $x_2$ are Role expressions, then $x_1 \land x_2$, $x_1 \lor x_2$, and $\neg x_1$ are also Role expressions:
   - $x_1 \in \mathcal{R}_{xp} \land x_2 \in \mathcal{R}_{xp} \Rightarrow x_1 \land x_2 \in \mathcal{R}_{xp}$
   - $x_1 \in \mathcal{R}_{xp} \land x_2 \in \mathcal{R}_{xp} \Rightarrow x_1 \lor x_2 \in \mathcal{R}_{xp}$
   - $x_1 \in \mathcal{R}_{xp} \Rightarrow \neg x_1 \in \mathcal{R}_{xp}$

If $r$ is a relational expression, and $F$ is a selection formula, then the relational expression $\sigma_{F}(r)$ is defined as:

- $\text{Schema}(\sigma_{F}(r)) = \text{Schema}(r)$
- $\text{Pop}(\sigma_{F}(r)) = \{t \in \text{Pop}(r) \mid F(t)\}$

<table>
<thead>
<tr>
<th>Person</th>
<th>$\sigma_{F}((\text{Person})_{\text{Year-of-study} \leq \text{Year-of-birth}})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Year-of-birth</td>
</tr>
<tr>
<td>B. Smith</td>
<td>1953</td>
</tr>
<tr>
<td>X. Li</td>
<td>1961</td>
</tr>
<tr>
<td>S. Lopez</td>
<td>1961</td>
</tr>
<tr>
<td>B. Smith</td>
<td>1948</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Year-of-birth</th>
<th>Year-of-study</th>
</tr>
</thead>
<tbody>
<tr>
<td>X. Li</td>
<td>1961</td>
<td>1961</td>
</tr>
</tbody>
</table>

Figure 7.3: Application of the selection

In figure 7.3, we illustrate the application of the selection on a Relationship Person, consisting of the Roles Name, Year-of-birth, and Year-of-study. In this example, the - for practical purposes somewhat obscure - selection formula $F$ is: Year-of-study $\leq$ Year-of-birth.
Projection

The relational operator projection maps a number of Roles from a relational expression onto a number of new Roles. First, the projection is defined formally. Subsequently, we provide an example.

Suppose that $r$ is a relational expression, that $p_j$ are some Roles in $r$, that $q_i$ are new Roles ($q_i \in \text{Fresh.roles} (\mathcal{M}, \mathcal{M}_1)$), which are all different, and that $\xi : q_i \rightarrow p_j$ is a function defining for each new Role one of the existing Roles. We then define the relational expression which results from applying the projection as:

- Schema($\pi_\xi (r)$) = dom($\xi$)
- Pop($\pi_\xi (r)$) = \{t | \exists s \in \text{Pop}(r) \forall 1 \leq i \leq |\text{dom}(\xi)| [t(q_i) = s(\xi(q_i))]\}

We use $\pi_{p_1, \ldots, p_n}(r)$ as a shorthand for $\pi_{p_1 \cdot p_2 \cdots p_n}(r)$. If $\tau$ is a set of Roles, then the notation $\pi_\tau (r)$ is also used.

As a result of applying a projection, the set of Roles $\mathcal{P}_1$ in the meta-model is united with the new Roles $q_i$, and the set of Relationships $\mathcal{F}_1$ is extended with the new Object type $\{q_i \in \text{Fresh.objs}(\mathcal{M}, \mathcal{M}_1)\}$. Each of the new Roles $q_i$ is assigned the base Object type of the corresponding Role $p_j$. Hence, the relation Base$_1$ is extended with the set $\{q_i : \text{Base}(\xi(q_i)) | 1 \leq i \leq |\text{dom}(\xi)|\}$.

<table>
<thead>
<tr>
<th>Person</th>
<th>$\pi_\xi (\text{Person})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Year-of-birth</td>
</tr>
<tr>
<td>B. Smith</td>
<td>1953</td>
</tr>
<tr>
<td>X. Li</td>
<td>1961</td>
</tr>
<tr>
<td>S. Lopez</td>
<td>1961</td>
</tr>
<tr>
<td>B. Smith</td>
<td>1948</td>
</tr>
<tr>
<td>Nm</td>
<td>Yos1</td>
</tr>
<tr>
<td>B. Smith</td>
<td>1973</td>
</tr>
<tr>
<td>X. Li</td>
<td>1961</td>
</tr>
<tr>
<td>S. Lopez</td>
<td>1981</td>
</tr>
</tbody>
</table>

Figure 7.4: Application of the projection

Figure 7.4 presents an example of the application of the projection on the Relationship Person. The projection results in a relational expression with Roles Nm, Yos1, and Yos2, according to the mapping $\xi = \{Nm : Name, Yos1 : Year-of-study, Yos2 : Year-of-study\}$. Note that this example leads to Role duplication since the new Roles Yos1 and Yos2 are both mapped onto the Role Year-of-study.
Extension

The relational operator \textit{extension} extends a relational expression \( r \) with a new Role \( p \in \text{Fresh.roles}(\mathcal{MM}_1) \). Role \( p \) has an existing base Object type \( o \). Each instance of the schema of \( r \) is then extended with a function from the new Role to its new value. The value of the new Role is determined by a predefined function \( f \). \( f \) calculates a value, based upon the values of the other Roles in the relational expression: \( f : \Omega^* \rightarrow \Omega \). The resulting value should be element of the domain of \( o \). Formally, the relational operator extension is defined as:

\begin{itemize}
  \item \( \text{Schema}(\chi_{p,f,o}(r)) = r \cup \{p\} \)
  \item \( \text{Pop}(\chi_{p,f,o}(r)) = \{ t \mid t[\text{Schema}(r)] \in \text{Pop}(r) \land t(p) = f(t[\text{Schema}(r)]) \} \)
\end{itemize}

Applying the extension operator results in extending the set of Roles \( \mathcal{P}_1 \) with the new Role \( p \). Furthermore, the set of Relationships is extended with the Relationship \( r \cup \{p\} \). The relation Base$_1$ is extended with one new element: \( \{p : o\} \).

\begin{table}[h]
\begin{center}
\begin{tabular}{|c|c|c|}
\hline
Name & Year-of-birth & Year-of-study \\
\hline
B. Smith & 1953 & 1973 \\
X. Li & 1961 & 1961 \\
S. Lopez & 1961 & 1981 \\
B. Smith & 1948 & 1973 \\
\hline
\end{tabular}
\end{center}
\end{table}

\begin{table}[h]
\begin{center}
\begin{tabular}{|c|c|c|c|}
\hline
Name & Year-of-birth & Year-of-
\begin{center}
\end{center}
\end{center}
\end{table}

\begin{table}[h]
\begin{center}
\begin{tabular}{|c|c|c|c|}
\hline
Name & Year-of-
\begin{center}
\end{center}
\end{table}

\begin{table}[h]
\begin{center}
\begin{tabular}{|c|c|c|c|}
\hline
Name & Year-of-
\begin{center}
\end{center}
\end{table}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure7_5.png}
\caption{Application of the extension}
\end{figure}

Figure 7.5 shows an example of the application of the extension operator on the same Relationship as used in the previous examples.

Join

The relational operator \textit{join} joins two relational expressions \( r \) and \( s \) resulting in one new relational expression \( r \Join s \), defined as:

\begin{itemize}
  \item \( \text{Schema}(r \Join s) = r \cup s \)
  \item \( \text{Pop}(r \Join s) = \{ t \mid t[\text{Schema}(r)] \in \text{Pop}(r) \land t[\text{Schema}(s)] \in \text{Pop}(s) \} \)
\end{itemize}
The join results in the addition of a new element to the set of Relationships: \( r \cup s \). Note that if Schema\((r)\) and Schema\((s)\) do not have any Role in common, the join operator is equal to the cartesian product.

![Figure 7.6: Application of the join](image)

Figure 7.6 shows an example of the application of the join operator on the Relationships *Transaction* and *Person*, *Name* being a common Role in these two Relationships.

**Unnest**

The relational operator *unnest* flattens nested Relationships, i.e. Relationships in which (at least) one of the Roles has a Relationship as its base Object type. Formally, let \( r \) be a relational expression, \( p \) a Role in Schema\((r)\), \( S \) the remaining set of Roles \( (S = \text{Schema}(r) \setminus \{p\}) \), and \( f \) a Relationship which is the base Object type of Role \( p \) (Base\((p) = f\)). We then define the relational expression resulting from applying the unnest operator as:

- \( \text{Schema}(\mu_{p,f}(r)) = f \cup S \)
- \( \text{Pop}(\mu_{p,f}(r)) = \{ t \cup s[S] \mid t \in \text{Pop}(f) \land s \in \text{Pop}(r) \land s(p) = t \} \)
As a result of applying the unnest operator, the set of Relationships is extended with \( f \cup S \).

In figure 7.7, we illustrate the application of the unnest operator on a Relationship \( \text{Transaction-price} \), in which the Role \( \text{has-Transaction} \) has the Relationship \( \text{Transaction} \) as its base Object type.

Note that hypothetical cases exist, in which Relationship \( f \) is \textit{not} the base Object type of Role \( p \), but is \textit{type related} with the base Object type of Role \( p \) instead. Consider for example figure 7.8, taken from [Hofstede, 1993]. It should be clear that if we apply the unnest, Relationship \( f \) has to be specified instead of base Object type \( C \): \( \mu_{r,f}(g) \). For this reason, it is necessary to specify \textit{both} Role \( p \) and Relationship \( f \) as parameters of the unnest. [Hofstede, 1993] provides a complete treatment of cases in which Object types are considered to be type related.

Another rare case is shown in figure 7.9. The result of unnesting Relationship \( g \) twice, via Role \( r \) and Role \( s \), is not defined. We refer to [Hofstede, 1993] who defines a strong unnest operator to handle such hypothetical cases.
Nest

The relational operator *nest* is the inverse operator of the unnest: it nests a Relationship, resulting in a relational expression. One of the Roles in the schema of the relational expression will have a Relationship as its base Object type. Defining the relational expression formally, let \( r \) be a relational expression, \( \tau \) a non-empty subset of Roles of the schema of \( r \) (\( \tau \subset \text{Schema}(r), \tau \neq \emptyset \)), \( S \) the set of remaining Roles (\( S = \text{Schema}(r) \setminus \tau \)), and \( p \) a new Role (\( p \in \text{Fresh}\_\text{roles}(MMI) \)). The schema of the relational expression \( \rho_{r,p}(r) \) contains the Roles \( S \cup \{p\} \). The Roles \( \tau \) constitute a new Relationship which is the base Object type of Role \( p \). We define the relational expression resulting from applying the nest operator as:

- \( \text{Schema}(\rho_{r,p}(r)) = S \cup \{p\} \)
Towards effective information modelling support

\begin{itemize}
\item \( \text{Pop}(\rho_{S,t}(\tau)) = \{ t \mid t[S] \in \text{Pop}(\Pi_S(\tau)) \land \text{Pop}(\mu_{p,t}(S \cup \{p\})) = \text{Pop}(\tau) \} \)
\end{itemize}

From this definition it should be clear that the set of Roles \( \mathcal{P}_1 \) and the set of Relationships \( \mathcal{F}_1 \) are extended as a result of applying the nest operator, respectively to \( \mathcal{P}_1 \cup \{p\} \) and \( \mathcal{F}_1 \cup \{S \cup \{p\}\} \). The nest operator, however, changes meta-model \( M, M_1 \) in more respects. \( \tau \) is added to \( \mathcal{F}_1 \) as another Relationship. Furthermore, the relation \( \text{Base}_1 \) is united with \( \{p : \tau\} \).

![Diagram showing nesting of a Relationship](image)

**Figure 7.10:** Abstract example of nesting a Relationship, which shows both the schemata and the populations involved.

Figure 7.10 presents an abstract example of nesting a Relationship. In this figure, Relationship \( f_1 \) has been nested \( (\rho_{(s,t),p}(f_1)) \). The schema of the resulting relational expression consists of Role \( p \), together with the set of remaining Roles \( \text{Schema}(f_1) \setminus \{s, t\} \).
7.4 Basic transformations

Basic transformations of a meta-model result in addition of an instance to, or deletion of an instance from one of the components of its Object structure, or in addition of a constraint to, or deletion of a constraint from its set of constraints. In this section, we define the syntax and the semantics of each basic transformation. As part of the definition of the semantics of a basic transformation, we always present the way in which a meta-model is transformed itself, and the way in which the corresponding population is transformed. We first discuss basic transformations involving the Object structure of a meta-model. Then, we discuss basic transformations involving the set of constraints of a meta-model.

For the following part of this chapter, we adopt what we call the restricted domain assumption. This assumption has the following three implications. Given two arbitrary sets $A$ and $B$, and given a relation $R \subseteq A \times B$, be $R$ a function or not, we assume that removing an element $a_i$ from set $A$ results also in the removal of all elements $(a_i, b)$ from relation $R$. If a set has a strict and total partition, then we assume that the removal of an element from that set results also in removal of that element from the appropriate subset. Given two arbitrary sets $A$ and $B$ with the property that $A$ is a subset of $B$, $A \subseteq B$, we assume that removing an element from $A$ results in removing that element from $B$.

7.4.1 Basic Object transformations

Basic Object transformations manipulate the Object structure of a meta-model. This section presents the syntax and the semantics of the various kinds of basic Object transformations.

7.4.1.1 Syntax of basic Object transformations

Basic Object transformations transform meta-models: $\mathcal{M}_1 \to \mathcal{M}_2$. They are listed in table 7.4.

Given a specific meta-model $\mathcal{M}_1$, the following parameters are used in table 7.4: $fo \in \text{Fresh.objs}(\mathcal{M}_1), fr \in \text{Fresh.roles}(\mathcal{M}_1), o, p \in \mathcal{O}_1, l, m \in \mathcal{L}_1, e \in \mathcal{E}_1, r \in \mathcal{P}_1, g \in \mathcal{G}_1, s \in S_1, c \in C_1, \omega \subseteq \mathcal{O}_1, f \in \mathcal{F}_1, \xi, \eta : \text{Fresh.roles}(\mathcal{M}_1) \to \mathcal{P}_1, b : \text{Fresh.roles}(\mathcal{M}_1) \to \mathcal{O}_1, d \in D, v \in d, \kappa \in Rxp, k : \Omega^* \to \Omega$.

7.4.1.2 Semantics of basic Object transformations

For each of the basic Object transformations, the semantics is defined. As pointed out in section 7.2.1, an Object structure consists basically of a set of Object types and a set of Roles. Object types and Roles are associated by six relations ($\text{Base}_1, \text{Elt}_1, \text{Schema}_1, \text{Spec}_1, \text{Gen}_1, \text{Dom}_1$).

In this section, we first define the semantics of transformations which manipulate the set of Object types in an Object structure. We then define the semantics of the transformations which manipulate the set of Roles in an Object structure. We finally define the semantics of transformations which involve one of the six above relations.
<table>
<thead>
<tr>
<th>Add.lab_{fo,d}</th>
<th>Add Label type</th>
<th>Del.lab_{t}</th>
<th>Delete Label type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add.ent_{fo}</td>
<td>Add Entity type</td>
<td>Del.ent_{e}</td>
<td>Delete Entity type</td>
</tr>
<tr>
<td>Add.set_{fo,o}</td>
<td>Add Set type</td>
<td>Del.set_{g}</td>
<td>Delete Set type</td>
</tr>
<tr>
<td>Add.seq_{fo,o}</td>
<td>Add Sequence type</td>
<td>Del.seq_{a}</td>
<td>Delete Sequence type</td>
</tr>
<tr>
<td>Add.sch_{fo,o}</td>
<td>Add Schema type</td>
<td>Del.sch_{c}</td>
<td>Delete Schema type</td>
</tr>
<tr>
<td>Add.rel_{fo,b}</td>
<td>Add Relationship</td>
<td>Del.rel_{f}</td>
<td>Delete Relationship</td>
</tr>
<tr>
<td>Uni.rel_{k,e}</td>
<td>Unite Relationship</td>
<td>Sel.rel_{k,e}</td>
<td>Select Relationship</td>
</tr>
<tr>
<td>Join.rel_{e}</td>
<td>Join Relationship</td>
<td>Proj.rel_{e}</td>
<td>Project Relationship</td>
</tr>
<tr>
<td>Unn.rel_{k,r,f}</td>
<td>Unnest Relationship</td>
<td>Nest.rel_{k,r,f}</td>
<td>Nest Relationship</td>
</tr>
<tr>
<td>Add.role_{f,r,k,o}</td>
<td>Add Role</td>
<td>Del.role_{r}</td>
<td>Delete Role</td>
</tr>
<tr>
<td>Add.spec_{fo,p}</td>
<td>Add Specialisation</td>
<td>Del.spec_{o,p}</td>
<td>Delete Generalisation</td>
</tr>
<tr>
<td>Add.gen_{o,p}</td>
<td>Add Generalisation</td>
<td>Del.val_{l,v}</td>
<td>Delete value</td>
</tr>
<tr>
<td>Add.val_{l,v}</td>
<td>Add value</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Uni.lab_{l,m,fo}</td>
<td>Unite Label type</td>
<td>Restr.lab_{l,d}</td>
<td>Restrict Label type</td>
</tr>
</tbody>
</table>

Table 7.4: Basic object transformations

Manipulate Object type

Object types may be added to or deleted from an Object structure. Since the set of Object types $O_1$ is partitioned into six subsets, we distinguish six basic Object transformations for adding an Object type, and six basic Object transformations for deleting an Object type.

The **addition** of a Label type or an Entity type to an Object structure results in the extension of the sets $L_1$ and $E_1$. The addition of a Label type may also result in the extension of the set $D_1$, since the instances in the new Label type may take their values from a new concrete domain. The addition of a Set type or a Sequence type to an Object structure leads to the extension of the sets $G_1$ and $S_1$. It is only possible to extend an Object structure with a Set type or a Sequence type if an existing Object type is specified to be the Element of that Set type or Sequence type. The analogon holds for the addition of a Schema type to the set of existing Schema types ($C_1$). Finally, the addition of a Relationship to an Object structure leads to the extension of the set $F_1$. Since a Relationship is a set of Roles by definition, $P_1$ is also extended. For each of the new Roles, the base Object type has to be specified.

Table 7.5 shows for each of the basic Object transformations which meta-model components are transformed, and how they are transformed. Naturally, the Object type added (in table 7.5 $l_1$, $e_1$, $s_1$, and $r_1$, respectively) is not already part of the meta-model: $l_1, e_1, s_1, r_1 \in \text{Fresh.objs}(M,M_1)$. The analogon holds for the Roles in Relationship $r_1: r_1 \subseteq \text{Fresh.roles}(M,M_1)$. In the case of the addition of a Relationship, $b_1$ defines the base Object types of the Roles in the Relationship to be added: $b_1 : r_1 \rightarrow O_1$. Finally, the Object types $o_i$ are part of the existing meta-model: $o_i \in O_1$.

In all these cases, the **population** of the Object type to be added is defined as the empty set.

186
### Table 7.5: Semantics of basic Object transformations (add Object type)

<table>
<thead>
<tr>
<th>Transformation</th>
<th>Changed components</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add.lab$_1$,d$_1$($MM_1$)</td>
<td>$L_1$</td>
<td>$L_1 \cup {l_1}$</td>
</tr>
<tr>
<td></td>
<td>$\text{Dom}_1$</td>
<td>$\text{Dom}_1 \cup {l_1 : d_1}$</td>
</tr>
<tr>
<td></td>
<td>$D_1$</td>
<td>$D_1 \cup {d_1}$</td>
</tr>
<tr>
<td>Add.ent$_{e_1}$($MM_1$)</td>
<td>$E_1$</td>
<td>$E_1 \cup {e_1}$</td>
</tr>
<tr>
<td>Add.set$_{s_1,o_1}$($MM_1$)</td>
<td>$G_1$</td>
<td>$G_1 \cup {s_1}$</td>
</tr>
<tr>
<td></td>
<td>$\text{Elt}_1$</td>
<td>$\text{Elt}_1 \cup {s_1 : o_1}$</td>
</tr>
<tr>
<td>Add.seq$_{s_1,o_1}$($MM_1$)</td>
<td>$S_1$</td>
<td>$S_1 \cup {s_1}$</td>
</tr>
<tr>
<td></td>
<td>$\text{Elt}_1$</td>
<td>$\text{Elt}_1 \cup {s_1 : o_1}$</td>
</tr>
<tr>
<td>Add.sch$_{s_1,o_1,\ldots,o_n}$($MM_1$)</td>
<td>$C_1$</td>
<td>$C_1 \cup {s_1}$</td>
</tr>
<tr>
<td></td>
<td>Schema$_1$</td>
<td>Schema$_1 \cup {s_1 : o}</td>
</tr>
<tr>
<td>Add.rel$_{r_1,b_1}$($MM_1$)</td>
<td>$F_1$</td>
<td>$F_1 \cup {r_1}$</td>
</tr>
<tr>
<td></td>
<td>$P_1$</td>
<td>$P_1 \cup r_1$</td>
</tr>
<tr>
<td></td>
<td>Base$_1$</td>
<td>Base$_1 \cup b_1$</td>
</tr>
</tbody>
</table>

### Table 7.6: Semantics of basic Object transformations (delete Object type)

<table>
<thead>
<tr>
<th>Transformation</th>
<th>Changed components</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>Del.lab$_1$($MM_1$)</td>
<td>$L_1$</td>
<td>$L_1 \setminus {l_1}$</td>
</tr>
<tr>
<td>Del.ent$_{e_1}$($MM_1$)</td>
<td>$E_1$</td>
<td>$E_1 \setminus {e_1}$</td>
</tr>
<tr>
<td>Del.set$_{s_1}$($MM_1$)</td>
<td>$G_1$</td>
<td>$G_1 \setminus {s_1}$</td>
</tr>
<tr>
<td>Del.seq$_{s_1}$($MM_1$)</td>
<td>$S_1$</td>
<td>$S_1 \setminus {s_1}$</td>
</tr>
<tr>
<td>Del.sch$_{s_1}$($MM_1$)</td>
<td>$C_1$</td>
<td>$C_1 \setminus {s_1}$</td>
</tr>
<tr>
<td>Del.rel$_{r_1}$($MM_1$)</td>
<td>$F_1$</td>
<td>$F_1 \setminus {r_1}$</td>
</tr>
<tr>
<td></td>
<td>$P_1$</td>
<td>$P_1 \setminus r_1$</td>
</tr>
</tbody>
</table>

Naturally, deleting Object types from an Object structure results in the removal of the Object type specified, see table 7.6. Due to the restricted domain assumption, we do not need to specify the deletion of elements from $\text{Dom}_1$, $\text{Elt}_1$, Schema$_1$, and Base$_1$ in this table explicitly. For the same reason, we do not need to define the population changes either.

Deleting Object types may, however, result in a non-valid Object structure. We illustrate this in figure 7.11. In the leftside meta-model, the Object types $C$, $E$, $F$, $b$, $c$, and $d$ depend on Object type $B$. Therefore, deleting Object type $B$ must result in deleting all dependent Object types, Roles, and constraints as well, to keep a valid Object structure. We specify this in table 7.7, using the notion of dependencies in meta-models from section 7.2.3.

Now that we have discussed adding elements to the set of Object types and deleting elements from the set of Object types, we turn our attention to the third and last category of
Figure 7.11: Deleting an Object type from an abstract meta-model

<table>
<thead>
<tr>
<th>Transformation</th>
<th>Changed components</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>Del.lab_{o_1}(M,M_1), Del.seq_{o_1}(M,M_1), Del.ent_{o_1}(M,M_1), Del.set_{o_1}(M,M_1), Del.rel_{o_1}(M,M_1)</td>
<td>O_1, P_1, R_1</td>
<td>O_1 \setminus \text{Dep.objects}(o_1), P_1 \setminus \text{Dep.roles}(o_1), R_1 \setminus \text{Dep.rules}(o_1)</td>
</tr>
</tbody>
</table>

Table 7.7: Semantics of basic Object transformations, continued (delete Object type)

Object type manipulations: manipulating Relationships. We pay particular attention to Relationships, since Relationships can be considered to be sets themselves (they are sets of Roles). We discuss the manipulation of individual Roles in a Relationship later in this chapter.

In this section, we focus on manipulations of Relationships as a whole. Based upon the relational algebra of section 7.3, we discuss six basic transformations. Each of these basic transformations results in the addition of one or two new Relationships to the meta-model at hand, without removing the Relationships upon which the construction of the new Relationships is based. Thus, one is able to use one and the same Relationship for several transformations.

We refer to the existing Relationships as \( f_1 \) and \( f_2 \). New Relationships are referred to as \( h_1 \) and \( h_2 \). Naturally, the new Relationships are not already part of the meta-model: \( h_1 \),
$h_2 \in \text{Fresh.objs}(\mathcal{M}_{M1})$. $h_1$ and $h_2$ are sets of Roles which are neither assumed to be part of the meta-model: $h_1, h_2 \subseteq \text{Fresh.roles}(\mathcal{M}_{M1})$.

We distinguish the following basic transformations, as far as the manipulation of Relationships is concerned:

- the union $\text{Uni.rel}_{\xi_1, \xi_2}$ of two Relationships $f_1$ and $f_2$, resulting in one new Relationship $h_1$. The bijections $\xi_1, \xi_2$ relate the Roles in the new Relationship $h_1$ to the Roles in the existing Relationships $f_1$ and $f_2$ ($\xi_1 : h_1 \rightarrow f_1$, $\xi_2 : h_1 \rightarrow f_2$).

- the selection $\text{Sel.rel}_{\xi_1, \kappa_1}$ of a new Relationship $h_1$. The population of $h_1$ is defined to be a subset of the population of an existing Relationship $f_1$, according to some decision criterion $\kappa_1$. The bijection $\xi_1$ relates the Roles in $h_1$ to the Roles in $f_1$ ($\xi_1 : h_1 \rightarrow f_1$).

- the projection $\text{Proj.rel}_{\xi_1}$ of some Roles $\tau_1$ in an existing Relationship $f_1$ onto a new Relationship $h_1$. The bijection $\xi_1$ relates the Roles in $h_1$ to those in $\tau_1$ ($\xi_1 : h_1 \rightarrow \tau_1$). Naturally, it is required that a Relationship exists in the current meta-model which contains the specified Roles ($\exists f_1 \in \mathcal{F} \mid \tau \subset f_1$).

- the join $\text{Join.rel}_{\xi_1}$ of two Relationships $f_1$ and $f_2$ to one new Relationship $h_1$. The bijection $\xi_1$ relates the Roles in the new Relationship $h_1$ to the Roles in the relational expression which results from applying the relational operator join on the old Relationships $f_1$ and $f_2$ ($\xi_1 : h_1 \rightarrow f_1 \Join f_2$).

- the unnest $\text{Unn.rel}_{\xi_1, \tau_1, f_2}$, flattening a Relationship $f_1$ in which Role $\tau_1$ has a base Object type which is type related with another Relationship $f_2$ itself, resulting in a new Relationship $h_1$. The bijection $\xi_1$ relates the Roles in $h_1$ to Roles in $f_1 \cup f_2$ ($\xi_1 : h_1 \rightarrow f_1 \cup f_2$).

- the nest $\text{Nest.rel}_{\xi_1, \xi_2, \tau_1}$, nesting one Relationship $f_1$ into two Relationships $h_1$ and $h_2$. Role $\tau_1$ in $h_2$ has Relationship $h_1$ as its base Object type. We assume that $\tau_1$ is a subset of Roles of $f_1$ ($\exists f_1 \in \mathcal{F} \mid \tau \subset f_1$). The bijection $\xi_1$ relates the Roles in $h_1$ to Roles in $\tau_1$ ($\xi_1 : h_1 \rightarrow \tau_1$). The bijection $\xi_2$ relates the Roles in $h_2$ (with the exception of $\tau_1$) to the remaining Roles in $f_1$ ($\xi_2 : h_2 \setminus \tau_1 \rightarrow f_1 \setminus \tau_1$). Role $\tau_1$ is related to itself ($\xi_2 : \tau_1 \rightarrow \tau_1$).

For applying the basic transformation $\text{Uni.rel}$ on two Relationships $f_1$ and $f_2$, it is required that a Role exists in $f_2$ for each Role in $f_1$ with the same base Object type, and vice versa. In figure 7.12 for example, the Roles $v$, $w$, and $x$ in Relationship $a$ have the same base Object types as the Roles $s$, $t$, and $u$ in the relationship $b$, namely $A$, $A$, and $C$. To put it formally, it is required that a bijection $\xi : f_1 \rightarrow f_2$ exists between the two Relationships to be joined, such that $\forall p \in f_1 \mid \text{Base}(p) = \text{Base}(\xi(p))$.

These basic Object transformations result in the addition of one or more new Relationships to the meta-model. For those components of the meta-model which are changed by these basic Object transformations, we present a formal definition of these changes in table 7.8.
Figure 7.12: Uniting two Relationships in an abstract meta-model

<table>
<thead>
<tr>
<th>Transformation</th>
<th>Changed components</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uni_rel_{ξ_1,ξ_2}, Sel_rel_{ξ_1,η_1}, Proj_rel_{ξ_1}, Join_rel_{ξ_1}, and Unn_rel_{ξ_1,r_1,f_1}</td>
<td>( \mathcal{F}_1 ) ( \mathcal{P}_1 ) ( \text{Base}_1 )</td>
<td>( \mathcal{F}_1 \cup { \text{dom}(\xi_1) } ) ( \mathcal{P}_1 \cup \text{dom}(\xi_1) ) ( \text{Base}_1 \cup { p : \text{Base}_1(\xi_1)(p) \mid p \in \text{dom}(\xi_1) } )</td>
</tr>
<tr>
<td>Nest_rel_{ξ_1,ξ_2,r_1}</td>
<td>( \mathcal{F}_1 ) ( \mathcal{P}_1 ) ( \text{Base}_1 )</td>
<td>( \mathcal{F}_1 \cup { \text{dom}(\xi_1), \text{dom}(\xi_2) } ) ( \mathcal{P}_1 \cup \text{dom}(\xi_1) \cup \text{dom}(\xi_2) ) ( \text{Base}_1 \cup { p : \text{Base}_1(\xi_1)(p) \mid p \in \text{dom}(\xi_1) } \cup { p : \text{Base}_1(\xi_2)(p) \mid p \in \text{dom}(\xi_2) \setminus { r_1 } } \cup { r_1 : \text{dom}(\xi_1) } )</td>
</tr>
</tbody>
</table>

Table 7.8: Semantics of basic Object transformations, concerning the manipulation of Relationships.

To define the population of the new Relationships, resulting from applying one of the six basic transformation we have discussed in this section, we use the relational algebra of section 7.3, see table 7.9.

**Manipulate Role**

Roles may be added to or deleted from a Relationship in an Object structure. The addition or deletion of a Role leads to a manipulation of the set of Roles \( \mathcal{P}_1 \). In the case of addition, it is required that the base Object type is also specified. As a consequence of the addition or deletion of a Role, a new Relationship is constructed. Therefore, the set \( \mathcal{F}_1 \) is manipulated as well. Note, however, that in the case of deletion of a Role from a unary Relationship, the construction of a new Relationship is not allowed because the empty set is not a valid
Table 7.9: Definition of the population of the new Relationships, resulting from applying one of the six basic Object transformations, concerning the manipulation of Relationships.

The semantics of these two basic Object transformations is presented in table 7.10. The definition assumes Role $p_1$ not to be part of the meta-model: $p_1 \in \text{Fresh.} \text{roles}(\mathcal{M, M}_1)$. The Object types $f_1$ and $o_1$ and Role $q_1$ on the other hand are assumed to be part of the meta-model: $o_1 \in \mathcal{O}_1 \land q_1 \in \mathcal{P}_1$. Finally, it is assumed that no single Object type, Role, or constraint depends on the Relationship to be changed.

Table 7.10: Semantics of basic Object transformations (manipulate Role)

Both the addition and the deletion of a Role result in a new Relationship. The function $k_1$ is used to determine the value of the new Role in the population of $f_1$. Formally, the population of a Relationship in the case of addition of a Role is defined as:

$$\text{Pop}(f_1 \cup \{p_1\}) = \text{Pop}(\chi_{p_1,k_1,o_1}(f_1))$$

To define the population of a Relationship in the case of the deletion of a Role, we project all Roles of the Relationship involved on themselves, except the Role to be deleted:

$$\text{Pop}(\text{Rel}(q_1) \setminus \{q_1\}) = \text{Pop}(\pi_{\text{Rel}(q_1) \setminus \{q_1\}}(\text{Rel}(q_1)))$$
Manipulate relations

The relations Base₁, Elt₁, Schema₁, Spec₁, Gen₁, and Dom₁ between existing Object types or Roles may be changed in some cases only. Base₁, Elt₁, and Schema₁ are total functions, which are changed automatically in the case of the manipulation of a Relationship, a Set type or Sequence type, or a Schema type, respectively. These transformations have been discussed. We therefore focus on the manipulation of the relations Spec₁, Gen₁, and Dom₁ in this section.

If two Object types participate in a specialisation hierarchy, then their populations satisfy the Specialisation Rule, see section 7.2.2. Existing Object types should therefore not be added as Subtype of another Object type. The addition of a specialisation relation amounts to the introduction of a new Object type as Subtype of an existing Object type. A Subtype defining rule is required to define the population of the new Object type. Analogously, deleting a specialisation relation is the same transformation as deleting an Object type. As a consequence, manipulating specialisation relations is realised only by the basic Object transformation Add.spec.

Generalisation relations on the other hand may be arbitrarily added or deleted, due to their nature. This is realised by applying the basic Object transformations Add.gen and Del.gen. Note however that the addition of a generalisation relation may lead to the violation of some of the graphical constraints in the meta-model. An example would be a Generalised object type playing a total role constraint.

Since the function Dom₁ involves sets of values, we discuss manipulations of these sets. Four basic transformations are distinguished:

- Add.val₁₁ : adding a value \( v₁ \) to the domain of a Label type \( l₁ \).
- Del.val₁₁ : deleting a value \( v₁ \) from the domain of Label type \( l₁ \).
- Restr.lab₁₁ : restricting the domain of a Label type \( l₁ \) to a domain \( d₁ \).
- `Uni.lab_{l_1, l_2, m_1}`: uniting the domains of two Label types \( l_1 \) and \( l_2 \) to the domain of a new Label type \( m_1 \).

Figure 7.13 illustrates the application of the basic transformation `Uni.lab`.

<table>
<thead>
<tr>
<th>Transformation</th>
<th>Changed components</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add.spec_{o_1, o_2}((\mathcal{M}, \mathcal{M}_1))</td>
<td>(O_1)</td>
<td>(O_1 \cup {o_3})</td>
</tr>
<tr>
<td>Spec</td>
<td>(Spec_1 \cup {(o_1, o_2)})</td>
<td></td>
</tr>
<tr>
<td>Add.gen_{o_1, o_2}((\mathcal{M}, \mathcal{M}_1))</td>
<td>(Gen_1)</td>
<td>(Gen_1 \cup {(o_1, o_2)})</td>
</tr>
<tr>
<td>Del.val_{i_1, v_1}((\mathcal{M}, \mathcal{M}_1))</td>
<td>(Dom_1)</td>
<td>(Dom_1 \uplus {l_1 : \text{Dom}(l_1) \cup {v_1}})</td>
</tr>
<tr>
<td>Del.val_{i_1, v_1}((\mathcal{M}, \mathcal{M}_1))</td>
<td>(Dom_1)</td>
<td>(Dom_1 \uplus {l_1 : \text{Dom}(l_1) \cup {v_1}})</td>
</tr>
<tr>
<td>Uni.lab_{l_1, l_2, m_1}((\mathcal{M}, \mathcal{M}_1))</td>
<td>(\mathcal{L}_1)</td>
<td>(\mathcal{L}_1 \cup {m_1}) (\setminus) ({l_1, l_2})</td>
</tr>
<tr>
<td>Dom_1</td>
<td>(Dom_1 \uplus {m_1 : (\text{Dom}(l_1) \cup \text{Dom}(l_2))})</td>
<td></td>
</tr>
<tr>
<td>Base_1</td>
<td>(Base_1 \uplus {r : m_1 \mid r : l_1 \in \text{Base}_1 \lor r : l_2 \in \text{Base}_1})</td>
<td></td>
</tr>
<tr>
<td>Restr.lab_{l_1, d_1}((\mathcal{M}, \mathcal{M}_1))</td>
<td>(Dom_1)</td>
<td>(Dom_1 \uplus {l_1 : d_1})</td>
</tr>
</tbody>
</table>

Table 7.11: Semantics of basic Object transformations (manipulate relation)

Table 7.11 defines the semantics of these basic Object transformations. Most parameters in this table are related to the existing meta-model: \(l_1, l_2 \in \mathcal{L}_1, o_1, o_2 \in O_1\), and \(d_1 \subseteq \text{Dom}(l_1)\). \(m_1\) is a new Label type and \(o_3\) is a new Object type: \(m_1, o_3 \in \text{Fresh objs}(\mathcal{M}, \mathcal{M}_1)\).

The population of Label type \(l_1\) resulting from applying the basic transformation `Add.val` is \(\text{Pop}(l_1) \cup \{v_1\}\). The population of Label type \(l_1\) resulting from applying the basic transformation `Del.val` is \(\text{Pop}(l_1) \setminus \{v_1\}\). The population of the Label type \(m_1\) which results from uniting the Label types \(l_1\) and \(l_2\) is defined as the union of the two separate populations: \(\text{Pop}(m_1) = \text{Pop}(l_1) \cup \text{Pop}(l_2)\). The population of the Label type \(l_1\) which results from applying the basic transformation `Restr.lab` is: \(\text{Pop}(m_1) = \text{Pop}(l_1) \cap d_1\).

### 7.4.2 Basic constraint transformations

This section defines both the syntax and the semantics of the basic constraint transformations.

#### 7.4.2.1 Syntax of basic constraints transformations

Basic constraint transformations add or delete a specific constraint to the set of constraints of a meta-model. Two basic constraint transformations are distinguished, one for addition, and one for deletion. They are listed in table 7.12. They transform meta-models: \(\mathcal{M}, \mathcal{M} \rightarrow \mathcal{M}, \mathcal{M}\). In table 7.12, parameter \(c\) is a new constraint, \(c \in \text{Fresh rules}(\mathcal{M}, \mathcal{M}_1)\), and parameter \(d\) is an existing constraint, \(d \in \mathcal{R}_1\).
7.4.2.2 Semantics of basic constraint transformations

The semantics of the basic constraint transformations are defined as shown in Table 7.13.

<table>
<thead>
<tr>
<th>Add_constr</th>
<th>Delete constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add_constr ((\mathcal{M}_1)) = (\langle OS_1, R_1 \cup {c_1}\rangle)</td>
<td></td>
</tr>
<tr>
<td>Del_constr ((\mathcal{M}_1)) = (\langle OS_1, R_1 \setminus {c_1}\rangle)</td>
<td></td>
</tr>
</tbody>
</table>

Table 7.13: Semantics of basic constraint transformations

In most cases, the addition of a constraint will have consequences for the population of the meta-model at hand. We do not explore these consequences, as they are situation specific: they are dependent on the constraint type as well as on the actual meta-model and its population.

7.5 Composed transformations

This section focuses on the composition of meta-model transformations, based upon the basic transformations of section 7.4. Since each basic transformation \(b : \mathcal{M} \rightarrow \mathcal{M}\), new transformations are easily composed by concatenating existing transformations. We use the operator \(\circ\) to this purpose. For example, the addition of two new Entity types \(e_1\) and \(e_2\) to a meta-model \(\mathcal{M}_1\) is denoted as: \(\text{Add} \text{ent}_{e_2} \circ \text{Add} \text{ent}_{e_1}(\mathcal{M},\mathcal{M}_1)\). Note that the order of the two basic Object transformations is from right to left. Therefore, Entity type \(e_1\) is added before Entity type \(e_2\) is added in this example.

We first pay attention to redefining the six basic transformations which manipulate Relationships. In the previous section, we have defined these basic transformations, which do not remove the Relationships which are used during the manipulation. As removal of these Relationships is often desirable in these cases (e.g. figure 7.12), we introduce six new composed transformations, one for each of the basic transformations mentioned. They are referred to as \(\text{Comp.Uni.rel}\), \(\text{Comp.Sel.rel}\), \(\text{Comp.Proj.rel}\), \(\text{Comp.Join.rel}\), \(\text{Comp.Unn.rel}\), and \(\text{Comp.Nest.rel}\).

The composed union operator \(\text{Comp.Uni.rel}\) is defined as the union of two Relationships according to two mappings \(\xi_1\) and \(\xi_2\), followed by the removal of the two Relationships used:

\[
\text{Comp.Uni.rel}_{\xi_1,\xi_2}(\mathcal{M},\mathcal{M}_1) \equiv \text{Del.rel}_{\text{ran}(\xi_2)} \circ \text{Del.rel}_{\text{ran}(\xi_1)} \circ \text{Uni.rel}_{\xi_1,\xi_2}(\mathcal{M},\mathcal{M}_1)
\]

The composed selection operator \(\text{Comp.Sel.rel}\) selects some instances from a Relationship, according to some decision criterion. Afterwards it removes the original Relationship:

\[
\text{Comp.Sel.rel}_{\xi_1,\tau_1}(\mathcal{M},\mathcal{M}_1) \equiv \text{Del.rel}_{\text{ran}(\xi_1)} \circ \text{Sel.rel}_{\xi_1,\tau_1}(\mathcal{M},\mathcal{M}_1)
\]
The composed projection operator \( \text{Comp.Proj.rel} \) projects a Relationships onto a new one, and removes the original Relationship afterwards:

\[
\text{Comp.Proj.rel}_\xi(M,M_1) \equiv \text{Del.rel}_{\text{ran}(\xi_1)} \circ \text{Proj.rel}_\xi(M,M_1)
\]

The composed join operator \( \text{Comp.Join.rel} \) joins two Relationships to a new one, and removes the two original ones:

\[
\text{Comp.Join.rel}_\xi(M,M_1) \equiv \text{Del.rel}_{\text{ran}(\xi_1)} \circ \text{Del.rel}_{\text{ran}(\xi_1)^2} \circ \text{Join.rel}_\xi(M,M_1)
\]

The composed unnest operator \( \text{Comp.Unn.rel} \) is defined analogously:

\[
\text{Comp.Unn.rel}_{\xi_1,r_1,f_2}(M,M_1) \equiv \text{Del.rel}_{\text{ran}(\xi_1)} \circ \text{Del.rel}_{\text{ran}(\xi_1)} \circ \text{Unn.rel}_{\xi_1,r_1,f_2}(M,M_1)
\]

The composed nest operator \( \text{Comp.Nest.rel} \) is defined as:

\[
\text{Comp.Nest.rel}_{\xi_1,\xi_2,r_1}(M,M_1) \equiv \text{Del.rel}_{\text{ran}(\xi_1) \cup \text{ran}(\xi_2)} \circ \text{Nest.rel}_{\xi_1,\xi_2,r_1}(M,M_1)
\]

Now that we have defined composed transformations for the manipulation of Relationships, we direct our attention to other composed transformations. Clearly, the basic transformations presented in the previous section can be concatenated in numerous ways. Therefore, we focus on some characteristic examples found in the literature on schema transformations. Several publications are available on relations between schemata in general, see for example [D’Atri and Sacca, 1984], [Jajodia et al., 1983], and [Kobayashi, 1986]. Since we base ourselves upon PSM, which is an Object-role modelling technique, we are particularly interested in literature on schema transformations which are based upon an Object-role modelling technique. Given this restriction, literature is scarce. Examples of schema transformations in this context are found in [Oei et al., 1992] and [Halpin, 1989]. The former are based on the Predicator Model (see [Bommel et al., 1991]) as an underlying modelling technique, whereas the latter are based on NIAM. We discuss several examples from [Oei et al., 1992] and [Halpin, 1989], and express these using our basic transformations.

In [Oei et al., 1992], three particular examples of what they call relations between meta-models are distinguished, which are referred to as partitioning, restriction, and degeneration. We explain them individually. In addition to this, we show how they are expressed using our basic transformations.

The partitioning relation expresses the extension of a meta-model with a specialisation hierarchy, in which all subtypes are considered to be leafs. All subtypes are assumed to be involved in a subtype exclusion constraint and a subtype total constraint. Figure 7.14 shows an example of the partitioning relation. This example is expressed as:

\[
\text{Add.constr.sub.exclusion}(\{B,C\}) \circ \text{Add.constr.sub.total}(\{B,C\}) \circ \text{Add.spec}_{B,A} \circ \text{Add.spec}_{C,A}(M,M_1).
\]

The restriction relation expresses the addition of a constraint to a meta-model. The restriction is equal to the basic transformation \( \text{Add.constr} \).
The *degeneration* relation expresses the change of an existing specialisation hierarchy in a meta-model. *Weak degeneration* results in the removal of the pater familias of the specialisation hierarchy, whereas *strong degeneration* results in the removal of one of the Subtypes in a specialisation hierarchy. Figure 7.15 shows an example of the weak degeneration relation and the strong degeneration relation. Strong degeneration can be expressed using the basic transformation $\text{DelEnt}$. In the example of figure 7.15, strong degeneration is expressed as $\text{DelEnt}_B(M, \mathcal{M}_1)$. Note that it is not possible to assign a meaningful semantics to weak degeneration in the context of PSM, since the population of a subtype depends on the population of its supertype.

In [Halpin, 1989], a large variety of suitable and sometimes complex schema transformations are presented. For a representative majority of these schema transformations, we
present how they can be expressed using our basic transformations. The following schema transformations given in [Halpin, 1989] are dealt with, using Halpin's terminology: Relationship nesting and Relationship flattening, Relationship splitting and Relationship combining, Code introduction and Code removal, and Described object type introduction and Described object type removal.

The schema transformations Relationship nesting and Relationship flattening are the same as our composed transformations Comp.Nest.rel and Comp.Unn.rel. The schema transformation Relationship combining is the same as our composed transformations Comp.Join.rel. Relationship splitting is achieved by a double projection, followed by removing the original Relationship:

\[
\text{Split.rel}_{\xi_1, \xi_2}(\mathcal{M}, \mathcal{M}_1) \equiv \text{Del.rel}_{\text{Rel}(\text{ran}(\xi_1)), \circ \text{Proj.rel}_{\xi_2} \circ \text{Proj.rel}_{\xi_1}}(\mathcal{M}, \mathcal{M}_1)
\]

Figure 7.16: Example of Code introduction versus Code removal

The schema transformation Code introduction can be applied in cases with compatible Relationships. This leads to the introduction of a Label type which is used to classify the Relationships. Code removal results in a number of Relationships, which were originally classified by one Label type. To illustrate these schema transformations we use an example from [Halpin, 1989], see figure 7.16. In this example, the Label type Status-code is used to classify the Relationships between an employee and a project.

Code introduction is expressed by concatenating several transformations. Starting from the rightside Object structure in the example of figure 7.16, two Label types are added, W-code and S-code. The related domains are enumerated sets, \{working\} and \{supervising\}, each having one value. Both Relationships in the rightside Object structure are extended with one Role, wcode-of and code-of, respectively. The two added Label types are the base Object types of these new Roles. Subsequently, the two Label types are united to one new Label type. Finally, the two Relationships, being transformed into compatible ones, are united. Figure 7.17 illustrates Code introduction as a concatenation of transformations.

Formally, Code introduction is expressed in this example as

\[
\text{Comp.Uni.rel}_{\xi_1, \xi_2} \circ \text{Uni.lab}_{W\text{-code}, S\text{-code}, \text{Status-code}} \circ \text{Add.role}_{W, \text{wcode-of}, k_1, W\text{-code}}
\]
Figure 7.17: Code introduction as a concatenation of transformations

\[
\text{Add}_{\text{role}}_S \circ \text{scode-of} \circ \text{Add}_{\text{lab}} \circ \text{code-of} \circ \text{Add}_{\text{lab}} \circ \text{working}\left(\mathcal{M}, \mathcal{M}_1\right).
\]

In this composed transformation, \(\xi_1\) and \(\xi_2\) are mappings which relate the Roles in the final Relationship to the Roles in the two compatible Relationships:

\[
\begin{align*}
\xi_1 &= \{\text{has : works-on, on : is-worked-on-by, of : wcode-of}\}, \\
\xi_2 &= \{\text{has : supervises, on : is-supervised-by, of : scode-of}\}.
\end{align*}
\]

The functions \(k_1\) and \(k_2\) fill the added Roles with the values “working” and “supervising”, independent of the values of the other Roles in the instances of the Relationships \(W\) and \(S\).

It is apparent from the above that Code introduction involves a large number of parameters. These parameters are the Label type introduced, in the above example Status-code, and a
6-tuple

\((\text{Relationship, Role, Population function, Label type, Domain, Role mapping})\)

for each Relationship.

In the above example these 6-tuples are

\((W, \text{wcode-of}, k_1, \text{W-code}, \{\text{working}\}, \xi_1)\)

and

\((S, \text{scode-of}, k_2, \text{S-code}, \{\text{supervising}\}, \xi_2)\)

The large number of parameters involved provides an indication of the complexity of the composed transformation Code introduction \((\text{Add-code})\).

Naturally, Code removal is expressed as the inverse transformation. Code removal \((\text{Del-code})\) also involves a large number of parameters. The parameters are the Label type to be removed, a selection formula, and a 2-tuple

\((\text{Role, Role mapping})\)

for each Relationship.

Starting from the leftside Object structure in figure 7.16, Code removal is expressed as

\[
\text{Del-code}_{\text{Status-code}, \text{of}} = \text{working}, \text{(wcode-of}, \xi_1), \text{(scode-of}, \xi_2)(\mathcal{M}, \mathcal{M}_1) = \\
\text{Del.lab}_{\text{Status-code}} \circ \text{Del.role}_{\text{scode-of}} \circ \text{Del.role}_{\text{wcode-of}} \circ \text{Comp Sel.rel}_{\xi_1^{-1}, \xi_2^{-1}} \text{of} = \text{working}(\mathcal{M}, \mathcal{M}_1).
\]

Figure 7.18: Example of Described object type introduction versus Described object type removal

The last pair of schema transformations from [Halpin, 1989] concerns so-called Described object types. We use an example from [Halpin, 1989] to explain the introduction and removal
of Described object types. The upper Object structure in figure 7.18 shows a Person being described by a Gender-code. In the lower Object structure, a Described Object type is introduced, Gender, which in turn is described by a Gender-code.

Starting from the upper Object structure in figure 7.18, Described object type introduction is expressed by introducing an Entity type Gender, then by adding a new Role, say Gender-role, to the Relationship between Person and Gender-code with Gender as its Base object type, and with values which are determined according to a bijection between the values of the other Roles, and these new ones, and finally by projecting the new ternary Relationship onto two new binary ones, resulting in the lower Object structure. Formally, Described object type introduction is expressed in this example as

$$\text{Del}_\text{rel} \circ \text{Proj}_\text{rel}(\text{has-code:Gender-role, of:gender}) \circ \text{Proj}_\text{rel}(\text{has-gender:has, of:person:Gender-role}) \circ$$

$$\text{Add}_\text{role}_\text{k}(\text{Gender-role, k, Gender}) \circ \text{Add}_\text{ent}_\text{Gender}(\mathcal{M}, \mathcal{M}_1).$$

In this example, function \(k\) maps values of Gender-code onto values of Gender.

Naturally, Described object type removal is expressed as the inverse transformation. Starting from the lower Object structure in figure 7.18, Described object type removal is expressed formally as

$$\text{Split}_\text{rel}(\text{has, of}) \circ \text{Comp}_\text{Join}_\text{rel}(\text{has-gender, t, of:person } \cup \text{ has-code, of:gender})(\mathcal{M}, \mathcal{M}_1).$$

### 7.6 Genericity and equivalence of transformations

This section discusses for each of the basic transformations (section 7.4) and the composed transformations (section 7.5), whether its application leads to a more generic meta-model, to a more individualised meta-model, or to an equivalent meta-model. Recall from section 7.1 that we refer to meta-model \(\mathcal{M}, \mathcal{M}_1\) as being more generic than meta-model \(\mathcal{M}, \mathcal{M}_2\), if each population of meta-model \(\mathcal{M}, \mathcal{M}_2\) can be transformed to a population of meta-model \(\mathcal{M}, \mathcal{M}_1\) in a unequivocal way. Two meta-models \(\mathcal{M}, \mathcal{M}_1\) and \(\mathcal{M}, \mathcal{M}_2\) are equivalent if meta-model \(\mathcal{M}, \mathcal{M}_1\) is generic compared to meta-model \(\mathcal{M}, \mathcal{M}_2\) and vice versa. Consequently, a transformation is referred to as “generic” if the meta-model resulting from the transformation is more generic than the meta-model transformed. The analogous explanation holds for individualised and equivalent transformations.

#### 7.6.1 Basic transformations

We first focus on the basic transformations. Table 7.14 lists for each basic transformation whether its application results in a more generic meta-model, in a more individualised meta-model, or an equivalent meta-model.

It should be clear that the basic transformations which add new Object types or new Roles to a meta-model, result in a more generic meta-model, whereas the basic transformations which delete Object types or Roles from a meta-model, result in a more individualised meta-model.

200
Table 7.14: Basic object transformations, categorised into "generic", "individualised", and "equivalent"

Furthermore, the addition of a new constraint to a meta-model results, generally, in a more individualised meta-model, since the new constraint restricts the number of possible populations, while the deletion of a constraint from a meta-model results, generally, in a more generic meta-model. Note that we use the word "generally" because we are well aware that the addition of a new constraint does not impose further restrictions on the set of possible populations in all circumstances. Consider for example the addition of the constraint frequency(\{w, x\}, 1, 1) to the abstract meta-model given in figure 7.1. Addition of this constraint does not lead to a further restriction of the set of possible populations because of the uniqueness constraint over Roles w and x in this meta-model.

Analogously, it should be clear that the basic transformations which extend the domain of a Label type result in a more generic meta-model, whereas the basic transformations which reduce the domain of a Label type result in a more individualised meta-model.

The six basic transformations which manipulate Relationships as a whole all lead to a more generic meta-model as well. The reason for this is that they add some Relationships to a meta-model without removing the Relationships upon which the construction of the new Relationships is based.

It might be more difficult to determine to which category the manipulation of generalisations and specialisations belong. Consider figure 7.19, which shows the addition of a Subtype D to Object type A. The population of Subtype D is a subset of the population of Object
Figure 7.19: Example of the addition of a specialisation relation to an abstract meta-model type A, defined in the Subtype defining rule. The population of the original meta-model remains unchanged. The addition of a specialisation relation thus results in an equivalent meta-model as such. Note, however, that the Subtype defining rule may require the addition of new Object types.

Figure 7.20: Example of the addition of a generalisation relation to an abstract meta-model

Consider figure 7.20, which shows the addition of a generalisation relation between Object type B and D. According to the Generalisation Rule, the cardinality of the population of Object type B has increased by \(|\text{Pop}(D)|\). Therefore, adding a generalisation relation results in a more generic meta-model. Consequently, removing a generalisation relation results in a more individualised meta-model.

### 7.6.2 Composed transformations

This section addresses the categorisation of some composed transformations, as generic, individualised, or equivalent. For most of the composed transformations addressed, we show that their application results in an equivalent meta-model, provided that some specific constraints are present in the original meta-model and in the resulting meta-model.
The six composed transformations concerning the manipulation of Relationships, are addressed first. The composed union is a generic transformation: the population of the resulting meta-model cannot be transformed back to the original meta-model, since it is not possible to decide for each instance in the united Relationship in the resulting meta-model, to the population of which Relationship in the original meta-model it belongs. The composed selection is an individualised transformation: the instances which have not been selected are lost. In the case of the composed join, it is required that the populations of the two Relationships satisfy an equality constraint over these subsets of Roles which have the same Base object types. These subsets are considered to be uniqueness constraints. In the example of figure 7.21, the equality constraint in the leftside meta-model involves subsets of Roles which have A and B as their Base object types. The composed projection is an individualised transformation, analogous to the composed selection.

The composed unnest and the composed nest are equivalent transformations, provided that the new Role in the meta-model with the nested Relationship satisfies a uniqueness constraint and a total role constraint, and that the Roles in the objectified Relationship satisfy a uniqueness constraint in both meta-models, see figure 7.22.

Application of the two specific schema transformation pairs from [Halpin, 1989]. Code introduction and Code removal, and Described object type introduction and Described object type removal, also results in equivalent meta-models, provided that some specific conditions hold. In the case of Code introduction, an enumeration constraint over the new Label type is required. The cardinality of the domain of this Label type should equal the number of Relationships involved, see figure 7.23. The constraints which should hold for Described object type introduction and removal being equivalent transformations are given in figure 7.24. In this example, it is required that each instance of Object type c is unequivocally related to one instance of Object type B and vice versa, in other words, that the populations of Object type B and C are involved in a bijection. The two total role constraints and uniqueness constraints satisfy this requirement.
Figure 7.22: Abstract example of equivalent meta-models concerning the composed transformations Comp_Unn_rel and Comp_Nest_rel

Figure 7.23: Abstract example of equivalent meta-models concerning Code introduction and Code removal

We omit the formal proof that application of these composed transformations results in equivalent meta-models. For each of the cases presented, the essence of the proof is that a bijection can be defined between the population of the original meta-model and the population of the meta-model resulting from the application of one of the composed transformations.

### 7.7 Applications

We have defined a language, consisting of meta-model transformations. This section presents several applications of meta-model transformations. We first present a generic meta-model in section 7.7.1. We apply several meta-model transformations on this generic meta-
model, and show the consequences for a way of modelling (section 7.7.2) and a way of working (section 7.7.3).

7.7.1 A generic meta-model

This section presents a generic meta-model, which serves as a starting point for the illustration of applying meta-model transformations in sections 7.7.2 and 7.7.3. We assume such a generic meta-model to be available. Recall the use of the word "meta-model" which refers to an Object structure and a set of graphical constraints.

The generic meta-model concerns Yourdon's way of modelling. Naturally, this meta-model is based upon the prescribed modelling knowledge given in chapter 3, and on the applied modelling knowledge given in chapters 4 and 5. We therefore do not consider it to be necessary to explain the modelling concepts and the relationships between them as they are given in the generic meta-model.

As pointed out in the beginning of this chapter, a meta-model is considered to be more generic if more choices with regard to the conceptualisation and the interpretation of a way of modelling are left open. Generally, making decisions with respect to such choices implies that the set of possible populations of the meta-model is restricted. As it is our aim to present a generic meta-model in this section, we do not add any graphical constraints, as these restrict the number of possible populations generally. Therefore, the generic meta-model is an Object structure. This Object structure consists of a large number of Object types. For a convenient presentation, the Object structure is given in three consecutive figures.

Figure 7.25 shows the generic meta-model, as far as ERDs are concerned. Figure 7.26 concerns DFDs, whereas figure 7.27 presents several relationships between ERDs and DFDs.
Figure 7.25: A generic meta-model of Yourdon's way of modelling: ERDs

The latter figure shows the Object type Data-flow twice for reasons of comprehensibility. The most important Object types are Entity-type, Relationship, Attribute (figure 7.25), Process, Data-store, Terminator, Data-flow, Control-process, Control-flow (figure 7.26), and Event (figure 7.27).

7.7.2 Way of modelling

As pointed out in chapter 6, the main differences in the ways of modelling prescribed and applied, are the use of different modelling concepts, the use of more refined modelling concepts, the application of a different set of constraints, and the use of different graphical notations. This section gives some examples in which the meta-model transformations support in creating a meta-model with an individualised set of modelling concepts and graphical constraints. Graphical notations are not dealt with, as we decided not to focus on graphical notations in chapter 7.
First, the use of different modelling concepts is illustrated. A specific way of modelling is characterised by the choice of a specific subset of modelling concepts from the generic metamodels. Choosing a subset from the generic meta-model is a simple operation, using the basic transformations which delete Object types. For example, Yourdon's way of modelling as it is prescribed does not employ any modelling concepts which serve communication purposes or which provide quantitative insight. The generic meta-model hence can be
Figure 7.27: A generic meta-model of Yourdon's way of modelling: the relationships between ERDs and DFDs

transformed as follows:

\[
\text{DelLab} \text{Frequency} \odot \text{DelLabSize} \odot \text{DelLabDuration} \odot \text{DelLabPriority} \odot \text{DelLabVolume} \odot
\]
Another example is the approach followed by information engineers B and C when constructing DFDs. Their approaches do not include Control processes and Control flows. Clearly, the generic meta-model is transformed as follows:

\[
\text{Del}_{\text{Sample-value}}(M_1M_1) \quad \text{Del}_{\text{entControl-process}} \circ \text{Del}_{\text{entControl-flow}}(M_1M_1)
\]

None of the information engineers used diverging or converging Data flows in their DFDs. Therefore, the Set type \text{Data-flow-group} can be removed from the generic meta-model. Note that this removal implies that the Relationship between \text{Data-flow} and \text{Data-flow-group} is also removed, due to the definition of the basic transformation involved. Given this removal,
it is possible to enforce in the Object structure that each Data flow connects *two* DFD Objects, by joining the two separate Relationships \( f = \{ \text{produces, is-output-of} \} \) and \( g = \{ \text{consumes, is-input-for} \} \) between Data-flow and DFD-Object. This pair of transformations results in a more individualised meta-model, see figure 7.28.

Formally, this individualisation is denoted as:

\[
\text{Comp}_\text{join-rel}_\xi \circ \text{Del-set}_{\text{Data-flow-group}}(\mathcal{M}, \mathcal{M}_1)
\]

Here, the bijection \( \xi \) defines how the Roles of the two separate Relationships \( f \) and \( g \) are mapped onto the Roles of the resulting Relationship.

![Diagram](image)

Figure 7.29: Adding an occurrence frequency constraint to enforce support for the binary ERD variant

Refinement of modelling concepts is achieved by adding specialisation hierarchies. For example, the use of ERDs by information engineer A (see figure 5.12, page 128) suggests the addition of three Subtypes to the Object type *Entity-type* in the generic meta-model:

\[
\text{Add-spec}_{\text{Concrete-entity-type}, \text{Entity-type}} \circ \text{Add-spec}_{\text{Associative-entity-type}, \text{Entity-type}} \circ \\
\text{Add-spec}_{\text{Conceptual-entity-type}, \text{Entity-type}}(\mathcal{M}, \mathcal{M}_1)
\]

The use of constraints can be illustrated in numerous ways. A characteristic example is the use of an occurrence frequency constraint to restrict the use of the ERD modelling technique to its binary variant (see figure 7.29):

\[
\text{Add-constr}_{\text{frequency}([\text{in}], 2, 2)}(\mathcal{M}, \mathcal{M}_1)
\]

Note that it is also possible to restrict the use of the ERD modelling technique to its binary variant in the Object structure itself. Two transformations are used to this end: (i) removal of the Code *Role*, resulting in two Relationships, and (ii) joining the two resulting Relationships, see figure 7.30. Formally, this transformation is written:

\[
\text{Comp}_\text{join-rel}_{\xi_3} \circ \text{Del-code}_{\text{Role}, \text{is-played-in}} = a, (a, \text{role}, \xi_1), (b, \text{role}, \xi_2)(\mathcal{M}, \mathcal{M}_1)
\]

The parameters \( a - \text{role} \) and \( b - \text{role} \) are intermediate Roles which have to be specified for Code removal. The parameters \( \xi_1, \xi_2, \) and \( \xi_3 \) denote Role mappings.
7.7.3 Way of working

As pointed out in chapter 6, various strategies that are part of a way of working are used to construct DFDs and ERDs. We observed that the differences between the various strategies result from different ways of modelling, in particular, we observed that information interdependencies (such as those given in figure 7.27) play an important role.

Therefore, in order to support a specific strategy as part of a way of working, it is neces-
sary to select (in other words, not to remove) some specific Object types from the generic meta-model. In section 6.2, we discussed how various strategies are related to specific Object types, see in particular the discussion of the items "Information engineer dependent strategies", "Different process modelling strategies", and "Different data modelling strategies".

7.8 Conclusions

In this chapter, we have defined a set of transformations which support the adaptation of generic modelling knowledge towards more individualised modelling knowledge. By this, we have formulated a theory about the use of modelling knowledge to achieve more effective information modelling support.

It can be concluded that the set of basic transformations is complete, with respect to the underlying meta-modelling technique. Each component of a meta-model can be manipulated by some basic transformation (section 7.4). To obtain a suitable language, we have discussed several composed transformations from literature. These turned out to be expressable using our basic transformations (section 7.5). In section 7.6 we have pointed out which transformations result in equivalent or more individualised meta-models. These transformations are of particular interest, since their application fits into our notion of "effective information modelling support". Finally, we have illustrated the use of meta-model transformations by applying them to a generic meta-model of Yourdon's information systems development method.

This chapter leads us to conclude that our second research question is answered positively, it is possible to define a set of transformations which aid in individualised support. Moreover, we can conclude that this set of transformations is complete and suitable.
Chapter 8

Epilogue

8.1 Introduction

This chapter aims at evaluating our research questions. Therefore, we return to our objective of achieving more effective information modelling support.

In chapter 1, we restricted the problem area addressed in this thesis to the support of the early stages of the information systems development life cycle, with regard to a way of modelling and a way of working. Methods, techniques, and CASE tools, though developed to deliver information modelling support, are shown to be used to a limited extent only. Modelling knowledge as applied by experienced information engineers turns out to deviate from modelling knowledge as prescribed by method handbooks. CASE tools are considered to provide rigid and unflexible support, and, as a consequence, information engineers have to adapt their working style to CASE tools instead of vice versa. Several researchers have elaborated on a new approach to information modelling support to diminish these problems, generally referred to as the meta-modelling approach. This approach focuses on possibilities for information engineers to specify, to adapt, and to refine, modelling knowledge as part of CASE tools, according to their preference and experience.

The meta-modelling approach, however, neglects the issue of whether it is worthwhile to take the effort to specify a detailed modelling knowledge base for each individual information engineer. As a second drawback, support which is effective for the individual does not necessarily meet an organisation's need for standardisation and for transferability of results.

Based on the above, we formulated our research questions in section 1.5. We repeat them here for convenience:

1. Is it possible to acquire a detailed insight into differences and similarities between prescribed modelling knowledge and applied modelling knowledge?

2. Is it possible to develop mechanisms that support the adaptation of generic modelling knowledge towards individual modelling support?
8.2 Evaluation

Our research questions are evaluated as follows.

Research question 1

To acquire a detailed insight into differences and similarities between prescribed modelling knowledge and applied modelling knowledge, we developed (chapter 2) and applied (chapters 3, 4, and 5) an approach to acquiring modelling knowledge.

In chapters 3 up to and including 5, detailed models of prescribed [Yourdon, 1989] and applied modelling knowledge were given. In chapter 6, we summarised our findings with regard to a way of modelling and a way of working in prescribed and applied modelling knowledge.

Different variants of the prescribed modelling techniques were applied in practice. The applied variants differ from the prescribed modelling techniques because they employ additional modelling concepts, more refined modelling concepts, and different constraints. The additional modelling concepts include concepts for representing quantitative information, for representing organisational aspects, and for representing problem situations. Modelling concepts were represented by different graphical notations.

Modelling strategies widely varied over the information engineers. These strategies deviated from the prescribed way of working to a considerable extent. The prescribed modelling approach is characterised by an almost strictly linear order of performing modelling tasks, whereas a very opportunistic order is actually used. This order seemed to depend on the problem domain as well as on the information engineer. The information engineers applied different process modelling strategies and different data modelling strategies. The extent to which process modelling dominates data modelling - or vice versa - also varied largely over the information engineers. This spectrum of differences between ways of working could be traced back partially to different information interdependencies between modelling techniques, and, therewith, to different underlying ways of modelling.

In chapter 6, we concluded that our approach has led to a detailed understanding of differences and similarities between prescribed and applied modelling knowledge, albeit at the expense of a time-intensive approach. Obviously, we can answer the first research question positively based upon our findings.

It is important to stress that this statement is true within the limits of our approach:

- We restricted ourselves to modelling knowledge as contained in the Yourdon method (section 2.2).
- We represented only those aspects of modelling knowledge that the expressive power of the meta-modelling technique chosen, permits one to represent (section 2.3).
- We acquired applied modelling knowledge according to a specific experimental setting (section 2.4) involving three experienced information engineers who solved two cases each.

**Research question 2**

To develop mechanisms that support the adaptation of generic modelling knowledge towards individual modelling support, we presented a language comprising a set of basic meta-model transformations in chapter 7. This language is considered to be complete with respect to the meta-modelling technique chosen. Furthermore, it is considered to be a suitable language, since the transformations are defined in a constructive way. Basic transformations can be concatenated in numerous ways. A number of complex transformations, presented in the literature on schema transformations as primitive, can be constructed by concatenating our basic transformations. We finally have shown a number of applications of meta-model transformations. Hence, we can answer the second research question positively.

We are well aware that our approach has been restricted in one particular aspect:

- We restricted ourselves to mechanisms that support the adaptation of modelling knowledge towards individual needs only as far as a way of modelling is concerned (section 7.1).

### 8.3 Future research

We conclude this chapter with the identification of a number of topics for further research. We showed a number of differences between prescribed and applied modelling knowledge. A closer focus on these differences might lead to a number of suggestions for useful extensions to prescribed modelling knowledge.

Although we showed a small number of applications of meta-model transformations, we did not address the issue of how to derive a generic meta-model. Instead, we assumed a generic meta-model to be available (section 7.7). Future research should focus on the identification of modelling concepts which may be part of a generic meta-model. Furthermore, an approach is needed for identifying which parts of modelling knowledge should be standardised, and which parts contain only recommendations.

Our mechanisms for individualisation of a generic meta-model are restricted to a way of modelling. Therefore, the development of mechanisms which aid in individualisation of a way of working is a topic of future research. A hypothesis for future research is that mechanisms for specific orderings of tasks and for giving priority to specific decision outcomes lead to more individualised support, without affecting the generic meta-model.

This thesis shows the feasibility of constructing mechanisms for individualising method knowledge. It was not investigated to what extent these mechanisms reduce the effort of specifying modelling knowledge. Economic aspects of the use of these mechanisms are there-
fore an issue for further research. We consider the incorporation of adaptation mechanisms in a CASE shell to be desirable for this purpose.
Appendix A

Verification rules prescribed modelling knowledge

In this appendix we present the verification rules that are part of the prescribed modelling knowledge from [Yourdon, 1989]. We present them graphically as far as possible, however if a verification rule is too complex to be presented graphically, we present its formal definition.

<table>
<thead>
<tr>
<th>Verification rule categories</th>
</tr>
</thead>
<tbody>
<tr>
<td>Labelling</td>
</tr>
<tr>
<td>Connections</td>
</tr>
<tr>
<td>Decomposition</td>
</tr>
<tr>
<td>Information interdependencies</td>
</tr>
<tr>
<td>Occurrences</td>
</tr>
</tbody>
</table>

Table A.1: Categories of verification rules

Verification rules are divided into five distinct categories in this appendix. These are itemised in table A.1. The category “Labelling” deals with Label types in relation to the corresponding Object types. The category “Connections” constrains the relationship between several Object types which are part of one and the same model. The category “Decomposition” constrains the relationship between several Object types which are part of different models of the same model type. The category “Information interdependencies” constrains the relationships between several Object types which are part of models of a different model type. Finally, the category “Occurrences” deals with models as a whole, in relation to their constituting Object types.

To avoid confusion, usage of “at most one” means “either zero or one”.
A.1 Task independent verification rules

This section describes the verification rules that are part of Yourdon's way of modelling. These rules apply independent of a specific stage within an information modelling process, therefore verification rules that are part of a way of modelling always have an a priori character.

This section is structured analogously to the presentation of Yourdon's way of modelling as such, see section 3.1. We cover a priori verification rules for Entity-Relationship Diagrams and Data Flow Diagrams first, the verification rules constraining the relationships between ERDs and DFDs are then presented.

A.1.1 Entity-Relationship Diagrams

This section presents the a priori verification rules for Entity-Relationship Diagrams, part of Yourdon's way of modelling. Figure A.1 shows the rules E1 up to and including E7 which can be represented graphically. To improve readability, the Relationships in the Object structure of ERDs (figure A.1) are not named. Instead, the first column in table A.2 presents these Relationship names, whereas the three other columns present the Object types involved.

<table>
<thead>
<tr>
<th>Relationship</th>
<th>Object type</th>
<th>Object type</th>
<th>Object type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERD-Object-name</td>
<td>ERD-Object</td>
<td>(E-Name)</td>
<td>~</td>
</tr>
<tr>
<td>Participation</td>
<td>Object-type</td>
<td>Relationship</td>
<td>Role</td>
</tr>
<tr>
<td>Associative-object-type-indicator</td>
<td>Object-type</td>
<td>Relationship</td>
<td>~</td>
</tr>
<tr>
<td>Supertype/subtype-indicator</td>
<td>Object-type</td>
<td>Object-type</td>
<td>~</td>
</tr>
<tr>
<td>Object-type-description</td>
<td>Object-type</td>
<td>Data-element</td>
<td>~</td>
</tr>
<tr>
<td>Object-type-identification</td>
<td>Object-type</td>
<td>Data-element</td>
<td>~</td>
</tr>
</tbody>
</table>

Table A.2: Relationship names in figure A.1

Labelling

[E1] ERD Objects have at most one Name.

Connections

[E2] Roles are played at most once in a Relationship.

[E3] Object types have at most one Supertype.

[E4] Object types replace at most one Relationship.

[E5] Relationships are replaced by at most one Object type.
Figure A.1: Object structure of ERDs with graphical a priori verification rules

[E6] Data elements which identify a Object type should also describe that Object type.

[E7] Object types which replace a Relationship are not allowed to be a Subtype.

Decomposition

[E8] Recursive subtype hierarchies of Object types are not allowed.

NO Object-type

ANY-REPETITION-OF (Object-type is-supertype-of Object-type)

THAT Object-type

A.1.2 Data Flow Diagrams

This section presents the a priori verification rules for Data Flow Diagrams, part of Yourdon's way of modelling. Figure A.2 shows the verification rules D1 up to and including D17 and D25 up to and including D33 which can be represented graphically. To improve readability, the Relationships in the Object structure of DFDs (figure A.2) are not named. Instead, the first column in table A.3 presents these Relationship names, whereas the second and the third column present the Object types involved.
Figure A.2: Object structure of DFDs with graphical a priori verification rules

Labelling

[D1] DFD Elements have at most one Name.

[D2] DFDs have at most one Name.

[D3] Processes have at most one Number.
<table>
<thead>
<tr>
<th>Relationship</th>
<th>Object type</th>
<th>Object type</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFD-name</td>
<td>DFD</td>
<td>(D-Name)</td>
</tr>
<tr>
<td>DFD-Element-name</td>
<td>DFD-Element</td>
<td>(D-Name)</td>
</tr>
<tr>
<td>Process-number</td>
<td>Process</td>
<td>(D-Number)</td>
</tr>
<tr>
<td>DFD-Number</td>
<td>DFD</td>
<td>(D-Number)</td>
</tr>
<tr>
<td>Process-spec</td>
<td>Process</td>
<td>Process-Specification</td>
</tr>
<tr>
<td>Process-decomposition</td>
<td>Data-store</td>
<td>DFD</td>
</tr>
<tr>
<td>External-data-store</td>
<td>Data-flow</td>
<td>Data-flow</td>
</tr>
<tr>
<td>Data-flow-correspondence</td>
<td>Data-flow</td>
<td>Data-flow-group</td>
</tr>
<tr>
<td>Complex-flow</td>
<td>Data-flow</td>
<td>DFD-Object</td>
</tr>
<tr>
<td>Incoming-flow</td>
<td>Data-flow</td>
<td>DFD-Object</td>
</tr>
<tr>
<td>Outgoing-flow</td>
<td>(D-name)</td>
<td>(D-name)</td>
</tr>
<tr>
<td>Store-flow-relation</td>
<td>Control-process</td>
<td>STD</td>
</tr>
<tr>
<td>Control-process-decomposition</td>
<td>Control-flow</td>
<td>DFD-Object</td>
</tr>
<tr>
<td>Control-flow-connection</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table A.3: Relationship names in figure A.2

[D4] Numbers are assigned to at most one Process.

[D5] DFDs have at most one Number.

[D6] Numbers are assigned to at most one DFD.

Connections

[D7] Data flows are output of at most one DFD Object.

[D8] Data flows are input for at most one DFD Object.

[D9] Control flows come from at most one DFD Object.

[D10] Control flows go to at most one DFD Object.

[D11] Data flows are not input for and output of the same DFD Object.

[D12] Control flows do not come from and go to the same DFD Object.

[D13] DFD Objects do not consume more than one Data flow with the same Name.

[D14] DFD Objects do not produce more than one Data flow with the same Name.
[D15] Data flows participate in at most one Data flow group.

[D16] Data flows are connected to at most one Data flow group.

[D17] Data flow groups are connected to at most one Data flow.

[D18] Data flows do not combine being connected to a Data flow group, being input for a DFD Object and being output for a DFD Object.
NO Data-flow (is-connected-to-group AND-ALSO is-input-for AND-ALSO is-output-for)

[D19] Data flows do not exist between two Data stores.
NO Data-store produces Data-flow is-input-for Data-store

[D20] Data flows do not exist between two Terminators.
NO Terminator produces Data-flow is-input-for Terminator

[D21] Between two Terminators, Data stores do not exist.
NO Terminator produces Data-flow is-input for Data-store produces Data-flow is-input for Terminator

[D22] Terminators can not receive a Control flow.
NO Control-flow flows-from DFD-Object to Terminator

[D23] Data stores are not connected to Control flows.
NO Data-store ASSOCIATED-WITH Control-flow

[D24] Control processes are not connected to Data flows.
NO Control-process ASSOCIATED-WITH Data-flow

[D25] Data flow names relate to at most one Data store name.

[D26] Data store names relate to at most one Data flow name.
Decomposition

[D27] Data flows are related to at most one higher level Data flow.

[D28] Data flows are related to at most two lower level Data flows.

[D29] Processes are decomposed into at most one DFD.

[D30] DFDs are the decomposition of at most one Process.

[D31] Processes are described in at most one Process specification.

[D32] Processes are not allowed to be specified in a Process specification and to be decomposed into a DFD at the same time.

[D33] Control processes are decomposed into at most one STD.

[D34] Recursive decomposition of Processes is not allowed.

NO Process
ANY-REPETITION-OF (Process is-decomposed-into DFD COMPRISING Process)
THAT Process

A.1.3 Relationship between ERDs and DFDs

This section presents the a priori verification rules which apply for the relationships between DFDs and ERDs, part of Yourdon's way of modelling. Figure A.3 shows the verification rules R1 up to and including R11 which can be represented graphically. To improve readability, the Relationships in the Object structure of figure A.3 are not named. In table A.4, the first column presents these Relationship names, whereas the second and the third column present the Object types involved.

Information interdependencies

[R1] Object types are related to at most one Data store.

[R2] Data stores are related to at most one Object type.

[R3] The instances of a Relationship are manipulated by at most one Event.

[R4] Events manipulate at most the instances of one Relationship.

[R5] Processes handle at most one Event.
Figure A.3: Object structure representing the relationships between DFDs and ERDs, with graphical a priori verification rules

<table>
<thead>
<tr>
<th>Relationship</th>
<th>Object type</th>
<th>Object type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relationship-manipulation</td>
<td>Event</td>
<td>Relationship</td>
</tr>
<tr>
<td>Object-type-derivation</td>
<td>Event</td>
<td>Object-type</td>
</tr>
<tr>
<td>Stimulus-recognition</td>
<td>Flow-oriented-event</td>
<td>Data-Flow</td>
</tr>
<tr>
<td>Event-processing</td>
<td>Flow-oriented-event</td>
<td>Data-Flow</td>
</tr>
<tr>
<td>Response-production</td>
<td>Flow-oriented-event</td>
<td>Data-Flow</td>
</tr>
<tr>
<td>Event-handling</td>
<td>Event</td>
<td>Process</td>
</tr>
<tr>
<td>Control-event-indication</td>
<td>Control-event</td>
<td>Control-flow</td>
</tr>
<tr>
<td>ERD-DFD-Coherence</td>
<td>Object-type</td>
<td>Data-store</td>
</tr>
</tbody>
</table>

Table A.4: Relationship names in figure A.3

[R6] Control events are associated with at most one Control flow.
[R7] Control flows are associated with at most one Control event.

[R8] Data flows are required for the processing of at most one Flow-oriented event.

[R9] Data flows recognise at most one Flow-oriented event.

[R10] Flow-oriented events are recognised by at most one Dataflow.

[R11] Data flows either recognise a Flow-oriented event, are required for processing by a Flow-oriented event or respond to a Flow-oriented event.

[R12] Outgoing flows do not lead to Flow oriented events that are recognised by the system.

NO Flow-oriented-event is-recognised-by Data-flow is-output-of Process

[R13] Outgoing flows do not lead to Flow oriented events that are processed by the system.

NO Flow-oriented-event is-processed-by Data-flow is-output-of Process

[R14] Incoming flows do not lead to Flow oriented events that are responded to by the system.

NO Flow-oriented-event is-responded-to-by Data-flow is-input-for Process

A.2 Task dependent verification rules

This section presents the verification rules which apply only during parts of an information modelling process. In other words, they are task dependent. These verification rules are expressed in terms of the Object structure which represents Yourdon's way of modelling, and they apply to specific Tasks which are part of Yourdon's way of working.

The verification rules are presented for each leaf modelling task (see table 3.1) separately. The majority of them are a posteriori verification rules.

a. Formulate statement of purpose

In terms of update statements (see section 2.3.4.1), the leaf modelling task "Formulate statement of purpose" is defined as follows:

ADD Statement-of-purpose s

This definition guarantees the existence of exactly one Statement of purpose. No additional verification rules apply for this leaf modelling task.
b. Construct context diagram

A number of verification rules should be satisfied after having constructed the context diagram. Figure A.4 shows the verification rules b1 up to and including b6 which can be represented graphically.
Labelling

[b1] All DFD Elements have a Name.

Connections

[b2] Each Data flow is input for a DFD Object.

[b3] Each Data flow is output of a DFD Object.

[b4] Each Control flow goes to a DFD Object.

[b5] Each Control flow comes from a DFD Object.

[b6] Each DFD Object is connected to a Data flow or to a Control flow.

[b7] "Spontaneous generation bubbles", i.e. Processes that have outputs but no inputs, must be avoided.
FOR-EACH p IN Process HOLDS p consumes Data-flow

[b8] "Infinite sinks", i.e. Processes that have inputs but no outputs, must be avoided.
FOR-EACH p IN Process HOLDS p produces Data-flow

[b9] Beware of "read-only data stores".
FOR-EACH s IN Data-store HOLDS s consumes Data-flow

[b10] Beware of "write-only data stores".
FOR-EACH s IN Data-store HOLDS s produces Data-flow

[b11] Each Control flow is connected to a Process.
FOR-EACH c IN Control-flow HOLDS c ASSOCIATED-WITH Process

[b12] The Name of a Data store is the plural form of the Name of one of the Data flows connected to that Data store.
Data-store ASSOCIATED-WITH Data-flow
INCLUDES
Data-store has D-Name relates-to-flow-with D-Name of Data-flow
Figure A.5: Object structure representing the relationships between DFDs and ERDs, with graphical a posteriori verification rules

**Occurrences**

[b13] Each DFD contains a Data flow.

 FOR-EACH \( d \) IN DFD HOLDS \( d \) COMPRISING Data-flow


 FOR-EACH \( d \) IN DFD HOLDS \( d \) COMPRISING Terminator

**c. Construct event list, context diagram based**

Events are added to the event list during the task "Construct event list, context diagram based". Additions are preferably, but not obligatorily, based on flows in the context diagram. The verification rules which apply to the construction of an Event list all involve information interdependencies between the two model types.
At the end of this task, it should be checked that all flows in the context diagram have been taken into account. This implies that each Flow-oriented event relates to a Data flow, and vice versa, and that each Control event relates to a Control flow, and vice versa. Figure A.5 shows the verification rules c1 up to and including c4 which can be represented graphically.

**Information interdependencies**

[c1] Each Flow-oriented event relates to a Data Flow.


[c3] Each Control event relates to a Control flow.

[c4] Each Control flow relates to a Control event.

d. Construct ERD

![ERD diagram](image_url)

Figure A.6: Object structure of ERDs with graphical a posteriori verification rules
The task "Construct ERD" is an auxiliary one, serving as a basis for the construction of an event list. The resulting ERD may be discarded afterwards, since Yourdon provides no clue for employing it during the construction of the behavioral model.

The way in which a first ERD may be constructed is restricted according to a number of verification rules. Figure A.6 shows the verification rules d2 and d3 which can be represented graphically.

Verification rules which should be satisfied after having constructed an ERD are:

**Labelling**

[d1] Each Object type has a Name.

\[
\text{FOR-EACH } o \text{ IN Object-type \quad HOLDS } o \text{ has E-name}
\]

**Connections**

[d2] Each Object type participates in at least one Relationship.

[d3] Each Relationship contains at least one Object type.

**Occurrences**


\[
\text{FOR-EACH } e \text{ IN ERD \quad HOLDS } e \text{ COMPRISING Object-type}
\]

[d5] Each ERD contains a Relationship.

\[
\text{FOR-EACH } e \text{ IN ERD \quad HOLDS } e \text{ COMPRISING Relationship}
\]

**e. Construct event list, ERD based**

The task "Construct event list, ERD based" results in the addition of events to the event list, based on manipulations of instances of Relationships which are part of the ERD. Only one verification rule is discerned which applies to the construction of an Event list, based on the existing ERD. This verification rule involves the information interdependency between Event lists and ERDs. Figure A.5 shows verification rule e1 which can be represented graphically:

**Information interdependencies**

[e1] Each Event manipulates instances of a Relationship.
f. Construct context diagram, event list based

The task “Construct context diagram, event list based” results in a context diagram, to which the flows have been added by determining for each event on the event list how it is recognised, processed, and responded to.

This task is bound to the same verification rules as the construction of a Context diagram from scratch (rules b1 up to and including b14). Furthermore, the rules which apply to the construction of an Event list based on a Context diagram (c1 up to and including c4) are applicable for this leaf modelling task.

g. Construct preliminary DFD

The task “Construct preliminary DFD” results in a DFD based on the existing Event list. The construction of a preliminary DFD is similar to the construction of a context diagram (see task b). A few important differences should be underlined however. First, Processes are added to the preliminary DFD using the existing Event list as a starting point (see rule g5). To make the asynchronous characteristic of inter-process communication explicit (rule g2), Processes may not be connected directly. Furthermore, converging or diverging Data flows may be part of the preliminary DFD, contrary to the context diagram. In the case of such a complexly structured Data flow, the Data flow is connected to a Data flow group. In this case, a Data flow is either produced by or consumed by a DFD Object.

The verification rules b1 and b4 up to and including b14 apply to this task. Verification rules b2 and b3 have been substituted by verification rule g3. Figure A.4 shows the verification rules g3 and g4, figure A.5 shows verification rule g5. These verification rules can all be represented graphically. The formulation of the verification rules for this task employs the variables “Preliminary DFD” and “Preliminary process”. The following verification rules are satisfied after having constructed the preliminary DFD:

Labelling

[g1] Each Process in the preliminary DFD has a Number.
    FOR-EACH p IN Preliminary-process HOLDS p has D-Number

Connections

[g2] Data flows do not exist between two Processes.
    NO Process produces Data-flow is-input-for Process

[g3] Each Data flow is connected to a DFD Object.

[g4] Each Data flow group is connected to a Data flow to constitute a complex Data flow.

Information interdependencies

[g5] Each Event is handled by a Process.
Occurrences

[g6] Each DFD contains a Process.

FOR-EACH d IN DFD HOLDS d COMPRISING Process

h. Check preliminary DFD against context diagram

The task “Check preliminary DFD against context diagram” realises that the preliminary DFD is related to the context diagram as its next lower decomposition. After that, it checks the name of the DFD against the name of the Process in the context diagram. Furthermore, it checks whether all Data flows which go into or come out of the preliminary DFD correspond to Data flows in the context diagram. The following verification rules apply:

Decomposition

[h1] A Process and its decomposition should have the same Name.

DFD has D-Name

INCLUDES

DFD is-decomposition-of Process has D-Name

[h2] All DFD Objects which provide input for a Process are part of the input providing environment of its decomposed DFD, and vice versa.

DFD is-decomposition-of Process consumes Data-flow is-output-of DFD-Object

EQUALS

(DFD COMPRISING DFD-Object consumes Data-flow is-output-of DFD-Object) BUT-NOT

(DFD COMPRISING DFD-Object)

[h3] All DFD Objects which provide output for a Process are part of the output providing environment of its decomposed DFD, and vice versa.

DFD is-decomposition-of Process produces Data-flow is-input-of DFD-Object

EQUALS

(DFD COMPRISING DFD-Object produces Data-flow is-input-of DFD-Object) BUT-NOT

(DFD COMPRISING DFD-Object)

[h4] All Data flows which are input for a Process are also input for its decomposed DFD, and vice versa.

DFD is-decomposition-of Process consumes Data-flow

EQUALS

DFD COMPRISING DFD-Object consumes Data-flow relates-to-higher-level Data-flow
All Data flows which are output for a Process are also output for its decomposed DFD, and vice versa.

DFD is-decomposition-of Process produces Data-flow
EQUALS
DFD COMPRISING DFD-Object produces Data-flow relates-to-higher-level Data-flow

i. Construct initial ERD

The construction of the initial ERD is based on the existing Event list. Verification rule i1 involves the information interdependencies between Event lists and ERDs, see figure A.5.

The construction of the initial ERD does not differ much from task d. The main difference is that the related Object structure contains more Object types. The construction of the initial ERD involves Associative object type indicators, and supertypes and subtypes. Therefore, all verification rules of task d apply for the construction of the initial ERD, except for rule d2, which has been replaced by rule i2, see figure A.6.

Connections

[i1] Each Object type occurs as a noun in the description of one or more Events.

Information interdependencies

[i2] Each Object type participates in a Relationship or replaces a Relationship or is the subtype of another Object type.

j. Cross-check preliminary DFD against initial ERD

Cross-checking the preliminary DFD and the initial ERD to each other involves an information interdependency between DFDs and ERDs. It is checked whether the verification rule which states that Data stores in the preliminary DFD should be related to Object types in the initial ERD is obeyed, and vice versa.

The resulting verification rules are represented graphically, see figure A.5.

Information interdependencies

[j1] Each Object type relates to a Data store.

[j2] Each Data store relates to a Object type.
k. Finish DFD

The task "Finish DFD" does not differ essentially from the task "Construct preliminary DFD". The main difference is that "Construct preliminary DFD" includes only one decomposition, from context diagram to preliminary DFD, whereas "Finish DFD" leads to an arbitrary number of decompositions. Several minor differences can be distinguished. The Event list is not of any relevance to this task, excluding verification rule g5. Furthermore, the numbering of DFDs, the use of Process specifications and the decomposition of Control processes are issues in this leaf modelling task, in addition to the ones of task g. The variables Low-level DFD and Low-level Process are employed in the formulation of some of the remaining a posteriori verification rules.

The following verification rules should be satisfied after having finished the DFD:

**Labelling**

[k1] Each Low-level DFD has a Number.

\[
\text{FOR-EACH } d \text{ IN Low-level-DFD HOLDS } d \text{ has } D\text{-Number}
\]

[k2] Each Low-level Process has a Number.

\[
\text{FOR-EACH } p \text{ IN Low-level-Process HOLDS } p \text{ has } D\text{-Number}
\]

**Decomposition**

[k3] Each Process is decomposed into a DFD or is specified in a Process Specification.

[k4] Each Control process is decomposed into a STD.


[k6] Each STD is the decomposition of a Control Process.

[k7] A Process and its decomposition should have the same Number.

\[
\text{DFD has } D\text{-Number}
\]

\[
\text{EQUALS}
\]

\[
\text{DFD is-decomposition-of Process has } D\text{-Number}
\]

**Occurrences**

[k8] Each Low-level DFD contains at most six Processes.

\[
\text{FOR-EACH } d \text{ IN Low-level-DFD HOLDS NUMBER-OF Process PART-OF } d \leq 6
\]

[k9] Each Low-level DFD contains at most six Data stores.

\[
\text{FOR-EACH } d \text{ IN Low-level-DFD HOLDS NUMBER-OF Data-store PART-OF } d \leq 6
\]
1. Finish ERD

The main distinction between task 1 and the previous data modelling tasks concerns the attention paid to attributes. Furthermore, Relationships are required to have a Name after having performed this task. This results in a number of verification rules, in addition to rules d2 up to and including d5 and i1 and i2.

Figure A.6 shows the verification rules l1 up to and including l4 which can be represented graphically.

Labelling

[l1] Each ERD Object has a Name.

Connections

[l2] Each Object type is described by a Data element.

[l3] Each Data element describes an Object type.

[l4] Each Object type is identified by a Data element.
Appendix B

Description case 1

This appendix describes case 1. Both the problem context, i.e. the organisation involved, and the problem statement are presented. The case concerns the Dutch Open University, and we refer to this organisation by its acronym OU.

B.1 Problem context: the Open University

Since 1985, it has been possible to take courses at the OU. The OU aims at developing and providing for remote education, at an academic level. The term “remote” refers to the fact that examinations are decentralised. The OU offers study opportunities for students who work in the day-time. Students undergo examinations at study centres in the neighbourhood of their residence. The OU has eighteen study centres in the Netherlands and six study centres in Belgium. Contrary to regular universities, no specific previous education is required to obtain the “doctorandus” degree at the OU. In 1990, approximately 60,000 students attended courses at the OU. The OU is a growing organisation. Each year, more courses are given and more students attend the courses.

The main office of the OU is situated in Heerlen, in the Netherlands. Several functions are performed centrally. The main office is, among others, responsible for course development, student administration, individual result administration, and the planning and organisation of the examination process.

B.2 Problem area: planning and organisation

The problem area involves the planning and organisation of the examination process. We first discuss some characteristics of the examination process as such, then we focus on the planning and organisation. The OU gives about 120 courses, which are examined three times a year (figures from 1990). Several types of examination exist. Normally, examinations are written, however, specific student groups such as students with a visual handicap, undergo oral examinations. Some courses may also be examined by multiple
choice using a computer. Examinations are held at various locations, in addition to the regular study centres. Typical examples include prisons, navy ships, embassies, consulates, and the University of the Antilles. Summarising, the examination process is characterised by several types of examination, and by a variety of locations.

Consequently, the planning and organisation of the examination process is a complex process in itself. Many external parties are involved. At the moment, the planning and organisation is carried out by one person, supported by a manual planning chart. This makes the entire OU dependent on one person.

It is believed that a (partially automated) planning support system may reduce this dependency, and might make the planning and organisation of the examination process more reliable. Such a planning support system should control all planning and organisation activities. It should deliver relevant information to all external parties involved, and it should provide for insight into the operational employees’ workload, both at organisation unit level and at employee level.
Appendix C

Description case 2

This appendix describes case 2. The problem context (i.e., the organisation involved), the problem area, the problem statement itself, and an impression of the solution of this problem as perceived by the problem owner are part of this description.

C.1 Problem context: the GAK

The case concerns the Dutch Industrial Insurance Administration Office (the Dutch acronym GAK stands for Gemeenschappelijk AdministratieKantoor). In the Netherlands, employers can be members of an industrial insurance board. Among other things, these industrial insurance boards are concerned with carrying out social insurances for their members.

Eighteen out of the twenty-three Dutch industrial insurance boards have contracted out these activities to the GAK. The primary function of the GAK is to carry out the requirements of several laws within Dutch social legislation. Several activities are carried out as part of the primary process, of which the most important are: collecting insurance contributions from employers, assessing the entitlement to national insurance benefits and the amount of national insurance benefits to be paid to employees, paying national insurance benefits to employees, administrating these benefits, and inspecting and looking after the medical situation of employees. The last activity applies to industrial disability insurance in particular.

The GAK has a hierarchical organisation structure, with one head office, about thirty district offices and about 400 local offices. Figure C.1 shows schematically the financial flow between the GAK and the parties in its environment.

Application of the information paradigm, see [Brussaard and Tas, 1980], will provide more insight into the role of the GAK in relation to its environment. According to this paradigm, any system can be modelled, consisting of a real system (RS) and an information system (IS), where the RS sends messages about its state to the IS. The IS interprets these messages. Based on this interpretation, the IS sends messages to the RS that changes the state of the RS. In short, the IS controls the RS.
In these terms, the Dutch working population, consisting of employers and employees, constitutes the RS, as figure C.2 shows. Messages from the Dutch working population are sent to the GAK, containing information on changes in the situation of employers and
employees. An example would be the message that an employee has fallen ill, the GAK registers this information, and based on this information, the GAK decides whether national insurance benefits should be paid.

The information to be registered to facilitate the decision making by the GAK is organised into two types of registrations: object registrations and subadministrations. The GAK distinguishes three different object registrations, containing information on employees, employers and employments, respectively. For each national insurance law, a subadministration is distinguished. Subadministrations contain information on the specific situation of employees, as far as relevant for that specific national insurance law, and on the payments to employees.

Figure C.3: Entity-relationship model, indicating the relationship between object registrations and subadministrations

Figure C.3 shows the relationship between object registrations and subadministrations, using the entity-relationship modelling technique according to [Chen, 1976]. This figure also shows how parties in the GAK environment (see figure C.1) relate.

Table C.1 shows the size of the object registrations and of the most important subadministrations (in 1990). The object registrations, named by the Dutch acronyms BRP
(BasisRegistratie Personen), BRDV (BasisRegistratie DienstVerbanden) and BRWG (BasisRegistratie WerkGevers), relate to the registrations of employees, employments and employers, respectively. The Dutch acronyms wW (Werkloosheidswet), aOW/wAO (Algemene Ouderdomswet/Wet op de ArbeidsOngeschiktheid) and zw (Ziektewet) relate to different national insurance laws, and are used to refer to some of the subadministrations.

<table>
<thead>
<tr>
<th>Object registration</th>
<th>Number of attributes</th>
<th>Number of instances (× 1,000)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BRP</td>
<td>20</td>
<td>5,300</td>
</tr>
<tr>
<td>BRDV</td>
<td>20</td>
<td>4,500</td>
</tr>
<tr>
<td>BRWG</td>
<td>60-80</td>
<td>200</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sub-administration</th>
<th>Number of attributes</th>
<th>Number of instances (× 1,000)</th>
</tr>
</thead>
<tbody>
<tr>
<td>WW</td>
<td>200</td>
<td>250</td>
</tr>
<tr>
<td>aOW/WAO</td>
<td>200</td>
<td>500</td>
</tr>
<tr>
<td>ZW</td>
<td>100</td>
<td>2,500</td>
</tr>
<tr>
<td>others</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table C.1: Size of object registrations and the most important subadministrations, in terms of number of attributes and number of instances

C.2 Problem area: the Department of Statistics and Research

The problem area is the GAK Department of Statistics and Research. An information system is being developed on behalf of this department. This information system is called ssS (BasisSysteem Statistiek). We first present the ssS in relation to the GAK and its environment, then we focus on its structure.

The purpose of the ssS is the delivery of various types of statistic reports for different GAK clients. The Central Statistical Office (CBS), the National Insurance Council and various Ministries are among these clients. Two types of statistic reports can be distinguished: periodic reports and research reports. Periodic reports are triggered by the ending of a period of time. Research reports are triggered by a client request. Examples of a client request are: “deliver an analysis on the effects of a certain insurance contribution increase”, or: “deliver an analysis on absenteeism per industry”. These examples illustrate that the clients can have complex information needs involving relationships between groups of employees and/or employers.

Again, the information paradigm is applicable to describing the GAK in relation to the clients in its environment. Applying the “recursion principle” of the information paradigm
results in figure C.4, which shows the RS-IS combination as described in the previous section (see figure C.2) as the new RS, whereas the clients are to be seen as the IS now. From this RS, statistic reports are sent to the IS. Based on these reports, the IS sends messages to the RS causing its state to change. An example of such a message is: the criteria that have to be used to assess one's entitlement to a national insurance benefit are changed.

Figure C.5 shows the structure of the bss in terms of the dataflow diagramming technique from [Yourdon, 1989]. It is clear that object registrations and subadministrations constitute the input for the bss.

Several statistic files are created which in turn are used for the delivery of reports for GAK clients. Statistic files relate either to an object registration or to a subadministration.

For each object registration, one statistic file exists in the bss which can be regarded as an extraction of the object registration database. In the case of the object registration BRF, the statistic file sorc (Statistiek en Onderzoek PersoonsGegevens) is created.

For each subadministration, two statistic files in the bss can be distinguished that are extractions from the subadministration databases. In the case of the AOW/WAO subadministration for example, these are the aosr (ArbeidsOngezichtheid Situatie- en Rechtsgegevens) and the aodis (ArbeidsOngezichtheid DIStributiegegevens) files. The prefix ao- indicates the specific national insurance law, whereas the suffixes -sr and -dis indicate the two types of statistic files to be distinguished for each national insurance law. The -sr files contain records corresponding to the entity type "Situation" from figure C.3. These records hold several parameters describing a specific situation, based on which the amount of the entitled payment can be calculated. The -dis files contain records corresponding to the entity type "Payment" from figure C.3. These files indicate for each month the real amounts of
national insurance benefits paid, and the number of days within that month a person has been entitled to those benefits.

C.3 Problem statement

The GAK Department of Statistics and Research acts within the case as the problem owner. The problem at hand is that the identification of employees in the object registration and the identification of situations in the subadministrations change. We will go into these two identification mutations separately.

Employees in the BRR are identified by one key attribute, the sofi-number (social fiscal number). For each employee, the sofi-number is determined by the Inland Revenue. As soon as employees are entitled to a national insurance benefit, the GAK requests verification of their sofi-number by the Inland Revenue. In some cases, the Inland Revenue changes the sofi-number in the verification phase. Therefore, sofi-numbers may change, while the national insurance benefits for that employee continue.

Situations in the subadministrations are identified by the sofi-number. This is not a sufficient identification as such, because one employee can be entitled to benefits of a national insurance law several times during his lifetime. Therefore, a situation is identified by the
combination of a sof-number and an administration number. For each situation in a sub-
administration, the administration number is determined by a district office. The way this
administration number is built up varies over the national insurance laws. In the ww sub-
administration, each district office uses random numbers of six digits. The administration
number in the aow/wao subadministration consists of an employee family name code, the
employee's birth date and a letter. In the zw subadministration, the administration number
is built up by means of the employer number and a number of eight digits. When employees
move to a part of the country that comes under another district office, their administra-
tion number may sometimes have been assigned to another employee in that district office.
Therefore, administration numbers may change, while the national insurance benefits for
that employee continue.

Summarising, both the key attribute sof-number and the key attribute administration
number may change. As a consequence, the GAK Department Statistics and Research
has a problematic situation. The sources for the statistics to be made, i.e. the object
registrations and the subadministrations (see figure C.5), apparently suffer from incorrect
key attributes. With respect to the bss to be developed, the following requirements were
formulated:

1. Integrity of the statistic files in the bss.
2. Ability to supply the GAK clients' complex information needs.
3. Easy incorporation in the bss of future object registrations and subadministrations.

C.4 A solution

All statistic files have to remain in existence for at least five years. This has resulted in a
number of history files corresponding to a memory space of about 60 Gigabyte. Therefore,
the solution to change the key fields of all records in the statistic files based on key mutations
in the object registrations and the subadministrations, will imply very time intensive update
processes. This solution is considered to be unrealistic.

The solution proposed by the GAK essentially amounts to the introduction of a new statis-
tic file in the bss for each national insurance law, the -vb files (VerwijzingsBestand), which
are intended to serve as an index. A new, internal number is introduced by the GAK De-
partment Statistics and Research, the sko-number (Statistiek & Onderzoek). This number
consists of a national insurance law identification and a number which identifies a situa-
tion. The records in the -vb files are identified by this new sko-number. For each record,
two fields are filled with the present sof-number and the present administration number.
These two attribute fields are updated each time the sof-number and the administration
number changed in the previous month. The sko-number also acts as an identification for
the -sr and the -bris files. In this way, a suitable identification for the extractions of the
subadministrations can be obtained. The -vb files keep track of the relationships between
old and new sof-number and administration numbers on the one hand, and the sko-number
on the other hand. As part of the monthly delivery of records from the object registrations, both the old and the new sofi-number are delivered. As part of the monthly delivery of records from the subadministrations, both the old and the new administration number are delivered. Furthermore, an indication is delivered, stating for each record whether it involves a new situation, or an existing situation with a changed key attribute. Only new situations are assigned a new s&c-number.

Because of the uniqueness of this new internal number, the statistical files do not suffer from key instability. Furthermore, future subadministrations can easily be dealt with because of the national insurance law identification which is part of the s&c-number.
Appendix D

Mathematical notations

In this appendix the mathematical notations used in this thesis will be described briefly.

D.1 Functions

A *partial function* $f$ from $A$ to $B$ is defined by $f : A \to B$. Formally, $f \subseteq A \times B$ such that $(a,b) \in f \land (a,c) \in f \Rightarrow b = c$. This property allows for writing $f(a) = b$ instead of $(a,b) \in f$. In this thesis a notational shorthand for denoting functions is used. As an example consider a function $f, f = \{(p,a),(q,b)\}$. This function is denoted as $\{p : a, q : b\}$.

If $f$ is a partial function, then applying the *overriding union* (see [Meyer, 1990]) operator results in another partial function $f \uplus \{a : b\}$ defined by:

$$f \uplus \{a : b\} = \{a : b\} \cup f \setminus \{x : y \in f \mid x = a\}$$

The function $f \uplus \{a : b\}$ therefore behaves the same as function $f$ except that it assigns $b$ to $a$.

The following abbreviations are used:

- $f(a)\downarrow \equiv \exists b \in B \ [f(a) = b]$
- $f(a)\uparrow \equiv \lnot f(a)\downarrow$
- $\text{dom}(f) \equiv \{a \in A \mid f(a)\downarrow\}$
- $\text{ran}(f) \equiv \{b \in B \mid \exists a \in A \ [f(a) = b]\}$

A *total function* $f$ from $A$ to $B$ is defined by $f : A \to B$. Formally, $f : A \to B$ for which $\text{dom}(f) = A$. The function $f[A']$ is the function $f$ restricted to a subdomain $A' \subseteq \text{dom}(f)$. This function is defined by:

$$f[A'] = \{a : b \in f \mid a \in A'\}$$
D.2 Sets

The powerset of a set $A$, i.e. the set of all subsets of $A$, is denoted as $\mathcal{P}(A)$. The set of all finite subsets of $A$ is given by $\mathcal{P}^\text{fin}(A)$. The set of all finite tuples of elements from $A$ is denoted by $A^*$, while the set of all non-empty finite tuples of elements from $A$ is denoted by $A^+$. The $i$-th element of a tuple $\langle a_1, \ldots, a_i, \ldots, a_n \rangle$, i.e. $a_i$, can be found by projection: $\langle a_1, \ldots, a_i, \ldots, a_n \rangle_{<i>}$. The length of a tuple, i.e. its number of elements, can be found as follows: $|\langle a_1, \ldots, a_i, \ldots, a_n \rangle| = n$. The set $B^A$ denotes the set of all (total) functions from $A$ to $B$, i.e.:

$$B^A = \left\{ f \in \mathcal{P}(A \times B) \mid f : A \rightarrow B \right\}$$

Other notational abbreviations are:

$$A \subset B \equiv A \subseteq B \land A \neq B$$
$$A \nsubseteq B \equiv \neg A \subset B$$
$$A \setminus B \equiv \{ a \in A \mid a \notin B \}$$
Bibliography


Bibliography


Bibliography


[Bibliography]


256


258


Index

◦ 194
Ω 172
  formal definition of 172
abstraction 5
activity type 51
ADD 42
Add_constr 194
Add_ent 187
Add_gen 193
Add_lab 187
Add_rel 187
Add_role 191
Add_sch 187
Add_seq 187
Add_set 187
Add_spec 193
Add_val 193
additional modelling concepts 159
AND 39
AND-ALSO 38
ANY-REPETITION-OF 38
application level 16
application model 18
applied modelling knowledge 11
ASSOCIATED-WITH 38
associative object type indicator 59
Bachman diagram 8
Base 169
BE 39
behavioral model 64
BUT-NOT 38
C 169
cartesian product 181
CASE shell 17

CASE tool 1
  benefits of, 1
CDIF 23
  checking 5
  Code introduction 197
  Code removal 197
  compatibility 177
  completeness 212
  complex concept 24
  Comp.Join.rel 195
  Comp.Nest.rel 195
  Comp.Proj.rel 195
  Comp.Select.rel 194
  Comp.Uni.rel 194
  Comp.Unn.rel 195
  component view 8
  COMPRISING 38
  concatenation 34
  concept oriented text analysis 52
  concept structure 23
  conceptualisation 2, 53
  consistency characteristic 6, 41
  CONTAINING 38
  context diagram 65
  context free grammar 24
  contingency factor 12
  control event 64
  control flow 62
  control process 62
  coordination 19
  correlation 38
  create 43
  current situation 23

D 172
D 170
data element 58
data flow 59
data flow group 62
data modelling strategy 162
    noun driven, 162
    object driven, 162
    process driven, 162
data store 59
decision 5, 40
    non-terminating, 40
    terminating, 40
degeneration 196
    strong, 196
    weak, 196
Del.constr 194
Del.ent 187
Del.gen 193
Del.lab 187
Del.rel 187
Del.role 191
Del.sch 187
Del.seq 187
Del.set 187
Del.val 193
DELETE 42
Dep.enumeration 176
Dep.equal 176
Dep.exclusion 176
Dep.frequency 176
Dep.member 176
Dep.objects 175
Dep.roles 175
Dep.rules 175
Dep.setcard 176
Dep.setcover 176
Dep.setexclusion 176
Dep.subexclusion 176
Dep.subset 176
Dep subtotal 176
Dep.total 176
Dep.unique 176
dependency 174
    formal definition of, 174
Described object type introduction 200
Described object type removal 200
DFD 59
DFD Element 62
dialogue flow 62
difference 177
different graphical notations 159
diverging flow 62
Dom 170

\( \mathcal{E} \) 169
early stages 10
effective information modelling support 15, 18
elicitation 47
Elt 170
entity structure 8
enumeration 170
enumeration constraint 31
environmental model 64
equal 170
equality constraint 33
EQUALS 39
equivalence 168
ERD 57
ERD-Object 59
error 175
essential model 64
event list 64
event partitioning approach 23
exclusion 170
exclusion constraint 33
expert 46
extension 180

\( \mathcal{F} \) 169
flexibility 19
flow-oriented event 64
FOR-EACH \( x \) IN \( \mathcal{R} \) HOLDS \( P \) 39
form conversion 5
formal approach 168
fragment 51
Fragment Toolkit 51
framework 3
Index

frequency 170
Fresh.objs 177
Fresh.roles 177
Fresh.rules 177

G 169
gen 170
Gen 170
Generalisation 28
generalisation hierarchy 28
Generalisation Rule 173
genericity 168
graphical constraint 30
    syntax of, 171

here 261
heuristic 13

identification 59
implementation 2
IN 38
INCLUDES 39
increasing structuredness 162
individual 18
inevitable interviews 165
information engineer 16
information engineer dependent 161
Information Engineering 22
information interdependency 6, 41–42, 162
information model 3
information modelling 3
information modelling support 10
information structure 24
information system 1
information systems development 1
    methods for, 1
information systems development method 22
initial ERD 67
initial item 40
instantiation 15
interpretation mechanism 18
interviewing 47
INVOLVED-IN 37
IPSE 14

IS-NAME-OF 37
IsPop 172
join 180
Join_rel 189
keywords 37

L 169
LET 39
level of abstraction 15
LISA-D 30
lower CASE tool 14

managerial tasks 4
MAX 38
member 170
membership constraint 33
meta-knowledge base 18
meta-level 16
meta-model 16
    equivalent, 168
    formal definition of, 169
    generic, 167
    individualised, 168
meta-model editor 18
meta-model transformation 168
    basic, 185
    Object, 185
    constraint, 193
    genericity of, 200
    composed, 194
    genericity of, 202
meta-modelling technique 16, 23
    comprehensibility of, 163
    evaluation of, 163
MetaPlex 17
MetaView 17
method engineer 16
method variant group 22
methodology jungle 1
MIN 38
model 2
model type 3
model type variant 156
model view 8
modelling 2
modelling context 6
modelling knowledge acquisition 46
modelling process history 18
modelling technique 3
modifying 43

nest 183
Nest.rel 189
NIAM 8, 23
NO 39
NOT 38
NUMBER-OF 38

O 172
O 169
Object abstraction 27
object composition 29
object registration 241
Object structure 25
  approach to comparing, 156
  formal definition of, 169
Object type 26
  dependent, 174
  described, 199
  generalised, 28
Object-type 58
objectification 26
Objects 175
occurrence frequency constraint 32
OF 37
operational tasks 4
opportunism 161
opportunistic control 13
OPRR 23
OR 39

P 172
P 169
PART-OF 38
partitioning 195
pater familias 28
  formal definition of, 170
Pop 172

population 15
  of a Label type, 172
  of a Relationship, 173
  of a Schema type, 173
  of a Sequence type, 173
  of a Set type, 173
  of a meta-model, 173
  of an Entity type, 173
  of an Object structure, 172
predicate 39
preliminary DFD 67
preparation 47
prescribed modelling knowledge 11
problem domain dependent 161
problem solving process 2
problem specification 2
process 59
process modelling strategy 161
  data driven, 161
  input driven, 161
  output driven, 161
process oriented view 2
process specification 62
product oriented view 2
product view 7
productivity 1
Proj.rel 189
projection 179
protocol analysis 47
protocol transcript 51
PSM 25
  expressive power, 24

quality 1

R 172
RAMATIC 17
rationale 52
READ 42
real-time system 23
Rel 170
relational algebra 175
relational expression 175
  compatibility of, 177
task types 5
temporal event 64
terminator 59
text-based model 51
THAT 38
theory level 16
time-intensive interpretation 164
tiring elicitation 164
top-down approach 23
total 170
total role constraint 32
transferability 18
transitive closure 38
trigger 40
type coercion 38
type related 182

Uni.lab 193
Uni.rel 189
union 177
unique 170
uniqueness constraint 32
Unn.rel 189
unnest 181
    strong, 182
update statement 42
upper CASE tool 14
user implementation model 64
using 43

Val[r](Pop) 175
variable 39

way of controlling 4, 8
way of modelling 3-4
way of supporting 4, 10
way of thinking 4, 9
way of working 4-5
WITH 37
workbench 1

Yourdon 22
    generic meta-model of, 205
## Author Index

<table>
<thead>
<tr>
<th>Name</th>
<th>Page Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alanis, M.</td>
<td>14, 252</td>
</tr>
<tr>
<td>Antill, L.</td>
<td>12, 260</td>
</tr>
<tr>
<td>Araujo, T.</td>
<td>15–16, 249</td>
</tr>
<tr>
<td>Ash, C.J.</td>
<td>24, 251</td>
</tr>
<tr>
<td>Assche, F.J.M. van</td>
<td>5, 10, 257</td>
</tr>
<tr>
<td>Avison, D.E.</td>
<td>1, 9, 12, 249, 260</td>
</tr>
<tr>
<td>Aytom, P.</td>
<td>47–48, 261</td>
</tr>
<tr>
<td>Ballay, J.M.</td>
<td>13, 249</td>
</tr>
<tr>
<td>Balzer, R.</td>
<td>11, 258</td>
</tr>
<tr>
<td>Bansler, J.P.</td>
<td>12, 249</td>
</tr>
<tr>
<td>Batra, D.</td>
<td>48, 249</td>
</tr>
<tr>
<td>Bekke, J.H. ter</td>
<td>24, 249</td>
</tr>
<tr>
<td>Belady, L.</td>
<td>11, 249</td>
</tr>
<tr>
<td>Benyon, D.</td>
<td>12, 249</td>
</tr>
<tr>
<td>Bergsten, P.</td>
<td>17, 249</td>
</tr>
<tr>
<td>Bhubuta, L.</td>
<td>10, 257</td>
</tr>
<tr>
<td>Bödker, K.</td>
<td>12, 249</td>
</tr>
<tr>
<td>Bomans, G.</td>
<td>13, 249</td>
</tr>
<tr>
<td>Bommel, P.</td>
<td>28, 31, 195, 250</td>
</tr>
<tr>
<td>Bots, P.W.G.</td>
<td>2, 19, 39, 250–251</td>
</tr>
<tr>
<td>Bouwman, M.J.</td>
<td>48, 250</td>
</tr>
<tr>
<td>Brickhill, C.J.</td>
<td>24, 251</td>
</tr>
<tr>
<td>Brinkkemper, S.</td>
<td>12, 15, 23, 195, 250, 257</td>
</tr>
<tr>
<td>Brooks, R.E.</td>
<td>48, 250</td>
</tr>
<tr>
<td>Brooks Jr, F.P.</td>
<td>11, 250</td>
</tr>
<tr>
<td>Brough, M.</td>
<td>15–16, 250</td>
</tr>
<tr>
<td>Brussaard, B.K.</td>
<td>9, 122, 239, 250</td>
</tr>
<tr>
<td>Bubenko, J.A. 2</td>
<td>9, 11–12, 17, 249–250</td>
</tr>
<tr>
<td>Buchanan, B.G.</td>
<td>47, 260</td>
</tr>
<tr>
<td>Burgess, K.</td>
<td>12, 252</td>
</tr>
<tr>
<td>Butler Cox 1</td>
<td>250</td>
</tr>
<tr>
<td>Carapuça, R.</td>
<td>15–16, 249</td>
</tr>
<tr>
<td>Case Jr, A.F.</td>
<td>12, 250</td>
</tr>
<tr>
<td>Chen, M. 2</td>
<td>14, 17, 251</td>
</tr>
<tr>
<td>Chen, P.P.</td>
<td>23, 241, 251</td>
</tr>
<tr>
<td>Chikofsky, E.J.</td>
<td>12, 251</td>
</tr>
<tr>
<td>Clancey, W.J.</td>
<td>47, 260</td>
</tr>
<tr>
<td>Clarkson, G.P.E.</td>
<td>48, 251</td>
</tr>
<tr>
<td>Clements, P.C.</td>
<td>11, 257</td>
</tr>
<tr>
<td>Cohen, B.</td>
<td>168, 251</td>
</tr>
<tr>
<td>Colter, M.A.</td>
<td>11, 251</td>
</tr>
<tr>
<td>Connor, D.</td>
<td>12, 251</td>
</tr>
<tr>
<td>Crossley, J.N.</td>
<td>24, 251</td>
</tr>
<tr>
<td>Curtis, B.</td>
<td>10, 253</td>
</tr>
<tr>
<td>D’Atri, A.</td>
<td>195, 251</td>
</tr>
<tr>
<td>Dahl, R.</td>
<td>17, 249</td>
</tr>
<tr>
<td>Davis, G.B.</td>
<td>12, 251</td>
</tr>
<tr>
<td>Davis, J.G.</td>
<td>48, 249</td>
</tr>
<tr>
<td>DeMarco, T.</td>
<td>22–23, 251</td>
</tr>
<tr>
<td>Dickson, G.</td>
<td>11, 48, 259</td>
</tr>
<tr>
<td>Dissen, H.G. van</td>
<td>11–12, 260</td>
</tr>
<tr>
<td>Duda, R.O.</td>
<td>47, 251</td>
</tr>
<tr>
<td>Dunn, R.</td>
<td>10, 251</td>
</tr>
<tr>
<td>Dur, R.C.J.</td>
<td>19, 251–252</td>
</tr>
<tr>
<td>Ehrlich, H.D.</td>
<td>168, 252</td>
</tr>
<tr>
<td>Eilers, H.B.</td>
<td>8, 259</td>
</tr>
<tr>
<td>Engels, G.</td>
<td>168, 252</td>
</tr>
<tr>
<td>Ericsson, K.A.</td>
<td>48, 252</td>
</tr>
<tr>
<td>Essink, L.J.B.</td>
<td>12, 252</td>
</tr>
<tr>
<td>Everest, G.C.</td>
<td>14, 252</td>
</tr>
<tr>
<td>Fairley, R.E.</td>
<td>12, 252</td>
</tr>
<tr>
<td>Falkenberg, E.D.</td>
<td>15, 195, 250, 257</td>
</tr>
<tr>
<td>Feigenbaum, E.A.</td>
<td>47, 252</td>
</tr>
<tr>
<td>Fiadeiro, J.</td>
<td>168, 257</td>
</tr>
<tr>
<td>Fitzgerald, G.</td>
<td>1, 9, 249</td>
</tr>
<tr>
<td>Flaatten, P.O.</td>
<td>12, 252</td>
</tr>
<tr>
<td>Floyd, C.</td>
<td>9, 252</td>
</tr>
<tr>
<td>Fox, J.</td>
<td>48, 256</td>
</tr>
<tr>
<td>Gane, C.</td>
<td>22–23, 252</td>
</tr>
</tbody>
</table>

268
Author Index

Gogolla, M. 168, 252
Gordon, C.L. 1, 256
Grabowski, M. 47, 252
Greaves, M.F. 48, 256
Grochow, J.M. 14, 252
Guindon, R. 10, 13, 253
Gustafsson, M. 17, 249

Hagelstein, J. 5, 10, 257
Halpin, T.A. 23, 195–197, 199, 203, 253, 256
Hanson, B. 11, 258
Harst, G.M.A. van der 51, 55, 90, 164, 253
Hart, A. 46, 253
Harley, D.J. 22–23, 253
Hayes-Roth, F. 47, 253
Heijes, H. 51, 253
Heunen, L.J.G.T. van 195, 257
Herlé, J.E. 51, 53, 100, 164, 253
Hershey, E.A. 17, 258
Heyn, M. 15–16, 253
Hice, G.F. 8, 259
Hohenstein, U. 168, 252
Holloway, S. 22, 254
Hull, R. 24, 254
Hülsmann, K. 168, 252

Imber, M. 15–16, 23, 255
Jackson, M.A. 29, 255
Jajodia, S. 195, 255
Johansson, L-Å. 17, 249

Kamp, G. 11, 258
Kensing, F. 8–9, 255
King, R. 24, 254
Kobayashi, I. 195, 255
Kumar, K. 11–12, 260
Kusters, R.J. 14, 255

Landry, M. 11, 255
Langerhorst, R.P. 8, 259
Larkin, J.H. 48, 255
Ledderhof, H.J.A. 31, 255
Lenat, D.B. 47, 253
Levy, A. 29, 255
Löhr-Richter, P. 168, 252
Lyytinen, K. 14–16, 23, 258

Macasovic, P. 17, 258
Macdonald, I.G. 5, 10, 257
Malouin, J.L. 11, 255
Marca, D.A. 5, 255
Martin, C.F. 2, 255
Martin, J. 12, 255
Martti, P. 14–16, 23, 258
Maryanski, F. 24, 257
Mathiassen, L. 9, 12, 255–256
McAllister, A.J. 17, 258
McClure, C. 2, 14, 256
McCubbrey, D.J. 12, 252
McDermott, J. 48, 255
McGowan, C.L. 5, 255
McMenamin, S. 22–23, 256
Meersman, R. 28, 168, 257, 259
Mellor, S.J. 22–23, 260
Meyer, B. 247, 256
Mitchell, A.A. 48, 256
Mitroff, I.I. 2, 256
Motshagen, P.A. 19, 256
Munk-Madsen, A. 12, 255
Myers, C.D. 48, 256

NGGO 14, 195, 256
Necco, C.R. 1, 256
Newell, A. 48, 256
Ng, P.E. 195, 255
Nieuwland, E.R. 24–25, 39, 57, 254
Nijssen, G.M. 23, 256
Norman, R.J. 14, 257
Nunamaker, J.F. 2, 14, 251, 257
Nunamaker Jr, J.F. 17, 251

O’Riordan, P.D. 12, 252
Oei, J.L.H. 195, 257
Olle, T.W. 5, 8, 10–11, 257
Oosterom, N.E. van 8, 23, 257, 260

269
Österle, H. 15–16, 253
Palmer, I. 12, 255
Palmer, J. 22–23, 256
Parnas, D.L. 11, 257
Peckham, J. 24, 257
Pegram, S.M. 48, 256
Pirbhai, I.A. 22–23, 253
Pong, L. 168, 259
Proper, H.A. 24, 33, 168, 254
Rolland, C. 5, 10, 257
Saake, G. 168, 252
Sacca, D. 195, 251
Sarson, T. 22–23, 252
Seligmann, P.S. 4, 257
Sernadas, A. 168, 257
Sernadas, C. 168, 257
Shomenta, J. 11, 258
Shortcliffe, E.H. 47, 251
Simon, D.P. 48, 255
Simon, H.A. 48, 252, 255–256
Simpson, B. 11, 258
Skidmore, S. 12, 249
Smolander, K. 14–16, 23, 258
Sol, H.G. 1–5, 8–12, 19, 46, 257–258
Sommerville, I. 34, 258
Sorenson, P.G. 17, 258
Springsteel, F.N. 195, 255
Stillwell, J.C. 24, 251
Stomph-Blessing, L.T.M. 13, 258
Swartout, W. 11, 258
Tahvanainen, V-P. 14–16, 23, 258
Tas, P.A. 9, 122, 239, 250
Teichroew, D. 17, 258
Tremblay, J.-P. 17, 258
Tromp, Th.J.M. 48, 258
Troyer, O.M.F. de 28, 259
Tsai, N.W. 1, 256
Tse, T.H. 168, 259
Tully, C.J. 11, 257
Turner, W.S. 8, 259

Uijttenbroek, A.A. 8, 259
Verhoef, T.F. 25, 55, 57, 64, 77, 119, 131, 141, 164, 254, 259
Verlinden, P. 28, 259
Verrijn-Stuart, A.A. 5, 10–11, 257
Vitalari, N. 11, 13, 47–48, 259
Voelcker, J. 14, 259
Vonk, R. 2, 260
Ward, P.T. 22–23, 260
Waterman, D.A. 47, 253
Weber, E.S. 2, 14, 251
Weelden, J.A. van 19, 260
Welke, R.J. 11–12, 23, 260
Wilkins, D.C. 47, 260
Williams, N.H. 24, 251
Wintraecken, J.J.V.R. 8, 30, 260
Wood-Harper, A.T. 12, 260
Wright, G. 47–48, 261
Yamamoto, Y. 17, 258
Yourdon, E. 12, 22–23, 26, 30, 44, 54, 57–59, 61, 64, 66, 68, 72–75, 155, 164, 214, 217, 243, 261, 273

270
Samenvatting

Probleemstelling

Het ontwikkelen van informatiesystemen wordt gezien als een complex en slecht-gestructureerd proces van probleempoplossen. Om de uitvoering van zo'n proces te ondersteunen zijn diverse methoden, technieken en geautomatiseerde hulpmiddelen, zoals CASE-tools, geïntroduceerd. Methoden, technieken en CASE-tools zijn onderwerp van dit proefschrift, waarbij nadruk ligt op de vroege fasen in een systeemontwikkelingstraject, en daarbinnen op de uitvoerende activiteiten.

De huidige methoden en technieken kampen met verscheidene tekortkomingen: concepten zijn soms onduidelijk of kunstmatig en gedetailleerde richtlijnen voor de uitvoering ontbreken veelal. Ervaringen informatiekundigen hebben vanuit hun praktijkervaring met dergelijke onvolkomenheden leren omgaan, en hun eigen invullingen ontwikkeld.

De huidige geautomatiseerde hulpmiddelen zijn als produktgericht te karakteriseren. Ze bevatten geen kennis over het te ondersteunen proces. Daarmee blijft hun bruikbaarheid beperkt tot hulpmiddelen voor documentatie en verificatie.

In de laatste jaren wordt er steeds meer aandacht besteed aan de vraag hoe de (geautomatiseerde) ondersteuning van modelleertaken verbeterd kan worden. Hierbij staat het idee centraal dat een goede aansluiting van geautomatiseerde hulpmiddelen bij het te ondersteunen proces van wezenlijk belang is voor effectief gebruik van die hulpmiddelen. Gedetailleerde kennis over de modelleer- en werkwijzen zoals toegepast door informatieïndigen is daarbij belangrijk. Deze ideeën hebben geleid tot de ontwikkeling van zogeheten CASE-shells, geautomatiseerde hulpmiddelen waarbinnen beschrijvingen van modelleerkennis expliciet beschikbaar dienen te zijn. Informatiekundigen kunnen dan ondersteund worden op basis van beschrijvingen, die toegesneden zijn op hun eigen, individuele behoeften, en die eenvoudig aangepast en uitgebreid kunnen worden. Beschrijvingen van modelleerkennis worden in deze benadering meta-modellen genoemd, technieken die gebruikt worden om deze beschrijvingen te maken heten meta-modelleringstechnieken.

De meta-modelleringsbenadering heeft zich gericht op flexibele ondersteuning van individuele informatiekundigen. In dit proefschrift worden twee nadelen van deze benadering gesignaleerd. Vanuit het oogpunt van de organisatie waarvoor een informatiekundige werkt kan het ondersteunen van individuele behoeften van informatiekundigen strijdig zijn met de
Samenvatting

behoeft aan standaardisatie vanuit de organisatie als geheel. Overdraagbaarheid van resultaten tussen informatiekundigen kan belemmerd worden door een variëteit aan individuele modeller- en werkwijzen te ondersteunen. Een tweede nadeel is dat het specificeren van een gedetailleerde modellerkennisbank om één individuele informatiekundige te ondersteunen een arbeidsintensieve taak is. Er is sprake van effectieve modellerondersteuning als (i) het mogelijk is om delen van modeller- en werkwijzen te standaardiseren, en (ii) het mogelijk is om delen van modeller- en werkwijzen naar individuele behoeften aan te passen en te verfijnen.

Om een meer effectieve modellerondersteuning te bereiken is het in de eerste plaats gewenst om een gedetailleerd inzicht te verkrijgen in de mate waarin en de wijze(n) waarop modellerkennis zoals toegepast door informatiekundigen in de praktijk afwijkt van modellerkennis zoals beschreven in methodehandboeken. Een dergelijk inzicht kan leiden tot aanwijzingen voor het aanpassen en verfijnen van generieke modellerkennis naar individuele behoeftens. Dit proefschrift onderzoekt:

1. of het mogelijk is een gedetailleerd inzicht te verkrijgen in overeenkomsten en verschillen tussen voorgeschreven en toegepaste modellerkennis;

2. of het mogelijk is mechanismen te ontwikkelen die de verbijzondering van generieke modellerkennis naar individuele behoeften ondersteunen.

Inzicht in voorgeschreven en toegepaste modellerkennis

Aanpak

Als onderdeel van de te volgen aanpak om een gedetailleerd inzicht te verkrijgen in overeenkomsten en verschillen tussen voorgeschreven en toegepaste modellerkennis, worden drie essentiële keuzen gemaakt: (i) een beperking tot inzicht in beschrijving en toepassing van één - representatieve - methode voor informatiesysteemontwikkeling, (ii) een keuze voor een techniek waarmee modellerkennis geregistreerd kan worden, en (iii) een keuze voor een aanpak om modellerkennis te verkrijgen, zoals toegepast door ervaren informatiekundigen uit de praktijk.

Methode voor informatiesysteemontwikkeling

Bij de keuze van een representatieve methode voor het ontwikkelen van informatiesystemen spelen drie criteria een rol. De methode moet de vroege fasen in een systeemontwikkelings-traject afdekken. Verder moet literatuur beschikbaar zijn waarin de methode gedetailleerd beschreven wordt. Tenslotte moet de methode in Nederland veel toegepast worden. Deze criteria leiden tot de keuze van Yourdon.
Techniek voor representatie van modelleerkennis

Bij de keuze van een geschikte techniek voor de representatie van modelleerkennis is het van belang, dat de techniek een begrippenapparaat aanreikt waarmee een werkwijze en een modelleerwijze adequaat gereserveerd kunnen worden, alsook de samenhang tussen een werkwijze en een modelleerwijze. De meta-modelleringsmethode die in dit proefschrift gebruikt wordt bestaat uit drie essentiële componenten, namelijk (i) het Predicator Set Model (PSM), (ii) de taakstructuurdiagramtechniek, en (iii) de Language for Information Structure and Access Descriptions (LISA-D). PSM is een datamodelleringsmethode die gebaseerd is op objecten en relaties tussen objecten. Deze techniek wordt gebruikt om een modelleerwijze te representeren. De taakstructuurdiagramtechniek is een procesmodelleringsmethode met als kernbegrippen: taak, beslissing, trigger, decompositie, synchronisator en initiële taak. Deze techniek wordt gebruikt om een werkwijze te beschrijven. LISA-D is een taal waarmee verificatieregels geformuleerd kunnen worden. Verificatieregels geven aan waaraan gemaakte modellen moeten voldoen.

Aanpak voor kennisacquisitie

De aanpak voor kennisacquisitie is gericht op het verkrijgen van een gedetailleerde beschrijving van modelleerkennis zoals toegepast door informatiekundigen die als Yourdon-expert gezien worden. De gekozen acquisitieaanpak bestaat uit de taken voorbereiding, elicitation, interpretatie en conceptualisatie. De eerste taak betreft technische en organisatorische voorbereidingen, alsook voorbereiding van de expert op de uit te voeren modelleertaak. De elicitation van modelleerkennis gebeurt wanneer de expert aan de hand van een casus een modelleertaak uitvoert. De elicitationssessie kent een experimentele opzet waarbij de expert gestimuleerd wordt hardop te denken. De communicatie met gebruiker(s) vindt daarbij plaats via een computer- en videoverbinding. De elicitationstaak resulteert in een protocolltranscript. De interpretatie van het protocolltranscript bestaat uit een taakgerichte tekstanalyse en een conceptgerichte tekstanalyse. De taakgerichte tekstanalyse is gericht op het markeren van tekstfragmenten die uitgevoerde modelleertaken en beslissingen weergeven. In de conceptgerichte tekstanalyse worden tekstfragmenten gecentreerd die gebrukte modelleerconcepten betreffen. In de conceptualisatetaak wordt uiteindelijk een conceptueel meta-model opgesteld op basis van het tekstmodel.

In het onderzoek worden drie ervaren informatiekundigen betrokken, die ieder twee modelleertaken met een doorlooptijd van vier dagen uitvoeren, aan de hand van twee verschillende casus. De aanpak voor kennisacquisitie wordt dus zes keer uitgevoerd.

Resultaten

De aanpak voor kennisrepresentatie en kennisacquisitie is gericht op een gedetailleerd inzicht in modelleerkennis, zowel ten aanzien van de handboekbeschrijving uit [Yourdon, 1989] als ten aanzien van de toepassing in de zes praktijkessenties. Voor wat betreft de representatie van modelleerkennis uit Yourdon ligt de nadruk op data-flow diagrams en entity-relationship diagrams (modelleerwijze), en op de fase waarin het essential model geconstrueerd wordt (werkwijze).
Samenvatting

Ten aanzien van de *modellierwijze* in Yourdon blijkt, dat de experts *eigen varianten* van DFD’s en ERD’s gebruiken die alle afwijken van die in het handboek. Deze afwijkingen worden verklaard door het gebruik van additionele maar ook van meer verfijnde modellerconcepten. De experts gebruiken ook *andere modellerconcepten* dan die in de genoemde modelleringstechnieken. Dit wordt deels verklaard vanuit hun behoefte om kwantitatieve inzichten te verkrijgen. Ook concepten om organisaties te beschrijven worden gebruikt. Het gebruik van *grafische conventies* om modellerconcepten te presenteren varieert aanzienlijk.


Hiermee kan geconcludeerd worden dat de eerste onderzoeksvraag positief beantwoord is.

**Mechanismen voor het verbijzonderen van generieke modelleerkennis**

Op basis van het inzicht dat verschillen en overeenkomsten tussen werkwijzen deels te verklaren zijn uit verschillende modellierwijzen, wordt een beperking aangebracht bij het ontwikkelen van mechanismen voor het verbijzonderen van generieke modelleerkennis tot kennis over modellierwijzen. *Meta-modellen* waarin minder beperkingen zijn opgelegd ten aanzien van een modellierwijze worden in dit proefschrift als generieker gezien. Het maken van specifieke keuzen ten aanzien van een modellierwijze leidt tot meer specifieke of geïndividualiseerde meta-modellen. Formeel wordt een meta-model gezien als meer generiek dan een ander meta-model als iedere populatie van het geïndividualiseerde meta-model ook als populatie kan dienen van het generieke meta-model.

Verschillende mechanismen (gesproken wordt van *transformaties*) voor het verbijzonderen van meta-modellen worden gedefinieerd. De keuze van de (volledige en orthogonale) set
van basistransformaties wordt bepaald door de onderliggende meta-modelleringstechniek. Deze techniek bestaat uit constraints enerzijds, en uit objecttypen en rollen anderzijds. Diverse soorten objecttypen worden onderkend, namelijk labeltypen, entiteittypen, settypen, sequentietypen, schematypen, en relatietypen. Met behulp van de set van basistransformaties is het mogelijk, constraints, objecttypen en rollen aan een meta-model toe te voegen of uit een meta-model te verwijderen. Relatietypen kunnen op verscheidene manieren gemanipuleerd worden. Om deze vormen van manipulatie in de set van basistransformaties te kunnen incorporeren wordt een onderliggende relationele algebra gedefinieerd. Voor ieder van deze basistransformaties wordt aangegeven hoe het schema van een meta-model verandert na uitvoering van de basistransformatie, en hoe de populatie verandert.

In de literatuur over schematransformaties zijn diverse complexe transformaties bekend. Hoewel deze tot dusverre als primitieve transformaties gepresenteerd zijn, wordt in dit proefschrift aangetoond dat deze transformaties in de gedefinieerde set van basistransformaties uitgedrukt kunnen worden.

Voor ieder van de onderkende basistransformaties wordt vastgesteld of uitvoering ervan leidt tot een meer generiek meta-model, een meer geïndividualiseerd meta-model of een equivalent meta-model. Tenslotte wordt de toepasbaarheid van de transformaties aan de hand van een aantal eenvoudige voorbeelden gedemonstreerd. Daarmee is de tweede onderzoeksvraag positief beantwoord. Door een complete en orthogonale set van mechanismen te definiëren is bovendien een basis gelegd voor de implementatie van CASE-shells die bijdragen aan een effectieve ondersteuning van informatiemodelleringstaken.
Curriculum Vitae

Dankwoord

Vele personen en instellingen hebben, vanuit verschillende invalshoeken, aan de totstandkoming van dit proefschrift bijgedragen. Dit dankwoord is gewijd aan iedereen met wie ik in de afgelopen jaren heb samengewerkt.

Allereerst wil ik mijn promotor, Henk Sol, bedanken voor zijn stimulerende en doelgerichte begeleiding. Met een schijnbaar onuitputtelijke energie voorzag hij de vele tussenversies van zijn vraagtekens, zijn suggesties en zijn constructieve kritiek. Zijn aandacht voor duidelijkheid, verantwoording en haalbaarheid is van grote waarde geweest.


Diverse partijen zijn betrokken geweest bij de experimenten. Allereerst gaat mijn dank uit naar de drie experts, Ton Akkerman (BSO/IS Arnhem, tegenwoordig KEMA), Klaas
Dankwoord

Kramer (het Informatica Instituut van Cap Gemini Pandata) en Nico Siepel (Volmac Nederland BV), voor hun deelname aan de experimenten, voor hun expertise op het gebied van informatiemondeling en voor het enthousiasme waarmee zij steeds weer tijd vrijmaakten voor bespreking en discussies. Ook de genoemde organisaties wil ik bedanken voor de bereidheid aan het onderzoek deel te nemen. Daarnaast wil ik het GAK en de Open Universiteit bedanken, in het bijzonder Alexander Garschagen (GAK), Jos Beek, Marie-Thérèse Maroofi en Henk Vos (OU) voor hun rol als gebruiker gedurende de experimenten. Martijn Berkelmans en Christine Fokkens droegen als velotypist zorg voor een voorspoedig verloop van de experimenten.

Bij het schrijven van dit proefschrift wil ik allereerst Eric Nieuwland en Erik Proper noemen vanwege hun vele tips bij het werken met \LaTeX en bij andere ongemakken. Thanks also go to Miranda Aldham-Breary for the numerous ways she pointed out I abused the English language, en naar John Bouwmans voor zijn gedegen spellingscontrole. Guido van der Harst ten slotte, bedankt voor het fraaie kaftontwerp.

Last but not least wil ik mijn familie en vrienden dank zeggen, voor het geduld in het afgelopen jaar wanneer ik weer eens niet thuis gaf. In het bijzonder wil ik mijn ouders, mijn stiefouders en mijn broer Peter bedanken voor hun permanente steun en interesse.

Pijnacker, mei 1993.