MODELLING SUPPORT

IN

INFORMATION SYSTEMS DEVELOPMENT
MODELLING SUPPORT

IN

INFORMATION SYSTEMS DEVELOPMENT

PROEFSCHRIJT

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
door het College van Dekanen daartoe aangewezen,
on donderdag 5 september 1991 te 14.00 uur

door

GERARDUS MARIA WIJERS

informatica ingenieur

geboren te Den Haag

Thesis Publishers Amsterdam

1991
Dit proefschrift is goedgekeurd door de promotor:

Prof. Dr. H.G. Sol

en door de leden van de promotiecommissie:

Prof. Dr. T.M.A. Bemelmans
Prof. Drs. B.K. Brussaard
Prof. Dr. Ir. J. van Katwijk
Prof. Dr. H. Koppelaar
Prof. Dr. Ir. M. Looijen
## CONTENTS

**PREFACE** ...................................................... vii

1 **MODELLING SUPPORT** ........................................ 1
   1.1 Methods ................................................. 1
   1.2 CASE tools ............................................. 2
   1.3 Intelligent tools ....................................... 3
   1.4 A problem solving perspective ......................... 6
   1.5 The practising information engineer ................. 7
   1.6 Intelligent support revisited ......................... 9
   1.7 Research approach and study outline ................. 10

2 **INFORMATION SYSTEMS DEVELOPMENT** .......................... 13
   2.1 Introduction ........................................... 13
   2.2 Understanding information systems development ...... 13
   2.3 Way of modelling ...................................... 15
   2.4 Way of working ........................................ 17
   2.5 Relationship between way of modelling and working .. 18
   2.6 Way of controlling ..................................... 20
   2.7 Way of thinking ....................................... 21
   2.8 Way of supporting ..................................... 24
   2.9 Automated support ..................................... 25
   2.10 Research question revisited ........................ 28

3 **REPRESENTATION OF MODELLING KNOWLEDGE** .................. 31
   3.1 Introduction ........................................... 31
   3.2 Concept structure ..................................... 32
   3.3 Verification rules ..................................... 37
   3.4 Task structure ......................................... 39
   3.5 Integrating task and concept structure ............... 46
   3.6 Summary ................................................. 57

4 **ACQUISITION OF MODELLING KNOWLEDGE** .......................... 59
   4.1 Introduction ........................................... 59
   4.2 Overview of knowledge acquisition methods ............ 59
   4.3 Acquisition approach for modelling knowledge ....... 66
   4.4 Summary ................................................. 74

5 **EXPERIMENT 1: D2S2** ........................................ 77
   5.1 Introduction ........................................... 77
   5.2 D2S2 .................................................. 77
   5.3 Conceptual meta-model of the D2S2 experiment ......... 81
   5.4 Summary ................................................. 108
6 EXPERIMENT 2: YOURDON .............................................. 113
   6.1 Introduction .................................................. 113
   6.2 YOURDON ..................................................... 113
   6.3 Conceptual meta-model of the YOURDON experiment ....... 117
   6.4 Summary ...................................................... 139

7 EXPERIMENT 3: ESM .................................................. 143
   7.1 Introduction .................................................. 143
   7.2 ESM .......................................................... 143
   7.3 Conceptual meta-model of the ESM experiment ............. 145
   7.4 Summary ...................................................... 166

8 DETAILED UNDERSTANDING OF MODELLING ....................... 169
   8.1 Introduction .................................................. 169
   8.2 Detailed modelling knowledge ................................ 169
   8.3 Adequate representation of modelling knowledge .......... 172
   8.4 Effective and efficient acquisition of modelling knowledge 178
   8.5 Conclusions .................................................. 181

9 DEVELOPMENT OF AN MSS ENVIRONMENT ........................ 183
   9.1 Introduction .................................................. 183
   9.2 Functionality of an MSS environment ....................... 183
   9.3 Prototype of an MSS environment ........................... 204
   9.4 Application of the prototype MSS environment ............ 222
   9.5 Conclusions .................................................. 223

APPENDIX A: A RESERVATION ORGANIZATION ....................... 225

REFERENCES ........................................................ 227

AUTHOR INDEX .................................................... 237

SUBJECT INDEX .................................................... 241

SAMENVATTING ..................................................... 249

CURRICULUM VITAE ................................................ 257
CASE tools are today the most prominent stimuli to the continuing improvements in information systems development methods. In this thesis, CASE tools and methods are considered mainly as tools offering partial methodical structure to information engineers in the practical problem solving process of effectively applying information technology to persons and organizations. This study concentrates on these practising information engineers. It presents an approach and environment enabling effective individual modelling (not managerial) support, based upon a detailed understanding of their way of working and modelling.

This thesis could not have been written without the help of many persons and organizations. Acknowledgements are firstly due to Henk Sol for his everlasting inspiration, confidence and involvement in this study, his talented managerial capabilities, and his continuing challenge to my intellectual capacity.

The Delft University of Technology was my scientific home base for almost four years. I thank all my former colleagues for an inspiring environment, in particular Philip Seligmann for our fruitful theoretical and methodological discussions, Ben van Tilborg for his excellent role in my experiments, and my Kongsai friends for sharing all aspects of being a PhD student in the Information Systems Department.

I would like to thank James Martin Associates, and particularly Frans Van Assche, for initiating my scientific career in general and my interest in methods, modelling techniques and CASE tools, more specifically. I am grateful to the NGGO for their efforts in finding the industrial background for my research. The NGGO demonstrated the benefits of a close interaction between the academic world and industry. I would particularly like to mention the following companies: BSO/AT Rotterdam, HighTech Automation, Siemens Nixdorf Informatiesystemen and Unisys Nederland.

Many students have been involved in my thesis; they have invested a great deal of time and energy in important parts of my research. Thanks go to Harry Heijes, Gijs Huizinga, Henk Ledderhof and Frank Schaaptherder. I implicitly thank all students who have attended the course in information systems design methods and CASE tools, enabling me to put various research ideas to the test.

Special thanks are due to the experts Louis Schijve, Johan van den Hoogenband and Elma van den Berg-Schipper who have participated in the experiments described in this thesis. I owe them greatly for the invaluable experience in information systems planning. I am grateful to their companies, CACI Nederland, CMG Den Haag and Unisys Nederland, who have given me much time of their valuable experts.
I would like to mention SERC, in particular the SOCRATES project, enabling me to continue the fascinating research into methods and tools. I remember the teamwork and constructive discussions with Sjaak Brinkkemper, Arthur ter Hofstede, and Denis Verhoef. Without them this thesis would not have been the same. I also thank Peter Weijland for his support in finishing the last parts of this thesis at SERC.

Most of all, I am aware of the duties I was forced to neglect, and the persons I used to be involved with socially. I thank Marleen for her absolutely essential love, and my parents, my family and friends for their support and patience during my study.

Gouda, July 1991

Gerard M. Wijers
1

MODELLING SUPPORT

1.1 METHODS

There are many reasons for the interest expressed in the capabilities of information systems development methods\(^1\) to systematize the information systems development process, including planning, analysis, design, implementation and maintenance activities. These reasons are generally concerned with the quality of information systems and the productivity of development processes, see Sol (1985) and Blokdijk and Blokdijk (1987). Methods claim to improve the management of the development process, cf. Butler Cox (1987), and to render them less dependent on the skills of individual information engineers responsible for information systems development - experts who are in short supply and expensive to train. Considering the perceived needs for methods as described by Avison and Fitzgerald (1988), these imply a need for better end products, better development processes, and, in addition, standardized ones.

Bubenko (1986) estimates that so far hundreds of method descriptions have been published. In practice, probably thousands of various approaches are being used. Most organizations have developed their own methods, and have declared their use as mandatory in their handbooks. In spite of these figures, only few scientific studies have reported on experiences in the use of methods in practice. We are actually quite ignorant of the applicability of methods and the validity of information systems' specifications.

Various shortcomings in methods have been reported in literature, such as fuzzy and artificial concepts, scarce guidelines, high complexity, incompleteness, and lack of quality control, see Sol (1985), Bubenko (1986) and Essink (1986). Fuzzy concepts can be found especially in the early stages of information systems development in which modelling techniques are mainly introduced by examples and some notation rules, see Brinkkemper (1990). Guidelines for model construction and intended strategies are generally only described in rough managerial terms with a neglect of quality control concepts.

The influence of the environment and the application domain on the suitability of methods is regularly overlooked, although their influence on specific information systems development processes has already been identified in Davis and Olson (1984), Sol (1984), Bemelmans (1987) and Fazlollahi and Tanniru

\(^1\)In this study we prefer to use the classical notion of "methodology" in its meaning of study of methods, also referring to Kensing (1984), Olle et al. (1988a), Wieringa and de Jonge (1989) and Andersen et al. (1990).
(1988). These contingency factors, as they are called, amongst other things
determine the tasks to be performed, the techniques to be used, and the way
users participate. Only recently, we observe a trend of different scenarios (Olle
et al. (1988a)) becoming part of one method. Evidence for this trend can be
found in Wood-Harper et al. (1985), MacDonald (1986) and Flaatten et al. (1989).
Information systems development methods can be considered as sets of tasks and
concepts, which, in any given situation, have to be reduced to approaches
uniquely adapted to prevailing situations, also see Checkland (1981) and Sol
(1988). Methods should meet quality and productivity needs by offering clear and
detailed concepts and guidelines, and by defining their applicability explicitly,
possibly by including various alternative strategies and scenarios for specific
environments or application domains, such as administrative or industrial
applications.

In practice, shortcomings such as fuzziness, incompleteness and inadequacies
have been identified in methods, and ad-hoc modifications and refinements have
been introduced in order to cure the problems. The result is often an incoherent
patchwork of sometimes extended specification techniques, together with
undocumented formal and informal guidelines, and haphazardly added tasks and
theories. This is a natural process of filling the gaps in methods, and improving
them by incorporating successful features of other methods.

The resulting situation is one in which the practical application of methods
differs significantly from its theoretical description, see Avison and Fitzgerald
(1988). The quality of the development process is largely dependent on the
quality and competence of the information engineer, see also Bubenko (1986) and

These observations clearly define our problem area: the application of
(incomplete) methods with fuzzy concepts and scarce guidelines results in the
individual interpretation and application of methods, which complicates a general
and standardized effective support of information engineers in practice.

1.2 CASE TOOLS

CASE tools, as they are called, are currently considered as the prime initiators
of attempts to formalize and standardize methods in a more detailed way, see
and Bubenko (1988) believe that these CASE tools are becoming the most
important contributors to the continuing development in information systems
development methods. Martin (1986a) introduces engineering-like methods, characterized by a coherent integrated set of techniques covering the complete development process. He argues that such methods depend on the availability of automated tools, because of the required level of consistency between various specifications, which would be difficult to verify manually. Butler Cox (1987) and Yourdon (1986) describe techniques such as entity-relationship diagrams, dataflow diagrams and structured English as tedious, time-consuming and even impractical, if not supported by automated tools.

CASE tools provide following general features: support for structured techniques by offering diagram (and text) editors, verification of specifications, storage of specifications, generation of documentation, prototyping support, automation of various specification transformations, and code generation (see Butler Cox (1987), Bubenko (1988), Vonk (1988a,b), Chen et al. (1989), McClure (1989)).

The major benefits of these tools are the documentation and verification support, the integration of various representation techniques and occasionally the opportunity of generating new specifications and code automatically. Using CASE tools implies standardization, but they are strongly product oriented, which has also been confirmed by the study of Wijers and van Dort (1990). This study among Dutch users of CASE tools concluded that they are mainly used for documentation and verification purposes. After a model has been constructed, it is specified by using an automated tool, and then verified. We suspect that this limited usage is largely due to the design of the tools not paying appropriate attention to the information engineer's needs.

Bubenko (1986) and Floyd (1986) endorse the view that automated tools support consistency in the development process, but identify the danger of computerizing unsuitable methods. Guidelines on why and how to perform various tasks, and how to determine the specification's quality, are not part of automated tools. Benyon and Skidmore (1987) suggest that before using automated tools, the range of the required facilities should be thoroughly understood. They propose an environment (automated or not) supporting the practising information engineer in the use of suitable techniques, depending on the current situation.

1.3 INTELLIGENT SUPPORT

In Sol (1985), Bubenko (1986), Knuth et al. (1986), Lockemann and Mayr (1986), Wijers and Sol (1987), Welke (1988), Vonk (1988a), Potts (1989), Chen et al. (1989) we find various research proposals which could be grouped under the
heading of "intelligent" automated support, all aiming at improving the automated support of information engineers. In the various suggestions made by these authors we observe the following areas of intelligent support: better verification possibilities of completed specifications, suggestions and advice during specification construction, explicit guidance in the development process, and enhanced facilities for redesign, transformation and integration.

Literature has so far described various software systems deserving the label "intelligent". The types of tasks supported by these systems can be classified as:

- the interpretation of restricted natural language with a predefined grammar (Kersten et al. (1986), Lubars and Harandi (1986), Proix and Rolland (1986)),

- the transformation of one structure into another, in which this transformation is normally broken up into concise and compact steps (Bouzeghoub et al. (1985), Falkenberg et al. (1988)),

- the generation of alternative structures based on one or more formal algorithms (Karimi and Konsynski (1988), Tan et al. (1989)),

- the verification of a specification based on a set of rules, sometimes also offering suggestions, advice and guidance on the resolution of inconsistencies, or on the way the development process should be continued (Stephens and Whitehead (1985), Lubars and Harandi (1986), Benner (1988), Budgen and Marashi (1988), Kramer et al. (1988), Loucopoulos and Champion (1988), Puncello et al. (1988), Wohed (1988), Yadav and Chand (1989)).

These systems normally apply hybrid knowledge representation languages, combining semantic networks and rules to represent knowledge about possible information system specifications. The types of knowledge found in these systems may be defined as domain knowledge (or re-usable components) and method knowledge.

- Domain knowledge

In the systems constructed by Kersten et al. (1986), Lubars and Harandi (1986), Proix and Rolland (1986), Falkenberg et al. (1988), Puncello et al. (1988), Loucopoulos and Champion (1988), Wohed (1988), and Benner (1988), the role of domain knowledge can be readily recognized. However, Lubars (1989) admits that the construction of "knowledge-based libraries" is one of the significant bottlenecks in their system.
Method knowledge

As far as method knowledge is concerned, various authors distinguish between rules based on the syntax and semantics of the specification language(s) supported by the system, and rules based on expert experience, see Wohed (1988), Benner (1988), Budgen and Marashi (1988). In most systems, method knowledge is in fact knowledge about feasible specifications, and is not concerned with knowledge about the development process. Suitable ways of working have been implemented at the outset. A notable exception is the system described in Kramer et al. (1988), who argue that "requirements analysis tools must be seen as intelligent assistants catering for users of widely varying degrees of expertise in the requirements method". Our ideas coincide with theirs in the sense that we also believe that "current work has tended to concentrate on specification structures and notations, and few methods offer more than collections of representational techniques. They tend to provide the practitioner with notations for encoding specifications and perhaps some validation procedures, but fail to suggest ways of getting the specifications in the first place".

Most "intelligent" support systems have been constructed especially for one method, and, in fact, generally even for only one specification technique. Sorensen et al. (1988) and Chen and Nunamaker (1989) have taken the view that there is a need for an additional abstraction level, which is normally referred to as "the meta-level". The expressive power of the systems they describe, MetaView and MetaPlex respectively, is, however, rather limited, and essentially only provides a semantic network in which knowledge about techniques and actual applications can be stored. They are not concerned with explicit knowledge of possible strategies in the development process, and cannot specify rules constraining allowable specification structures further.

In the systems discussed above, from which some envision the complete automation of certain small tasks, and others expect intelligent assistance, the problem of knowledge acquisition was never addressed. One notable exception is the study by Budgen and Marashi (1988) who have experimented on the creation and validation of the set of rules required for their Advisor system. They obtained a number of designs for a single problem. The designers were asked to provide details of their intermediate documents and the reasons for the decisions involved. Their analysis of the experiments, however, focussed on the product of the design process.

Summarizing, we find that current attempts to improve automated support are not based on a general view on the practical meaning of methods and CASE
tools in information systems development. They are furthermore careless in the
identification of useful heuristics and rules (knowledge acquisition), and
knowledge about the information systems development process.

We will now discuss the relationship between the practical use and meaning of
methods on the one hand, and automated support on the other. We will consider
modelling as a problem solving process to which the use of methods and CASE
tools is related.

1.4 A PROBLEM SOLVING PERSPECTIVE

We have discussed the importance of clear concepts and clear guidelines in
information systems development methods. We also argued that, depending on
the problem, various strategies might be necessary because other aspects of the
problem area need to be taken into consideration, or because other perspectives
have become predominant. The development, and particularly the analysis and
design of information systems, present various aspects and interrelated problems
to be solved, see Benyon and Skidmore (1987).

The problem solving view on information systems development, see Sol (1982),
suggests that we do not apply methods and CASE tools as a goal in itself, but
to support the problem solving process by defining ways of using information
technology effectively in organizations. Methods provide structure by identifying
useful strategies and techniques for specific problem classes.

According to Bots (1989), two facets of the problem structure must be
recognized: the first is "what", i.e. the problem model, while the second is
"how", i.e. the problem solving strategy. Sol (1982) views problem solving as a
chain of transformations in which the products of the various activities in the
process superimpose in successive layers. And again, we note product oriented
and process oriented views on problem solving.

The products of problem solving processes will be referred to as models in this
thesis, whereas the processes of problem solving will be called modelling.

Definition 1.1

A model is defined as a simplified, stylized representation of a system,
abstracting the essence of the system's problem studied.
Definition 1.2

Modeling is defined as the problem solving process of a problem as perceived in a given system.

Modelling in the context of this thesis is equivalent to information systems development in its meaning of a problem solving process of applying information technology effectively to persons and organizations. It is important to note that we apply the term "model" broadly, for example, including simple notes, verbal data or mental models of problem situations. The types of model consequently range from mental models not represented externally, to formal executable models with complete syntactic and semantic definitions.

1.5 THE PRACTISING INFORMATION ENGINEER

Various studies have analysed the behaviour of practising designers in various disciplines, see Vitalari and Dickson (1983), Ballay (1987), Stomph-Blessing (1988), and Vitalari (1988). The results clearly demonstrate the cognitive limitations of designers, and the observed behaviour can be best described by the paradigm of bounded rationality, see Sol (1982) and Bots (1989). According to these studies, modelling can be characterized by the following statements:

- Various models for an area of concern are possible and cannot be defined beforehand.

- It is impossible to consider a model as optimal, it can only be satisfactory.

- Intermediate models become part of the modelling environment and therefore influence the subsequent course of the modelling process and the nature of the subsequent models. Most models do not start "from scratch" but build on information extracted from previous models.

- Intermediate models are defined in an organizational context and involve several participants. Their domain knowledge and the domain knowledge of the information engineer(s) define the realm of domain knowledge influencing the actual modelling process. As argued in Stomph-Blessing (1988), the definition of domain dependent knowledge is unclear, even if theoretically possible.
• Experienced information engineers use several types of model representations because each type highlights specific design aspects. In the study of Ballay, designers either continuously alternate between types of representation while focussing on one subproblem, or deal with one type of representation and switch between the various design subproblems.

• Information engineers are uncertain of model details until part of the model has been built; "knowing" is in the observing of the model.

• Well-rehearsed routines and primary processes are alternated with planning periods, during which criteria and constraints are extensively reviewed, and major decisions about future processes are made.

• Information engineers make a choice between types of representation and levels of precision, inclusion and coherence, during the planning stage. Ballay (1987) argues that information engineers include so much ambiguity in their sketches by exclusion, incoherence and imprecision, that they benefit from opportunities sparking their inventiveness, until the very end of the modelling process.

We can conclude from the above that the actual models are strongly influenced by decisions made during the modelling process at various levels. For example, decisions can be made to conduct an information systems planning project, to work in a process oriented way or even to define a relationship between two entities as a one-to-many. These decisions determine the remaining modelling process and the final results.

Although the actual decisions made and the actual models constructed during the modelling process cannot be defined in advance, information engineers develop well-rehearsed problem solving behaviour as they gain experience, adding to their skill-repertoire, see Vitalari and Dickson (1983).

As Aguero and Dasgupta (1987) note: "it would be difficult to imagine a designer who has absolute certainty about the correctness and quality of his or her design. However, it is also equally difficult to imagine a designer who does not care about developing a certain degree of confidence in his or her design".

We tend to re-phrase the above: although domain knowledge strongly influences actual modelling processes, information engineers know about typical model structures and typical tasks to be performed, and the typical decisions to be made. Typical model structures can be viewed as patterns in equilibrium with
sets of heuristics or guidelines generally applied by information engineers. These heuristics can be interpreted as common patterns with appropriate levels of inclusion and precision, and sufficient amounts of coherence with previous models. This type of knowledge about typical model structures, typical tasks and typical decisions is called modelling knowledge.

1.8 INTELLIGENT SUPPORT REVISITED

In the previous sections we discussed a problem solving view on information systems development, in which we concentrated on the behaviour of practising information engineers, the way they develop confidence in the ongoing process, and how they achieve a certain level of quality. If we aim at improving automated support for information engineers, these tools must be able to support the processes to which these practitioners contribute.

This implies that automated tools should consider modelling as a problem solving process in which several strategies can be applied, various types of representation can occur, models gradually gain in accuracy and coherence, and existing models are extensively used in the construction of new models. These tools should include both process oriented and product oriented knowledge about the type of modelling to be supported. We think that, if automated tools are built using another view on modelling, they will remain documentation and verification aids. But, in our view, tools are essential for the improvement of the methodical support to be realized in information systems development. Books and courseware material cannot adequately contain more detailed modelling knowledge.

Our problem solving perspective furthermore implies that methods and CASE tools are only tools offering (partial) modelling knowledge which can be applied by information engineers in practice. Various shortcomings of methods and CASE tools have been discussed, such as fuzzy concepts, scarce guidelines, incompleteness and product-orientation. These shortcomings have been addressed by information engineers in practice. Their modelling knowledge is based on books and course material on the one hand, and on practical experience on the other.

If modelling knowledge contained in methods or CASE tools is not in equilibrium with the modelling knowledge as applied by information engineers, complex integration and tuning processes are called for. CASE tools have the problem that the view of the process to be supported has been hard-coded in these tools, and therefore cannot be changed or customized. Information engineers are
consequently left with the problem of finding a way of applying these tools in the information engineering practice.

Automated tools are preferably built according to a meta-system architecture in which the tool's behaviour can be modified and extended because the tool includes explicit and adaptable modelling knowledge. If we can include the information engineer's detailed modelling knowledge in automated tools, we can improve the quality of the modelling process by reducing its reliance on the engineer's quality and competence. In fact, we are convinced that this is the only appropriate way of making their modelling knowledge available. We quote Feigenbaum (1979): "what masters really know is not written in the textbooks of the masters".

1.7 RESEARCH APPROACH AND STUDY OUTLINE

The observed problems associated with methods and CASE tools are the main motives for the research presented in this thesis. The main objective of our study is to:

improve the automated modelling support offered to information engineers in information systems development.

The observation that methods and CASE tools are only tools providing partial structure to information systems development for typical classes of information systems development problems is of extreme relevance. Information systems development is in itself a problem solving process best described by the paradigm of bounded rationality. Both process oriented and product oriented perspectives have been identified as cornerstones of structuring this process.

Experienced practitioners apply methods during modelling, but refine the strategies and concepts offered by these methods in greater detail. We need to be in a position also to include this type of knowledge to increase the level of automated support in support environments for information engineers. We assume that the effectiveness of automated support is directly dependent on the available understanding of the modelling process to be supported. Based on these considerations, our research questions are essentially:

1. How can we describe modelling knowledge so that the description adequately provides structure to modelling in information systems development?
2. How can we acquire modelling knowledge so that it corresponds to the actual behaviour of practising information engineers?

3. How can we apply acquired modelling knowledge in support environments so that information engineers are effectively supported?

In chapter 2 we refine our view on modelling and the use of methods and CASE tools during a modelling process in greater detail. Based on this theory, the above tentative research questions will be refined to more concise research hypotheses, part of what Lakatos (1970) would call a "research programme" for the improvement of automated support in information systems development.

In chapter 3 we discuss a technique used for the description of modelling knowledge. In chapter 4 we address the problem of modelling knowledge acquisition, and present a useful approach for the construction of conceptual (technology-independent) models of experienced practitioners' available modelling knowledge.

The developed technique and approach are applied to three experienced information engineers in three methods in chapters 5 to 7. Knowledge acquisition is performed for a limited modelling task. In chapter 8 we evaluate the use of our technique and approach.

In chapter 9 the implications for automated tools are considered. We develop a solution for the incorporation of modelling knowledge in automated tools based on our view on modelling. The architecture of an environment for what are called modelling support systems (MSS) is proposed, and a prototype of an MSS environment is developed. It is evaluated by simulating the modelling behaviour as examined in the three cases in chapters 5 to 7. Finally, we summarize our research findings at the end of chapter 9.
2

INFORMATION SYSTEMS DEVELOPMENT

2.1 INTRODUCTION

We presented information systems development methods as facilities for structuring, standardizing and improving information systems development processes and its products. We pointed to various problems found in current methods, and we expect automated tools to become indispensable in improving methods and the methodical aspects of information systems development processes.

In this chapter, we propose a framework for understanding information systems development in which a systems development process is characterized by a way of thinking, modelling, working, controlling and supporting. Methods should include a way of thinking, modelling, working and controlling. Automated tools are expected to support these four "ways" and are therefore part of a way of supporting. We will refine our research questions and view on modelling support in systems development in greater detail by this framework.

The emphasis on understanding stems from our preference to know and understand before we solve, see Sol (1982). The framework is therefore a useful tool solving specific methodical problems. This framework could well be used as a first step in assessing and comparing methods, or for performing a method companionship analysis between methods and CASE tools, see Brinkkemper (1990). Its application as a means of uniformly highlighting specific distinctive characteristics of a heterogeneous set of existing methods, which is particularly useful in education, has been demonstrated in Seligmann et al. (1989).

Our main objective is to improve modelling support for information engineers. The framework constitutes our view on information systems development to which information engineers contribute. We emphasize the role of methods and automated tools in such processes for information engineers. The framework will be used to refine the research questions introduced in chapter 1.

2.2 UNDERSTANDING INFORMATION SYSTEMS DEVELOPMENT

We have introduced systems development as a problem solving process which is too complex to be conducted without plans and structure, see Sol (1982). Specific aspects of a system and parts of a system must be considered one at a time.
Many situations with interrelated problems must be evaluated in information systems development, see Benyon and Skidmore (1987).

Structuring systems development really means structuring the types of problems to be solved. More specifically, it implies structuring the types of models required for problem specification and solution, and structuring the types of activities for the solution of problems. This corresponds to the product and process oriented nature of problem solving processes, see section 1.4. We refer to these two perspectives as the way of modelling and the way of working, as shown in figure 2.1. Each will be discussed in more detail in sections 2.3 and 2.4. If we only consider these two perspectives on information systems development we will call this modelling.

The way of modelling and working concerns the actual problem of how to apply information technology effectively for organizations. But managerial activities are also required, such as setting up project teams, allocating tasks and managing finance, see also Kensing (1984). These activities are part of a way of controlling, and will be discussed in more detail in section 2.6.
Sec. 2.2 UNDERSTANDING INFORMATION SYSTEMS DEVELOPMENT

The other two perspectives within the framework are the way of thinking and the way of supporting. The way of thinking is based on the recognition that the features considered important to the process of information systems development, depend on an underlying philosophy used to inspect organizations and information systems. The way of supporting relies on possibly automated tools to be used in supporting a way of modelling, working and controlling. The way of thinking and way of supporting are discussed in greater detail in sections 2.7 and 2.8.

2.3 WAY OF MODELLING

A way of modelling structures the models which can be used in information systems development. Several models are usually required for problem specification and solution in the application area, because specific features have specific significance at various levels of abstraction, and for different views of the problem area. According to Benyon and Skidmore (1987), models are used in various ways, and many models and languages are therefore appropriate in existing situations, as Martin and McClure (1985) confirm.

2.3.1 CATEGORIES OF MODELLING LANGUAGES

Four categories of modelling languages are of particular interest in information systems development, each category having its specific characteristics and corresponding analytical capabilities. We again find a large number of specific languages within each of the categories, each with specific concepts, conventions and terminology. We will now discuss these categories briefly, see also Shubik (1979), Wilson (1984) and de Leeuw (1986).

- Free models

Free models are only restricted in their structure by the model builders' imagination. Typical examples are verbal and pictorial models. Free models are less accurate, but more flexible and subtle than, say, models built by applying structured techniques, computer languages and mathematics. Delicate nuances in meaning are more effectively expressed by essays and drawings than by constructs bound by specific rules. Free models, however, can in essence not be verified by means of a computer, unlike structured, mathematical and dynamic ones.
• Structured models

Structured models, generally represented in diagrams, tables, matrices or structured text, are constrained by the types of concepts used, and the properties valid for these interrelated concepts. Structured models are specifically used for clarification, illustration and definition of overall logical structures. Structured models can be analysed for consistency, and partly by additional logic reasoning.

• Mathematical models

Mathematical models are defined by applying languages based on mathematical constructs: formal specifications are a typical example. These models are particularly useful for deducing specific systems characteristics.

• Dynamic models

Dynamic models, such as simulation models and prototypes offer experimentation facilities. They can be used to analyse the dynamic behaviour of certain system properties, see Wilson (1984).

The above categories are not mutually exclusive, of course. Some mathematical formalisms\(^1\) are executable as well, while some dynamic modelling techniques are mathematically based. In other cases, initially purely structured modelling techniques, such as NIAM, have been based on mathematics, see Bommel et al. (1991), and mathematical formalisms have been extended by graphical front-ends, see Harel (1988). Languages can consequently benefit from both analytical capabilities.

2.3.2 INTERRELATED STRUCTURE OF MODELLING CONCEPTS

We are generally not restricted in information systems development to applying a single modelling language. Various more or less interdependent models are developed in the development process. For example, structure charts of processes in dataflow diagrams are drawn up, event lists are used to define the stimuli in context diagrams, Structured English specifications are given for processes in dataflow diagrams, and PASCAL programs are written for Nassi-Shneidermann diagrams.

\(^1\) Throughout this thesis we will use synonymous words for modelling language such as modelling technique, structured technique, specification formalism, programming language, etc. depending on the context.
It is therefore essential not only to be familiar with the *modelling concepts* of one specific modelling language, but also with the *interrelationships* between the modelling language concepts and the models constructed with the various languages. The expression "*way of modelling*" represents this *interrelated* set of modelling concepts, describing the types of models which can potentially be built in development processes (depending on specific problem situations). A way of modelling defines:

- the *modelling concepts* to be used in information systems development, and
- the *interrelationships* between these concepts.

*Instances* of these modelling concepts and their interrelationships come into being during modelling, and models are therefore just sets of such interrelated modelling concept *instances*.

### 2.4 WAY OF WORKING

The process oriented nature of information systems development is generally referred to as a *way of working*, see Blank and Krijger (1982), Bemelmans (1987) and Sol (1988). It *structures* the strategy determining the manner in which information systems are developed.

We discussed systems development strategies in chapter 1, together with the associated tasks to be performed and the choices to be made. We prefer to name a relevant, self-contained part of the development process a *task*, in accordance with our problem solving perspective, see Newell and Simon (1972) and Bots (1989). Strategy deals with the identification of the relevant tasks in the development process, and their *feasible order*. Whenever strategies contain various alternative tasks, a choice of the preferred alternative must be made. *Decisions* are a natural part of a way of working. Tasks can be divided into subtasks, which means that the problem to be solved by the specific task can be divided into subproblems, each handled by a subtask. In addition to these structural aspects of systems development tasks, informal *guidelines* about task objectives and results, and suggestions about involving the various *participants* in the process, are important components of a way of working.

---

2. The following synonymous words are found for our notion of task: phase, activity, procedure, step or action.
Various *types of tasks* can be identified in systems development. Some are concerned with model construction, refinement and transformation, others with model verification or experimentation. The tasks specifically concerned with such model manipulations will be defined as *modelling tasks*. Other typical *non-modelling tasks*, for example, are concerned with explanation and education, and with management reporting.

We view a *way of working* to consist of:

- the tasks possibly to be performed during a development process,
- the decomposition of tasks into subtasks and decisions,
- the possible order of tasks and decisions, and
- the informal guidelines and suggestions on how to perform tasks, and how to make the decisions.

Specific tasks are performed and specific choices are made depending on specific situations.

### 2.5 RELATIONSHIP BETWEEN WAY OF MODELLING AND WORKING

In section 1.5, we have characterized modelling on the one hand based on a problem solving perspective relying on the paradigm of bounded rationality, and on the other hand based on general studies on design processes in various disciplines. This characterization of modelling in section 1.5 is our starting point for our *view* on the relationship between the way of modelling and working. We recall the following points of particular importance:

- information engineers select specific types of *representation* and levels of *inclusion*, *precision* and *coherence* to be used within the modelling task at hand, and

- intermediate models become part of the *modelling environment* and therefore *influence* the subsequent course of the modelling process and the nature of the subsequent models.

These subjects will be discussed in more detail in the following sections.
2.5 LEVELS OF INCLUSION, PRECISION AND COHERENCE

Modelling tasks in a modelling process result in models associated with certain levels of inclusion, precision and coherence. Ballay (1987) also inverted the reasoning: it is natural to apply certain amounts of exclusion, incoherence and imprecision during modelling, in order to achieve sufficient levels of ambiguity allowing inventiveness to grasp opportunities until the very end of the modelling process. We will now define inclusion, precision and coherence in terms of specific relationships between the way of working and modelling.

- **Inclusion**

  Inclusion is related to the type of concepts actually applied or manipulated in a specific modelling task. Inclusion defines the scope used when analysing a problem in a specific task. For example, while one task may only be interested in events, the other is concerned with data stored in the system. Both tasks have a completely different scope of the modelling concepts offered by a way of modelling.

- **Precision**

  Precision is concerned with the consistency level to be achieved during a task and after, implying that verification rules must define precision levels during these two stages. These precision requirements will be referred to as the invariants and postconditions of specific tasks. An example of an invariant is that each entity is named, and of a postcondition that each entity can be identified. The verification rules proper should be regarded as essential properties of the modelling concepts and their relationships as defined in a way of modelling.

- **Coherence**

  Coherence applies to the interdependence between results of one task and that of previous ones, and is therefore concerned with specific constraints of previous modelling results on the results of a new task. For example, a process in a dataflow diagram is selected in order to define a new dataflow diagram for the specific process: all incoming and outgoing dataflows of the process are included in the new dataflow diagram. While the new dataflow diagram is set up, the context of the previously selected process is taken into account. In other words, the previous task defines a specific context for the current task, which implies that specific task output can define a context (input) for other tasks. Coherence requires information to be exchanged between tasks.
2.5.2 TYPE OF REPRESENTATION

Models must evidently be represented in one way or another: diagrams, matrices, tables, lists, and program specifications are examples. A clear distinction should be made between the modelling concepts and their external notation. Sutcliffe et al. (1989) argue that some methods allow alternative equivalent notations, but that on the other hand similar graphical and textual topologies can represent different types of models. We would even take this one step further: alternative representations can, and sometimes must be selected for the various tasks in the modelling process. For example, processes are represented by rounded boxes in functional decompositions, but by circles in dataflow diagrams.

2.5.3 INFLUENCE OF THE MODELLING ENVIRONMENT

Decisions, for example, to follow certain strategies, to perform specific tasks, and to add certain concepts to models, are strongly dependent on previous results. This type of information is part of what we call the modelling environment, where we find both process history, and the models built so far. Criteria and guidelines supporting decision making, called decision rules, refer to this type of information. For example, in the YOURDON method, the decision to add a stimulus to a context diagram is determined by the existence of events in the previously constructed event list. In order to clarify the relationship between a way of working and modelling, we re-phrase the essence of the influence of the modelling environment: decision rules supporting the making of a decision which is defined within a way of working may refer to the existence of specific model structures as prescribed by a way of modelling.

In the sections 2.3 to 2.5 we have introduced our perspectives on information systems development related to the actual problem solving process of how to apply information technology effectively for organizations and persons. As discussed earlier, with only a way of modelling and working in mind, we will generally refer to modelling rather than to information systems development. We will now discuss the three other perspectives on information systems development making up our framework for understanding information systems development.

2.6 WAY OF CONTROLLING

The management of a development process attracts quite some interest in information systems development, which is represented in our framework by the way of controlling. It includes planning and plan evaluation, establishing the
project by indicating how the various persons and groups should interact, and how the generally limited resources should be employed and allocated, see also Mathiassen (1981), Kensing (1984) and Sol (1988). Planning and plan evaluation are realized by dividing development processes into phases with checkpoints and baselines. Andersen et al. (1990) find that "the organization of systems development projects should ensure direct and close interaction between performance and management activities". In terms of our framework, this encourages close interaction between a way of working and modelling on the one hand, and a way of controlling on the other.

The interaction becomes apparent because a way of controlling requires at least a rough view of the way of working and modelling, to achieve effective planning, and a way of working requires that plans are available while work is in progress, so that management can be informed about divergences from the plan. Because information systems development inherently deals with uncertainty, the interaction between the various "ways" is particularly important for understanding the development process correctly. The resulting interrelationships between a way of modelling, working and controlling are presented in figure 2.2, using de Leeuw's (1986) control paradigm.

![Figure 2.2 The relationship between the way of controlling, way of working and way of modelling](image)

2.7 WAY OF THINKING

We have so far presented three ways in which information systems development can be structured and interrelated. As mentioned in section 2.1, methods should always include these three ways. However, one question has so far not been answered, although it is essential for understanding methods - it is the one we
informally refer to as the "why question". More specifically: what are the salient aspects in a specific way of modelling, working and controlling of a specific method?

The hidden assumptions embedded in these three "ways" become explicit on answering this question, and can be discussed as appropriate to certain systems development problems. In Seligmann et al. (1989) we have suggested that methods can have considerably differing starting points and basic objectives. They may, for example, differ in the requirement to include explicit definition of the information systems' objectives, and in whether social and organizational matters should be considered, in whether information systems should be regarded primarily as computer systems, or as systems of persons and computers, and in whether the system's environment should be taken into consideration. The major assumptions underlying a certain approach are represented in what we call a way of thinking.

2.7.1 SOME VIEWS ON THE WAY OF THINKING

Ways of thinking embedded in methods have already been studied by various other authors. In this section, we summarize some of these findings in literature in order to produce an adequate definition of the way of thinking ourselves in the next section.

Sol (1983) concludes that methods cannot be discussed and evaluated if Weltanschauung (or paradigm) and construct paradigm (or model cycle) are ignored. He observes that in discussions about methods, paradigms and model cycles are not extensively debated, although they are essential. Sol (1982) distinguishes the paradigm of bounded rationality (or procedural rationality) as opposed to substantive rationality. An inductive-hypothetic model cycle is distinguished from a hypothetico-deductive one.

In Mathiassen (1981) methods are globally characterized by their application area and by their underlying perspective. Application areas are sometimes additionally characterized by their size, the implied functionality and technical setting of the automated system, the type of organizational change, and the project setting, see Kensig (1984) and Floyd (1986). The underlying perspective is concerned with the total perception of given phenomena, which are information systems in their environment in this case.

According to Avison and Fitzgerald (1988) "the choice of the area covered by a methodology, its systems, data or people orientation, its bias or otherwise towards computation, and other aspects, are made on the basis of the philosophy
of the method”. They regard the philosophy as a principle, or set of principles, underlying the method. They distinguish four factors part of the philosophy of a method: 

paradigm, objective, domain and target. In accordance with Wood-Harper and Fitzgerald (1982), they identify two relevant paradigms: the science paradigm and the systems paradigm. According to Wilson (1984), this is the difference between the statement that a model attempts to describe what exists, and that a model attempts to describe a perception of what exists. The method’s objective deals with its ultimate result which could consist of a computerized system, or a solution for problems in an object system. The method’s domain is related to the object of study, which can be a single problem area or even a complete organization. Avison and Fitzgerald (1988) refine the methods’ target to problem type, environment, and organization type and size - some methods require no specific target and are consequently more general.

Bubenko (1986) explains trends in methods by referring to our evolved perception of information systems and their corresponding system architectures and processing principles. He finds that "a method is always designed on the basis of some assumptions about the characteristics of its target class of applications and information systems, about the skills and capabilities of the professional designers, and about the know-how and attitudes of the users and consumers”.

For completeness’ sake we emphasize that other authors have obviously also discussed the “way of thinking”, such as Blank and Krijger (1982), Maddison et al. (1983) and Bemelmans (1987). As our study is not primarily concerned with a detailed study to underlying ways of thinking, we shall not discuss these explicitly.

2.7.2 ESSENTIALS OF THE WAY OF THINKING

When reviewing the literature presented above, we observe incomparable differences in the levels of abstraction, standing in the way of a clear and compact definition of the way of thinking, based on this review. Some authors explicitly consider certain characteristics such as application area, organization type, and analyst skills, whereas others limit themselves to identifying two different paradigms. The remaining question is why these authors have highlighted these aspects in the first place.

To resolve this problem we return to our first introduction of the way of thinking: it reflects the answer to the “why question”, resulting in the explanation of the salient points of the method. Applying the well-known philosophical method of asking oneself the why-question repeatedly, Seligmann et al. (1989), concluded that the way of thinking should, in fact, encompass a basic view of
the problem to be solved. As methods are intended to support information systems development, a way of thinking should define the assumptions of a method with respect to:

- what constitutes an information system,
- what is the function of an information system with regard to its environment,
- what constitutes the environment of an information system, and
- what are the major characteristics of the component parts of an information system and its environment.

In terms of Nurminen (1988), this is viewed as a method description "reduced to its bare essentials". A simple visualization of the essentials of a method, regarding the information system's functionality and its role in the environment, has helped Seligmann et al. (1989) in describing a way of thinking.

2.8 WAY OF SUPPORTING

Tools should support information systems development, which itself can be understood by distinguishing a way of modelling, working and controlling, all embedded in a specific way of thinking. A collection of tools supporting systems development is referred to as a way of supporting.

Tools are supporting means required to perform and facilitate certain tasks. In programming we find (syntax-directed) editors, debuggers, and compilers. In the database area we have database management systems, data dictionaries, report and screen generators, and 4th generation systems. Word processors can be considered as tools for editing baseline documents and other interim reports. As to the use of structured techniques, the main tools available only a decade ago were preprinted standard forms and stencils. Today's automated tools include support for the use of these techniques, see Bubenko (1988), Fisher (1988), Vonk (1988a,b), McClure (1989). A way of supporting may obviously include tools for the way of controlling, such as project management workbenches. In addition to automated tools, the less sophisticated blackboards, paper, pencils, and whiteboards are also considered to belong to a way of supporting.
The importance of automated tools is confirmed by Martin (1986a), and Avison and Fitzgerald (1988). The CRIS-88 conference, see Olle et al. (1988b), concentrated on automated information systems development and by that emphasized the increasing importance of automated tools. In the next section we will discuss the role of automated tools in more detail. We conclude that the main criterion for effective support is a way of supporting fitting neatly into the three other "ways".

2.9 AUTOMATED SUPPORT

We introduced CASE tools (Computer-Aided Systems Engineering) earlier. Various other names, such as I-CASE, AWB, PWB, IPSE, Upper-CASE, and Method Companion are found for it in the information systems area, classifying specific products offering automated support. We are not overly concerned with terminologic struggles in automated systems development in this study. A thorough analysis of the main characteristics of these tools is considered to be more important in our opinion: the functionality required in CASE tools should be defined clearly. Although we realize that the name CASE tool is an arbitrary choice, we prefer to use it in this thesis because it has become common usage. Using the name CASE tool and defining its functionality, we essentially describe the functionality of an automated way of supporting.

Bubenko (1988) considers the main tasks of CASE tools to accept various specifications, to analyse and transform these, and to maintain a large, expanding set of interrelated specifications. He furthermore finds that "good" tools must also support and guide the users. Following features are distinguished in CASE tools: user interaction, verification support, validation support, design support and development project management. As far as support components are concerned, he differentiates between the method's object schema, method knowledge, domain knowledge, and reusable specifications.

In order to provide "full" support, CASE tools should have the following capabilities, according to McClure (1989): graphics capability, error checking, information repository, tightly integrated toolset, full life cycle coverage, prototyping support, automatic code generation, and structured methodology support. In the discussion about potential improvements in the current CASE products, she expects these in the areas of integration and intelligence. The following "intelligent" options are distinguished: natural language communication, method knowledge bases, application domain knowledge bases, expert knowledge bases, habitable environments, methodology drivers and reusability.
Chen et al. (1989) recognize four basic functions of CASE tools: *elicitation, analysis, transformation* and *information storage*. They argue that CASE tools differ in system type, life cycle coverage, activity type, intelligence level, integration level, mode of use, information source, and degree of flexibility.

Vonk (1988a) describes CASE tools schematically as a database or *dictionary* surrounded with a set of functions accessible by a common user interface, such as *diagram editing*, (intelligent) *text editing*, *screen and report design*, *prototyping*, *analysis*, generation and *transformation*, *report generation*, multi-user support and *navigation*.

We observe that the various authors define ideal types, in which various possible future research directions have generally been included. At the centre of each CASE tool description is the *repository*, which is simply storage for various items. We will distinguish between *information used* and *information constructed*. In figure 2.3, information used is found at what we call the *meta-level*, while information constructed is at the *application level*.

![Diagram](image)

**Figure 2.3** The components of a CASE repository

Information at the meta-level has been based upon *management, process* and *product oriented knowledge*.

Management oriented knowledge is related to *project management support*. At the meta-level, one may wish to specify the type of aspects to be managed, such as time, responsibility, money, persons, and means, and the type of activities to be performed in order to manage these aspects. It corresponds to an explicit specification of a way of controlling.
Process oriented knowledge is concerned with the above mentioned *methodology driver*, *navigator* and *guide*, and corresponds to a way of working.

As regards product oriented knowledge, both Bubenko (1988) and McClure (1989) distinguish *syntactical modelling language knowledge* from *expert based knowledge* and *reusable domain knowledge*. Although this distinction may appear too rigorous, it is important to note that there are differences. Certain syntactically correct constructs are simply "not done".

As discussed in section 1.5, practising information engineers have knowledge about *typical model structures*, perhaps only valid in specific application domains. Some of this knowledge about typical model structures is based on modelling language syntax. Other knowledge is based on *experience*, and may be regarded as an *extension* to the formal syntactical knowledge. We find these extensions in textbooks as well, because most books illustrate and explain their modelling techniques by *examples*. In Martin (1986b), for instance, various functional decompositions for various types of organizations are presented. Knowing the type of organization implies knowing a typical model structure. Product oriented knowledge at meta-level corresponds to a way of modelling.

The *application level* contains all information generated in projects for specific organizations and applications. Information at the application level is an *instantiation* of information at the meta-level. In other words, the meta-level defines the set of possibilities at application level. The same tripartition as found at meta-level is found at application level.

Considering the current CASE tools, hardly any knowledge at meta-level has been stored explicitly. This type of knowledge has been hard-coded in these tools, which can consequently not be adapted to specific types of information systems development.

We expect future CASE tools mainly to grow to meta-systems, see Sorenson et al. (1988), Chen and Nunamaker (1989) and Falkenberg (1989), in which support is based on explicit knowledge at the meta-level. Based on this type of knowledge in their repository, CASE tools should at least offer the following types of support:

- management support,
- process guidance,
- model construction,
(heuristic) model verification, and

model derivation.

Dynamic interpretation and prototyping should also be supported by CASE tools, see Vonk (1988a) and McClure (1989). However, these activities remain strongly dependent on the dynamic semantics of the modelling language used. The support for these activities using a meta-system approach is still questionable, see also Cohen et al. (1986), and will probably have to be provided by specific supporting tools.

The above considerations have resulted in a CASE environment architecture as shown in figure 2.4. It shows that CASE tools should preferably function by means of an interpretation mechanism interpreting explicit knowledge of the way of modelling, working and controlling to be supported.

![Diagram showing CASE environment architecture](image)

Figure 2.4 General architecture for a CASE environment

### 2.10 RESEARCH QUESTION REVISITED

The framework presented should particularly help in understanding information systems development processes. We have demonstrated that understanding systems development requires a distinction between a way of modelling, working and controlling, all embedded in a way of thinking and supported by a way of supporting.
Methods must offer a specific way of thinking, modelling, working and controlling to solve specific types of information systems problems. CASE tools should offer automated support as part of a way of supporting.

We have discussed an architecture for CASE environments in section 2.9, capable of supporting any way of modelling, working and controlling, because of the explicit method knowledge available. Focussing on modelling in systems development implies concentrating on a way of modelling and a way of working, including their specific interrelationships as discussed in the sections 2.3 to 2.5. CASE environments directed at modelling support will be referred to as MSS (Modelling Support System) environments, which include explicit modelling knowledge.

Information engineers apply methods and tools to structure their systems development and modelling process.

In chapter 1 we discussed various problems around systems development methods and CASE tools. Methods apply fuzzy and artificial concepts, produce few guidelines, and are incomplete. The way of working is generally only described in rough managerial terms. The influence of situational factors on the applicability of methods has been neglected. CASE tools are strongly product oriented, and are mainly used for documentation and verification. Guidelines on why and how to perform various tasks, and how to determine the quality of specifications are not part of automated tools. Specific views of the process to be supported have been hard-coded in CASE tools, which can therefore not be customized.

These methodical shortcomings have been addressed by experienced information engineers in practice. Their modelling knowledge depends on books and course material, and on practical experience, so that individual interpretations, deviations and refinements are common practice. Their modelling knowledge should be included in automated tools if we wish to improve automated support by corresponding to their actual practice, or by making their detailed knowledge available to others. Automated tools should offer facilities for the application of detailed modelling knowledge, unavailable in books and courses. Modelling support to information systems development can thus be improved effectively in practice.

In this thesis we will elaborate on our theory aimed at improving the automated support for modelling in information systems development. As part of this theory we will consider the following hypotheses, the first two of which are related to representation and acquisition of modelling knowledge, and the third is related to MSS environments.
Hypothesis 1:

Modelling knowledge can be represented adequately by considering it as a description of a way of modelling and working, including their interrelationships.

Hypothesis 2:

Detailed modelling knowledge can be acquired from experienced practitioners in the specific field of modelling, i.e. knowledge acquisition from experienced practitioners leads to detailed understanding of the actual modelling process to be supported.

Hypothesis 3:

It is possible to construct an interpretation mechanism as part of an MSS environment, which offers automated support by interpreting explicit modelling knowledge.

In this study, we will first attempt to obtain and verify an approach by which modelling knowledge can be acquired and represented. This is related to the first two hypotheses, and is dealt with in chapters 3 to 8. Chapter 3 proposes a detailed modelling knowledge representation technique. Chapter 4 relates to the modelling knowledge acquisition approach. In the next three chapters, this approach is applied to three modelling tasks performed by three experienced information engineers with different methodical backgrounds (D2S2, YOURDON and ESM). The three tasks may be considered as roughly equivalent. They all concern the construction of what we call an information architecture, possibly supported by CASE tools such as SDW, IEW and BLUES. We have limited ourselves to this task as a test case because of criteria such as fuzzy concepts, few verification rules, various modelling techniques and incompletely defined guidelines. We draw our conclusions on these three experiments in chapter 8, and judge whether our technique is suitable for the description of modelling knowledge, and whether our acquisition approach leads to detailed modelling knowledge.

After testing the first two hypotheses, we will discuss the third one in chapter 9, where we will concentrate on the detailed definition of an architecture for MSS environments, and the mechanisms required for the interpretation mechanism. The feasibility of the implementation will be demonstrated by simulating the modelling processes observed in the experiments discussed in chapters 5 to 7. We will evaluate our findings at the end of chapter 9.
3

REPRESENTATION OF MODELLING KNOWLEDGE

3.1 INTRODUCTION

This chapter describes a technique for modelling knowledge representation, formalizing a way of modelling, a way of working, and their interrelationships, see chapter 2. Figure 3.1 shows the research area discussed in this chapter.

![Diagram showing three levels of abstraction]

Figure 3.1 Three levels of abstraction

At application level we are dealing with an actual information systems development problem. As mentioned earlier, we distinguish between a process and a product oriented perspective prescribed by a way of working and a way of modelling, respectively.

A specific way of modelling and working, as the figure indicates, is found at the meta-level. Again we find a process and product oriented perspective. Modelling knowledge\(^1\) constitutes the product oriented perspective at meta-level. The acquisition of modelling knowledge is part of the process oriented perspective at meta-level.

\(^1\) We will refer to formally represented modelling knowledge as a modelling knowledge specification or a meta-model.
The theory level is concerned with a theory applicable at meta-level, see Chen and Nunamaker (1989). A theory for the acquisition of modelling knowledge is to be found in chapter 4, whereas we concentrate on a representation technique for modelling knowledge in this chapter.

The modelling knowledge representation technique, or in short meta-modelling technique, will be discussed in three parts: (1) the representation of a way of modelling, (2) the representation of a way of working, and (3) the representation of the specific interrelationships between the two.

Section 3.2 is concerned with representing a way of modelling. We will introduce concept structures to represent ways of modelling formally. A concept structure can be found at the meta-level. Models constructed at the application level by information engineers are instantiations of a concept structure. More complex model restrictions, generally called verification rules, can be defined for concept structures. The verification rule language defining the form of verification rules is explained in section 3.3.

Section 3.4 is concerned with the specification of ways of working. Task structures constitute our means for representing ways of working formally. An actual modelling process at application level is an instantiation of a task structure at meta-level.

In section 3.5 we will demonstrate the part of the meta-modelling technique enabling the specification of links between ways of modelling and working. We will define task scopes, a-priori and a-posteriori verification rules, information places, basic procedures and decision rules. In section 3.6 we will summarize the main characteristics of the meta-modelling technique.

Throughout this chapter, illustrations from the JSD method will be used, see Jackson (1983) and Cameron (1983); the illustrations are based on an examination of literature rather than on the acquisition of practical JSD expert knowledge.

3.2 CONCEPT STRUCTURE

A way of modelling prescribes the (interrelated) concepts to be used in a modelling process. Models constructed by an information engineer are instantiations of the concepts prescribed by a specific way of modelling. The concept parts of a way of modelling are interrelated, leading to associations,
linking two or more concepts. Concepts play a specific role within associations, see also Welke (1988). A role clarifies the involvement of a concept in an association. Associations may in turn have associations themselves, leading to objectified associations involved in associations with other concepts or even with other objectified associations. Objectification offers the opportunity to treat associations as concepts. Objectified associations, which are the same as associative entities in some entity-relationship modelling dialects, can also be found in Nijssen and Halpin (1989) and Welke (1988). Besides association and objectification, a third well-known abstraction mechanism in knowledge representation and data-modelling is the one of generalization versus specialization. Inheritance of properties can be established by specialization relationships between two concepts. The subtypes, as they are called, inherit associations from its supertypes.

We apply the mechanisms of association, objectification and specialization in defining a network of modelling concepts applied by an information engineer, the network being called a concept structure. The above notions are summarized in figure 3.2, using the external representation conventions defined by the NIAM method, see Wintraecken (1985), and Nijssen and Halpin (1989).

![Diagram](attachment:image.png)

Figure 3.2 The notions part of a concept structure

3.2.1 A JSD EXAMPLE

As an illustration of the above, we will present a JSD example taken from JSD's entity structure diagramming technique. In these structure diagrams the order of entity actions of an entity is shown by means of a tree-like structure, the root
of which represents the entity, and the leaves represent the actions of the entity. The intermediate tree boxes define sequence, choice and iteration between its subparts. An example of a JSD entity structure diagram is shown in figure 3.3, according to Jackson (1983). The example should be placed at the application level, see figure 3.1.

![JSD entity structure diagram](image)

Figure 3.3 Example of a JSD entity structure diagram

In figure 3.4 we show the corresponding part of the JSD concept structure. This figure should be placed at the meta-level.

![JSD concept structure](image)

Figure 3.4 Part of the JSD concept structure concerned with the JSD entity structure diagrams
3.2.2 THE CONCEPT STRUCTURE DEFINITION

After this introduction by intuition and example, we will now define concept structures formally, using first order predicate logic. We will assume all sets to be finite and disjoint, unless the context suggests the opposite.

Definition 3.1

A concept structure $C$ is a 5-tuple $C = (C, A, R, S, Q)$, where:

- $C$ is a set of concepts, and
- $A$ is a set of associations.
- $R : A \times N\setminus\{0\} \rightarrow C$ is the partial function for the mapping from roles to concepts. The function is referred to as role with $\text{role}(a, 2) = c$ indicating the second role of association $a$ being played by concept $c$.
- $S \subseteq C \times C$ is the specialization relation defining the subtype network. The predicate $\text{subtype}$ will be used to represent this relation with $\text{subtype}(c_1, c_2)$ implying that $c_1$ is a subtype of $c_2$.
- $Q : A \leftrightarrow C$ is the partial objectification function (a bijection) representing objectified associations. The function $Q$ will be referred to as $\text{objectified}_{\text{into}}$ with $\text{objectified}_{\text{into}}(a) = c$ meaning that association $a$ is objectified into concept $c$. The inverse function $Q^{-1}$ will be denoted by the predicate $\text{objectification}_{\text{of}}$.

We will first define two auxiliary predicates, and subsequently the properties valid for a correct concept structure $C$.

Definition 3.2

Concepts are their own ancestors and of other concepts, if they are supertypes of the other concepts or ancestors of one of their supertypes.

$$\text{ancestor} \subseteq C \times C$$

$$\text{ancestor}(c_1, c_2) \iff c_1 = c_2 \lor \exists c_3 \in C [\text{subtype}(c_3, c_1) \land \text{ancestor}(c_3, c_2)]$$

Concepts are called other concepts' pater familias, see de Troyer et al. (1988), if they are ancestors of other concepts and have no supertypes.
\[ \text{pater\_familias} \subseteq C \times C \\
pater\_familias(c_1, c_2) = \text{ancestor}(c_1, c_2) \land \neg \exists c_3 \in C [\text{subtype}(c_1, c_3)] \]

Property 3.1

Each association has at least two roles.

\[ \forall a \in A \ \exists c_1 \in C \ \exists c_2 \in C [\text{role}(a,1)=c_1 \land \text{role}(a,2)=c_2] \]

Property 3.2

Roles are numbered. Role numbering within an association starts at 1 and continues in steps of 1.

\[ \forall a \in A \ \forall c_1 \in C \ \forall n \in \mathbb{N} [\text{role}(a,n)=c_1 \land n>1 \Rightarrow \exists c_2 \in C \ \exists m \in \mathbb{N} [m=n-1 \land \text{role}(a,m)=c_2]] \]

Property 3.3

Each concept has exactly one pater\_familias. This implies that each subtype network has exactly one concept as its top.

\[ \forall c_1 \in C \ \exists! c_2 \in C [\text{pater\_familias}(c_2,c_1)] \]

Property 3.4

There are no cycles in subtype networks.

\[ \forall c_1, c_2 \in C \ [\text{ancestor}(c_1,c_2) \land c_1 \neq c_2 \Rightarrow \neg \text{ancestor}(c_2,c_1)] \]

Property 3.5

Objectifications cannot be subtypes of other concepts.

\[ \forall c_1 \in C \ [\exists a \in A [\text{objectified\_into}(a)=c_1] \Rightarrow \neg \exists c_2 \in C [\text{subtype}(c_1,c_2)]] \]

Property 3.6

Cyclic objectification structures are excluded, see also Bommel et al. (1990).

There is a function \( f: C \cup A \rightarrow \mathbb{N} \), such that

\[ \forall a \in A \ \forall c \in C [\text{objectified\_into}(a)=c \land f(c) = f(a)] \]
\[ \forall c_1, c_2 \in C \ [\text{ancestor}(c_2, c_1) \Rightarrow f(c_1) = f(c_2)] \]

\[ \forall a \in A \ \forall c \in C \ \forall n \in \mathbb{N} \ [\text{role}(a, n) = c \Rightarrow f(c) < f(a)] \]

The above properties define the set of syntactically correct concept structures. A concept structure determines the set of possible application models. Models constructed during a modelling process can be considered as a set of concept and association instances. More details about the formal relationships between a concept structure and an application model can be found in chapter 9, and Wijers et al. (1990).

### 3.3 Verification Rules

Verification rules are used for the additional restriction of possible models defined by a concept structure. Although verification is related to the tasks part of a way of working (see section 3.5), the verification rules proper depend only on the concepts and associations defined in a concept structure.

The rule language for the specification of model constraints will be discussed in this section. The essence of a verification rule is its verification of the consistency of a given model as an instance of a given concept structure. We have decided to use first order predicate calculus (with equality) for our rule language syntax. More details about the formal language of first order predicate calculus can be found in Carnap (1958), Kowalski (1979), Gray (1984) and Nolt and Rohatyn (1988).

Using the predicate calculus as our rule language, implies that our modelling knowledge representation technique has (1) high expressive power and (2) is formally defined. These two considerations are of paramount significance. For example, the use of graphical constraints as defined for NIAM could have provided an alternative. These constraints could be mapped onto statements in the first order predicate calculus, but the NIAM constraints are less powerful - which is the reason for our above decision.

#### 3.3.1 A JSD Example

For example, we can define rules applicable to the concept structure in figure 3.4. One such rule valid for JSD entity structures says:
• Entities should be specified by structure diagrams, and there must be exactly one structure diagram for each entity.

The association between "entity" and "structure diagram" has not been named explicitly in figure 3.4, but we will name the association "specified_by" in our example. The verification rule in first order predicate calculus is therefore:

\[ \forall e \ [ \text{entity}(e) \rightarrow \exists s \ [ \text{structure_diagram}(s) \land \text{specified_by}(e, s)] \].

Another rule stipulates that each not-a-leaf is either an iteration, selection or sequence. This results in:

\[ \forall n \ [ \text{not_a_leaf}(n) \Rightarrow \text{iteration}(n) \oplus \text{selection}(n) \oplus \text{sequence}(n)] \].

### 3.3.2 The Verification Rule Definition

The set of predicates allowed in verification rules is restricted to a certain type. Each sentence in first order predicate calculus using only these predicates constitutes a proper verification rule about a concept structure. The following predicates are allowed in verification rules:

**Definition 3.3**

*Conceptname* may be used as a one-place predicate in verification rules if *conceptname* occurs in the corresponding concept structure. *Conceptname*(x) is true for each instance of *conceptname* and for each instance of a descendant of *conceptname* (descendant is the inverse of ancestor, see definition 3.2.2).

**Definition 3.4**

*Associationname* may be used as an n-place predicate in verification rules if the n-ary association *associationname* occurs in the corresponding concept structure. *Associationname*(x₁,...,xₙ) is true if each concept instance xᵢ plays the i-th role in an instance of association *associationname*.

**Definition 3.5**

*Conceptname~associationname* may be used as an (n+1)-place predicate in verification rules if the n-ary objectified association *associationname* occurs in the corresponding concept structure. *Associationname* is objectified into *conceptname*. *Conceptname~associationname*(x,y₁,...,yₙ) is true if concept
instance $x$ is an objectification of an association instance, with each concept instance $y_i$ playing the $i$-th role.

### 3.4 TASK STRUCTURE

A way of working determines the tasks which can be performed in a modelling process, and structures the potential modelling strategies. More specifically: we are interested in the various relationships determining task order. We should be able to specify:

- parallelism between tasks,
- sequence of tasks,
- iteration of tasks,
- alternative tasks,
- optional tasks, and
- synchronization of tasks.

Task structures are capable of representing such aspects of a way of working, as we will indicate in this section. After Bots (1989), we consider tasks as actions to be undertaken in order to achieve certain objectives. Tasks can be defined recursively in terms of subtasks which should accomplish certain sub-objectives. If performing a task involves alternative strategies, choices will be represented by decisions. Time sequences between tasks and/or decisions are represented by triggers. If a task is decomposed into subtasks, decisions and triggers, then the task's initial item(s) must be defined.

We have illustrated tasks, decomposition relationships, decisions, triggers and initial items in figure 3.5, according to the external representation conventions defined in Bots (1989). The figure shows task $A$ which is decomposed into six subtasks and two decisions. The execution of task $A$ starts with task $B$, its initial item; then tasks $C$ and $D$ can be performed simultaneously, because they are parallel tasks. After task $C$ either task $E$ or task $F$ is carried out, determined by decision $d_1$. Decision $d_2$ finds the suitable instant to perform task $G$. 

39
Various criteria influencing decision d2 can be envisaged: for example, perform task G only after task E completion, otherwise do nothing, or, perform task G after task D completion, and tasks E or F completion. Decision d2 will be activated twice, because of our informally presented dynamic view on task structures. The first time after task D completion, and the second time after completion of tasks E or F. Task A will have been completed when all tasks will have been carried out, and all decisions taken. A formal definition of the dynamic interpretation of task structures is presented in chapter 9.

3.4.1 A JSD EXAMPLE

To assist in understanding task structures, we will discuss some examples of the JSD method. In Jackson (1983), JSD is introduced as a six step procedure, the first five of which are concerned with specification, the last one with implementation. No alternative strategies are presented at this level, and iteration is not explicitly covered.

JSD therefore consists of two successive tasks at the highest level: specification and implementation, see figure 3.6. The structure of the specification task is shown in figure 3.7. The specification task is a sequence of 5 tasks with the entity action step as the initial item. The following steps are taken in succession: the entity structure step, the initial model step, the function step, and the system timing step.
The entity action step consists of two parallel tasks: one constructs the entity list, the other the action list. The entity action step consequently consists of two tasks only, both being initial items. The structure is shown in figure 3.8.
Decomposing the task "Make the action list", we arrive at a cyclic task structure diagram in which each task can change the existing action list. The tasks "Define an action" and "Reject an action" are not further decomposed. We will refer to them as basic tasks, not consisting of subtasks. The task structure diagram of "Make the action list" shown in figure 3.9 contains the decision "How to adapt the action list?" as initial item.

The task "Describe an action" can be decomposed into tasks concerned with the specification of action attributes, and involved actions. The definition of a task structure is cumbersome at this level, if we only rely on Jackson (1983) and Cameron (1983). For example, an information engineer could move attributes from one action to another. However, such tasks have not been included, because we cannot detect such tasks if we only depend on literature. Similar arguments are valid for high level task structures. The simple task structure of figure 3.7 is based on literature, and could well differ from observing practitioners in real life, see chapter 4.

The JSD examples should therefore only be interpreted as illustrations of task structures and not as representations of actual ways of working when applying JSD - at least, in our opinion.
3.4.2 THE TASK STRUCTURE DEFINITION

After this introduction by intuition and example, we will now define task structures formally, using first order predicate logic.

Definition 3.6

A task structure $\mathcal{T}$ is a 7-tuple $\mathcal{T} = (T, D, L, N, F, G, I)$, where:

- $T$ is a non-empty set of tasks,
- $D$ is a set of decisions, and
- $L$ is a set of labels.

The set $T \cup D$ will be denoted by $O$ as the set of task objects. Both tasks and decisions are considered to be task objects.

- $N : O \rightarrow L$ is the function assigning labels to task objects. Different task objects can have the same names. The predicate \textit{label} will be used to denote this function with \textit{label}(o)=l stating that the name of task object o is equal to l.

- $F \subseteq O \times O$ is the relation defining the triggers. The predicate \textit{trigger} will represent this relation: \textit{trigger}(o_1,o_2) implies that task object $o_1$ triggers another task object $o_2$.

- $G : O \rightarrow L_t$ is the partial function defining the decomposition with $L_t = \{l \in L | \exists e \in T [\text{label}(t) = l]\}$. The function will be referred to by the predicate \textit{part_of}. The formula \textit{part_of}(o)=l means that task object $o$ is part of the decomposition of the class of tasks with name $l$. It is important to realize that although there can be various tasks with the same name, there is only one decomposition for all these tasks.

- $I \subseteq G$ is the partial function defining the initial items. It is a subset of the decomposition function. The predicate \textit{initial} is used to represent this function. \textit{initial}(o)=l denotes that task object $o$ is an initial item for the class of tasks named $l$.

We will first define some auxiliary predicates, and then the properties that should be valid for a correct task structure $\mathcal{T}$.
Definition 3.7

Tasks are other tasks' children if these tasks occur in the decomposition of those tasks or one of their subtasks. The predicate parent will be used to denote the inverse relation.

\[
\text{child} \subseteq T \times L_t \\
\text{child}(t_1,l_1) \equiv \text{part_of}(t_1)=l_1 \lor \exists l_2 \in L \exists t_2 \in T \left[ \text{part_of}(t_1)=l_2 \land \text{label}(t_2)=l_2 \land \text{child}(t_2,l_1) \right]
\]

Two task objects are connected if there is a directed path via triggers between them.

\[
\text{connected} \subseteq O \times O \\
\text{connected}(o_1,o_2) \equiv \text{trigger}(o_1,o_2) \lor \exists o_3 \in O \left[ \text{trigger}(o_1,o_3) \land \text{connected}(o_3,o_2) \right]
\]

The following predicate is true for two, possibly the same, tasks if they are connected by a directed path, only consisting of tasks.

\[
\text{task\_path} \subseteq T \times T \\
\text{task\_path}(t_1,t_2) \equiv \text{trigger}(t_1,t_2) \lor \exists t_3 \in T \left[ \text{trigger}(t_1,t_3) \land \text{task\_path}(t_3,t_2) \right]
\]

The following predicate is true for two, possibly the same, decisions if they are connected by a directed path, only consisting of decisions.

\[
\text{decision\_path} \subseteq D \times D \\
\text{decision\_path}(d_1,d_2) \equiv \text{trigger}(d_1,d_2) \lor \exists d_3 \in D \left[ \text{trigger}(d_1,d_3) \land \text{decision\_path}(d_3,d_2) \right]
\]

We can now formulate the properties valid for a task structure by using these auxiliary predicates.

Property 3.7

If tasks are children of other tasks they can never be parent tasks of those tasks. This property excludes loops in decomposition hierarchies.

\[
\forall t_1,t_2 \in T \forall l_1,l_2 \in L_t \left[ \left( \text{child}(t_1,l_2) \land \text{label}(t_1)=l_1 \land \text{label}(t_2)=l_2 \right) \Rightarrow \neg \text{child}(t_2,l_1) \right]
\]

Property 3.8

There is exactly one task not part of an other task. We will call this task the overall task.
\( \exists ! t_1 \in T \quad \forall l \in L_t \quad [\text{part}_o(t_1) = l] \)

**Property 3.9**

Decomposed tasks have at least two task objects as parts of those tasks.

\( \forall o_1 \in O \quad \forall l \in L_t \quad [\text{part}_o(o_1) = l \implies \exists o_2 \in O \setminus \{o_1\} \quad [\text{part}_o(o_2) = l]] \)

**Property 3.10**

Two task objects related to a trigger are part of same supertask.

\( \forall o_1, o_2 \in O \quad \forall l \in L_t \quad [(\text{trigger}(o_1, o_2) \land \text{part}_o(o_1) = l) \implies \text{part}_o(o_2) = l] \)

**Property 3.11**

All decisions are indirectly connected to tasks by triggers.

\( \forall d \in D \quad \exists t \in T \quad [\text{connected}(d, t)] \)

**Property 3.12**

All cycles (task objects connected to themselves) must at least contain one task and one decision.

\( \forall t \in T \quad [\neg \text{task}_o(t, t)] \land \forall d \in D \quad [\neg \text{decision}_o(d, d)] \)

**Property 3.13**

Task objects not being the overall task are either initial items or connected to initial items.

\( \forall o_1 \in O \quad \exists l \in L_t \quad [(\text{initial}(o_1) = l \land \neg \exists o_2 \in O \quad [\text{initial}(o_2) = l \land o_1 \neq o_2 \land \text{connected}(o_2, o_1)]) \lor (\neg \text{initial}(o_1) = l \land \exists o_2 \in O \quad [\text{initial}(o_2) = l \land \text{connected}(o_2, o_1)])] \)

**Property 3.14**

Decisions are always parts of supertasks.

\( \forall d \in D \quad \exists l \in L_t \quad [\text{part}_o(d) = l] \)
Property 3.15

Decisions cannot have triggers to two or more identically named tasks.

\[ \forall d \in D \ \forall t_1, t_2 \in T \ ((\text{trigger}(d, t_1) \land \text{trigger}(d, t_2)) \Rightarrow \text{label}(t_1) \neq \text{label}(t_2)) \]

The above nine properties define the set of syntactically correct task structures. Task structures define the set of possible modelling strategies. A specific modelling process can therefore be considered as a set of task and decision instances. More details about the formal relationships between a specific modelling process and a task structure can be found in chapter 9, and Wijers et al. (1990).

3.5 INTEGRATING TASK AND CONCEPT STRUCTURE

We are about to establish links between a task structure and a concept structure, based on the relationships between a way of working and a way of modelling, as identified in section 2.5. The main subject of section 2.5 were the levels of inclusion, precision and coherence, and the influence of the modelling environment on modelling strategies. Also types of graphical representation were identified as a specific link between a way of modelling and a way of working. We will not discuss these representational problems in detail here, as they are concerned with layout and user interface problems rather than with identifying the required modelling concepts and the way these are applied in a modelling process. We do not wish to be distracted by these matters in discussing conceptual meta-models of modelling processes to be supported.

3.5.1 THE LEVEL OF INCLUSION AND TASK SCOPES

Tasks consist of activities to be performed in order to achieve certain objectives. Modelling task objectives involve the manipulation of models which are generally instances of only a part of the concept structure. Tasks have a specific view by which problems are analysed. That part of a concept structure available for manipulation by a task will be called the task scope. The set of association and concept instances outside a task scope cannot be changed by the task. Figure 3.4 shows the scope of the JSD task "The entity structure step". Because a task can be decomposed into subtasks performing parts of the task, the subtask scopes are subsets of the supertask one. On the other hand, a task should be capable of fully manipulating the concepts and associations within its scope, and consequently this scope must constitute the union of the subtask scopes.
3.5.2 The level of precision and a-priori and a-posteriori rules

A certain level of precision must be maintained during and after task execution. Models manipulated by a task must satisfy a set of verification rules. We consequently have related verification rules to tasks: a-priori and a-posteriori ones, monitored while tasks execute and thereafter, respectively. The verification rule language is described in section 3.3. A-priori rules must have been met at the outset of a task. As these rules should also be valid during execution, a certain inheritance of a-priori rules is recognized: a-priori rules for supertasks apply to their subtasks as well. A task may only terminate if no a-priori and a-posteriori rules have been violated.

The JSD example in figure 3.4 demonstrates that typical a-priori rules define (1) that not-a-leaves are either iterations, selections or sequences, and (2) that only one root is available for each structure diagram. A typical a-posteriori rule limits the number of entities for each specified structure diagram to exactly one.

3.5.3 The level of coherence and information places

Task coherence implies that the output of specific tasks is adhered to by other tasks. Coherence requires explicit information to be exchanged between tasks, the information only to be retained temporarily in what we call information places. These can be considered as temporary repositories of certain concept instances. The type of an information place must be identical with a concept in the concept structure. Tasks can deposit instances in these places (output), refer to them (refer-to), and remove them (consume).

![Diagram](image)

Figure 3.10 A possible task structure for the JSD entity structure step
The graphical conventions of task structures have been extended to include the graphical definition of information place manipulation, see the JSD example in figure 3.10. Information places are represented by rectangles, output and consume manipulations by single arrows and refer-to manipulations by double arrows.

The simple JSD example demonstrates the use of information places. It shows a possible task structure for the JSD entity structure step in which structure diagrams must be created for all entities. We recall the concept "entity" of figure 3.4, associated with the concept "entity structure diagram". The task "Select an entity" deposits a selected entity in the information place "current entity", and the task "Define a structure diagram" defines a structure diagram for the selected entity.

Information places which are input to tasks constrain the possible manipulations within these tasks. For example, in the task "Define a structure diagram" of figure 3.10, the association between "structure diagram" and "entity" can only be populated by the entity in the information place "Current entity". In addition, the only removable association instances are the ones which involve this entity.

The consumption of, or reference to information places sometimes requires the definition of the roles for which the information place can be consumed or referred to. Intended roles should be included if the instances in the refer-to or consume information places may only play specific roles in the task. Homogeneous associations and multiple associations between identical concepts can particularly cause problems. In homogeneous associations two or more roles are played by concepts with the same pater familias, see figure 3.11.

![Diagram of intended roles and homogeneous associations](image)

Figure 3.11 Intended roles and homogeneous associations
The contents of information places consumed by a task are emptied at task termination, while the ones referred to will retain their contents. If output information places have been defined for a task, they will be filled with the appropriate concept instances at task termination. Output to information places takes place in a predefined way. The instances created during a task corresponding to the type of the information place will be entered in the information place. Intended roles can again restrict the set of output instances by requiring that the output instances play specific roles in specific associations.

The above intuitively presented meaning of input and output information places has been formally defined in Wijers et al. (1990). This default meaning can be overruled by basic procedures defined for basic tasks. Basic procedures perform specific model manipulations, such as the creation, removal and selection of concept instances.

Information places can be referred to explicitly or implicitly by a given task. Explicit reference implies the definition of an explicit refer-to manipulation between task and information place. Information places are implicitly referred to if they provide explicit input for one of the task's parent tasks.

Figure 3.12 The difference between information places implicitly referred to and explicitly referred to
For example, the information place "current diagram" in part (1) of figure 3.12 is referred to implicitly by all subtasks of "specify structure diagram". The information place is emptied after termination of the task "specify structure diagram". In part (2), the same information place is referred to by each "define box" task as well, but explicitly. In this case it is difficult to empty the information place. A dummy "empty place" task is required to empty the information place "current diagram".

Information places can be local for tasks, i.e. defined at the same level, or global, i.e. defined at a higher level in the task hierarchy. Tasks can manipulate both local and global information places.

Summarizing, the set of information places related to a task can be:

- defined as part of the task,
- implicitly referred to by the task,
- locally or globally referred to by the task,
- locally or globally consumed by the task, and
- locally or globally filled by the task.

Information places may only contain specific types of concepts, and intended roles can be defined for explicit refer-to, consume and output manipulations.

**3.5.4 THE INFLUENCE OF THE MODELLING ENVIRONMENT AND DECISION RULES**

We have so far strongly emphasized the way tasks are allowed to change models and the contents of information places. Decisions are still considered as black boxes with certain outcomes. The possible set of outcomes \( O(d) \) of decision \( d \) is defined as \( O(d) = \{ o \in L | trigger(d, o) \land label(o) = l \} \cup \{ "nothing" \} \).

The set of possible outcomes of a decision is defined by its outgoing triggers and the label "nothing". The outcome "nothing" should be interpreted as the conclusion that no task or decision resulting from the decision was considered as appropriate. For instance, our JSD examples contain decisions such as "How to adapt the action list?" in figure 3.9, with the possible outcomes: "Define an action", "Describe an action", "Reject an action", and "nothing"; or "Are there more entities to be analysed?" in figure 3.10, with the possible outcomes: "Select an entity", and "nothing".

50
Not all outcomes are equally likely in all circumstances. The outcome of a decision depends on the modelling environment, which includes the process history, previous models, information place contents, and the domain and real world knowledge of users and information engineers.

We will demonstrate how decisions can be opened up by defining decision rules for them, which should be considered as heuristics determining the likelihood of an outcome in a given situation.

A decision rule consists of a condition and action part. The condition part can be a simple condition, or a composite logical expression consisting of various simple conditions concatenated by and-, or- and not-operators. Simple conditions may be purely informal natural language statements to be evaluated within a specific real world situation, or formal statements to be evaluated on the basis of process history, constructed models and information place contents.

The action part of a decision rule recommends specific outcomes. Decision rules can suggest various outcomes with varying certainty. The decision rule language handles uncertainty according to the certainty factor model of Buchanan and Shortliffe, see Lucas and van der Gaag (1988). The certainty factor model is considered as adequate, fairly easy to use, and reasonably matching our intuitive notion of uncertainty.

The above considerations resulted in the following decision rule syntax:

```
<rule> ::= If <conditions> Then <outcomes> '.'
<conditions> ::= <simple_condition>
               | ('<conditions>' 'or' '<conditions>' '{or' '<conditions>' '}'
               | ('<conditions>' 'and' '<conditions>' '{and' '<conditions>' '}'
               | 'not' '('<conditions>' ')')
<outcomes> ::= <outcome> [with cf <cf>]
             | [and <outcome> [with cf <cf>]]
```

More details on the formal evaluation of decision rules can be found in Wijers et al. (1990) and Huizinga (1990).

### 3.5.5 THE META-MODEL DEFINITION

We will discuss the syntax of an interlinked task and concept structure in this section. This integrated structure is called a meta-model.
Definition 3.8

A meta-model $M$ is a 18-tuple with

$M = (T, C, T_v, P, T_{prl}, T_{pos}, B, B_p, I_p, L_p, I_n, I_d, I_1, I_r, I_t, I_b, D_T, D_a)$,

where:

- $T$ is a task structure with $T = (T, D, L, N, F, G, I)$.
- $C$ is a concept structure with $C = (C, A, R, S, Q)$.
- $T_v \subseteq L_t \times (C \cup A)$ denotes task scopes. The predicate $\text{include}$ is used to represent this relation where $\text{include}(l, o)$ implies that model object $o$ is part of the scope of the class of tasks labelled $l$.
- $P$ is a set of verification rules.
- $T_{prl} \subseteq L_t \times P$ is the relation defining the a-priori rules for a task.
- $T_{pos} \subseteq L_t \times P$ is the relation defining the a-posteriori rules for a task.
- $B$ is a set of basic procedures.
- $B_p : T_b \rightarrow B$ is the function mapping the set of basic tasks $T_b$ to basic procedures where $T_b = \{i \in L_t \mid \exists o \in O \text{ part_of}(o) = i\}$.
- $I_p$ is the set of information places.
- $L_p$ is the set of information place labels.
- $I_n : I_p \rightarrow L_p$ is the function assigning labels to information places. We use the predicate $\text{place_name}$ to denote this function with $\text{place_name}(i) = l$ meaning that information place $i$ is labelled $l$.
- $I_d \subseteq T \times I_p \times \{\text{consume, refer, output}\}$ is the relation defining the manipulation by tasks of information places part of the same supertask. The relation is referred to by the predicate $\text{local_manipulation}$.
- $I_1 \subseteq T \times I_1 \times \{\text{consume, refer, output}\}$ is the relation defining the manipulation by tasks of information places higher in the hierarchy with $I_1 : \{i \in L_p \mid \exists p \in I_p[I_n(p) = l]\}$. The relation is referred to by the predicate $\text{global_manipulation}$.
• $I_r : (I_d \cup I_t) \rightarrow \mathcal{P}(A \times N)$ is the function defining the intended roles with $\mathcal{P}$ our symbol for the powerset. The predicate roles indicates this function.

• $I_t : I_t \rightarrow C$ is the function defining the type of an information place. The function is identified by the predicate type.

• $I_s : I_p \rightarrow L_t$ is the function specifying the supertask of an information place. The predicate supertask denotes this function.

• $D_t$ is a set of decision rules.

• $D_s \leq D \times D_t$ is the relation indicating the decision rules for a decision.

We will now define the properties required for a correct meta-model $M$ in two parts: properties concerning task scopes in section 3.5.5.1, and properties relating to information places in section 3.5.5.2.

3.5.5.1 PROPERTIES CONCERNING TASK SCOPES

Before specifying the properties valid for task scopes, we define an auxiliary function.

Definition 3.9

The following function yields the set of model objects part of the scope of a task.

\[
\text{task\_scope} : (T \cup L_t) \rightarrow \mathcal{P}(C \cup A) \\
\text{task\_scope}(t) = \{o \in C \cup A | \text{include}(t, o)\} \text{ if } t \in L_t \\
\text{task\_scope}(\text{label}(t)) \text{ if } t \in T
\]

Property 3.16

Every task should have a scope.

\[
\forall t \in T \ [\text{task\_scope}(t) \neq \emptyset]
\]

Property 3.17

The scope of a task is a subset of the scope of its supertask.
\[ \forall t \in T \ \forall l \in L_t \ [\text{part_of}(t)=l \Rightarrow \text{task_scope}(t) \subseteq \text{task_scope}(l)] \]

**Property 3.18**

The scope of the overall_task is the complete concept structure.

\[ C \cup A = \text{task_scope(overall_task)} \]

**Property 3.19**

The scope of a task equals the union of the scopes of its subtasks.

\[ \forall l \in L_t \ [\text{task_scope}(l)=\{o \in C \cup A | \exists t \in T \ [\text{part_of}(t)=l \land o \in \text{task_scope}(t)]\}] \]

**Property 3.20**

If a subtype is part of a task scope, then its supertype should also be part of that task scope.

\[ \forall l \in L_t \ \forall c_1, c_2 \in C \ [(c_1 \in \text{task_scope}(l) \land \text{subtype}(c_1, c_2)) \Rightarrow c_2 \in \text{task_scope}(l)] \]

**Property 3.21**

If a concept is an objectified association and part of a task scope, then the association should also be part of that task scope and vice versa.

\[ \forall l \in L_t \ \forall c \in C \ \forall a \in A \ [(c \in \text{task_scope}(l) \land \text{objectification_of}(c)=a) \Rightarrow a \in \text{task_scope}(l)] \land \]
\[ \forall l \in L_t \ \forall c \in C \ \forall a \in A \ [(a \in \text{task_scope}(l) \land \text{objectified_into}(a)=c) \Rightarrow c \in \text{task_scope}(l)] \]

### 3.5.5.2 Properties Concerning Information Places

We will first define some auxiliary functions, and then the properties of information places.

**Definition 3.10**

A parent task relevant for a certain task with respect to a specific information place is an indirect parent of that task and is supertask of the specific information place.

\[ \text{relevant_parent} \subseteq T \times T \times I_1 \]
relevant_parent(s,t,p) = parent(s,t) \land part_of(t) \neq label(s) \land \exists i \in I_p \noindent \hspace{1cm} \text{place_name}(i) = p \land \text{supertask}(i) = label(s)]

A direct parent for a certain task concerning a specific information place is the relevant parent which has no other relevant parents as child.

\[
\text{direct_parent} \subseteq T \times T \times I_1
\]

\[
\text{direct_parent}(s,t,p) \equiv \text{relevant_parent}(s,t,p) \land \neg \exists u \in T [\text{parent}(s,u) \land \text{parent}(u,t) \land \text{relevant_parent}(u,t,p)]
\]

**Property 3.22**

For local manipulations, the involved task and information place should be part of the same supertask.

\[
\forall t \in T \forall i \in I_p \forall x \in \{\text{consume}, \text{refer}, \text{output}\} [\text{local_manipulation}(t,i,x) \Rightarrow part_of(t) = \text{supertask}(i)]
\]

**Property 3.23**

For global manipulations, the involved information place should be higher in the task decomposition hierarchy than the involved task.

\[
\forall t \in T \forall i \in I_1 \forall x \in \{\text{consume}, \text{refer}, \text{output}\} [\text{global_manipulation}(t,i,x) \Rightarrow \exists s \in T [\text{relevant_parent}(s,t,i)]]
\]

**Property 3.24**

Two information places with identical names cannot exist in the same task.

\[
\forall i, j \in I_p [\text{supertask}(i) = \text{supertask}(j) \Rightarrow \text{place_name}(i) \neq \text{place_name}(j)]
\]

**Property 3.25**

If an information place is globally manipulated by a task then every path up in the decomposition hierarchy should contain a task (not the direct supertask of the involved task) which is supertask of that information place.

\[
\forall t \in T \forall i \in I_1 \forall x \in \{\text{consume}, \text{refer}, \text{output}\} [\text{global_manipulation}(t,i,x) \Rightarrow \forall s \in T [(\text{parent}(s,t) \land part_of(t) \neq label(s)) \Rightarrow \exists u \in T [\text{relevant_parent}(u,t,i) \land \text{parent}(u,s) \lor \text{parent}(s,u) \lor s = u]]]
\]
Property 3.26

Intended roles linked to an information place manipulation have to involve the concept being the type of the information place.

\[
\forall i \in I_p \ \forall t \in T \ \forall x \in \{\text{consume, refer, output}\} \ \forall a \in A \ \forall n \in N \\
(((local\_manipulation(t,i,x) \land (a,n) \in \text{roles}(t,i,x)) \Rightarrow role(a,n) = \text{type}(\text{place\_name}(i))) \land \\
\forall i \in I_1 \ \forall t \in T \ \forall x \in \{\text{consume, refer, output}\} \ \forall a \in A \ \forall n \in N \\
(((global\_manipulation(t,i,x) \land (a,n) \in \text{roles}(t,i,x)) \Rightarrow role(a,n) = \text{type}(i))
\]

Property 3.27

An information place may not be input or output of both a task and one of its parents.

\[
\forall i \in I_1 \ \forall s, t \in T \ \forall x \in \{\text{consume, refer, output}\} \\
(((global\_manipulation(t,i,x) \land \text{direct\_parent}(s,t,i)) \Rightarrow \\
\neg \exists u \in T \ \exists y \in \{\text{consume, refer, output}\} \ [\text{parent}(u,t) \land \text{parent}(s,u) \land \\
\text{global\_manipulation}(u,i,y) \lor \exists p \in I_p [\text{local\_manipulation}(u,p,y) \land \\
\text{place\_name}(p) = i)])
\]

Property 3.28

Every information place is input for and output of a task.

\[
\forall i \in I_p \ \exists t \in T \ \exists x \in \{\text{consume, refer}\} [\text{local\_manipulation}(t,i,x) \lor \exists p \in I_p \exists s \in T \\
[\text{place\_name}(i) = p \land \text{global\_manipulation}(s,p,x) \land \text{direct\_parent}(s,t,p) \land \\
\text{supertask}(i) = t)] \land \\
\forall i \in I_p \ \exists t \in T \ \exists x \in \{\text{output}\} [\text{local\_manipulation}(t,i,x) \lor \exists p \in I_p \exists s \in T \\
[\text{place\_name}(i) = p \land \text{global\_manipulation}(s,p,x) \land \text{direct\_parent}(s,t,p) \land \\
\text{supertask}(i) = t)]
\]

Property 3.29

An information place cannot be consumed and referred to by the same task.

\[
\forall t \in T \ \forall i \in I [-(\text{global\_manipulation}(t,i,\text{consume}) \land \\
\text{global\_manipulation}(t,i,\text{refer})) \land \\
\forall t \in T \ \forall i \in I_p [-(\text{local\_manipulation}(t,i,\text{consume}) \land \text{local\_manipulation}(t,i,\text{refer}))]
\]
3.6 SUMMARY

In this chapter we have presented a technique for the representation of modelling knowledge, primarily extending and formalizing the concepts introduced in chapter 2. This meta-modelling technique is related to our first hypothesis about modelling knowledge being adequately represented by considering it as encompassing a way of modelling, a way of working, and their interrelationships. We conclude that the suggested technique allows such an integrated vision.

Concept structures can be used to describe ways of modelling. They consist of concepts, associations, specializations and objectified associations. Task structures describing ways of working have been introduced; they are specified by tasks, decisions, triggers, decomposition relationships and initial items. Various links between concept and task structures can be defined. We have identified task scopes, a-priori and a-posteriori verification rules, information places, basic procedures and decision rules.

In the next chapter we will concentrate on our second hypothesis; it postulates that detailed modelling knowledge can be acquired from experienced practitioners in specific fields of modelling. We will discuss the acquisition of modelling knowledge, and present a suitable approach relying strongly on the meta-modelling technique introduced in this chapter.
4

ACQUISITION OF MODELLING KNOWLEDGE

4.1 INTRODUCTION

In this chapter, we discuss an approach for the *acquisition* of modelling knowledge strongly influenced by general knowledge acquisition methods, see Breuker and Wielinga (1984), Ericsson and Simon (1984), Kidd (1987a), Slatter (1987), Wright and Ayton (1987), Guida and Tasso (1989a), and Hart (1989). We review these methods, and compare their strengths and weaknesses, in section 4.2.

The specific acquisition approach for modelling knowledge is explained in section 4.3. Roughly speaking, four stages can be distinguished: *preparation*, *elicitaiton*, *interpretation* and *conceptualisation*. Elicitation is based on concurrent protocol analysis. In the interpretation and conceptualization stages, the meta-modelling technique discussed in the previous chapter constitutes the main ingredient of the approach. Section 4.4 summarizes the findings described in this chapter and introduces the specific experimental setting for the experiments presented in chapters 5 to 7.

4.2 OVERVIEW OF KNOWLEDGE ACQUISITION METHODS

In Kidd (1987b), knowledge acquisition is defined as a process involving eliciting, analysing and interpreting the knowledge human experts use in solving particular problems, and then transform it to suitable machine representations. Wright and Ayton (1987), and Grabowski (1988) refer to knowledge acquisition as a process eliciting and modelling expert knowledge for subsequent representation in rule-based expert systems. Both definitions rely on technological considerations. They implicitly adhere to a way of thinking in which the ultimate system is expected to *emulate* the expert, see Slatter (1987). Two problems arise consequently:

- Knowledge acquisition is considered to be only effective in *expert systems development*. Other applications, more related to general cognitive studies, are ignored.

- The acquired knowledge is implicitly considered only useful in expert *emulation*, while other effective functions are left out.

The first problem has also been identified in Hart (1989), and Breuker and Wielinga (1987), who distinguish two major functions in knowledge acquisition:
ACQUISITION OF MODELLING KNOWLEDGE

- data elicitation on knowledge in certain domains, and

- data analysis, i.e. the transformation of data to interpretative frameworks, varying from weak classification schemes to solid models.

Hart (1989) regards knowledge acquisition as a process in which knowledge is extracted and represented in suitable conceptual models, strongly depending on their structure on specific situations. Knowledge acquisition should only be considered as the process of extracting and modelling experts' problem solving knowledge without technical considerations.

Concerning the second problem, Hart (1989) finds it unlikely and undesirable for programs to replace human experts completely, but they can be applied very successfully as supporting tools. Kidd (1987b) argues that knowledge engineers need to decide on the appropriate modality for proposed systems, e.g. automated experts, consultants, or tutors. Furthermore, users are active agents in interactive systems, and their capabilities should be used to advantage. Should proposed systems be applied to support experts, then knowledge engineers should identify specific weaknesses in human reasoning, and design systems to complement these.

We consequently prefer the term knowledge based system (KBS) to expert system, because the latter is too strongly reminiscent of systems that solve problems in isolation by emulating experts. We consider knowledge based systems to contain knowledge of certain domains which can be applied in various ways.

4.2.1 KNOWLEDGE ACQUISITION IN A KBS DEVELOPMENT PROCESS

In Hayes-Roth et al. (1983), Bolesian (1987), and Guida and Tasso (1989b), various stages in KBS development projects are identified, see table 4.1. The Bolesian (1987) approach was primarily based on Breuker and Wielinga (1987). A more complete survey of various KBS development approaches can be found in Guida and Tasso (1989b).

Hayes-Roth et al., and Guida and Tasso strive for a highly iterative development path often referred to as rapid prototyping. Hayes-Roth et al. explicitly mention iterations from conceptualization to testing.

Guida and Tasso wish to introduce a way of controlling into their method, and they have consequently divided their rapid prototyping approach into three stages, corresponding to a demonstration prototype, a full prototype and a target system implementation.
Table 4.1 Stages identified in a KBS development process

Bolesian favours a structured development path with knowledge acquisition and implementation strictly separated. The stages of orientation, and problem identification and analysis are iterative, because each stage identifies elicitation and analysis tasks. The major reasons for separating acquisition from implementation are explained in Breuker and Wielinga (1989):

- Rapid prototyping means that AI solutions are applied and systems are constructed as soon as knowledge engineers have found a way of structuring some of the (verbal) data. Because the implementation formalisms do not extend beyond the "logical" level, rapid prototyping easily leads to backtracking and abandoning of systems in more complex domains. At least, it does not result in an effectively controlled system development life cycle.

- Knowledge acquisition in rapid prototyping often deteriorates because of implementation formalisms. Today's more advanced, less rigid knowledge engineering environments such as KEE, ART and Knowledge Craft therefore render knowledge acquisition problems more urgent, because there is an embarras de choix in AI solution methods.

Based on these reasons, Breuker and Wielinga advocate a complete, implementation independent conceptual model resulting from knowledge acquisition. Knowledge engineers develop abstract models of expertise from (verbal) data, possibly followed by a transformation to a KBS architecture in the design phase. The subsequent implementation is carried out using expert systems shells, high level AI environments, or more traditional implementation tools as used in Weelderan and Sol (1990).
Summarizing, two approaches are generally applied to KBS development, both heavily relying on knowledge acquisition: rapid prototyping, and structured development. The rapid prototyping approach depends on incremental acquisition leading to quickly produced initial prototypes. As mentioned earlier, the approach tends to be implementation oriented in the knowledge acquisition stage already, and appears to ignore conceptual problems. The structured development approach advocates thorough acquisition before system construction. High level technology independent languages can be found particularly in the structured development approaches which focus on knowledge acquisition, as presented in Breuker and Wielinga (1989), Motta et al. (1989), and Hart (1989). These approaches appear to ignore the potential role of prototypes for elicitation.

We prefer a structured approach in our study because we wish to ignore technical considerations before having solved the acquisition problem. We believe that knowledge acquisition should result in conceptual models "independent" of the ultimate implementation of this knowledge in systems, see also the meta-modelling technique discussed in chapter 3. The design of this technique has been based on conceptual considerations about the aspects we consider important in describing modelling knowledge. In chapter 9, after having demonstrated the feasibility of the meta-modelling technique and the acquisition approach, we will study ways of applying the acquired knowledge in MSS environments.

We will now concentrate on knowledge acquisition as described in the above structured approaches. As mentioned before, knowledge acquisition is essentially divided into data elicitation and analysis, leading to conceptual models of the analysed tasks. We will first present a few popular elicitation methods in knowledge acquisition, and we will then discuss various analysis methods found in literature.

4.2.2 KNOWLEDGE ELICITATION METHODS

Although much knowledge available in a knowledge domain may be explicit and accessible in public documents, it is evident in KBS development that most knowledge in expert performance is not directly observable. In particular, the type of knowledge, and the way it is used, and under which conditions, is generally not well coded and documented, and must therefore be elicited from experts and other specialists.

The most popular methods used in eliciting expert knowledge are listed below, see also Breuker and Wielinga (1984), Slatter (1987), Wright and Ayton (1987) and Hart (1989).
Interviewing

Interviewing is the most familiar method, widely used because it is relaxed and acceptable. Interviews are recorded question-answer sessions. There are two types of interview: unstructured and structured. Unstructured (or focussed, directed) interviews closely resemble normal conversation. A list of topics is prepared in advance, and discussed in a breadth-first way. The method provides rough insight into domains. Structured interviews can be compared with interrogations. Knowledge engineers attempt to elicit knowledge about concepts or models by continuously soliciting clarifications, explanations, consequences, and justifications. Lists of specific questions are prepared. The evaluation of answers is paramount in these interviews.

Slatter (1987) summarizes the main advantages of interviews: they are effective in rapidly eliciting basic domain structures, and much of the knowledge explicit to experts can be elicited easily. Interviewing is unsuitable for eliciting detailed or poorly accessible domain knowledge, and it relies heavily on uncued recall, which is a disadvantage. This has also been mentioned in Wright and Ayton (1987), who found that interviews especially encourage experts to speculate on and theorize about their cognitive processes. Nisbett and Wilson (1977) have argued that we have no conscious access to mental processes, but only to their mental products.

Think aloud protocols

In these protocols, experts are asked to think aloud (or talk) about every thought and action while working on problems in the field of expertise. This verbalization is recorded and then typed out. The problems may be real or imaginary ones. An authoritative work on protocol analysis is by Ericsson and Simon (1984). Ericsson and Simon (1980) have argued that think aloud protocols do not change task performances, although they are slowed down. However, asking experts to explain why they are doing what they do, requires them to attempt to access additional knowledge, which disturbs task performance. So, although experts are frequently encouraged by phrases like "keep talking", or "what are you thinking about .....?", these reminders should not ask for explanations, see Ericsson and Simon (1984). Thinking aloud does not elicit explanations: asking for explanations does.

Think aloud protocols are particularly suited to eliciting information about the when's and how's of the use of specific knowledge, and to abstract the reasoning strategies and task decompositions followed. Protocol analyses have the disadvantages of being time consuming and that, similarly to interviews, knowledge engineers can only capture a series of verbal statements which must
be converted to knowledge modules by an interpretative process of sifting, selection and re-representation. Furthermore, in think aloud protocols, knowledge engineers should be aware of the experts' inexperience in self-reporting and of the inaccessibility of proceduralized knowledge.

Think aloud protocols as defined above are known as concurrent protocols. Some other think aloud protocol forms are selective protocols, simulation by teletype and retrospective protocols (i.e. observation and review). Only part of the task is performed in a think aloud way in selective protocols. Simulation by teletype has been used in dialogues where experts could not be provided with completely specified problem statements at forehand, see Winkels et al. (1986). The dialogues are held via teletypes, and both parties are asked to think aloud.

In retrospective protocols, task performances are recorded on audio or video. Reviewing the tape after having solved the problem, the experts are invited to comment on their thoughts and actions. However, retrospective protocols are not so reliable as concurrent verbalizations, because they depend too much on human memory and biases, see Hoc and Leplat (1983) and Ericsson and Simon (1984).

Introspection protocols

Introspection involves thinking aloud while solving artificial problems, and is therefore unlike thinking aloud in problem solving because the process is not based on actual problems. Experts are encouraged to suggest feasible solutions, possibly relying on previous experience. Introspection protocols are shorter than think aloud ones, and contain many "meta-descriptions" and process comments, see Newell and Simon (1972). Introspection is effective in getting experts to sketch their views on the strategies they apply in problem solving, including their justifications.

Observational studies

Observations are similar to think aloud protocols, but experts are not expected to think aloud while working. They can be recorded on video, and could include telephone conversations. Observations are particularly useful in determining the experts' roles and activities in processes, but poor in establishing the applied reasoning strategies.

Psychological methods

Some cognitive psychology methods have been used in KBS development as well. The main ones are multi-dimensional scaling, the repertory grid technique, card sorting, and laddering. These methods are especially valuable in finding subtle
discriminations between various concepts of the experts, and the corresponding discriminating factors.

4.2.3 ANALYSIS OF ELICITED KNOWLEDGE

Analysis in a KBS development process concerns abstracting and interpreting the results of elicitations into a conceptual model of expert knowledge. Most elicitation methods, especially the popular interviews and think aloud protocols, produce verbal data from which the elicited knowledge has to be obtained by further analysis. This is called text analysis, also done in collecting knowledge from other written material, such as journal articles, books and documents. Because text analysis starts with unstructured verbal data, the analysis task in KBS development typically follows a bottom-up approach.

In text analysis, knowledge engineers organize the textual material by recognizing and highlighting salient words, lines and paragraphs, and by annotating and grouping of related text fragments (also called chunks or segments). Units of knowledge are isolated from the raw text by defining potentially interrelated fragments of text. Text analysis results in what we call text based models of the analysed domains.

Text analysis is currently supported by knowledge acquisition tools such as KEATS-1, Acquist, KRITON, KPT and Shelley, see Anjewierden (1987), Motta et al. (1989), Diederich and Linster (1989), and Anjewierden (1990), respectively. These tools help in setting up a text based model consisting of text fragments, objects, groups and links.

In analysing texts, knowledge engineers decide on the relevant portions of interview or protocol transcripts. Interpretation is essential in text analysis, and explicit documentation of the extracted fragments is therefore vital. The fragments are assigned to text objects which can be considered as labels representing items of interest. Groups can be regarded as higher order objects providing an abstract rationale for a number of text objects. Groups are classes labelling sets of text objects. The main use of groups is to factor text objects into meaningful categories, facilitating the acquisition of knowledge in large domains. Fragments, text objects and groups can be linked in various ways. Knowledge engineers can specify a wide range of relationships between objects, groups and fragments by using links, and are consequently not restricted in representing the knowledge contained in the transcripts.

Knowledge engineers transform a text based model into a conceptual model, expressed in a specific conceptual modelling language (sometimes directly into
an implementation language, as in most rapid prototyping approaches). The importance of conceptual modelling languages, independent of ultimate implementations, is emphasized by Schreiber et al. (1987) and Weelderen (1991). Well-known diagrammatic conceptual modelling languages used in KBS development are semantic nets, inference structures, and hierarchies of e.g. objectives, tasks and objects. Textual modelling languages are essentially used for specifying rules. Interpretation and conceptualization can lead to identifying missing information, which can result in additional elicitation tasks.

4.3 ACQUISITION APPROACH FOR MODELLING KNOWLEDGE

An approach for modelling knowledge acquisition is described in this section. We have identified three major tasks in section 4.2: elicitation, interpretation, and conceptualization, see figure 4.1, all requiring an appropriate preparation.

![Diagram of the four tasks within the modelling knowledge acquisition approach](image)

Figure 4.1 The four tasks within the modelling knowledge acquisition approach
Elicitation captures verbal data. Interpretation leads to what we call a text based model which is further refined and transformed into a conceptual meta-model in the conceptualization task. Both interpretation and conceptualization may require additional elicitation tasks, possibly leading to new text based models and modification of the conceptual meta-model.

Each of the four tasks will be discussed in more detail in the following sections.

4.3.1 THE PREPARATION TASK

The preparation task is comparable to the identification or initiation stages, as shown in table 4.1 in section 4.2.1. Experienced information engineers (we call them experts), in whose modelling capabilities we are interested, are selected in the preparation task according to the following criteria:

- more than 10 years' experience in the field,
- modelling capabilities proven in a wide range of applications, and
- position of leading authority within the company.

Preparation consists of estimating the time required for acquisition, of familiarizing information engineers with the elicitation, interpretation and conceptualization tasks, and of defining technical and organizational arrangements for the elicitation sessions.

4.3.2 THE ELICITATION TASK

The elicitation task consists of a concurrent think aloud protocol using simulation by teletype. The main reason for starting with a think aloud protocol is that information engineers do not have explicit knowledge about the way modelling tasks are being carried out. This knowledge must be cued by actual modelling. Adequate problems to be solved by information engineers during the elicitation task must therefore be selected and prepared.

Observational studies of information engineers setting up an organization's information architecture have been described in Wijers (1988a). Although these studies provide insight about information engineers' activities and communication with users, they fail to uncover underlying thoughts. In order to reveal these thoughts, the information engineers have to think aloud while modelling. However, they generally also communicate with the user community, and communication and thinking aloud obviously cannot be done simultaneously by
using only one communication medium such as speech. Wijers (1988a) demonstrates that normal dialogue situations interfere with think aloud processes. These considerations have led to a protocol setting where communication and thinking aloud are conducted via separate media. As thinking aloud can only be done by speech, communication is restricted to telecommunication. Messages can be exchanged between information engineer and user using terminals. Diagrams can be exhibited on whiteboards and television. Figure 4.2 illustrates the protocol setting for concurrent think aloud protocols with information engineers.

![Diagram of protocol setting](image)

**Figure 4.2 Protocol setting**

The knowledge engineer in the expert's room observes the information engineer's behaviour, and occasionally encourages the latter to think aloud, while operating the video camera and tape recorder. The typist supports the information engineer (or user) by typing messages. The teletype connection allows concurrent typing, so that incoming messages can be interrupted, as in normal dialogues. The information engineer and the user both think aloud, while the tape recorder and the video camera register. Protocol sessions may take from a few hours to several days, depending on the nature of the modelling task. Because thinking
aloud does not come naturally, the participants exercise thinking aloud before the sessions start. Recordings are transcribed at the end of a protocol session, resulting in three transcripts: (1) the information engineer's think aloud protocol, (2) the one of the user, and (3) the communication between information engineer and user. These transcripts are then synchronized to link the individual think aloud parts and the communication part into one protocol transcript.

It is important to define models as accurately as possible. I understand from your words that a customer wants to make a reservation and that you then undertake the required actions to carry out the reservation, I don't quite understand reservation of a service in your model, is a service, are reservation and service two separate items or do you want to leave the model as it is? Could you explain?

You dropped your notes, ha, ha, oh, yes, I think that reservation and service are essentially the same thing but that we have several relationships, wow, that's complicated, that's going to take a while, these relationships being tied to the concept life cycle, it is started by the customer, that's status 1, it is either confirmed or not, and that's status 2, status 3 the customer pays, or doesn't, I'd prefer to have a single concept because otherwise we'll be in trouble later.

actions, I think I'll switch to the life cycle, so that we can take up a couple of processes,

reservation: 1 what the customer, 2 the administrative registration and control by us, yes, he registers, control is a, eh, it's a homonym, so its is a

reservation: I agree with the "required actions", I do not understand reservation of a service in your model, so, let's explain what we mean by service.

Reservation: 1 what the customer does, 2. the administrative registration and control by us.

Reservation: 1. what the customer does, 2. the administrative registration and control by us. So these are, it is in fact a homonym, so two meanings.

Figure 4.3 Transcript of a protocol session

Figure 4.3 shows part of a transcribed protocol translated in English. The left column is the information engineer's transcript, the right one that of the user, and the middle one represents the communication between the two. In the
middle column an arrow indicates the direction of communication. These transcripts are the input for the interpretation task.

In summary, an elicitation task consists of (1) performing a protocol session, (2) transcribing the recordings, and (3) synchronizing the transcripts.

4.3.3 THE INTERPRETATION TASK

Transcripts are interpreted by a text analysis in the interpretation task, see also section 4.2. The meta-modelling technique enables us to structure the text analysis in a specific way as explained in the following. Text analysis is strongly dependent on finding tasks and modelling concepts, together with their interrelationships. We will refer to these two parallel activities as task oriented and concept oriented text analysis, respectively.

We apply structured and unstructured interviews in the interpretation task, to obtain additional verbal data from information engineers for the clarification of specific transcript contents.

4.3.3.1 THE TASK ORIENTED TEXT ANALYSIS

In the task oriented text analysis we are especially interested in finding the main modelling tasks. A transcript is first split into disjoint fragments which are either interpreted as the construction of a specific part of a model using one type of representation, or as being completely unrelated to the explicit construction of a model, which is to be interpreted as a decision, as an implicit modelling task, or as a non-modelling task. Disjoint text fragments can be identified best by thoughts such as "I think I will switch to ...", "This is enough for the moment ...", "First I would like to know ...", and "Should I include other things in ...". Disjoint fragments either represent tasks or decisions.

In the case of decisions, fragments containing arguments for certain outcomes are looked for. For tasks, we try determine their relevance, and, if found in the transcript, we register the corresponding fragments. We examine task termination similarly, and register the reasons for ending tasks. Fragments identified as tasks are documented by rough descriptions of their main activities and summaries of results. Having obtained what we call task fragments, we attempt to identify smaller fragments corresponding to more specific tasks, or to combine various fragments leading to more general tasks.

During the general discussion of text analysis in section 4.2, this type of activity was presented as linking fragments to text objects which can be grouped and
linked additionally. In the task oriented text analysis, we try to group the identified text objects corresponding to specific text fragments either as "tasks", "decisions" or "arguments". In figure 4.4 we illustrate the interpretation of a protocol transcript into a text based model as part of a task oriented text analysis. Text objects in a text based model are always part of a group. In figure 4.4 the text objects shown are instance of the group labelled "task".

![Figure 4.4 The task oriented interpretation of a transcript into a text based model](image)

In a text based model, fragments are linked to text objects which are instances of groups. For task oriented text analysis in particular the groups of tasks, decisions and arguments are relevant. Arbitrary links can be specified between these objects. Important link types in the task oriented text analysis are the "precedence" and "decomposed" links.

### 4.3.3.2 THE CONCEPT ORIENTED TEXT ANALYSIS

In a concept oriented text analysis we try to find nouns or short phrases denoting concepts or associations used by information engineers. During interpretation we should be extremely conscious of the difference between the meta and application level, see figure 3.1. Knowledge acquisition is a process of eliciting, interpreting and conceptualizing the knowledge of experts in certain domains. For information engineers working in information systems development, their knowledge domain is modelling knowledge at the meta-level.
However, communication with the users normally occurs at application level, and most nouns and phrases should be therefore be placed at the application level, and should be regarded as instances of concepts at the meta-level. So, often text fragments in a protocol transcript have to be interpreted as instances of concepts at the meta-level. In text-based model construction, we directly link text fragments to concepts or associations at the meta-level. An example of a concept-oriented text interpretation is shown in Figure 4.5.

![Diagram showing concept-oriented interpretation of a transcript into a text-based model.]

Figure 4.5 The concept-oriented interpretation of a transcript into a text-based model.

A concept-oriented text analysis also involves the identification of basic concept manipulations performed by information engineers, which are registered as basic tasks. For example, the fragment in Figure 4.5 also corresponds to a basic task "identify a stimulus". We will again attempt to register the reason for performing that task simultaneously.

In summary, a text-based model is constructed from a protocol transcript in the interpretation task. Text fragments are marked as text objects, which are to be grouped as tasks, decisions, reasons for starting and terminating tasks, basic tasks, concepts, and associations. Arbitrary links can be defined between these objects and groups. A text-based model is input for the conceptualization task.

4.3.4 THE CONCEPTUALIZATION TASK

In the conceptualization task, text-based models are transformed or integrated into a conceptual meta-model, applying the meta-modelling technique introduced in Chapter 3.
We will now briefly describe the way in which the various components of the ultimate meta-model are obtained. The concept and task structure will be discussed first, and then the integration of the two, leading to one coherent conceptual meta-model. The activities are presented in arbitrary order because they are normally performed iteratively.

Structured and unstructured interviews can be held with information engineers for clarification purposes, and for validation of the ultimate conceptual meta-model.

4.3.4.1 CONSTRUCTION OF THE CONCEPT STRUCTURE

A concept structure is set up from the text objects in the text based models grouped as concepts and associations. By examining the various links between these text objects, both specialization hierarchies and associations not explicitly marked as text objects in the text based models can be found.

4.3.4.2 CONSTRUCTION OF THE TASK STRUCTURE

A task structure is devised in a top-down manner from text objects in the text based models marked as tasks or decisions. These "task objects" and "decision objects" are related by precedence links; this linear structure is analysed for the occurrence of iterations, i.e. the repeated execution of certain tasks and decisions. If repetitive structures are found, decisions governing the execution of iterations are included. Tasks can be decomposed by identifying existing "decomposed" links. Task structures are set up for these decomposable tasks in a similar way as described above.

4.3.4.3 LINKING CONCEPT STRUCTURE AND TASK STRUCTURE

Task scopes, verification rules, information places, basic procedures and decision rules further interrelate concept structure and task structure into one complete meta-model. Each of these components will be discussed briefly.

Task scopes

Task scopes can be determined easily because "concept objects" and "association objects" in the text based models are implicitly linked to corresponding "task objects" if the text fragments of a "concept object" or an "association object" are included in the text fragment of the "task object".
Verification rules

The "argument objects" embodied in the text based models for representing reasons of task termination, are transformed into a-posteriori verification rules if these reasons are concerned with specific characteristics of the models constructed. A-priori verification rules can be found only implicitly as rules never violated by information engineers. A-priori verification rules must particularly be validated by information engineers in structured interviews as they do not follow directly from the transcripts.

Information places

Information places are relevant when information engineers explicitly work on specific topics of interest for further study. This requires the identification of tasks performing selections. The selected items are stored in information places which are input for subsequent tasks.

Basic procedures

Basic procedures describe the predefined modelling behaviour exhibited by basic tasks. Knowledge engineers should document basic tasks with short descriptions of corresponding procedures. An intuitive meaning of the procedure to be executed in the basic task suffices for the conceptual meta-model.

Decision rules

Arguments for certain decision outcomes registered in the text based models are transformed into decision rules. Reasons for starting tasks can also be transformed into decision rules, if these reasons are related to the interpretation of environmental conditions. The occurrence of iterations in a task structure should also be justified. Reasons for performing an iteration are transformed into decision rules.

The conceptualization task results in a representation of the information engineer's modelling knowledge used in the protocol sessions, which is expressed in the meta-modelling technique defined in chapter 3.

4.4 SUMMARY

In this chapter we have presented our approach for the acquisition of modelling knowledge. Based on a review of current approaches in knowledge based systems
development we conclude that knowledge acquisition results in a conceptual model independent of the ultimate implementation of this knowledge in software.

Acquisition can be divided into elicitation and analysis. Various elicitation and analysis methods have been described in section 4.2. We selected a few of these methods, and integrated them into our approach, as described in section 4.3.

The acquisition approach for modelling knowledge consists of four major tasks: (1) preparation, (2) elicitation, (3) interpretation, and (4) conceptualization. The preparation task is concerned with the selection of an information engineer, and the preparation of the technical and organizational aspects of the other tasks. In an elicitation task we perform a concurrent think aloud protocol together with simulation by teletype.

The meta-modelling technique, as defined in chapter 3, plays an important role in the interpretation and conceptualization tasks. In the interpretation task, a text analysis is performed on the transcripts resulting from the elicitation task. This text analysis leads to a text based model consisting of interlinked text fragments, text objects and groups. The text analysis focusses on groups categorizing text objects which represent one of the notions in the meta-modelling technique. In a conceptualization task, a text based model is completely transformed and integrated into a conceptual meta-model obeying all conventions of a correct meta-model, see chapter 3.

In the next three chapters we will apply the presented acquisition approach to three experienced information engineers with a different methodical background (D2S2, YOURDON and ESM). The study involves the construction of an information architecture for specific organizations, as mentioned in section 2.10.

The information engineers participating in the experiments are instructed as follows:

"The person in the other room represents an organization for which you must develop an information architecture. We expect decomposition diagrams, certain types of dataflow and entity relationship diagrams to be used as typical modelling techniques. You are, however, free to work the way you like, because that is what we are particularly interested in. Please remember to think aloud while working. Good luck!".

The experiments are restricted to one elicitation session with each information engineer, as described in section 4.3.2. All information engineers must solve the same problem. The organization requiring an information architecture is briefly described in appendix A.
5

EXPERIMENT 1: D2S2

5.1 INTRODUCTION

This chapter describes the acquisition of D2S2 modelling knowledge required for the construction of an information architecture. D2S2 is a method marketed by CACI, and explained in section 5.2. The acquisition task was performed with a senior CACI consultant.

Section 5.3 describes the acquisition results, although the scope of this thesis limits its completeness. We will therefore concentrate on the overall strategy, and on certain interesting details in the specialities of the information engineer concerned. We will present the results by following the conceptualized task structure. This chapter concludes with a summary of the major results of the experiment in section 5.4.

5.2 D2S2

Authorities on the D2S2 method (development of data sharing systems) are Palmer, Rock-Evans and MacDonald; D2S2 overviews are given by MacDonald and Palmer (1982), and Lith (1983).

The method was developed in 1974 following the application of a number of database systems development techniques, such as entity relationship and process logic diagrams. It gradually developed into a method, including information strategy planning and covering information systems development from analysis to production, in more or less detail.

D2S2 has strongly influenced the construction of the Information Engineering method (IE), see Martin (1986a, 1986b, 1988) and MacDonald (1986, 1988), and the Integrated Resource Analysis and Design method (IRAAD), see DCE (1986) and Rock-Evans (1987a, 1987b, 1987c). D2S2 is still practised by CACI consultants in ISP studies and information systems development projects. The method is currently primarily documented in courseware and overview articles.

An evaluation of D2S2’s way of thinking shows that the method is not primarily concerned with the development of specific systems in individual areas. It regards information as one of the cornerstones of organizations, besides money, material, and persons.
Because organizations are becoming increasingly dependent on computerized information, computers and telecommunication are bound to change the mode of operation of organizations. The information resources of organizations may consist of various systems sharing computerized information, leading to integration and improved communication.

The view of D2S2 on information systems is illustrated in figure 5.1. D2S2 supports the computerization of information resources of entire organizations. As (computerized) information is an organization asset, all organizational activities should ultimately be supported.

![Diagram of D2S2 way of thinking](image)

*Figure 5.1 Essentials of the D2S2 way of thinking*

The D2S2 way of modelling is rather complex for a full coverage in an introduction, especially because we are only interested in a fraction of the method. The interested reader is referred to Rock-Evans (1987a, 1987b) for a detailed description of the techniques used in IRAAD, which are almost identical to the ones required by D2S2.

We will just indicate its way of modelling, applying the illustration technique as used in Seligmann *et al.* (1989). The result is presented in figure 5.2. The general structure in the D2S2 way of modelling is based on two principles:
• separation between technology dependent and technology independent models

• tripartition into function, entity and integration models, integrating functions and entities.

In particular the modelling techniques for the technology independent models have been described in detail by D2S2. Five techniques are presented: (1) entity modelling, (2) function modelling, (3) entity life cycle analysis, (4) function dependency analysis, and (5) function logic analysis. The techniques for the technology dependent models have only been explained roughly, and have been drawn from descriptions of the way of working.

In the global information model part of the strategic model we distinguish function models, entity models, dependency models, and matrices. Entity life cycles and function logic diagrams are part of the analysis model.

![Diagram](image)

Figure 5.2 An overview of the D2S2 way of modelling

MacDonald and Palmer (1982) present the way of working of D2S2 as a life cycle consisting of six stages: strategy, analysis, design, construction, transition and production.
A salient principle in the D2S2 way of working is a "divide and conquer" strategy. Following the strategy stage, an enterprise or organization is divided into application areas for which a development strategy is developed. Subsequent stages concern only a single application area.

The strategy stage is divided into three tasks: (1) preparation, (2) overview, and (3) applications planning. The analysis and design stages have been specified in greater detail: 36 and 40 tasks, respectively. The subsequent stages are only described in general terms: it is unclear whether activities are serial or parallel, and whether there are any decision points.

Because the modelling task to be analysed is related to the overview task part of the strategy stage, we quote the part describing this task:

"The objective of the overview task is to provide a broad understanding of the enterprise as a whole, and to identify the data and functional areas requiring attention. The potential applications in the development strategy are expressed in terms of the high level business functions which could benefit from computer support, and of the major entities about which data could usefully be held in the new system. Hence, the overview is best expressed in terms of a general entity model identifying the major entities apparent in the business and showing the essential relationships between them, together with a function hierarchy defining the major business functions decomposed to about three levels of detail. The entities involved with each of these functions are listed, in order to draw up an entity-function matrix which, although lacking detail, is adequate for providing a good indication of the potential for data sharing. To complete this view, a function dependency diagram is necessary as both entity models and function hierarchies tend to provide only a static impression of the environment. This diagram portrays functional dependencies by showing which functions must precede others in time, and by indicating where the entities created or modified by one function become input to another."

The way of controlling is almost ignored in D2S2, but attention is paid to the definition of documents to be delivered, and the construction of time scales of the strategy stage. No attention is paid to the construction of project teams, the definition of decision points, or management reviews.

Although the way of supporting is not explicitly discussed, at least some use might be made of tools such as IEW, IEF, SDW and Blues in the application of D2S2, see for a detailed description of these tools NGGO (1988) and Seligmann et al. (1990). Tools for the construction of function logic diagrams and entity life cycle models are unavailable. The above tools are mainly used as documentation aids because they are only superficially linked to the D2S2 ways of modelling and working.
5.3 CONCEPTUAL META-MODEL OF THE D2S2 EXPERIMENT

In this section we describe the conceptual meta-model elicited and conceptualized in the D2S2 experiment according to the approach described in chapter 4. The overall task structure is shown in figure 5.3.

![Diagram showing the overall task](image)

Figure 5.3 The overall task

The subtasks will be discussed in greater detail later on. The subtasks are listed in table 5.1 together with the subsections describing these tasks more specifically.

Table 5.1 allows easy access to the individual sections of the D2S2 experiment results. The last two columns contain the figures corresponding to the specific tasks: the first points to the task structure, the second to the task scope (i.e. the relevant part of the concept structure).
## Table 5.1 Overview of contents of section 5.3

<table>
<thead>
<tr>
<th>Task</th>
<th>Section</th>
<th>Task structure</th>
<th>Task scope</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>D2S2 construction of an information architecture</strong></td>
<td>5.3</td>
<td>5.3</td>
<td></td>
</tr>
<tr>
<td><strong>Orientation</strong></td>
<td>5.3.1</td>
<td>5.4</td>
<td></td>
</tr>
<tr>
<td>Study of the primary process</td>
<td>5.3.1.1</td>
<td>5.6</td>
<td>5.5</td>
</tr>
<tr>
<td>Objectives analysis</td>
<td>5.3.1.2</td>
<td>5.7</td>
<td>5.8</td>
</tr>
<tr>
<td>Environmental analysis</td>
<td>5.3.1.3</td>
<td>5.10</td>
<td>5.9</td>
</tr>
<tr>
<td>Examination of financial situation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Orientation on technical infrastructure</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Qualification of discussion partner</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Construct global models for demarcation</strong></td>
<td>5.3.2</td>
<td>5.11</td>
<td></td>
</tr>
<tr>
<td>Explain modelling techniques to user</td>
<td>5.3.2.1</td>
<td>5.13, 5.14</td>
<td>5.12</td>
</tr>
<tr>
<td>Construct global entity model</td>
<td>5.3.2.1</td>
<td>5.16, 5.17, 5.18</td>
<td>5.15</td>
</tr>
<tr>
<td>Construct function model</td>
<td>5.3.2.2</td>
<td>5.19</td>
<td>5.20</td>
</tr>
<tr>
<td>Construct function hierarchy</td>
<td>5.3.2.3</td>
<td>5.21</td>
<td></td>
</tr>
<tr>
<td><strong>Construct integrated global model</strong></td>
<td>5.3.3</td>
<td>5.22</td>
<td></td>
</tr>
<tr>
<td>Essentialize function hierarchy</td>
<td>5.3.3.1</td>
<td>5.23, 5.24</td>
<td>5.22</td>
</tr>
<tr>
<td>Relate processes to entities</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Refine entity model</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Refine function hierarchy</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Refine function model</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Construct process-entity matrix</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Label relationships in entity model</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Correct global model</strong></td>
<td>5.3.4</td>
<td>5.25</td>
<td></td>
</tr>
<tr>
<td>Adopt function hierarchy</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Adopt entity model</td>
<td></td>
<td></td>
<td>5.26</td>
</tr>
<tr>
<td>Adopt matrix</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Detail global model</strong></td>
<td>5.3.5</td>
<td>5.27</td>
<td></td>
</tr>
<tr>
<td>Detail process into subprocesses</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Construct partial entity model for subprocess</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Integrate partial model into overall entity model</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 5.3.1 THE ORIENTATION TASK

The first phase of the overall task involves an *orientation*, analysing various aspects of the organization. It is what we call an implicit modelling task: short notes, rather than diagrams, are produced. Organizations are primarily analysed for their *external links*, and no questions are asked about the internal organization.

The orientation task is subdivided as shown in figure 5.4: six aspects can be analysed, and moving from one aspect to another is possible because of the cyclic nature of the task structure. Decision rules for the decision "Which aspect should be analysed next?" suggest that the primary process should be studied first, followed by analyses of objectives and the environment.
The subtask "Orientation in the technical infrastructure" is only performed if the organization frequently exchanges information with its environment, which follows from the results of the subtask "Environmental analysis".

A cyclic structure is used in figure 5.4 because we observed experimentally that objectives identified during the objectives analysis led to the examination of various other aspects in the other subtasks. New objectives were sometimes also introduced by such examinations. The results of the environmental and financial analyzes are also dependent on each other. These dependencies have been modelled by information places, some of which we will discuss in the remainder
of the orientation task. We will not discuss all orientation subtasks fully, but concentrate on the first three of figure 5.4.

5.3.1.1 STUDY OF THE PRIMARY PROCESS

The concepts

When studying the primary process in the organization, the task scope as shown in figure 5.5 is applied. The task is obviously responsible for the construction of a hierarchy of typical products and services offered by the organization.

A typical a-priori rule for this task defines products and services as specializations of at most one other product or service, formally expressed as:

- \( \forall p_1, p_2 \left[ \text{product}(p_1) \land \text{product}(p_2) \land \text{is}_a(p_1, p_2) \rightarrow \neg \exists p_3 \left[ \text{product}(p_3) \land \text{is}_a(p_1, p_3) \right] \right] \)

Another a-priori rule requires that products are not their own specializations:

- \( \forall p \left[ \text{product}(p) \rightarrow \neg \text{is}_a(p, p) \right] \)

The tasks

The task structure in figure 5.6 is strongly influenced by what might be called a generalization strategy: attempt to minimize the number of different product groups.

The decision "Can the product be generalized into a certain product group?" leads to identifying more general product terms after product determination, while the decision "Change product scope" encourages (by using decision rules) the establishment of generalizations if several distinct products are available.
The strategy is based on avoiding excessive detail in the orientation task. A critical identification of the business’ major activities is required.

![Diagram](image1)

Figure 5.6 Study of the primary process

### 5.3.1.2 OBJECTIVES ANALYSIS

**The tasks**

Two types of reasoning are possible in the objectives analysis, see figure 5.7, of which one is clearly preferred: first identify objectives, potentially leading to the analysis of problems. The other type of reasoning accepts problem situations and then attempts to identify corresponding objectives.

![Diagram](image2)

Figure 5.7 Objectives analysis
The concepts

More attention is obviously paid to objectives than to problems, as also follows from the task scope corresponding to the objectives analysis in figure 5.8. Four subtypes of the concept "objective" have been identified, which do not constitute a total partition of the set of objectives: other types of objectives are also possible.

![Figure 5.8 The scope of the objectives analysis](image)

"Product change objectives" are stored in an ormation place in order to influence the task "Study of the primary process" to add products to those already specified. This is an example of the influence exerted by results of one task on the performance of others.

5.3.1.3 ENVIRONMENTAL ANALYSIS

The concepts

The task "Environmental analysis" identifies external parties related to the organization. Typical external parties are clients, suppliers, etc. (see figure 5.9).

We are especially interested in the formal relationships between external parties and the organization, such as contracts, orders, and invoices. External parties are expected to have a particular expectation of the organization.
The tasks

The rather straightforward task structure of the environmental analysis is represented in figure 5.10. We can directly access the task "Analyse relationships with organization" from the decision "Other external parties?" because external parties can be identified in both the "Environmental analysis" and "Examine financial situation" tasks.

5.3.2 CONSTRUCT GLOBAL MODELS FOR DEMARCATION

The orientation task is followed by the construction of global models primarily intended for demarcation. Three modelling techniques are applied: entity modelling, function dependency modelling, and function modelling (see section 5.2).
It should be noted that the naming of these techniques appeared to differ from the D2S2 documentation during the experiment. Function dependency models, as they are known formally, are called function models in this section, while function models are referred to as function hierarchies. Other synonyms for "function model" have also been used by the expert: process model and dependency model.

The task "Construct general models for demarcation" first explains the modelling techniques to be used (see figure 5.11). This subtask is typically non-modelling and will not be subdivided here. Three modelling tasks are carried out subsequently: (1) for the global entity model, (2) for the function model, and (3) for the function hierarchy.

It should be realized that the objective of this modelling task is the demarcation of the study area. Tasks are performed with the objective of identifying the most important entities, processes and functions, without aiming at completeness.

Figure 5.11 The construction of global models for demarcation

The linear task structure in figure 5.11 suggests that entities, for example, are only identified in the task "Construct global entity model".
However, the scopes of the various tasks overlap, and we will see that entities can also be identified in the other tasks. In such cases the identification of entities is for example a consequence of the function hierarchy construction. Iterations can generally be modelled in two ways in the meta-modelling technique: (1) by cyclic task structures as shown in figure 5.4, or (2) by overlapping task scopes. The second type of iteration is found in the task structure of "Construct global models for demarcation".

Figure 5.12 Scope of global entity modelling
5.3.2.1 CONSTRUCT GLOBAL ENTITY MODEL

The concepts

The task scope as represented in figure 5.12 (see above) specifies the modelling concepts of the global entity modelling task detailedly. The concept "entity model" enables us to construct various entity relationship diagrams. Entity models consist of a number of entity model objects, which are generalizations of "entities" and "relationships".

Entities can be documented by entity descriptions, and, if required, by examples. Relationships relate two entities. This association between "entity" and "relationship" is objectified so that it can be given a name, the "role label", enabling us to label relationships twice.

Two divisions of the concept "entity" are considered as relevant: (1) into concrete and conceptual objects, and (2) into primary process, financial and external objects. The first division is a total partition, the second a non-total one. Concrete objects are entities representing physical objects or actual, real events, and conceptual objects are entities concerning categorical notions, such as product group, and client type. The second division arises from an abstraction of orientation task results. External parties, organizational relationships, and financial flows can be abstracted into entities and relationships. Decision rules in the task structure will initiate the identification of such entities. External, primary process, and financial objects should be available at the completion of this task.

![Diagram](image)

Figure 5.13 Construct the global entity model
The tasks

The construction of a global entity model is based on two reasoning mechanisms: (1) the identification of relevant entities, and (2) what we will refer to as the detailing of the entity model (see figure 5.13).

In the task "Identify entity" (see figure 5.14), the decision "What kind of entity?" depends primarily on the results of the orientation task. For example, one of the decision rules states that "If there is an external party category with many individuals, which has as yet not been abstracted into an entity, an external object should be added".

![Diagram](image-url)  
*Figure 5.14 Identify entity*
Figure 5.14 also indicates that instance examples are only specified for complex and important entities. We analyse the relationships between added entities and existing ones directly.

Additional detailing of the entity model (recall figure 5.13) is required if certain relationships or entities are unclear. In such cases, relationships are decomposed into one or more entities and relationships, the life cycles of entities are analysed, entities are documented, and conditional relationships are researched for their counterparts and opposite relationships. This is done in order to gain more insight into specific parts of the existing entity model, the tasks therefore being referred to as "Detail entity model".

5.3.2.2 CONSTRUCT FUNCTION MODEL

The concepts

Function model construction is based on the analysis of external events and the way they are handled by the organization. The sequence of processes handling one type of external event is called a process flow. External events, called i-events (incoming events), originate from external parties in certain physical implementations, such as telephone calls or letters (see figure 5.15).

We can have several function models (or rather, sheets of function models) consisting of one or more process flows. Three types of process flows are distinguished: (1) primary, (2) financial, and (3) secondary, supporting the primary process flows. Process flows are triggered by i-events from external parties. Process flows consist of processes interrelated by flows (see the objectified association). Processes can produce outward-bound organization signals, the o-events (outgoing events).

Two subdivisions of the concept "process" are of interest: (1) into externally triggered, internally triggered, and time or data driven processes, and (2) into creating or transforming, retrieving, co-ordinating and reporting processes.

Co-ordinating processes only evaluate incoming information on specific characteristics, and then trigger other specific processes. Reporting processes only multiply and send information to external parties. The category of creating and transforming processes includes deleting processes, although these are considered to be of limited interest.

Flows between two necessarily different processes are either triggering or supporting flows. Triggering flows imply that the "start process" is directly
followed by the "end process", which then handles the information originating from the "start process". Supporting flows imply that the information originating from the start process is required by the end process, this process being triggered or started by other means.

Figure 5.15 Scope of the construction of the function model
A multitude of rules can be given for processes and flows. A few of these are:

- Process parts of non-secondary process flows are initiated by i-events, or constitute flow endpoints.
  \[\forall p \forall f \left[ \text{process}(p) \land \text{part of}(p,f) \land \neg \text{secondary process flow}(f) \implies \exists i \left[ \text{i-event}(i) \land \text{triggers}_{p}(i,p) \right] \lor \exists l \exists q \left[ \text{flow}(l) \land \text{flow-start-end}(l,q,p) \right] \right]\]

- All process parts of secondary process flows have outgoing support flows.
  \[\forall p \forall f \left[ \text{process}(p) \land \text{part of}(p,f) \land \text{secondary process flow}(f) \implies \exists l \exists q \left[ \text{support flow}(l) \land \text{flow-start-end}(l,p,q) \right] \right]\]

- Processes initiated by i-events are part of the process flows triggered by these i-events.
  \[\forall p \forall i \forall f \left[ \text{process}(p) \land \text{i-event}(i) \land \text{triggers}_{p}(i,p) \land \text{triggers}_{f}(i,f) \implies \text{part of}(p,f) \right]\]

Flows can be labelled and can be optional. They can be exclusive, especially after co-ordinating processes. Outgoing flows can correspond to entity states, particularly for creating and transforming processes.

Processes should be interpreted as entity creation or transformation agents, the creation and transformation being considered as triggers for other processes. This view is paramount in the construction of the information architecture: processes create or transform entities, the entity creation and transformation being events triggering processes.

The tasks

The task structure for function model construction consists of four major tasks (see figure 5.16).

In the task "Add process flow" new process flows are specified. The task "Refine process flow" extends existing process flows by the identification of new processes part of that process flow. Processes can be refined in the task "Refine process", defining details of specific processes.

The task "Integrate process flows" analyses the interrelationships between process flows: information resulting from one process flow can be required for the execution of other process flows. These supporting flows relate the various process flows.
We will now describe the task structures of "Add process flow" and "Refine process" in more detail. Figure 5.17 shows the task structure of the addition of a new process flow.

The identification of a new process flow can be based on the global entity model of the previous task. These entities related directly to external objects induce the identification of relevant external events. New process flows may require new sheets (function models) for their specification.

If a process flow contains a clear external event, which is usual for primary process flows and financial flows, this i-event is specified unless the i-event can be integrated with existing i-events. In such a case the process flows are integrated by a co-ordinating process handling all i-events with an identical physical implementation, like "process incoming mail".

Having specified an incoming event and its corresponding process, a subsequent sequence of processes linked by flows can be defined. The initiation of other processes by the specified process can be determined by (1) examining the entity model, or by (2) evaluating relevant user remarks.

When examining the entity model, it is important to establish whether the specified process creates an entity related to other entities of the same category. Such processes are likely to trigger other processes, creating the other entities (certainly if the other entities have not already been manipulated by certain processes).

Relevant user remarks that have to be taken into account are those concentrating on trigger flows because they determine the order of the subsequent actions.
Figure 5.17 Add a process flow
Figure 5.18 shows the structure of the task "Refine process". The following refinements are considered as relevant: the division of one process into several more detailed ones, the addition of outgoing letters or reports as o-events, the addition of i-events to time or data driven processes, and the renaming of processes for an improved indication of the process objectives.

An example of the addition of i-events to time driven processes concerns the event of "Payment arrives" and the process "Process payments". For example, payments are processed at the end of each week. The i-event does not directly initiate the process under such conditions, although it is directly related to the process. The process "Process payments" is a time driven process.

5.3.2.3 CONSTRUCT FUNCTION HIERARCHY

The last activity part of the demarcation task is the construction of a function hierarchy.

The tasks

Because function models are available, the way of working in setting up a function hierarchy mainly involves the insertion of processes into a function tree. The relevant functions are determined first, followed by the allocation of the specified processes to the individual functions. The corresponding task structure is represented in figure 5.19.
The allocation of processes to functions is not always self-evident, because (1) functions may be missing, and (2) the meaning of processes may be unclear. In the first case, new functions must be introduced. In the second, the entities involved in the processes must be determined, which can even result in the modification of the entity model. Such analysis generally solves the process allocation problems.

![Diagram of function hierarchy]

*Figure 5.19 Construct a function hierarchy*

The concepts

The identification of relevant functions is performed by referring to *prototypical organization functions*, as shown in the task scope in figure 5.20. Each function
considered as relevant to the organization is chosen from one of seven subtypes. The expert has based his specialization of functions into a number of prototypical functions on Nielen (1976). Most function types are unique in an organization; the P and E-functions are exceptions because they can be subdivided.

Figure 5.20 The scope of the function hierarchy construction

The Z-function (self-maintenance) is concerned with the strategic and long term financial aspects of an organization. Although each organization implicitly strives for self-maintenance, it is not necessarily part of the function hierarchy. The O-function deals with the development of new types of product and service. The
O-function is treated similarly to the Z-function. The P-function is concerned with the primary processes of an organization and the corresponding financial flows. Each organization has a P-function. The E-function concentrates on the environment of an organization, and the corresponding external parties. Almost each organization has at least one E-function. The L-function is involved in production planning and control, and therefore relevant to complex production functions. The I-function is concerned with information management, and the S/C-function with the social aspects of an organization, i.e. personnel management.

5.3.3 CONSTRUCT INTEGRATED GLOBAL MODEL

The next major task identified in the D2S2 experiment deals with the integration of the various models resulting from the previous task. The integration is performed in order to realize a consistent model of the organization. In the next task, "Correct global model", this is validated, and possibly corrected together with the user.

The integration is primarily based on the assumption that processes create or transform entities and vice versa, as is evident from the task structure shown in figure 5.21. The function hierarchy is essentialized first, which primarily means that the hierarchy is freed from the reporting and co-ordinating processes, so that only the processes really manipulating entities are retained.

The next task explicitly relates processes to entities. A new entity model is created, and the processes manipulating or using the entities are registered.

In the task "Refine entity model", relationship cardinalities are defined, and possible internal entity structures are identified and represented as subtypes or as recursive relationships. In a recursive relationship both roles are played by the same entity, for example: a part consists of parts, which is a common structure in product modelling.

Hierarchies are analysed for what we call their balance, in the task "Refine function hierarchy". Functions containing too many processes should be split up, but two functions can also be combined. The task "Refine the function model" is optional.

The construction of the process-entity matrix is a simple activity as it has already been performed implicitly in the task "Relate processes and entities". The relevant information is therefore available, but presented in a different way in this task.
The last task of the integrated global model construction labels all relationships of the entity model as a further refinement to the entity model.
"Relate processes and entities" is the most important subtask of "Construct an integrated general model", as its main objective is to integrate the process and data oriented models. We will therefore now discuss this task in greater detail.

Figure 5.22 The scope of relating processes to entities.
5.3.3.1 RELATE PROCESSES AND ENTITIES

The concepts

The modelling concepts of the task "Relate processes to entities" is shown in figure 5.22. On the one hand it is a refinement to the task scope of figure 5.12, on the other hand it explicitly demonstrates the relationship between processes and entities.

A comparison of figures 5.12 and 5.22 shows that the concept "relationship" has been particularly refined. Associative relationships are distinguished from structural and recursive ones.

One entity depends on the existence of another in associative relationships. The independent and dependent entities are called permanent and associative, respectively. Instance-of relationships are special cases of associative relationships.

Besides the new specialization hierarchy of relationships, the entity model variant used in this task allows the specification of entity subtypes, and of exclusion constraints between roles.

Three types of what we call "manipulation relationships" can be discerned between entities and processes: create (c), update (u), and delete (d).

The tasks

The main objective of the task "Relate processes and entities" (see figure 5.23) is to produce a new entity model based on the latest entity model and function hierarchy. Processes and entities are related by manipulation relationships during the entity model construction.

Two types of verification are finally applied. One concerns the relationship between conceptual and concrete entities: conceptual and concrete entities related to each other by instance-of relationships must not be created by processes in the same function (logical horizon).

The second type of verification concerns the highly associative entities, which are associatively related entities, where the permanent entity comes into existence shortly before the associative entity. Highly associative entities must be created by processes within the same function (logical horizon).
Concentrating on the task "Identify entities based on the latest function hierarchy", we identify a task structure in which entity manipulations are formulated for all processes in the functions. The task structure for the formulation of the entity manipulations of a specific process is shown in figure 5.24.

An attempt is made to identify a specific entity manipulated or used by the process first; this entity can be found by inspecting the flows defined in the function models. Support flows with associated entities (see figure 5.16) suggest a "use" manipulation relationship. On the other hand, outgoing trigger flows with associated entities point to a "create" manipulation relationship. These guidelines have been specified as decision rules for the decision "Does the process manipulate or use entities?".

In case the creation of the identified entity can still not be traced in the defined processes, the function hierarchy must be extended by such process. If the entity was not included in the new entity model, it is analysed for relationships with already placed entities.
Figure 5.24 Formulate entity manipulation by process
5.3.4 CORRECT GLOBAL MODEL

The consistent integrated model resulting from the task "Construct integrated global model" is validated and corrected in this task in close co-operation with the user. Three tasks can be distinguished in a cyclic structure: adapt function hierarchy, adapt entity model and adapt matrix (see figure 5.25). The first two are the major tasks, while the third only adapts the matrix according to adaptations in the function hierarchy and the entity model.

![Diagram of Correct Global Model](image)

Figure 5.25 Correct global model

The function hierarchy can be adapted in four ways: (1) by the removal of partly overlapping processes, (2) by the addition of processes, (3) by applying more detail to processes as fresh information from the user becomes available, and (4) by renaming processes to clarify their objectives. The adaptation of the entity model can be performed in various ways, as can be seen in figure 5.26.

Remarks can lead to various adaptations of the entity model, such as the simple addition of relationships, or the introduction of disjoint subtypes, followed by an analysis of existing relationships of the supertype.

5.3.5 DETAIL GLOBAL MODEL

The final task in the construction of an information architecture adds more detail to the obtained models. Processes appearing complex and differentiated are analysed in more detail, as can be seen in figure 5.27. Subprocesses must be identified for such processes, for which, if necessary, a partial entity model must be established subsequently integrated with the overall entity model.
Time constraints prohibited the completion of the task "Detail global model" during the elicitation session of the D2S2 experiment, and additional sessions would be required to analyse this task fully. However, this was considered to be beyond the scope of our study.
After detailing the global model, the information architecture is complete, and is usually followed by an applications planning (see section 5.2), but this was not part of our study.

5.4 SUMMARY

We have described the results of the D2S2 experiment in this chapter. We will now summarize the major characteristics of the resulting conceptual meta-model of this experiment.

The experiment has yielded detailed insight into the construction of an information architecture, using D2S2 techniques. The construction of an information architecture has only been described superficially in formal D2S2 publications (see section 5.2). Based on this experiment, we have identified five major tasks during the construction of an information architecture, each subdivided into various subtasks.
The approach applied by the D2S2 expert can be characterized by the following statements:

- **Outside-in approach**

  The externals of an organization are analysed before internal structures are taken into account. It is vital to be aware of the various external aspects of an organization, such as the products and their added values, the major external parties, and the various primary and financial flows with these parties. Various types of external parties and relationships have been distinguished (see figure 5.9). The external aspects are subsequently represented in a global entity model. By determining the way external events are handled within an organization (in a function model), the internal activities are analysed in greater detail. The function hierarchy is set up only after the construction the function (dependency) model.

- **Critical business view**

  Organization's objectives and products are critically analysed particularly in the orientation task. Organizations are expected to be aware of the justification of their operations, which are carefully judged. The objectives analysis is an important subtask of the orientation task.

- **Focus on the business' future**

  Models represent aspects of future conditions of an organization. Although the current situation is generally implicitly chosen as starting point, models should finally include organization strategies.

- **Fundamental entity modelling technique**

  The entity model is paramount in all tasks, except the "Orientation". A global entity model is set up in the demarcation task. An overall entity model serves as basis for the integration task. In the validation task, corrections in the various models, are directly translated towards required changes in the overall entity model. Although the starting point for the detailing task is the function hierarchy, additional details are directly included in the overall entity model.

- **Detailed entity modelling technique**

  Both the concepts "entity" and "relationship" have been specialized in a number of subtypes (see figures 5.12 and 5.22). Entities are classified as (1)
conceptual or concrete, and as (2) primary process, external, financial or other objects. Relationships are classified as associative, structural or recursive. Subtype relationships can be defined between entities, and relationships can be exclusive.

- Event-driven function model construction

Important concepts in the function (dependency) model are "event" and "trigger". Function models are set up by analysing the processing of incoming events within organizations. External events are generally considered as starting points for model extensions.

- Detailed function modelling technique

The function modelling technique has been refined in several ways (see figure 5.15). Process flows are used to specify the order of processes initiated by incoming events, chained by triggers, and finally leading to outgoing events. Three types of process flow have been identified: primary process, financial, and secondary process flows. Furthermore, various types of processes and flows with specific characteristics have been distinguished.

- Use of a detailed set of prototypical business functions

The construction of a function hierarchy is strongly dependent on a number of prototypical business functions applicable to organizations in general. Seven of these functions have been identified: the Z-, O-, P-, E-, L-, I-, and S or C-functions (for more details see figure 5.20).

- Limited use of C/U matrix

The C/U matrix mapping processes to entities has mainly been used to represent available information in a different way (see figure 5.21). During the task "Construct integrated global model", processes and entities have already been related to each other in the subtask "Relate processes and entities", before the matrix was constructed.

- Strong relationships between the various models

The results of the various tasks are highly intertwined. For example, the entities in the global entity model are based on the products, external parties and external relationships identified in the orientation task. Flows in function models generally correspond to (states of) given entities, while processes are expected to create or transform specific entities. The creation
and transformation processes generally corresponds to associative relationships in the entity model. Processes in a function model become part of the function hierarchy. More complex interrelationships have been described in detail in section 5.3.

- Education and self-activity

Our final conclusion concerns the crucial importance of the results being considered as the users' mental property. This can only be achieved if users are trained in the modelling techniques in use, and if they are encouraged to participate actively, and deploy self-activity.
6

EXPERIMENT 2: YOURDON

6.1 INTRODUCTION

In this chapter we analyse the construction of an information architecture by an experienced information engineer, using the modelling techniques as prescribed by the YOURDON method. The expert information engineer in the YOURDON experiment is a senior consultant of CMG Den Haag BV.

We discuss the YOURDON method in section 6.2, or rather the YOURDON-like methods described in the literature, while the acquisition task results with the above mentioned expert are evaluated in section 6.3. The results emphasize the overall strategy and indicate particularly interesting details. The last section (6.4) deals with the major conclusions of this chapter.

6.2 YOURDON

A unique and comprehensive description of "the YOURDON method" does not really exist, because many authors have described their own variants of the "YOURDON school". Some of the well-known books on Structured Analysis (SA) and Structured Design (SD) are Gane and Sarson (1978), DeMarco (1979), McMenamin and Palmer (1984), Ward and Mellor (1985a, 1985b, 1986), and Yourdon (1989).

The existing variety is mainly due to the ongoing developments and the experience gained. For example, the early YOURDON school advocated an analysis of the current situation, and a top-down approach. In the latest book by Yourdon, Yourdon (1989), the current situation is only modelled in certain specific situations, and event partitioning, as introduced by McMenamin and Palmer (1984), is applied rather than a top-down approach. Furthermore, we see special variants for real time systems, as in Ward and Mellor (1985a, 1985b, 1986) and Hatley and Pirbhai (1987). The terms "YOURDON" and "YOURDON method" will be used throughout this chapter to indicate the above class of methods. Our description is primarily based on more recent versions, as found in Ward and Mellor (1985a, 1985b, 1986) and Yourdon (1989).

The YOURDON method, developed and marketed by Yourdon Inc. (USA), is widely applied in Europe and the United States. It concentrates on the development of both data processing and real-time systems, depending on the selected variant.
YOURDON primarily offers an elaborate set of modelling techniques and strategies for the analysis and design of the above mentioned systems, including rough guidelines for their implementation and maintenance.

The YOURDON strategy is based on the recognition that systems must function in specific environments, and that they must be considered as *stimulus–response mechanisms*, accomplishing specific purposes by responding to events in their environment. The system proper consists of processors, e.g. persons and computers, and communication links between these. The system communicates with its environment through communication links with the processors.

The essence of the YOURDON way of thinking is illustrated in figure 6.1.

![Diagram of environment and system](image)

**Figure 6.1 Essentials of the YOURDON way of thinking**

The YOURDON way of modelling is based on the separation of a system's essence and its technology. Given that systems must function in a specific environment, and must accomplish certain purposes, the definition of their activities and the nature of the associated data should be possible regardless of the technology applied in system implementation. This definition is called the *essential model*. Systems can also be described in terms of the applied technology -resulting in an *implementation model*, which can be considered as a mapping of an essential model on the technological environment. Figure 6.2 reflects YOURDON's way of modelling.
An essential model is divided into an *environmental* and *behaviour* model. The interfaces between a system and its environment is described in the environmental model, while the behaviour model defines the system’s internals, represented by dataflow diagrams, entity-relationship diagrams, a data dictionary, process specifications and, if required, state transition diagrams.

Three sections can be distinguished in the implementation model. The processes in the essential model are mapped onto human and computer processors, and the required human-computer interfaces are analysed in the user implementation model.

The *systems implementation model* concentrates on the computer processors for which their processes are grouped into a network of tasks. A task is defined by a module hierarchy (structure chart) in the *program implementation model*, removing concurrence and continuous processing, and tasks can therefore be implemented in sequential code.

The way of working defined by YOURDON in Yourdon (1989) consists of nine partly sequential and parallel activities represented in a task structure in figure 6.3.

The task ”Analysis” has been described in particular detail by Yourdon (1989), specifying the tasks ”Construct environmental model”, ”Build a preliminary behaviour model”, ”Complete the behaviour model”, and ”Construct the user
implementation model'. A more detailed specification of these tasks is not provided here, as the particulars can be found in Verhoef (1991), discussing a detailed meta-model of Yourdon (1989).

Figure 6.3 The overall way of working of YOURDON
The strategy applied by Yourdon (1989) in the analysis task is based on event partitioning as defined in McMenamin and Palmer (1984). In essence, event partitioning implies that dataflow diagrams are set up for external events (stimuli), integrated, and subsequently reallocated into a new hierarchy of dataflow diagrams.

As this strategy is only concerned with dataflow diagram creation, the role of entity-relationship diagrams and the data dictionary are not quite clear. They are generally constructed in parallel in one way or another. They are not really integrated with the clear strategy for dataflow diagrams.

A way of controlling is not available in YOURDON - it is even explicitly excluded. Yourdon (1989) mentions known management techniques such as PERT charts, Gantt charts and cost-benefit calculations, but these are not integrated with the other chapters in Yourdon (1989).

Many CASE tools support the YOURDON modelling techniques, owing to the popularity of the method. A few of the well-known tools are Blues, Excelerator, Promod, SDW, Teamwork, and the Yourdon A/D Toolkit. Detailed descriptions of these can be found in NGGO (1988), and Seligmann et al. (1990). The tools differ somewhat in their support of the diagram conventions and verification rules, obviously also due to the existing YOURDON variants. The way of working according to Yourdon (1989), applying event partitioning, is not supported by any of these tools, because they all assume the top-down approach, as presented in earlier Yourdon variants (see Gane and Sarson (1978) and DeMarco (1979)).

6.3 CONCEPTUAL META-MODEL OF THE YOURDON EXPERIMENT

The overall strategy for setting up an information architecture as elicited and conceptualized during the YOURDON experiment is shown in figure 6.4. Some details of the experiment can also be found in Ledderhof (1989). The first activity in the overall task is a brief orientation of the organization to collect the information allowing the decision whether to follow the data or process oriented approach. The process oriented approach is suitable for organizations receiving explicit stimuli from their environment. The data (or information systems) approach is chosen in other cases. The process oriented approach was applied by the expert in the YOURDON experiment; the task is described in more detail in this section. An additional elicitation session concentrating on the data oriented approach would have been required to apply sufficient detail to that task, but this was considered to be beyond the scope of this study.
The subtasks identified in the YOURDON experiment are listed with their corresponding sections and figures in table 6.1 serving as an index to section 6.3.

<table>
<thead>
<tr>
<th>Task</th>
<th>Section</th>
<th>Task structure</th>
<th>Task scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>YOURDON construction of an information architecture</td>
<td>6.3</td>
<td>6.4</td>
<td></td>
</tr>
<tr>
<td>Orientation</td>
<td>6.3.1</td>
<td>6.5</td>
<td>6.6</td>
</tr>
<tr>
<td>Define organization and its primary process</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Determine external parties</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Explore external relationships</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specify organizational structure</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Apply data or information systems oriented approach</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Apply process oriented approach</td>
<td>6.3.2</td>
<td>6.7</td>
<td></td>
</tr>
<tr>
<td>Orientation on relevant subprocesses</td>
<td>6.3.3</td>
<td>6.8</td>
<td>6.10</td>
</tr>
<tr>
<td>Identify subprocess</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Globally analyse subprocess</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Construct preliminary physical model</td>
<td>6.3.4</td>
<td>6.11</td>
<td></td>
</tr>
<tr>
<td>Construct context diagram</td>
<td>6.3.4.1</td>
<td>6.12, 6.13, 6.14</td>
<td>6.15</td>
</tr>
<tr>
<td>Develop preliminary dataflow diagrams</td>
<td>6.3.4.2</td>
<td>6.17, 6.18</td>
<td>6.16</td>
</tr>
<tr>
<td>Develop preliminary data models</td>
<td>6.3.4.3</td>
<td>6.20, 6.21</td>
<td>6.19</td>
</tr>
<tr>
<td>Construct integrated detailed model</td>
<td>6.3.5</td>
<td>6.22</td>
<td></td>
</tr>
<tr>
<td>Construct integrated process model</td>
<td>6.3.5.1</td>
<td>6.23</td>
<td></td>
</tr>
<tr>
<td>Construct integrated data model</td>
<td>6.3.5.2</td>
<td>6.24</td>
<td></td>
</tr>
<tr>
<td>Integrate process model and data model</td>
<td>6.3.5.3</td>
<td>6.25</td>
<td></td>
</tr>
<tr>
<td>Essentialize and construct logic model</td>
<td>6.3.6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Regroup logical processes</td>
<td>6.27</td>
<td>6.28</td>
<td></td>
</tr>
<tr>
<td>Construct logical dataflow diagrams</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Adept data model</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 6.1 Overview of contents of section 6.3
6.3.1 ORIENTATION

The tasks

The structure of the orientation task is shown in figure 6.5. The organization's primary process is determined first, followed by an analysis of the environment for relevant external parties, particularly those directly related to the primary process. The relationships between external parties and the organization are further explored in the subtask "Explore external relationships". The orientation task ends by specifying the organization structure.

Figure 6.5 The orientation

The concepts

The modelling concepts applied during orientation are presented in figure 6.6. Two subtypes of external party are particularly important: (1) the requesting parties, and (2) the offering parties. Other subtypes are not explicitly required at this stage but could obviously emerge in the subtask "Determine external parties". The relationships between external parties and the primary process, analysed in the task "Explore external relationships", are objectified into the concept "external relationship". The manner in which external relationships are generated, determines the type of contact, of which three are of special interest: (1) telephone, (2) letter, form or telefax, and (3) electronic network.
Two types of organizational structure can be distinguished in the subtask "Specify organizational structure": (1) product oriented and (2) function oriented organizations. An organizational structure consists of departments exerting
control over other departments. An organizational structure needs not be specified exhaustively in the orientation task, only the main departments must be defined.

6.3.2 THE PROCESS ORIENTED APPROACH

The task structure for the process oriented approach is shown in figure 6.7. The four subtasks are performed in a linear sequence. Iterations between the various modelling techniques are implemented in these tasks by overlapping task scopes. Dataflow and entity relationship diagrams are generated in all tasks except "Orientation on relevant subprocesses". The four tasks will be described in detail in the following sections.

![Figure 6.7 The process oriented approach](image)

6.3.3 ORIENTATION ON RELEVANT SUBPROCESSES

The tasks

"Orientation on relevant subprocesses" is an implicit modelling task because no explicit diagrams are produced. It can be regarded as information gathering for the remaining three tasks in the process oriented approach. The relevant task structure is shown in figure 6.8.
An important subtask is the identification of a subprocess. A subprocess is a clustering of several processes in an organization. Note that the expert regards subprocesses to consist of processes!

![Diagram showing subprocess identification and analysis](image)

Figure 6.8 The orientation on relevant subprocesses

Two strategies can be applied to identify a subprocess: (1) an organizational department is considered as subprocess, and (2) the handling of an external relationship is considered as a subprocess.

A rough analysis of a subprocess in the task "Globally analyse subprocess" leads to four major aspects: (1) the process part of the subprocess, (2) the flows to and from the environment, (3) the essential internal information, and (4) the role of existing automated systems supporting the subprocess. The analysis of these aspects is performed in parallel, as shown in figure 6.9. The major flows and processes are considered only - completeness is not required at this stage.

![Diagram showing subprocess analysis](image)

Figure 6.9 The global analysis of a subprocess

The concepts

The corresponding task scope (relevant part of the concept structure) of the task "Orientation on the relevant subprocesses" can be found in figure 6.10. A subprocess either specifies the activities in a department, or the handling of an
external relationship. Two types of subprocess are vital to organizations: (1) the primary, and (2) the financial subprocesses. Subprocesses consist of several processes and can be supported by (automated) information systems. Many flows are handled in a subprocess.

![Diagram of flow processes](image)

Figure 6.10 The scope of the global analysis of subprocesses

Two subdivisions of the concept "flow" have been considered: (1) the one between internal and external flows, and (2) the one between primary process, financial, and support or information flows. External flows go out to (responses), or come in from (stimuli) the environment, while internal flows represent information created and used in the organization.

Flows entering the organization either initiate primary or financial subprocesses, or they are required by these subprocesses. This difference is the reason for the
distinction between primary process, financial and supporting or information flows.

6.3.4 CONSTRUCT PRELIMINARY PHYSICAL MODEL

An initial set of diagrams is prepared as part of "Construct preliminary physical model", listing the processing of stimuli entering the organization, and the data required to support the processing. "Physical" implies that the models should be closely related to the organization's current "physical" situation. The models must be understandable especially for the users, and they should accurately define the processing as actually performed by the users. The processors (persons and machines) belonging to processes part of the organization are therefore also identified during this task. The preliminary physical model is constructed as shown in figure 6.11.

![Diagram](image)

Figure 6.11 The construction of the preliminary physical model

A context diagram is first set up for demarcation reasons. More detail is introduced by defining physical dataflow diagrams in the task "Develop preliminary dataflow diagrams". In data bound organizations, an initial inventory of the required data is conducted as well, resulting in a number of preliminary data models. These tasks will be discussed in more detail in the following subsections.
6.3.4.1 CONSTRUCT CONTEXT DIAGRAM

The tasks

A context diagram is developed as summarized in figure 6.12. Two reasoning strategies are distinguished: a stimulus entering, or response leaving the organization is identified, followed by an analysis of its consequences.

Figure 6.12 The construction of the context diagram

Stimuli and responses can be recognized from the results of the "Orientation on relevant subprocesses" task, or because additional user information has become available.

Figure 6.13 The identification of a stimulus
Not all stimuli (or responses) are explicitly shown in a context diagram: stimuli can be abstracted into generalized ones in order to avoid overloading the context diagram (see figure 6.13).

The task structure for response recognition is built up similarly to that for stimulus recognition, which is shown above. This is also true for the tasks analysing the consequences of a new stimulus and response. Figure 6.14 therefore only shows the analysis of the consequences of a new stimulus.

Figure 6.14 Handling the consequences of a new stimulus

Figure 6.14 underlines the importance of finding (1) responses having led to identified stimulus, such as invoice payments, and (2) responses resulting from stimulus, such as deliveries following orders. New responses are stored in an information place, influencing the decision "New stimulus or response?" in figure 6.12. A decision rule suggests an analysis of the consequences of recently identified responses.

The concepts

The task scope belonging to the construction of the context diagram is shown in figure 6.15. This part of the concept structure emphasizes the importance of a number of homogeneous associations between "flows" in a context diagram; they indicate that flows can (1) lead to other flows, (2) be required for flow processing, and (3) be abstracted in order to keep the context diagram well organized.

6.3.4.2 DEVELOP PRELIMINARY DATAFLOW DIAGRAMS

The next essential activity in preliminary physical model creation is the development of the preliminary dataflow diagrams.
The concepts

We will first introduce the modelling concepts used during the construction of physical dataflow diagrams. The relevant modelling concepts are represented in figure 6.16.

It should be interpreted as follows: physical dataflow diagrams consist of several process model objects, which can be processes, datastores, or automated systems. Processes are performed by processors, which can either be employees or departments, depending on the level of detail. Flows are defined between process model objects. Flows can be interrelated in three ways, as discussed in figure 6.15.

The concept "flow" can be divided in three ways. The division between internal and external flows has been introduced before, as well as the one between primary process, financial, and supporting or information flows. The division between normal, and time or data driven flows must now be added.
These flows originate from datastores and automated systems, and their existence is determined by specific conditions. Normal flows originate from processes, automated systems, or the environment.

Figure 6.16 The scope for the construction of physical dataflow diagrams
Various verification rules apply to physical dataflow diagrams, but a complete list would become too detailed. Two examples are given as an illustration:

- Internal flows link two process model objects, and must not start or terminate at the same process model object.
  \[ \forall f \, [\text{internal\_flow}(f) \Rightarrow \exists p_1 \exists p_2 \, [\text{start\_of}(p_1,f) \land \text{end\_of}(p_2,f) \land p_1 \neq p_2]] \]

- External flows run either to or from process model objects.
  \[ \forall f \, [\text{external\_flow}(f) \Rightarrow \exists p \, [\text{start\_of}(p,f)] \oplus \exists p \, [\text{end\_of}(p,f)]] \]

A final, interesting detail of figure 6.16 concerns the subtype "dialogue" of the objectified homogeneous association "flow dependency". A dialogue is a stimulus-response combination without time delay.

The tasks

The task structure for the construction of preliminary dataflow diagrams is shown in figure 6.17. New diagrams are developed, or existing ones are extended because fresh information from users has been obtained. Having completed the construction or revision of a dataflow diagram, the context diagram can be rendered consistent with these. Changes to the context diagram result from the introduction and removal of stimuli and responses.

![Diagram](image-url)  

Figure 6.17 The development of the preliminary dataflow diagrams
The tasks "Extend existing dataflow diagram" and "Construct new dataflow diagram" are essentially equal. The latter differs in its absence of new instances of the concept "physical dataflow diagram". We will therefore only discuss the construction of new physical dataflow diagrams in detail here.

Figure 6.18 The construction of a physical dataflow diagram
Sec. 6.3 CONCEPTUAL META-MODEL OF THE YOURDON EXPERIMENT

The identification of a major stimulus to be processed by the organization is the first step in the construction of a new dataflow diagram. The dataflow diagram handling this external flow is subsequently constructed. The corresponding task structure is shown in figure 6.18.

The construction of a physical dataflow diagram is an iterative process, coordinated by the decision "How to adapt the dataflow diagram?". The diagram is generally extended by the identification of new relevant incoming flows (stimuli), see the task "Identify new incoming flow".

We subsequently analyse the processing of this flow in the task "Analyse handling of flow", which is also performed if (1) incoming flows resulting from responses within the dataflow diagram are identified, or (2) new flows resulting from the processing of other flows are identified.

The task "Analyse handling of flow" demonstrates that identified flows (1) are handled as simple registrations in automated systems, (2) are handled in new processes, (3) constitute responses of organizations to the environment, and (4) are responses with a direct stimulus, i.e. dialogues.

We call the above reasoning a trigger analysis. In addition, we must identify supporting flows assisting flow processing, and time or data driven flows starting processes because of changes in time or data. Such time or data driven processes are subsequently analysed in more detail in the task "Analyse handling in time or data driven process".

The last subtask of figure 6.18 not discussed yet, is concerned with the modification and deletion of existing model components. We can delete flows, processes, and datastores in this task, and we can improve the naming of flows and other components.

6.3.4.3 DEVELOP PRELIMINARY DATA MODELS

Having discussed the construction of preliminary dataflow diagrams, we arrive at the last, optional, subtask of "Construct preliminary physical model", the task "Develop preliminary data models".

The concepts

Figure 6.19 shows the scope of the preliminary data model construction. Entity relationship diagrams consist of entities related by relationships with role labels and cardinalities. Entities either represent external parties or flows.
Normalization and a detailed analysis of dependencies between entities are not considered as important at this stage of data modelling. In fact, it is preferable to show the notions and concepts used by the users explicitly. This implies that various stages of possibly one entity are modelled as different entities here. For example, claims, unpaid claims, and paid claims can exist as three entities in a data model.

![Diagram](image)

*Figure 6.19 The scope of the construction of the preliminary data models*

**The tasks**

The task structure for building preliminary data models is shown in figure 6.20. It is obvious from this structure that we can construct various data models each handling specific aspects of the organization.

Data models are set up according to the previously constructed dataflow diagrams, which also follows from the title "Construct data model based on dataflow diagrams", and can subsequently be integrated with other related data models.
The cyclic task "Construct data model based on dataflow diagrams" selects and analyses dataflow diagrams for relevant entities. A dataflow diagram is analysed for data as shown in figure 6.21.
Two tasks may consequently lead to the identification of entities by examining the selected dataflow diagram: "Identify entities belonging to a flow" and "Abstract external party". Flows to data stores and systems, and responses from systems result particularly in entities, as do external parties belonging to stimuli and responses.

Relationships are handled in a separate task "Identify relationship with existing part of model". If entities belong to interrelated flows, they are likely to be interrelated themselves. The correction task "Correct model" in figure 6.21 allows the removal or renaming of existing relationships and entities.

6.3.5 CONSTRUCT INTEGRATED DETAILED MODEL

Having completed the preliminary models, the process oriented approach now integrates and refines these, as we recall from figure 6.7. Integration refers to the construction of one large model containing the partial models developed previously, while refinement concerns the improvement in names, and the removal of physical model elements. Dataflow diagramming is the first activity here (see figure 6.22), as we also experienced in the preliminary model construction.

```
Construct Integrated process model

Construct Integrated data model

Integrate process model and data model
```

Figure 6.22 The construction of the integrated detailed model

The integration of dataflow diagrams is followed by data model integration. As a last step, both models are interrelated by judging the feasibility of the processes on availability of data, which can lead to entity and process model modification.
6.3.5.1 CONSTRUCT INTEGRATED PROCESS MODEL

The construction of the integrated process model consists of creating a process list and, optionally, a complete dataflow diagram. This optional task was not performed during our experiment, and we will therefore only discuss the construction of the process list.

A process list sums up the "logical processes" performed in the organization. The addition "logical" implies that processes only perform one function, or only handle one flow. In physical dataflow diagrams, processes can include various functions because it is more important to relate processes to individual processors than to functions. Multi-functional processes and systems are now opened up and replaced by logical ones.

The task structure of creating a process list is detailed in figure 6.23.

![Diagram](image)

Figure 6.23 The construction of the process list

Processes and automated systems are transformed into one or more logical processes. Each outgoing flow and incoming stimulus of a physical process must be handled by a different logical process.
6.3.5.2 CONSTRUCT INTEGRATED DATA MODEL

The integration of data models results in one complete entity relationship diagram, integrating all partial ones, one diagram at a time. Models may be integrated in various ways. Entities and relationships in existing diagrams can be copied into the overall model and renamed, if required. Entities of partial models can be abstracted into entities part of the overall model. Entities and relationships may be added following recently acquired information from users.

6.3.5.3 INTEGRATE PROCESS MODEL AND DATA MODEL

The last task part of the integrated model construction is "Integrate process model and data model". The corresponding scope is shown in figure 6.24.

![Diagram](image)

Figure 6.24 The scope for the integration of the process model and the data model

The task links processes to entities, and it is therefore important to identify how the logical processes manipulate the entities (part of the overall entity relationship diagram). Two types of entity manipulation are relevant: (1) the use, and (2) the creation of entities.

The task structure for interrelating processes and entities is shown in figure 6.25. The entities used and created by all logical processes are determined. Failing this, specific processes must be removed, or the entity model adapted. All logical processes must have been analysed in this way before the task can be terminated.
6.3.6 ESSENTIALIZE AND CONSTRUCT LOGICAL MODEL

The last task in creating an information architecture consists of constructing a new set of levelled "logical" dataflow diagrams by grouping the processes in the integrated process model into groups of about $7 \pm 2$.

The logical processes are first regrouped, followed by a bottom-up drawing of new dataflow diagrams for these groups. The entity relationship diagram is finally adapted according to the new logical dataflow diagrams.

The expert could only perform the first part of this task, i.e. the regrouping, in the experiment because of time constraints.

Additional elicitation sessions would have been required to analyse the other parts of the logic model construction in greater detail, but that was considered to be outside the scope of this study. We will therefore now concentrate on the regrouping of logical processes.

The concepts

The task scope applied to the regrouping of the logical processes is shown in figure 6.26.
Figure 6.26 The scope of regrouping the logical processes

Figure 6.26 indicates that logical processes can consist of other logical processes. It should be realized, however, that logical processes consisting of other processes are no longer limited to single functions, but that they have become *multi-functional*.

Their grouping however is not based on the single processor requirement, but on functional considerations such as (1) based on one stimulus, (2) closely related in time, (3) concerned with identical external parties, and (4) dealing with the same type of activity, such as handling of updates, or sending of reminders, based on time triggers.

*The tasks*

The task structure corresponding to the regrouping task, shown in figure 6.27, starts by selecting a logical process which can either be included in an existing group, or for which a new group can be set up. In the latter situation we attempt to identify other processes belonging to the same group. Completed groups are named. After the creation of a group, another process is selected for grouping. Processes can be moved between groups if some of these have become too large.

The ultimate logic model corresponds to the information architecture. According to this model, the information planning can be continued by assigning certain groups of processes to systems. These newly identified systems can then be scheduled in an information systems plan.

For completeness' sake, we repeat that the exact role of the information architecture in the sequel of an information planning study has not been examined in detail, as it is not part of our experimental setting (see section 4.4).
6.4 SUMMARY

We have described the results of the YOURDON experiment in this chapter. This section contains a summary of its main characteristics.

The experiment resulted in detailed insight into the application of YOURDON techniques to the construction of an information architecture at organizational level. The YOURDON method proper (see section 6.2) is not explicitly concerned with this organizational analysis in setting up an information architecture.
This experiment shows that the YOURDON techniques and strategies can be used within information planning projects for the creation of an information architecture. Of course, this had already been demonstrated by our expert in the numerous information planning projects he had run with the YOURDON principles. We have made it explicit.

The approach applied by the YOURDON expert can best be characterized by the following statements:

- The starting point is the current situation

The observed way of working basically corresponds to the general strategy as presented in DeMarco (1979) in which the starting point is a physical model of the current situation, which is then essentialized into the essential model. The essential model, which is referred to by the expert as the logic model, is subsequently detailed into the physical model for the future situation.

- Preference for a process oriented approach

The process oriented approach is preferred above the data oriented approach. Process descriptions are considered as more recognizable for the users and less abstract (see also the next point).

- Recognizability for the users

Model recognizability for the users is a very important underlying principle. Initial models should not be too abstract and should show the activities as actually performed by the users. These models can be subsequently abstracted. Intensive user participation is essential in these processes.

- Additional concepts based on organizational theory

Some of the modelling concepts applied in the YOURDON experiment originate from organizational theory, for example: requesting and offering parties, primary and financial processes, and product oriented and function oriented organization structures. These concepts were a consequence of the application of the YOURDON techniques at an organization-wide level.

- Event-driven construction of dataflow diagrams

Dataflow diagrams are not constructed top-down, but event partitioning of McMenamin and Palmer (1984) is applied (see section 6.2). Physical dataflow
diagrams are created for stimuli entering the organization first, after which they are integrated, essentialized and relevelled.

- Detailed dataflow diagramming technique

The concepts used in dataflow diagramming have been refined. We emphasize (1) the ability to model automated systems in physical dataflow diagrams, (2) the distinction between normal and time or data driven flows, and (3) the various explicit dependencies between dataflows.

- Strong integration of dataflow diagrams and entity relationship diagrams

The use of entity relationship diagrams has been clearly integrated with the use of dataflow diagrams, contrary to our observation about the YOURDON literature in section 6.2. We can even choose between a data and a process oriented approach.

The creation of entity relationship diagrams in the process oriented approach strongly depends on existing dataflow diagrams, which is evident from the relationships existing between external parties and flows in dataflow diagrams, and entities in entity relationship diagrams.

Normalization of entity relationship diagrams is irrelevant during the construction of an information architecture; their close correspondence to dataflow diagrams is more valuable. If different processes establishing certain entity state transitions have been identified, these different states are modelled as different entities.

- Various shortcuts embedded in modelling strategy

Tasks are optional in various stages of the way of working, such as the construction of preliminary data models and the integration of the physical dataflow diagrams. The modelling process can consequently be speeded up at the risk of loosing contact with the users.
EXPERIMENT 3: ESM

7.1 INTRODUCTION

The third experiment in this study is related to the construction of an information architecture according to the Evolutionary Systems Planning (ESP) method part of the Evolutionary Systems Methodology (ESM), which has been marketed by Unisys since 1985 (see Unisys (1985)). The expert participating in the ESM experiment is a Unisys business consultant. In the next section, we will present an overview of ESM and highlight the part analysed in this experiment. Section 7.3 discusses the results obtained with the expert during acquisition. We point to some interesting features and explain the expert's ways of working and modelling. Section 7.4 lists the main conclusions of the ESM experiment.

7.2 ESM

ESM is a method directed at what is called business engineering (see Unisys (1985) and Unisys (1989)). ESM's way of thinking primarily focuses on the continuously changing organizations, requiring rapid adaptation of information technology applications, especially now that organizations depend heavily on information technology. Information systems should reflect the activities within organizations perfectly. Information systems are inseparable from their organizations, and the implementation of information systems can therefore not be isolated from businesses' optimization and improvement. The automated systems are considered as modular systems supporting specific organizational functions, preferably implemented in a 4GL environment.

We visualize ESM's way of thinking in a model "reduced to its bare essentials" (recall chapter 2) in figure 7.1. It emphasizes organizations in complex and changing environments with several internal functional areas supported by a modular information system. The importance of the persons working in, and responsible for these functional areas are also highlighted in figure 7.1.

ESM implements its way of thinking in two areas of concern: (1) ESP, Evolutionary Systems Planning, applying business and information strategy planning methods and techniques, and (2) ESI, Evolutionary Systems Improvement, dealing with the development and implementation of what is called "improvement projects" in a 4GL environment, promoting information systems development with a structured approach to prototyping.
ESP consists of five stages: preparation, internal studies, external studies, design and planning. These stages confirm the close relationship between the way of working and way of controlling as discussed in chapter 2. Preparation and planning both include management tasks such as project setup, resource allocation, and project evaluation. The internal and external studies are analysed in this ESM experiment, as well as the part of the design stage involved with the information architecture. We will leave out the construction of the organizational and technical architectures also part of the ESP design stage.

ESI takes a structured approach to prototyping, but, as this chapter is not concerned with this ESM feature, we will only summarize the main ESI project steps for completeness' sake. ESI projects are conducted in seven stages: (1) resolution, (2) scenario, (3) blueprint, (4) plastic system, (5) working prototype, (6) pilot system, and (7) production system and integration. More details can be found in Unisys (1985) and Unisys (1989).

The bipartition between ESP and ESI can be retraced in ESM's way of modelling. Screen layouts and dialogue flow diagrams are used in ESI prototype design, and 4GL languages are applied to specify and implement the prototypes and final system. An ESI project handles a single improvement project as identified in an ESP project. ESP's way of modelling recommends techniques applied in management and marketing, in addition to those applied in systems analysis and design. Typical techniques mentioned in the external studies are "Product
portfolio and market analysis", "Competitor performance analysis", and "Distribution analysis". Techniques such as "Dataflow diagrams", "Entity relationship analysis", and "Objectives analysis" are mentioned in the internal studies. ESP allows alternative techniques for the various tasks, and it can therefore be considered as a modular method allowing various modelling techniques to carry out the specific stages. The ESM documentation generally fails to define relationships between the various techniques, and criteria for the appropriateness of individual techniques. The ESP techniques can be grouped into a number of categories encompassing ESP's way of modelling, as shown in figure 7.2.

![Diagram](image)

**Figure 7.2 ESP's way of modelling at a global level**

Figure 7.2 indicates that the environment, internal aspects and objectives of an organization must be modelled, after which the design architecture can be set up.

We conclude the overview of ESM by discussing its way of supporting. As to automated tools, ESM refers to general CASE tools supporting dataflow and entity relationship diagram construction in ESP, and to 4GL environments in ESI. LINC and MAPPER are typical Unisys tools supporting ESI. Although IEW is currently used in ESP projects, it is not specifically geared to ESP and therefore only provides partial support. Word processors, general drawing tools and whiteboards are used in ESP projects, in addition to IEW.

### 7.3 CONCEPTUAL META-MODEL OF THE ESM EXPERIMENT

The results of the ESM experiment, see also Heyes (1990), are discussed in this section. The overall strategy, elicited and conceptualized in the experiment (see figure 7.3) consists of three subtasks: (1) orientation, (2) external and internal studies, and (3) activity and data analysis.
Table 7.1 lists the subtasks identified during the ESM experiment with the corresponding subsections and figures.

<table>
<thead>
<tr>
<th>Task</th>
<th>Section</th>
<th>Task structure</th>
<th>Task Scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESM construction of an information architecture</td>
<td>7.3</td>
<td>7.3</td>
<td></td>
</tr>
<tr>
<td>Orientation</td>
<td>7.3.1</td>
<td>7.4</td>
<td></td>
</tr>
<tr>
<td>Specify organization chart</td>
<td>7.3.1.1</td>
<td>7.5</td>
<td></td>
</tr>
<tr>
<td>Analyse organization characteristics</td>
<td>7.3.1.2</td>
<td>7.6</td>
<td></td>
</tr>
<tr>
<td>Determine objectives</td>
<td>7.3.1.3</td>
<td>7.7</td>
<td></td>
</tr>
<tr>
<td>Analyse environment</td>
<td>7.3.1.4</td>
<td>7.7</td>
<td></td>
</tr>
<tr>
<td>Define products or services</td>
<td>7.3.1.5</td>
<td>7.7</td>
<td></td>
</tr>
<tr>
<td>External and internal studies</td>
<td>7.3.2</td>
<td>7.8</td>
<td></td>
</tr>
<tr>
<td>Describe blackbox</td>
<td>7.3.2.1</td>
<td>7.10</td>
<td>7.9</td>
</tr>
<tr>
<td>Detail relevance of products or services</td>
<td>7.3.2.2</td>
<td>7.11</td>
<td>7.12</td>
</tr>
<tr>
<td>Determine objectives</td>
<td>7.3.2.3</td>
<td>7.12</td>
<td>7.12</td>
</tr>
<tr>
<td>Analyse current situation in detail</td>
<td>7.3.2.4</td>
<td>7.12</td>
<td>7.12</td>
</tr>
<tr>
<td>Activity and data analysis</td>
<td>7.3.3</td>
<td>7.13</td>
<td></td>
</tr>
<tr>
<td>Top down activity analysis</td>
<td>7.3.3.1</td>
<td>7.14</td>
<td></td>
</tr>
<tr>
<td>Construct functional decomposition</td>
<td>7.3.3.2</td>
<td>7.15</td>
<td>7.16</td>
</tr>
<tr>
<td>Define IPO specifications</td>
<td>7.3.3.3</td>
<td>7.17</td>
<td>7.18</td>
</tr>
<tr>
<td>Construct business flow diagrams</td>
<td>7.3.3.4</td>
<td>7.20, 7.21</td>
<td>7.19</td>
</tr>
<tr>
<td>Bottom up activity analysis</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Define internal data</td>
<td>7.3.3.5</td>
<td>7.22</td>
<td></td>
</tr>
<tr>
<td>Specify activity/data matrix</td>
<td>7.3.3.6</td>
<td>7.23</td>
<td>7.24</td>
</tr>
<tr>
<td>Define scope and detail of matrix</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fill in cells of matrix</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Construct information model</td>
<td>7.3.3.7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 7.1 Overview of contents of section 7.3
7.3.1 ORIENTATION

The first step in constructing an information architecture is the orientation, which is primarily an implicit modelling task, i.e. diagrams and tables are not produced, but only informal notes.

Organizational aspects are roughly analysed in this task: the nature and justification of the organization's activities, and its strategies. The task is further detailed in figure 7.4. Figure 7.4 suggests that the five subtasks can be performed at random, but decision rules for the decision "Analyse which aspect now?" point to a preferred sequence. One of these rules determines that the task "Analyse organization characteristics" is preferred in case other subtasks have not yet been executed. An other rule defines that the task "Determine objectives" is not required during orientation, if a detailed analysis of the business objectives has been planned.
The "Orientation" subtasks will be discussed briefly in the following subsections. The task scope, i.e. the relevant part of the concept structure, will be presented explicitly for those subtasks that are ESP-specific. The relevant concepts of other subtasks will only be discussed informally here.

7.3.1.1 SPECIFY ORGANIZATION CHART

The task "Specify organization chart" results in a standard organization chart additionally annotated with the names of the responsible managers. The modelling concepts shown in figure 6.6 of the YOURDON experiment apply here as well, extended by the concept "responsible manager". The main task objective is the identification of the persons involved in the project, and their organizational roles.

7.3.1.2 ANALYSE ORGANIZATION CHARACTERISTICS

A typical ESP task is "Analyze organization characteristics", the task scope being shown in figure 7.5. The organization is viewed from a management perspective in this task. Five different but interrelated aspects are analysed. We clearly observe the broad orientation of ESP emphasizing the importance of achieving a high degree of management awareness.

![Figure 7.5 The scope of the analysis of the main organizational characteristics](image)

The mission of an organization concerns the reasons for their being in business, i.e. what is the main reason for their existence, and what is their added value? The organizational profile determine the relationship with the environment: how is an organization profiled? Is it strictly commercial, or is it a friendly non-profit organization? An initial inventory of the current level of information technology applications is also taken, which is translated into information technology objectives in the subtask "Determine objectives".

Managerial (or leadership) style can be considered as a continuum ranging from task oriented, autocratic, directive leadership, to person oriented, participative, permissive leadership (see Blake and Mouton (1964), and Vroom and Yetton (1973)). Managerial style is related to organizational culture. While conducting ESP studies, we should realize that organizations consist of persons working in
groups, feeling part of the organization. Organizations have standards and values determining their members' conduct. Organizational culture is the broader framework within which membership feelings are experienced, and personal behaviour in organizations is influenced. Persons involved in information planning studies largely determine the ultimate success. Therefore organizational self-activity is vital to creating information plans. Effective stimulation of self-activity in ESP studies depends on the information engineer's interpretation of the existing managerial style and organizational culture.

7.3.1.3 DETERMINE OBJECTIVES

The main organizational objectives are determined in the subtask "Determine objectives", which was essentially initiated by the above task determining the organizational mission, its profile and the current state of information technology.

Organizational objectives are defined accurately in this task, and quantified, if possible. The task is optional at this stage, and could well be postponed until the task "External and internal studies" (see section 7.3.2). Figure 7.6 presents the modelling concepts applied during the determination of objectives.

Figure 7.6 The scope of the determination of objectives

Figure 7.6 suggests the construction of a network of objectives, objectives being specializations of, or shaping other objectives. Two types of objectives are shown explicitly: marketing and information technology objectives. An objective can be quantified by a measurement at a certain moment or over a certain time interval.
7.3.1.4 ANALYSE ENVIRONMENT

The subtask "Analyse environment" analyses the external parties of an organization. The subtypes of "external parties" are summarized in figure 7.7. The relevance of these subtypes to the organization is determined. Also potential external parties must be identified.

![Diagram of external parties](image)

Figure 7.7 The scope for the analysis of the environment

7.3.1.5 DEFINE PRODUCTS OR SERVICES

Finally we discuss the "Orientation" subtask defining the various types of products or services offered by the organization. Products or services are only considered as different if they really require different procedures. It is important to look for both existing and new products.

![Diagram of product/services](image)

Figure 7.8 The external and internal studies
7.3.2 EXTERNAL AND INTERNAL STUDIES

The task following "Orientation" is referred to as "External and internal studies". In this task the externals and internals of the organization are examined in more detail. Four subtasks are distinguished; their interrelationships are shown in figure 7.8. The subtasks "Describe black box" and "Detail relevance of products or services" are part of the external study, "Detail objectives" and "Analyse current situation in detail" of the internal study. It is recommended to carry out the subtasks of the external study first.

![Diagram](image)

Figure 7.9 The scope for the description of the black box
7.3.2.1 DESCRIBE BLACK BOX

The subtask "Describe black box" results in a black box description of the organization in a kind of Input-Process-Output (IPO) diagram type (see Cap Gemini Pandata (1989)) representing data entering and leaving the organization.

The concepts

Figure 7.9 presents the modelling concepts used in the construction of the black box. It demonstrates that a black box consists of external data divided into input and output data.
The concept "External data" is a subtype of the concept "data" and therefore inherits the characteristic that "data" can consist of other "data", or can be part of other "data" (a data aggregation characteristic). The occurrence of financial transaction data in an organization is emphasized.

Both input and output data are specialized in greater detail. Important classes of input data are data on (1) external parties, (2) marketing information, (3) market requests, and (4) required products and services. The association between "external party" and "input data" in figure 7.9 implies that external parties identified in the orientation task (see figure 7.4) result in external party data in this task.

"Output data" has two special subtypes: "product catalogue" and "product data". Product data relevant to be defined in the black box description rely on the existence of products and services, as defined earlier in "Orientation".

The tasks

The creation process of a black box description is described in the task structure in figure 7.10. First of all, output and input data can be identified and added to the black box description. If required, existing data can be defined in more detail by specifying constituent data, or can be regrouped into new sets of data. The blackbox can be corrected by data renaming and removal.

7.3.2.2 DETAIL RELEVANCE OF PRODUCTS AND SERVICES

The task "Detail relevance of products and services" consists of the two subtasks shown in figure 7.11. The relevance of the products and their relationships with the environment are specified first. Then, according to the results of this task, a portfolio analysis is performed.

![Diagram](Figure 7.11 Detail relevance of products and services)
The modelling concepts applied in the task "Detail relevance of products or services" are shown in figure 7.12. Special interest is in the current relative values of the organization's products and services, for example expressed as the current contribution to the annual turnover, together with their corresponding relative values in five years. Furthermore, the relationships between products and customers, supplier cooperators, competitors, and possibly the government must be defined. These aspects of figure 7.12 are analysed in the subtask "Define relevance of and environment of products".

![Figure 7.12 The scope of determining the relevance of products and services](image)

The subtask "Perform portfolio analysis" allocates products to one of the four available portfolio management cell values: a wild cat, a star, a cash cow, and a dead dog, see Heijes (1990) for an explanation of their meaning.

Products are classified according to the four categories, depending on their current and future relative product values. For example, if the current relative
product value is low and its future relative value is high, the product is
categorized as a wild cat. The movement of products across the portfolio
management matrix is also part of the portfolio analysis. Typical movements
are, for example, from wild cat to star, and from star to cash cow. The
difference between a current relative value and a future (expected) relative
value determines a product's cell movement.

7.3.2.3 DETERMINE OBJECTIVES

The task "Determine objectives" has already been mentioned in the task
"Orientation" (see the description in section 7.3.1.3). Should the task have been
performed during the orientation, it may be repeated here - if not, it must be
done now.

7.3.2.4 ANALYSE CURRENT SITUATION IN DETAIL

The subtask "Analyse current situation in detail" is only executed if the future
situation is likely to differ significantly from the current one, which follows
from the results of the portfolio and objectives analyses. If this task is carried
out, a functional decomposition of the current situation is constructed. As this
was not done in our experiment, we will defer the discussion of the functional
decomposition to the next major task after the external and internal studies,
in which a functional decomposition for the future situation is constructed.

7.3.3 ACTIVITY AND DATA ANALYSIS

The external and internal studies are followed by the task "Activity and data
analysis", in which an information architecture for the future situation is set up.
Various strategies are available and the choice is made depending on user
competence, available time, and organizational complexity.

The overall task structure for the activity and data analysis is shown in figure
7.13. The activity analysis is carried out first; two types are distinguished: (1)
top-down, and (2) bottom-up. The preferred type of analysis, top-down, is
applied in 90% of the cases. The activity analysis can be followed by the data
analysis, leading to an information model which is essentially a rough entity
relationship diagram. This part of the activity and data analysis is generally
omitted if time is scarce, or if the activity model suggests possible improvement
projects in a sufficient way. In cases where a data analysis is carried out, two
optional tasks can be carried out prior to the construction of the information
model: "Define internal data" and "Specify activity/data matrix".

155
The tasks "Top down activity analysis", "Define internal data" and "Specify activity/data matrix", which were carried out during our experiment, are discussed in further detail below.

7.3.3.1 TOP-DOWN ACTIVITY ANALYSIS

The top-down activity analysis starts by a general description of organizational activities using a functional decomposition, which can be defined in greater detail by Input-Process-Output diagrams (IPOs) and Business Flow Diagrams (BFDs). The task structure is presented in figure 7.14.
Figure 7.14 The top-down activity analysis

Figure 7.15 The construction of the functional decomposition
7.3.3.2 CONSTRUCT FUNCTIONAL DECOMPOSITION

The tasks

The construction of the functional decomposition itself is explained in figure 7.15. The determination of business functions is alternated with a more detailed description of a specific function, which involves the production of either a BFD, or the generally preferred list of business processes.
The concepts

A part of the task scope of setting up a functional decomposition is shown in figure 7.16; the BFD component is given separately in figure 7.19 for clarity reasons. Figure 7.16 mainly shows a detailed specialization hierarchy of the concept "business activity". The specialization of business functions is self-explanatory, and will therefore not be discussed in detail here. The specialization of business processes is quite ESP-specific. Five types of business processes have been identified. Operational and signal processes constitute the operational part of business functions. Signal processes indicate the arrival or departure of a signal. Arrival signal processes are paired with departure ones. Signals are used to synchronize the operational business processes in different functions. Strategic and tactical, and control processes plan and control business functions, which must therefore be part of these functions. Query processes provide facilities for statistical data retrieval, and for querying data produced by operational business processes.

The functional decomposition enables the analysis of data entering and exiting business processes in IPOs and BFDs (see figure 7.14). BFDs which have been set up during the development of the functional decomposition render this part of figure 7.14 superfluous. Otherwise there are two possible strategies: (1) construct BFDs only, or (2) construct IPOs, possibly followed by BFDs. We will now describe the definition of IPOs and the creation of BFDs.

![Diagram](image-url)

Figure 7.17 The definition of the IPO specifications
7.3.3.3 DEFINE IPO SPECIFICATIONS

The tasks

Figure 7.17. shows the task structure for the definition of IPO specifications. The scope of the entire IPO study is first determined in the task "Define scope of IPO specifications"; the scope of each individual IPO is then defined by selecting a set of specific functions and products.

The concepts

Figure 7.18 clarifies the structure of IPO diagrams. An IPO contains a list of operational processes, for which data inputs and outputs can be specified; data can also be defined more precisely (aggregation, see also figure 7.9).

All processes part of an IPO should belong to the functions determining the scope of the IPO. The following verification rule formalizes this requirement:

\[ \forall o \forall i \forall f \left[ (\text{operational\_business\_process}(o) \land \text{ipo}(i) \land \text{business\_function}(f) \land \text{is\_analysed\_in}(o,i) \land \text{details}(i,f) \Rightarrow \text{is\_part\_of}(o,f) \right] \]
7.3.3.4 CONSTRUCT BUSINESS FLOW DIAGRAMS

The concepts

The modelling concepts used for setting up BFDs (figure 7.19) shows that business flow diagrams have a scope consisting of a set of business functions and products.
Business flow diagrams include business activities, business activity groups, data, and data groups. Flows between business activity (groups) and data (groups) are defined as input flows if the direction is from data to business activities, and as output flows in the reverse case. Business activity groups are used for abstraction from their specific business activity parts, while data groups indicate that specific sets of data are created or used simultaneously. Data groups are not named specifically, contrary to business activity groups, but they are identified by specific notational conventions. Data is classified as external or internal.

The tasks

![Diagram of business flow diagram construction]

Figure 7.20 The construction of business flow diagrams

The task structure for setting up BFDs is given in figure 7.20. The scope of a BFD must be defined before the diagram can be set up, but it can be extended during the modelling process in the task "Extend scope". The functional decomposition
can be adapted according to new and extended BFDs (see the task "Adapt functional decomposition"). BFD construction is shown in more detail in figure 7.21.

![Diagram showing process of BFD construction](image)

Figure 7.21 The construction of a business flow diagram

Business flow diagramming relies on the identification of a business activity (group) followed by an analysis of its required inputs and resulting outputs. It
is subsequently advisable to look for shared use of data for the new identified outputs (see the task "Analyse usage of specific data (group)"). In addition to this basic reasoning pattern, activity and data groups can both be extended, new flows between existing components can be identified, and corrections can be applied by removal and renaming.

Having described the top-down activity analysis of figure 7.13, we will now discuss the tasks "Define internal data", "Specify activity/data matrix" and "Construct information model".

7.3.3.5 DEFINE INTERNAL DATA

Internal data, such as standards, rates, and tables, are only available within specific business activities and must be defined for undecomposed business activities. Its scope is presented in figure 7.22.

![Figure 7.22: The scope of the definition of internal data](image)

7.3.3.6 SPECIFY ACTIVITY/DATA MATRIX

An activity and data matrix is used as a cross-reference between the activities and the data identified in previous tasks, underlining specific interrelationships between sets of data. The matrix consequently paves the way to the creation of the information model.

![Figure 7.23: The specification of the activity/data matrix](image)
The tasks

After defining the scope of the matrix, its cells are filled. The task sequence is shown in figure 7.23.

The concepts

Figure 7.24 shows that the data are listed in the rows of the matrix, with their data definitions, sources and owners. The source of data can be internal or external, while the owner can be a specific department or person. The columns of the matrix are used for the business activities, a salient characteristic being that their level of detail is not bound to rules. Business functions may occupy a single column or multiple columns, each linked to an individual business process. The matrix cells can contain manipulation relationships representing either a create, use, or delete.

Figure 7.24 The scope of for the construction of the activity/data matrix
7.3.3.7 CONSTRUCT INFORMATION MODEL

The last task in creating an ESP information architecture involves the setting up of an information model (at least if a data oriented analysis has been decided on). An information model is an unnormalized entity relationship model, possibly without cardinalities. Its purpose is to achieve a general understanding of the relevant data without concentrating on details as customary for normalized data models.

The information model was not actually created during our experiment because of time constraints. Additional elicitation sessions would have been required to analyse the details of this task, which was, however, considered to be outside the scope of this study.

Based on the results of the activity and data analysis, an ESP study generally proceeds by constructing, for example, an organizational and technical architecture, and by identifying and planning improvement projects (see section 7.2). This completes our description of the conceptual meta-model resulting from the ESP experiment.

7.4 SUMMARY

Having discussed the results detailed in the previous section, we will now summarize the salient ESM characteristics as emerged from the experiment.

ESM was originally only loosely documented. The experiment resulted in a detailed insight in the construction of an information architecture in ESP projects. The applied strategy can be characterized as follows:

- Flexible strategy

Possible strategies must be selected at various points during the modelling process. For example, a decision whether or not to study the current internal situation in depth, whether to follow a top-down or bottom-up activity analysis, and whether to apply an intermediate representation technique between the functional decomposition and the business flow diagrams.

There are clear preferences for certain strategies, but user characteristics and time constraints largely determine the choice between specific strategies.
SUMMARY

- Process oriented

Various types of models are used for structuring organizational activities and the data flows between these. According to the expert, the information architecture’s fundamental model is the functional decomposition, which has also been specialized to a large extent (see figure 7.16). It is the technique with the most specialized concepts if compared to the other modelling techniques applied.

- External and internal models

In addition to the "standard" modelling techniques for structuring the relevant organizational activities and data (internal models), various models emphasizing the externals of organizations are created, for example, the portfolio matrix and the black box.

- Management techniques

A number of management techniques have been applied to increase the organization’s self-consciousness about its products, market and strategy.

- The Pareto rule

The Pareto rule specifies that it takes 20% of the time to examine 80% of a given situation, and that it consequently takes 80% of the time to process the remaining 20%. This implies that not all areas should be given the same amount of thought during the creation of an information architecture. Various tasks require firstly the determination of its scope before the specific study (for example, see the construction of IPOs and BFDs).

- The self-activity of the organization

The importance of the organization’s self-activity is stressed during the ESM experiment. Several diagrams (particularly the IPOs) were constructed by the users, and consequently due attention should be paid to user training and education.
8

DETAILED UNDERSTANDING OF MODELLING

8.1 INTRODUCTION

We shall now draw our conclusions from the theory and experiments described in the chapters 3 to 7. We are primarily interested in having achieved a detailed understanding in the way of working and the way of modelling (i.e. modelling knowledge) as applied by the experienced information engineers involved in this study. For this reason we will evaluate three specific areas of concern which all contribute to this objective.

In the first place we are concerned with the results as obtained from the three experiments. The review of these results in section 8.2 focusses on the new insights that have been realized in the specific field of modelling studied and which are a consequence of the experiments performed.

In section 8.3 we perform a kind of strength and weakness analysis on the meta-modelling technique applied during the experiments. We shortly recapitulate its main starting points and subsequently evaluate to what extent the technique adequately represents elicited modelling knowledge. Section 8.4 is concerned with conclusions on the applied acquisition approach. It addresses the effectiveness and efficiency of the acquisition process.

Based on sections 8.2 to 8.4 we review the first two hypotheses of the study in section 8.5 and assess the extent to which these are upheld by our findings. Furthermore we will relate these findings to the main objective of this thesis concerned with the improvement of modelling support to practising information engineers in information systems development.

8.2 DETAILED MODELLING KNOWLEDGE

The summarizing sections in chapters 5 to 7 indicate that the experiments have resulted in various new detailed insights in the way of working and the way of modelling as applied by the experienced information engineers in the specific area of information architecture construction.

These results are not too surprising because D2S2 and ESM only provide short descriptions of this specific field of modelling, whereas YOURDON is not concerned with the construction of an information architecture at all.
Within this section it is more important to arrive at a thorough understanding of the nature of these new insights. In such a way we make clear what type of results can be realized when applying the meta-modelling technique and the acquisition approach. Table 8.1 itemizes these findings. We will discuss each of these conclusions in more detail below.

<table>
<thead>
<tr>
<th>Refined concepts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Important non-diagramming concepts</td>
</tr>
<tr>
<td>Large difference between concept and notation</td>
</tr>
<tr>
<td>Preference for specific modelling technique</td>
</tr>
<tr>
<td>Strong relationships between modelling techniques</td>
</tr>
<tr>
<td>Gradual change in use of modelling techniques</td>
</tr>
<tr>
<td>Explicit strategy</td>
</tr>
<tr>
<td>Own strategy</td>
</tr>
<tr>
<td>Many arguments stem from practice</td>
</tr>
<tr>
<td>Much variety between individual experts</td>
</tr>
</tbody>
</table>

Table 8.1 Types of detailed results achieved in the experiments

- Refined concepts

The modelling concepts to be used in the diagrams as officially prescribed by the various methods have mostly been refined and customized by the experts. Typical examples demonstrating these refinements and changes can be found in figures 5.15, 5.22, 6.16, 6.19, 7.9 and 7.16. In this way the experts have partially solved the sometimes general and fuzzy concepts applied in diagramming techniques.

- Important non-diagramming concepts

The experts are also interested in aspects of the organization which cannot be explicitly modelled in diagrams. For example, each expert starts with an orientation in which only a mental model of certain organizational aspects is made, sometimes partially documented in the form of simple notes. This type of gathered information is essential to the subsequent modelling process, and have therefore been taken into account in the resulting conceptual meta-model of the experiments. Some of the typical non-diagramming concepts found are "objective", "problem", "primary process", and "external party".

- Large difference between concept and notation

In all diagramming techniques applied by the experts, we have found a mapping of various applied modelling concepts to only one graphical notation. For example, in the functional decomposition applied in the ESM experiment, no graphical distinction is made between primary functions and support functions although these are different modelling concepts.
• Preference for a specific modelling technique

Each expert has a preference for a certain modelling technique. While the first concentrates on ER models, the others prefer dataflow diagrams, and functional decompositions respectively. The modelling technique preferred by an expert includes a large number of specialized modelling concepts if compared to the techniques with a lower preference.

• Strong relationships between different modelling techniques

Various relationships between modelling concepts originating from different diagramming techniques have been identified in the experiments. In most cases these relationships are heuristic by nature; for example: an external party in a context diagram generally corresponds to an entity in an entity relationship diagram (see figure 6.19).

• Gradual change in use of modelling techniques

The way in which a modelling technique is used, gradually changes during the modelling process. A good example of this can be found in the D2S2 experiment when tracing the use of the entity-relationship diagramming technique in the sections 5.3.2, 5.3.3 and 5.3.4 respectively. The change in use is reflected in a changing set of modelling concepts and verification rules.

• Explicit strategy

Each expert explicitly applies a well-tried strategy to arrive at an information architecture. Sometimes choices must be made if alternative strategies are available, but the experts are always quite conscious of the strategy they would like to apply.

• Own strategy

Although the experts apply a clear strategy for the construction of an information architecture, we also conclude that they are very much individual, self-made strategies. The experts have found their own solutions for methodical shortcomings of the methods and techniques applied. All expert strategies identified in the experiments include a "divide and conquer" strategy and aim at a strong user involvement.
Many argumentations stem from practice

In harmony with our observation about individual self-made strategies, explanations and reasons justifying specific strategies originate from practical project experience. Experts often reason negatively in justifying their strategies by pointing to the risks of not adhering to these.

Much variety between experts

The general impression of the experiments is that the experts use a variety of strategies and concepts. Each has his preferences and reasons for concentrating on specific organizational aspects and particular strategies. Table 8.2 shows some differences as found between the experts:

<table>
<thead>
<tr>
<th>Focus on internal organization</th>
<th>Great attention for external side of organization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Critical view on business</td>
<td>Mainly interested in information processing</td>
</tr>
<tr>
<td>Data oriented aspects predominant</td>
<td>Process oriented aspects predominant</td>
</tr>
<tr>
<td>Trigger analysis for process model construction</td>
<td>Input-output analysis for process model construction</td>
</tr>
<tr>
<td>Outside-in process model construction</td>
<td>Top-down process model construction</td>
</tr>
<tr>
<td>Explicit analysis of physical current situation</td>
<td>Only interested in logical model of future situation</td>
</tr>
<tr>
<td>Taking the position of expert</td>
<td>Taking the position of facilitator</td>
</tr>
</tbody>
</table>

Table 8.2 Characteristic differences between the observed expert strategies

Considering the above observations, we conclude that we have achieved a greatly improved insight in the way of modelling and the way of working together with their interrelationships as applied by the experts. Each expert appears to have a strongly personalized approach and terminology. The various own strategies, personal preferences, changes and refinements result in a very differentiated view of the approaches that might be applied for the construction of an information architecture. Possible consequences for automated support are dealt with in section 8.5.

8.3 ADEQUATE REPRESENTATION OF MODELLING KNOWLEDGE

An adequate technique for the representation of modelling knowledge is a major ingredient in achieving the detailed understanding as described in the previous section. For example, if the technique would not have included notions to express a way of working, we would not have been able to gain a better insight in possible strategies for constructing an information architecture. A major criterion determining the adequacy of the technique is therefore its expressive power, in other words: are we able to express all the things we would like to express? Other criteria contributing to the adequacy of the technique are its flexibility and its communicatability.
In this section we evaluate the meta-modelling technique on its adequateness as far as it concerns its expressive power, flexibility and communicability.

In order to be able to evaluate the expressive power of the meta-modelling technique we recapitulate its major starting points. The meta-modelling technique should primarily encompass constructs for describing the process and product oriented aspects of a modelling process in a coherent way. Modelling has been considered as a problem solving process in which several strategies can be applied, various types of models can occur, models gradually gain in accuracy and coherence, and existing models are extensively used in the construction of new models, see sections 1.5 and 1.6. This view has been refined in the framework for understanding information systems development in chapter 2 in a way of working, a way of modelling and their interrelationships expressed in terms of levels of inclusion, precision and coherence and influence of the modelling environment (see sections 2.3 to 2.5). Evaluating the expressive power of the meta-modelling technique therefore implies evaluating its constructs offered for representing a way of modelling, a way of working and the interrelationships between both. A review of the strengths and weaknesses of these three elements of the meta-modelling technique, based on the results of the three experiments, is itemized in table 8.3, while these observations are briefly explained in the subsequent subsections.

<table>
<thead>
<tr>
<th>Adequate representation of a way of modelling, way of working and their interrelationships</th>
</tr>
</thead>
<tbody>
<tr>
<td>A concept structure to represent a way of modelling</td>
</tr>
<tr>
<td>Adequate expressive power and flexible representation</td>
</tr>
<tr>
<td>Specialization as a powerful abstraction mechanism</td>
</tr>
<tr>
<td>Objectification as a limited abstraction mechanism</td>
</tr>
<tr>
<td>Comprehensibility because of graphical conventions and use of scopes</td>
</tr>
<tr>
<td>A task structure to represent a way of working</td>
</tr>
<tr>
<td>Sufficient expressive power and flexibility in modelling</td>
</tr>
<tr>
<td>Comprehensibility because of graphical conventions and decomposition</td>
</tr>
<tr>
<td>A meta-model as an interrelated concept structure and task structure</td>
</tr>
<tr>
<td>Strong mechanisms for tuning task and concept structure to each other</td>
</tr>
<tr>
<td>Flexibility in defining verification rules</td>
</tr>
<tr>
<td>Good stop heuristics for decomposing tasks</td>
</tr>
<tr>
<td>Effective modelling of dependencies between tasks</td>
</tr>
<tr>
<td>Formal definition of verification and decision rules of limited use</td>
</tr>
</tbody>
</table>

Table 8.3 Conclusions on the meta-modelling technique

8.3.1 A CONCEPT STRUCTURE TO REPRESENT A WAY OF MODELLING

In the meta-modelling technique a way of modelling is formally defined in a concept structure, which is expressed in terms of concepts, associations,
specializations and objectifications. A formal tuple definition is given in section 3.2.2 together with six properties that have to be satisfied for a correct concept structure. Conclusions about the use of concept structures are:

- Adequate expressive power and flexible representation

A concept structure quite naturally extends beyond a definition of the concepts used in separate diagramming techniques. Concepts such as "problem" and "objective" can also be considered, although they are not applied in explicit diagramming techniques and only in quick orientations. The concepts used in different techniques can be interrelated by associations in a simple manner, see for example the figures 5.15, 6.19 and 7.18.

- Specialization as a powerful abstraction mechanism

Specialization is an abstraction mechanism which has been used very often: about 75% of the concepts identified in the experiments are involved in specialization hierarchies.

- Objectification as a limited abstraction mechanism

Objectification is only applied sparingly. In its current definition, i.e. the interchange of association and concept, it is clearly limited. However, we identified another frequently occurring type of association related to the idea of objectification, namely the "consists of/part of" association. About 25% of the associations in the experiments were of this type; it essentially specifies the internal structure of a specific concept, such as the observation that "An entity model consists of entities and relationships". This observation led us to conclude tentatively that the formalism for concept structures could be usefully extended with a notion of "complex concepts" consisting of other (complex) concepts, associations and specializations. The consequence of this extension to the meta-modelling technique is as yet unclear, and is therefore a possible subject for future research.

- Comprehensibility because of graphical conventions and use of scopes

The graphical representation of a concept structure gives good insight to the modelling concepts and their associations (if meaningfully named). Specialization hierarchies quite explicitly show the identified specialized modelling concepts. A complete concept structure generally consists of so many concepts that their presentation is quite ineffective; we have therefore decided to present and discuss only specific task scopes of a concept structure with the experts (see also the presentation of the experimental
results in chapters 5 to 7). We have thus been able to split a concept structure into sections rarely exceeding 25 concepts, which has improved the communication of a concept structure to the experts.

Based upon the above observations we conclude that the expressive power, flexibility and communicability of concept structures have been realized in a sufficient way. Its expressive power can still be improved slightly by introducing the notion of complex concept by which a concept structure is not necessarily a flat structure anymore.

8.3.2 A TASK STRUCTURE TO REPRESENT A WAY OF WORKING

A way of working is formally represented in a task structure, which deals with tasks, decisions, triggers, decomposition relationships and initial items. A formal tuple definition is given in section 3.4.2 with nine properties that must be satisfied for a correct task structure. Conclusions on using task structures are:

- Adequate expressive power and flexible modelling

  Task structures quite effectively can model all kind of time orders between tasks. Parallelism, sequence, choice, iteration and synchronization are all ordering aspects of tasks which can be expressed in a task structure.

  Decomposition and initial item are two constructs not strictly required for more expressive power. They quite effectively increase however the flexibility and efficiency in defining a task structure. Knowledge engineers are completely free in the use of decomposition: it initially depends on personal preference whether or not tasks and decisions are collected in more general tasks. The existence of parallel tasks, cyclic task structures and repeatedly occurring task structures indicate the usefulness of a more general task encompassing these parallel tasks, cyclic task structures or repeatedly occurring task structures.

- Comprehensibility because of graphical conventions and decomposition

  The graphical notation of a task structure particularly contributes to the communication with the experts: by the use of decomposition, task structure diagrams are relatively small and therefore easily understood.

These observations concerning the use of task structures lead to our conclusion that the expressive power, flexibility and communicability of task structures have been realized in an adequate way.
8.3.3 A META-MODEL AS AN INTERRELATED CONCEPT STRUCTURE AND TASK STRUCTURE

A meta-model constitutes the formal representation of modelling knowledge. It encompasses a concept structure and a task structure. Five constructs exist to interrelate these two structures, namely (1) task scopes, (2) verification rules, (3) information places, (4) basic procedures and (5) decision rules, see sections 3.3 and 3.5. The formal definition of a meta-model is given in section 3.5.5 together with 14 properties that have to be satisfied for a correct meta-model. Conclusions related to the use of these constructs are:

- **Strong mechanisms for tuning task and concept structure to each other**

  Task scopes appear to be an extremely powerful means of linking task and concept structure, because they clearly indicate those parts of the concept structure that can be manipulated by a task.

  The two syntax rules defining (1) that the scope of a subtask should be part of the scope of its supertask, and (2) that the scope of a supertask should be the union of the scopes of its subtasks, implies that task and concept structure are highly tuned to each other.

- **Flexibility in defining verification rules**

  Verification rules are defined after that task structure and concept structure have been interrelated to each other by having specified task scopes. Verification rules are identified for each task separately by examining the task scope on valid statements for the specific part of the concept structure. In this way, and by the distinction between a-priori and a-posteriori verification rules (invariant and post condition), the knowledge engineer need not question the overall validity of verification rules. The appropriateness of these rules can be adjusted completely to the specific tasks within a modelling process.

- **Good stop heuristics for decomposing tasks**

  Tasks must be decomposed until they can be related to basic procedures. The set of useful basic procedures is unrestricted in a conceptual meta-model. This implies that, at a certain level of detail, a meaningful task name and a small task scope generally indicate sufficiently the activities to be carried out within the task and therefore allow to stop additional task decomposition.
Sec. 8.3 ADEQUATE REPRESENTATION OF MODELLING KNOWLEDGE

- Effective modelling of dependencies between tasks

Information is exchanged between tasks by means of information places. Their use is essential to specifying the context of specific tasks, to saving selections for future use, or to triggering specific task behaviour depending on the results of other tasks. The additional graphical representation of information places in a task structure often results in overloaded diagrams. *Meaningful task naming* generally guarantees sufficiently the identification of relevant information places. Therefore we have not used the graphical conventions for information places in communication with the experts (see also chapters 5 to 7). *If* a graphical specification of information places would be preferred, a possible solution would be the use of an extra diagramming convention in which *only* the *information exchange* between tasks is modelled; this opposed to the current task structure diagrams in which only *time dependencies* between tasks are modelled.

- Formal definition of verification and decision rules of limited use

Although the use of predicate logic in defining verification rules satisfied our need for a highly expressive formal language, these formal rules must be replaced by natural language statements in discussions with the experts, because these are more adequate. The same applies to the formal language used for the definition of decision rules.

Based upon the above considerations, we conclude that we have achieved strong means for integrating a task structure and a concept structure. By the definition of a scope and verification rules for each task separately, we are able to define relevant levels of inclusion and precision in a very flexible way. Besides time dependencies between tasks, which can be modelled adequately in a task structure, information places offer good means to define result dependencies between tasks. The specification of both time dependencies and result dependencies in one diagram results in overloaded diagrams. During the communication with the experts these two aspects of a meta-model are discussed separately. The formal language of verification rules and decision rules provides for a high expressive power but is not absolutely necessary for a *conceptual* meta-model. The availability of these formal languages however does not hinder the construction of these meta-models and they will reasonably simplify the development of the MSS environment in chapter 9.

Summarizing, the meta-modelling technique has an excellent expressive power, includes strong internal consistency checks, allows for a natural communication with the experts and can represent modelling knowledge in a flexible way.
8.4 EFFECTIVE AND EFFICIENT ACQUISITION OF MODELLING KNOWLEDGE

The last part of our evaluation concerns the acquisition process leading to the detailed understanding as described in section 8.2. An effective and efficient process contributes to the applicability of the acquisition approach. We summarize our conclusions on acquisition process in table 8.4, and discuss them in greater detail below.

| Elicitation of valuable and practical verbal data |
| Good preparation required |
| Intensive elicitation process |
| More sessions lead to more detail |
| Satisfying cooperation of experts |
| Strong iteration between interpretation and conceptualization |
| Vague distinction between application and meta level |
| Knowledge engineer dependent |
| Time intensive approach |
| Automated tool needed |

Table 8.2 Conclusions on the process of modelling knowledge acquisition

- Elicitation of valuable and practical verbal data

Of central importance in the entire acquisition approach is our interest in practised information engineers. In order to elicit their modelling knowledge, we apply think aloud protocols using simulation by teletype. We conclude that this elicitation method indeed improves the expert's think aloud process, and avoids theoretical discussions on the applied concepts and strategies, because actual problems must be solved. The think aloud protocols result in explicit reporting on decisions made and strategies considered during a modelling process. Implicit modelling knowledge can consequently been made explicit.

- Good preparation required

Thinking aloud while solving problems is not a natural thing to do. We therefore exhaustively discuss the experimental setting of the elicitation sessions during the preparation of the experts. All experts have to get accustomed to verbalizing their thoughts. Prompts such as "What are you thinking about", "Keep talking", and "Say it aloud" are necessary, especially in the initial phases of the experiment.

- Intensive elicitation process

The experimental setting has the disadvantage of slowing down the communication speed. Persons easily become irritated by a deterioration
of their normal speed of information exchange. We prevent irritation during the experiment by demonstrating that the terminal connection is similar to verbal communication, only differing in the speed of communication. Both thinking aloud and slower communication are quite tiring for the experts.

- **More sessions lead to more detail**

We have used two day protocol sessions for the experiments, primarily based on experience in Wijers (1988a). We deduce from the experimental result descriptions that two days are hardly sufficient for constructing a complete information architecture. Elicitation sessions can generally be conducted better without time constraints. Furthermore, a single elicitation session does not necessarily lead to a meta-model including all strategies and concepts applied by an expert. More elicitation sessions may further detail acquired modelling knowledge.

- **Strong iteration between interpretation and conceptualization**

We re-interpret elicited verbal data several times during an experiment. In a first, rapid scan the most apparent tasks and concepts are identified and marked. This results in a first understanding of the expert’s way of modelling and working. Then iterations are performed for more detailed interpretation of the verbal data. For example, at first instance a dialogue between user and information engineer can be interpreted as "the identification of the primary products of an organization". In a second round the same verbal data is interpreted as "the identification of the various product groups of an organization, with their relative importance and added value to the customers". After the second round, the text is split into various text fragments, while the text only consists of one fragment in the first case. Three interrelated concepts can be identified in the second case, "product group", "relative value" and "added value"; only one concept is available in the first case, namely "primary product". The interpretation of verbal data is a crucial step in the acquisition approach and cannot be realized in one single scan through the verbal data.

- **Vague distinction between application and meta level**

In theory (see chapter 3) there is a clear distinction between the application and the meta level. However, this distinction is not always abundantly clear in the verbal data resulting from elicitation sessions: some fragments are clearly concept *instances*, others pure *concepts* used by the expert. On the other hand, we also regularly come across fragments which are not readily recognizable as concept *instances* or *concepts*. For example, the concepts
"client" and "supplier" in chapter 5 have been modelled as *subtypes* of "external party" and are considered to belong to the meta level. However, it would have been acceptable to interpret "client" and "supplier" as *instances* of "external party", placing them at the application level. One should aim at including only those concepts at the meta level, which are likely to be relevant for other applications as well. However, the boundary between application and method-specific items cannot always be drawn with certainty.

- Knowledge engineer dependent

Relevant text fragments are more easily recognized by a knowledge engineer familiar with the field studied. Sometimes it is simply impossible to interpret specific verbal data without having additional interviews with the expert. Verbal data interpretation is troublesome in cases where the expert's concepts are beyond the experience of the knowledge engineer, and reading of background literature and interviews are recommended in such cases. For this reason it is important to perform the interpretation in close liaison with the expert.

- Time-intensive approach

The transcripts of the think aloud protocols performed in the experiments contain about 90 pages of text. The analysis of this text, and the background literature study and interviews result in detailed modelling knowledge (see section 8.2). However, text analysis is time consuming, certainly because the detailed reading of voluminous documents is a slow process, but mainly because the identification and marking of relevant text fragments is an intensive and iterative process.

- Automated tool needed

We have used TASK, a CASE tool provided as part of Bots (1989), for setting up task structures, and SDW-NIAM, one of the modules of the SDW CASE tool marketed by Cap Gemini Pandata, for concept structures. Both tools have been used as documentation aids, and their limitations have become quite clear during the experiments.

We particularly observe a need for automated support tools, checking all syntax rules as specified in chapter 3, and supporting the specification of the various formal links between task and concept structures. Complete conceptual meta-models are so complex that full verification without automated tools is quite unmanageable.
Automated support is also required for the construction of a text based model, which is an intermediate between verbal data and a full conceptual meta-model. The arrangement of verbal data, text based models and meta-models in one repository strongly enhances the iterative process between elicitation, interpretation and conceptualization.

Our main conclusion of the above analysis is that the acquisition process is directed effectively towards the acquisition of detailed modelling knowledge of experienced practising information engineers. Starting point is their modelling behaviour in actual modelling sessions. Successive iterative tasks effectively transform elicited verbal data into a conceptual meta-model. Acquiring detailed understanding of a specific field of modelling is an intensive process taking much time. The use of automated tools might result in a more efficient and better manageable process, in particularly if these tools are built to support our specific acquisition approach and meta-modelling technique.

8.5 CONCLUSIONS

In this chapter we conclude that the representation technique and acquisition approach developed for the representation and acquisition of modelling knowledge contribute to a strongly improved understanding of that specific field of modelling to which technique and approach are applied; the meta-modelling technique because it adequately represents the process and product oriented aspects of modelling, the modelling knowledge acquisition approach because it is directed to the elicitation of implicit and detailed insights of an experienced information engineer which is transformed stepwisely into a modelling knowledge specification represented according to the applied meta-modelling technique conventions. Based on these findings reported in sections 8.2 to 8.4 we conclude that the first two hypotheses of this thesis (see section 2.10) cannot be rejected.

The acquired insights display a set of detailed, personalized strategies and preferences in which we can identify some communities as far as it concerns the applied modelling concepts. Alas we cannot conclude the same for the applied strategies. An MSS supporting the three experts equally well, will therefore probably not exceed the level of support realizable with current CASE tools. Improved modelling support must be found in "individualized" tools which have been customized to specific ways of working and modelling. In the next chapter we will further address this issue of individualized modelling support by the development of a prototype MSS environment in which modelling knowledge is explicitly available and adaptable and in which this knowledge rules the type of modelling support offered.
DEVELOPMENT OF AN MSS ENVIRONMENT

9.1 INTRODUCTION

In this chapter we concentrate on modelling support systems (MSS). Automated modelling support has to be provided in the fields of process guidance, model construction, model verification and model derivation (see section 2.9). We envision MSSs to function as, what we call, a meta-system which offers modelling support by an interpretation mechanism interpreting explicitly available modelling knowledge.

In section 9.2 we elaborate on a typical architecture and functionality of an MSS environment in which meta-models can be specified according to our metamodelling technique conventions and in which these meta-models can subsequently be used by an interpretation mechanism in order to provide modelling support in the above mentioned areas of concern. A prototype of this environment, developed in section 9.3, demonstrates the viability of such an MSS environment. The development of the prototype is directed at the automatic interpretation and execution of valid meta-models. In section 9.4, we apply the prototype MSS environment, concentrating on implementing the results of the three experiments, and on replaying these experiments. Based on these findings we revisit the third hypothesis of our study and assess the extent to which it is upheld by the prototype MSS environment developed in this chapter.

In section 9.5 we conclude with an overall evaluation of the findings reported in this thesis in relationship to our concern of improving modelling support for information engineers in information systems development.

9.2 FUNCTIONALITY OF AN MSS ENVIRONMENT

An MSS environment consists of the five major components shown in figure 9.1, recall section 2.9. Two types of users can be identified: firstly, the "modelling knowledge engineers" applying a meta-model editor to specify meta-models. These meta-models are stored in the meta-knowledge base.

The second type of users are the information engineers seeking modelling support. The interpretation mechanism provides support via a standardized user interface, by interpreting available meta-models. Results gradually emerging in this way during a modelling process are stored in the application knowledge base. Data in the application knowledge base includes the process history and
the application model. Data in the application knowledge base is a true instantiation of a meta-model in the meta-knowledge base, with the same distinction between process and product oriented data.

![Diagram](image-url)

**Figure 9.1** The general architecture of an MSS environment

The exact functionality and structure of an MSS environment very much depends on the meta-modelling technique applied. Based on the technique defined in chapter 3, we shall discuss the functionality of the meta-model editor and the interpretation mechanism in more detail in the next two subsections. We will not cover user interface details in this chapter because the emphasis is on the formal interpretation and execution of meta-models defined according to our meta-modelling technique. User interface details can be found in Schaapherder (1990).

### 9.2.1 FUNCTIONALITY OF THE META-MODEL EDITOR

The meta-model editor supports the construction of meta-models. For the meta-modelling technique at hand, this implies that we require:

- A task structure editor

  The task structure editor supports the definition of tasks and the decomposition of tasks into more detailed task structures. For each task the task structure editor furthermore supports the definition of:

  - the information places part of the task,
  - the manipulation of information places by the subtasks,
  - the scope of the task,
  - the verification rules for the task,
  - the basic procedures for the basic subtasks, and
  - the decision rules for the (sub)decisions.
 Sec. 9.2  FUNCTIONALITY OF AN MSS ENVIRONMENT

- A concept structure editor

  The concept structure editor specifies the concepts, associations, specializations and objectifications of a concept structure.

- A consistency checker

  The consistency checker verifies a specified meta-model according to the syntax rules as defined in chapter 3.

9.2.2 FUNCTIONALITY OF THE INTERPRETATION MECHANISM

The interpretation mechanism interprets a meta-model loaded from the meta-knowledge base. The interpretation mechanism is subdivided into mechanisms for:

- an agenda,
- basic procedure execution,
- information place creation,
- information place adaptation,
- history adaptation,
- verification and derivation, and
- decision making.

Each of these mechanisms will be discussed in detail in the following sections.

9.2.2.1 THE AGENDA MECHANISM

The agenda mechanism interprets a task structure defined within a loaded meta-model. During a modelling process, task and decision instances are created and executed leading to specific states for these instances, such as current, waiting, or active. An information engineer continuously executes task and decision instances.

The agenda mechanism keeps track of the states of task and decision instances. The set of active and waiting task and decision instances is referred to as the agenda. Waiting and active instances can be selected by the information engineer for execution. A selected task or decision instance becomes current. At most one instance can be current at the same time. The agenda mechanism handles the creation of new waiting instances and the termination of completed task instances. As mentioned before, the full history of performed task instances and decision instances is recorded by the agenda mechanism.
The agenda mechanism supports seven functions:

- the activation of a waiting task instance part of the current task instance,
- the activation of an active task instance part of the current task instance,
- the termination of the current task instance,
- the handling of a waiting decision instance part of the current task instance,
- the switch from the current instance to its active supertask instance,
- the abortion of the current instance, and
- the abortion of the last task/decision instance of the current task instance.

These functions represent state transitions in which existing task object instances, i.e. task instances or decision instances, change state, and in which new task object instances are generated.

For an accurate description of the state transitions initiated by the agenda mechanism, we have defined a Predicate/Transition (PrT) net formally describing the dynamic interpretation of a task structure \( T = (T, D, L, N, F, G, I) \) (see chapter 3) by the agenda mechanism. The use of PrT-nets as a way of formalizing the dynamic semantics of task structures find its origin in Hofstede and van der Weide (1990). Before explaining the PrT net developed for the dynamic interpretation of task structures, we will briefly describe the theory of PrT nets (see Genrich (1987) and Kappel and Schrefl (1990)).

9.2.2.1.1 INTRODUCTION TO PREDICATE/TRANSITION NETS

Informally, PrT nets are Petri nets in which the places correspond to predicates with variable extensions, and transitions represent classes of elementary changes of these predicate extensions. A simple PrT net is shown in figure 9.2, where places are represented by circles, and transitions by rectangles. There are four places and three transitions in this example. The place customer in need of service corresponds to a two-place predicate. For example, a token \(<c,n>\) in this place denotes customer \(c\) being in need of service \(n\). Transitions can remove tokens from places and add these. In the example, the transition "start service" removes a token \(<c,n>\) from the place customer in need of service and a token \(<d>\) from the place clerk inactive. As a result a token \(<c,n,d>\) is added to the place service. An additional condition, called the transition formula, has been added to this transition: clerk \(d\) should be capable of handling service \(n\). The predicate capable has a fixed extension or, in other words, is a static predicate.

After this introduction by example, we will present the theory of PrT net formally (see also Kappel and Schrefl (1990)). The formal theory should be thoroughly familiar in order to understand the contents of the sections 9.2.2.1.2 to 9.2.2.1.7.
Definition 9.1

A PrT net is defined as a 7-tuple $\mathcal{P} = (S, T, F, C, A_S, A_T, A_F)$ consisting of:

- A bipartite graph $(S, T, F)$ where $S$ and $T$ are two disjoint sets of vertices $(S \cap T = \emptyset)$, called places and transitions respectively, and $F$ is a set of directed arcs, each connecting a place $p \in S$ to a transition $t \in T$ or vice versa $(F \subseteq S \times T \cup T \times S)$.

- A structure $C$, the support of $\mathcal{P}$, defining the collection of individuals, and the appropriate operators and static predicates.

- $A_S$, a bijection between the set of places $S$ and a set of predicates with variable extensions.

- $A_T$, inscribing the set of transitions $T$ by mapping them to logic formulae built up in $C$ (transition formulae).

- $A_F$, the mapping of the set of arcs $F$ into formal sums of tuples of variables: the length of each tuple is the arity of the predicate annotating the place connected to the arc.
A token \( t = \langle a_1, a_2, \ldots, a_n \rangle \) in a place \( p \) denotes the fact that the corresponding predicate is true over the individuals \( a_1 \) to \( a_n \). A distribution of tokens over the places is called a marking \( M \) of the net, which can result in an other marking \( M' \) by firing transitions. For each transition \( t \), \( I(t) = \{ s \in S | (s, t) \in F \} \) denotes the input set of \( t \), and \( O(t) = \{ s \in S | (t, s) \in F \} \) the output set. The set of all variables occurring in the tuples annotating the incoming and outgoing arcs of \( t \) is called the index of \( t \).

**Definition 9.2**

A transition \( t \) is enabled for a substitution \( \sigma \), if \( \sigma \) is a substitution replacing all variables of the index of \( t \) by individuals, such that:

- \( \sigma \) satisfies the transition formula of \( t \).
- Each place \( s \in I(t) \) contains the tokens specified by the label on the corresponding incoming arc \((s, t)\) with the variables substituted by individuals according to substitution \( \sigma \).
- By firing the transition, no token is inserted in a place in which it is already contained.

**Definition 9.3**

A transition can fire with a substitution \( \sigma \) if it is enabled for \( \sigma \). If a transition is fired:

- The variables in its incoming and outgoing arcs are substituted as specified by \( \sigma \).
- From each place \( s \in I(t) \) the tokens specified by the label on the arc \((s, t)\) are removed from \( s \).
- To each place \( s \in O(t) \) the tokens specified by the label on the arc \((t, s)\) are added to \( p \).

Having concluded the general introduction to PrT nets, we will now discuss the PrT net specifically aimed at the dynamic interpretation of task structures.

The supporting static structure for this PrT net consists of a task structure \( T \) together with \( N \), the set of natural numbers including zero, the addition operator and two auxiliary static predicates representing (1) the number of initial task objects for each task, and (2) the number of triggered task objects for each task object.
The PrT net is presented in six disjoint parts describing the functions of the agenda mechanism. The parts are concerned with (1) the selection of a task object instance from the agenda to render it current, and the initialization of initial items in the case of selecting a waiting task instance, (2) the termination of a current task instance to render it passive, (3) the handling of a current decision instance, (4) the switch from a current task object instance to its active supertask instance, (5) the abortion of a current task object instance, and (6) the abortion of the last subtask object instance of a current task instance.

Although the presentation is divided into six distinct parts, the various PrT nets should still be regarded as part of one undivisible PrT net. Places with identical names in different figures refer to one and the same place.

9.2.2.1.2 ACTIVATION OF A TASK INSTANCE ON THE AGENDA

The agenda can be thought of as a blackboard containing task and decision instances, which can be taken into consideration. In terms of the PrT net of figure 9.3, the agenda contains task object instances which are either waiting or active. Waiting instances are subject to initiation, whereas active instances have been initiated already, but have not been completed.

Two types of task object instances can be selected from the agenda: (1) waiting task instances without supertask, and (2) waiting or active task object instances part of the current task instance. The first case is modelled as the transition "start" in figure 9.3, the second in the transitions "select waiting" and "select active".

The transitions "start" and "select waiting" result in a task object instance in the place task object to be initiated; the instance is said to be in the initiation stage. While task instances are being initiated, task object instances for all initial items of these task instances must be created and introduced in the place waiting.

Three counters for each newly created task object instance must also be initialized to zero: (1) the number of subtask object instances added to the agenda, (2) the number of passive subtask object instances, and (3) the number of triggered task object instances. The initiation of task instances is handled in the transitions "initiate task" and "put subtask object on agenda".

Selected task instances are rendered current after the initiation stage by the transition "activate task". Full history is stored in the places realized, initiated and decomposed.
Figure 9.3 The PrT net for the activation of task instances on the agenda

In table 9.1 we summarize all places defined for this part of the PrT net, and the meaning of the associated dynamic predicates.

The initial marking of the PrT net is explained as follows. The tuple \(<0>\) can be found in the place \(\text{waiting}\), and the tuple \(<0,0>\) in the places \(\#\text{put on agenda}, \#\text{passive}\) and \(\#\text{triggered}\). Furthermore, the place \(\text{new_inst_number}\) contains the tuple \(<1>\), and the place \(\text{class_of}\) the tuple \(<0,\text{overall_task}>\).
Table 9.1 Legend of the PrT-net for the activation of task instances

<table>
<thead>
<tr>
<th>Place Name</th>
<th>Tuple Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>waiting</td>
<td>s&lt;n&gt; task object instance n is waiting</td>
</tr>
<tr>
<td>active</td>
<td>s&lt;n&gt; task object instance n is active</td>
</tr>
<tr>
<td>current</td>
<td>s&lt;n&gt; task object instances part of task instance n can be selected</td>
</tr>
<tr>
<td>taskobject to be initiated</td>
<td>s&lt;n&gt; task object instance n has been selected and has to be initiated</td>
</tr>
<tr>
<td>realized</td>
<td>s&lt;n,r&gt; task object instance n has initiated or triggered task object instance r</td>
</tr>
<tr>
<td>initiated</td>
<td>s&lt;n,p&gt; an instance of task object p has been initiated by task object instance n</td>
</tr>
<tr>
<td>decomposed</td>
<td>s&lt;n,p&gt; task object instance n is part of task instance r</td>
</tr>
<tr>
<td>class of</td>
<td>s&lt;n,o&gt; task object instance n is an instance of task object o</td>
</tr>
<tr>
<td># to be put on agenda</td>
<td>s&lt;n,p&gt; still q initial task instances must be put on the agenda by task instance n</td>
</tr>
<tr>
<td># put on agenda</td>
<td>s&lt;n,q&gt; task object instances have been put on the agenda as part of instance n</td>
</tr>
<tr>
<td># passive</td>
<td>s&lt;n,p&gt; task object instances have become passive as part of task instance n</td>
</tr>
<tr>
<td># triggered</td>
<td>s&lt;n,q&gt; task object instances have been triggered by task object instance n</td>
</tr>
<tr>
<td>new_inst_number</td>
<td>s&lt;n&gt; the n-th instance can be created</td>
</tr>
</tbody>
</table>

9.2.2.1.3 TERMINATION OF THE CURRENT TASK INSTANCE

A current task instance can only be terminated if no waiting or active subtask object instances are left. In PrT net terminology, task instances can terminate if they are current, and if the number of passive subtask object instances equals the number of subtask object instances which have been put on the agenda. The corresponding PrT net can be found in figure 9.4.

Basic task instances are particularly interesting because the condition for termination is true immediately after they have become current, which is evident when we combine figures 9.3 and 9.4. In fact, basic tasks are considered as black boxes as far as their dynamic handling is concerned. The way in which basic tasks are carried out is considered as irrelevant to the agenda mechanism. We will discuss the way basic tasks are performed within the basic task execution mechanism in section 9.2.2.3.

The transition “terminate task” in figure 9.4 renders a task instance passive, implying that it is made part of the places history and last. We refer to table 9.2 for an explanation of the meaning of these places.

If a task instance is part of a supertask instance (see the place decomposed), the termination of the task instance could lead to the inclusion of new task object instances in the agenda. This is checked in the transition “define consequences”. The transition “trigger next task object” creates new task object instances for trigger relations, and adds these to the agenda. Similarly to figure 9.3, the places realized, initiated and decomposed are updated and the three counters #put on agenda, #passive and #triggered are initialized to zero for the newly created task object instances. After completing all triggers, the transition “activate supertask” renders the supertask instance current.
Figure 9.4 The PrT-net for the termination of task instances

Table 9.2 specifies the places part of figure 9.4 omitted in figure 9.3.

<table>
<thead>
<tr>
<th>Place Name</th>
<th>Tuple</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>task terminated</td>
<td>&lt;n&gt;</td>
<td>Task instance n has been terminated</td>
</tr>
<tr>
<td>k to be triggered</td>
<td>&lt;n, p&gt;</td>
<td>Still q task object instances have to be triggered by task instance n</td>
</tr>
<tr>
<td>last history</td>
<td>&lt;n&gt;</td>
<td>Task object instance n terminated most recently</td>
</tr>
<tr>
<td></td>
<td>&lt;n, r&gt;</td>
<td>Task object instance n terminated directly before task object instance r</td>
</tr>
</tbody>
</table>

Table 9.2 Legend of the PrT-net for the termination of task instances
To keep track of the history, the initial marking was expanded by an artificial task triggering the overall task. For the initial marking this implies that there is a tuple 
\(-1\) in the place last, a tuple 
\(-1,0\) in the places put on agenda and passive, a tuple 
\(-1,1\) in the place triggered, a tuple 
\(-1,0\) in the places initiated and realized.

9.2.2.1.4 THE HANDLING OF A DECISION INSTANCE

The third part of the PrT net is related to decisions (see figure 9.5), which are interpreted as choices of one of their possible outcomes. The set of possible outcomes \(O(d)\) of a decision \(d\) is defined as \(O(d) = \{l \in L | trigger(d, o) \land label(o) = l\} \cup \{"nothing"\}\).

![Diagram of PrT-net for decision making](image)

Figure 9.5 The PrT-net of making a decision instance

The outcome "nothing" should be interpreted as the conclusion that none of the tasks and decisions possibly resulting from the decision are considered as appropriate. As a result, the supertask instance is re-activated, implying that it moves from active to current.
Decision instances become passive on completion, like task instances becoming passive after termination (i.e. an update of the places last and history). If the outcome of the decision instance points to a task or decision to be initiated, a new task object instance is created in the transition "next is decision or task". This transition results in a task object instance in the place taskobject to be initiated. Table 9.3 summarizes the meaning of the places part of figure 9.5, omitted in figures 9.3 and 9.4.

<table>
<thead>
<tr>
<th>place name</th>
<th>tuple</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>decision made</td>
<td>&lt;n,a&gt;</td>
<td>decision instance n has been made with outcome a</td>
</tr>
</tbody>
</table>

Table 9.3 Legend of the PrT-net of making a decision instance

9.2.2.1.5 THE SWITCH FROM THE CURRENT INSTANCE TO ITS ACTIVE SUPERTASK INSTANCE

Various task object instances can be active at any given moment, but only one instance can be current. To change the current instance, it must be exited first of all, which can be done by returning to its active supertask instance or, in case of a task instance, by activating one of its subtask instances. The latter option has already been presented in figure 9.2. The return to the active supertask instance is shown in figure 9.6.

![Figure 9.6 The PrT-net of switching to the active supertask instance](image)

9.2.2.1.6 ABORT THE CURRENT TASK OBJECT INSTANCE

Only current task object instances can be aborted. Aborts are conceptually related to decision making under uncertain conditions. Certain circumstances in tasks or decisions, often identified by checking verification rules or firing decision rules, can prohibit the further execution of task and decision instances and necessitate an abort. Aborted task object instances enter the abort state. They return to the
waiting state by a mechanism to be explained later; as if no activity had taken place after their original initialization. As soon as the aborted task object instance has become waiting by this mechanism, its supertask becomes current. These actions have been modelled in the PrT net in figure 9.7.

![PrT-net for the abortion of the current task object](image)

Figure 9.7 PrT-net for the abortion of the current task object

Newly introduced places are explained in table 9.4.

<table>
<thead>
<tr>
<th>place name</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>overall abort</td>
<td>task object instance n is to be aborted</td>
</tr>
<tr>
<td>task object aborted</td>
<td>task object instance n is in the abort status</td>
</tr>
</tbody>
</table>

Table 9.4 Legend of the PrT-net for the abortion of the current task object

The mechanism of changing the state of a task object instance from abort to waiting is described in figures 9.8 to 9.12. The crucial point in understanding this mechanism is the influence of parallel task instances on the history. While the aborted task object instance was active, various task object instances belonging to different supertasks may have been executed and terminated. Because various parallel task instances can act on the same model, the only safe abort, is to abort all task instances and decision instances having been performed during the active period of the task object instance to be aborted. This situation has been modelled in figure 9.8.
Figure 9.8 shows that as long as there is a task object instance in the abort state with \#pA ssive \neq 0, the last task object instance in the history is transferred to the abort state and \#pA ssive of its supertask is decreased by one.

![Figure 9.8 PrT net for aborting the last task object instance in the history](image)

Aborted task object instances may have initiated active task object instances, either by trigger or decomposition relations. Because the initiated instance results from a task object instance to be aborted, the active instance itself requires also to be aborted. This is handled by the transition shown in figure 9.9.

![Figure 9.9 PrT net for aborting an active instance realized by an aborted instance](image)
Aborted task object instances can be related to active task object instances as shown above, but they can also have triggered waiting task object instances, or have initiated their initial items still waiting. These instances also need to be removed. Triggered instances in the waiting state are removed by the transition shown in Figure 9.10.

![Figure 9.10 PrT net for the removal of a waiting instance triggered by an aborted instance](image)

Waiting instances initiated by task object instances to be aborted are removed by the transition shown in Figure 9.11.

![Figure 9.11 PrT net for the removal of a waiting instance initiated by an aborted instance](image)
We must finally consider the return of aborted instances to the waiting state. This transition is only legal if the three counters of the aborted instance are zero and all relevant history has consequently been removed. This aspect of the abortion mechanism is shown in figure 9.12.

![Diagram](image)

Figure 9.12 PrT net for changing the abort state of an aborted instance into the waiting state

9.2.2.1.7 ABORT THE LAST SUBTASKOBJECT INSTANCES OF CURRENT TASK OBJECT INSTANCES

In order to remain in the current task instance, and have the opportunity of aborting the last task object instance performed in the task, the PrT net in figure 9.13 was created.

This mechanism can only be applied if \( \# \text{passive} \neq 0 \), which is checked in the transition "abort last subtask". If there are relevant subtask object instances to be aborted, the other three transitions of the PrT net will "walk" back in history until a task object instance has been aborted which is part of the current task instance. So, "walking" back in history implies forgetting history. Actually, the mechanism should be interpreted as providing an opportunity for the deletion of specific activities carried out so far. The newly introduced places part of the PrT net of figure 9.13 are explained in figure 9.5.

<table>
<thead>
<tr>
<th>place name</th>
<th>tuple</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>subtask to be aborted</td>
<td>( e, n )</td>
<td>the last subtask of task instance ( n ) is to be aborted</td>
</tr>
<tr>
<td>last to be removed for sub</td>
<td>( e, n )</td>
<td>task object instance ( n ) has to be aborted because the last subtask of task instance ( n ) is to be aborted</td>
</tr>
</tbody>
</table>

Table 9.5 Legend for the PrT net for the abortion of the last subtask object instance

198
This completes the description of the agenda functionality. The other components of the interpretation mechanism in the MSS environment depend on the agenda mechanism because they are initiated by it.

9.2.2.2 THE BASIC PROCEDURE EXECUTION MECHANISM

The basic procedure execution mechanism is started when non-decomposable current task instances terminate (see also figure 9.4, the transition "terminate task"). The basic procedure execution mechanism supports the execution of procedures related to basic tasks (see section 3.5).

It is important to the MSS environment's interpretation mechanism that the type of basic procedures to be used in the support of basic tasks is predefined. A meta-system architecture becomes feasible only by using predefined procedures for basic tasks.
We have identified nine basic procedures in this thesis, constituting the minimal set required for the envisioned MSS environment. These basic procedures are defined informally below. A formal definition of their intended behaviour can be found in Wijers et al. (1990). We have defined specific syntax rules for some basic procedures, in addition to those defined in chapter 3. They will be presented as part of the explanation of the specific procedure.

- **create-concept** and **create-association**

The objective of these two basic procedures is to add an instance of a concept or association to the existing application model. When a new instance of an association is created, the allowed instances for a specific role are limited to the concept instances within the input information place instances with that role as intended (or without any intended roles). The task scope of a create-concept procedure contains one concept, and the task scope of a create-association procedure contains one association. The task scope defines the concept or association for which an instance must be created.


Selections are natural to modelling (also see the results of chapters 5 to 7). Select procedures can select concept instances from the application model or from information place instances, and store their results in other information place instances. The task scope of a select procedure contains one concept defining the type of instances to be selected.

Four types of select procedures have been distinguished, relying on two characteristics: the first concerns the difference between the **select-one** and the **select-all**. A select-one selects only one "appropriate" instance, while a select-all selects all. The second characteristic considers the distinction between the **select-by-context** and the **select-by-set**, and defines what we mean by "appropriate" (see below).

- **select-by-context**

The select-by-context is best explained by an example. Suppose one wished to select a box part of a specific JSD structure diagram (see figure 3.2). This can be realized by calling a select-one-by-context procedure (for boxes) with an information place containing the specific JSD structure diagram as input. This JSD structure diagram is then considered to be the context of the select procedure. The context and the
concept to be selected should have a direct association in the concept structure.

- select-by-set

A select-by-set selects instances from input information places. The type of the information places related to the select-by-set should be equal to the concepts to be selected by the select procedure. A select-by-set without incoming information places is equivalent to a select-by-context without incoming information places; both select instances from a concept in the application model.

Select procedures can have more information places as input. In that case the intersection of the contents of these places is considered as input.

- delete-concept and delete-association

A delete procedure removes a concept or association instance from the existing application model. The task scope of the corresponding task defines the type of instance to be deleted. Furthermore, the instance to be deleted is primarily determined by incoming information places. For example, a box is selected from a JSD structure diagram and stored in an information place. This information place is then input to a delete-concept procedure which deletes the box from the application model. The selected box is the only one which can be deleted by that delete procedure. If there are no input information places, every instance of the concept in the application model may be deleted. Information places which are input to a delete-association procedure restrict the set of association instances that can be deleted to those related to the concept instances in the input information places.

- classify-concept

Classify procedures have been defined to further specialize an existing instance in the application model. The task scope determines the subtypes to which the instance may be specialized.

9.2.2.3 THE INFORMATION PLACE CREATION MECHANISM

The information place creation mechanism is responsible for the creation of new instances of information places. Information place instances are always part of exactly one task instance.
When a task instance is activated from the agenda, the information place creation mechanism looks for the information places needed within this task, and creates instances for these places accordingly. An explicit link between task and information place instances is also established. The information place creation mechanism is initiated in the transition "activate task" (see figure 9.3).

9.2.2.4 THE INFORMATION PLACE ADAPTATION MECHANISM

The information place adaptation mechanism is activated after the termination of task instances (immediately after the basic procedure execution mechanism). When task instances terminate, the identity of the concept and association instances having been created, deleted or selected is known, as well as the type of task instance terminated. The contents of the input and output information place instances can be adapted accordingly. The following rules are applied in the information place adaptation mechanism (see Wijers et al. (1990) for a formal definition):

- The contents of output information places of tasks not implemented as select procedures, is expanded by the concept instances created during task execution, if occurring in the population of the roles specified for those information places.

- Concept instances removed from the application model must no longer occur in information place instances.

- The concept instances selected by select procedures are added to the task's output information places.

- Consuming information places means that for all tasks, except the select-one-by-set, the contents of these information places are removed. An additional exception applies for information places direct input to delete-concept and classify-concept procedures, in case the type of these places equals the procedure's task scope (see below).

- A select-one-by-set procedure selects one element from the intersection of the contents of its direct input information places, and adds this element to its output information places. The selected element is removed from the information places consumed by the task.

- If a delete-concept procedure has direct input information places of the correct type, the delete-concept procedure remove an instance from the intersection of these information places and from the application model.
• If a classify-concept procedure has direct input information places of the correct type, the classify-concept procedure removes an instance from the intersection of these information places and creates a subtype instance for the specific instance.

9.2.2.5 THE HISTORY ADAPTATION MECHANISM

The history adaptation mechanism is part of the abort facility presented in sections 9.2.2.1.6 and 9.2.2.1.7, and is initiated by the transition "make aborted waiting" in figure 9.12.

As part of the abort facility, task object instances undergo state changes from current, active or passive to waiting. These state changes influence the contents of the application model and information place instances. The consequences are as follows:

• all information place instances part of aborted task instances are deleted,
• the concept and association instances created by aborted task instances are removed from the application model,
• the concept and association instances deleted by aborted task instances are added to the application model,
• the concept instances added to output information place instances by the aborted task instances are removed from these place instances,
• the concept instances consumed from input information place instances by the aborted task instances are added to these place instances, and finally
• various other facts part of the process history concerned with manipulations performed by the aborted task instances are removed.

9.2.2.6 THE VERIFICATION AND DERIVATION MECHANISM

Verification rules are specified in first order predicate logic (see section 3.3). Although this language is powerful in defining rules, an efficient resolution principle is not available. Practical implementations therefore apply resolution principles based on a subset of the first order predicate logic, the Horn clauses (see Minker (1988) and Clocksin and Mellish (1987)).

We shall restrict the verification rules in the MSS environment to Horn clauses, for implementation reasons. Horn clauses will be used both for the verification of the application model and the derivation of auxiliary model structures. Derivation rules compare to auxiliary predicates, verification rules to properties as used in chapter 3. Both type of rules are in the meta-modelling technique represented in first order predicate logic.
The verification and derivation mechanism is started when (1) waiting or active task instances are selected from the agenda, and (2) task instances are terminated. Under the first condition, only the a-priori verification rules of task instances are verified, in the second, both the a-priori and a-posteriori ones. Violation of the verification rules prohibits task instances from becoming current or passive. It should be remembered that a-priori rules are inherited by subtasks from their supertasks. If a verification rule requires an auxiliary predicate the corresponding derivation rules will be applied.

9.2.2.7 THE DECISION MAKING MECHANISM

The decision making mechanism supports the handling of decision instances. It has been based on a model for heuristic production rules, described in Shortliffe and Buchanan (1984). By firing the decision rules belonging to the current decision instance, the mechanism produces certainty factors for the possible outcomes, the most likely one being reported.

Decision rules are also used in more formal situations in which a certain choice has to be enforced. In such case, the condition part of a decision rule consists only of automatically verifiable conditions (see section 3.5.4) and the certainty factors of the action part are restricted to the values -1 (impossible) and 1 (strictly necessary). These formal decision rules combined with the use of information places and basic procedures offer sufficient expressive power to perform completely structured parts of a modelling process automatically.

9.3 PROTOTYPE OF AN MSS ENVIRONMENT

We shall now present two prototypes developed to demonstrate the technical feasibility of the meta-model editor and the interpretation mechanism. Prolog, more specifically Turbo Prolog (see Borland (1988)), was used in the implementation of the prototypes.

According to Bratko (1989) Prolog has strong abilities for: structuring of complex data, pattern matching, symbolic manipulations, implementing inference mechanisms, and knowledge representation formalisms. Prolog programs, on the other hand, are limited by their relative inefficiency and, in most cases, lack of sophisticated user interaction facilities. However, we have considered these aspects less important in the development of the prototypes.

Turbo Prolog is not a pure Prolog representative. It is however a language with many standard Prolog properties and with the benefits mentioned above.
Furthermore it is faster (see Burkholder et al. (1987)) and offers various easy-to-use predicates for windows and menus in PC environments.

We will first discuss META, the prototype meta-model editor, and then SHELL, the prototype interpreting correct meta-models set up by META.

9.3.1 IMPLEMENTATION OF THE META-MODEL EDITOR

META consists of a task structure editor, a concept structure editor, and a consistency checker. It also has standard facilities for loading and saving meta-models (see figure 9.14).

![META tool menu](image)

ESC - Quit the meta tool - Use arrow keys to select and RETURN to start

Figure 9.14 Main menu of META

9.3.1.1 THE META DATABASE

The META modules are grouped around a Prolog database, containing facts about a meta-model under construction. The predicates used for representing a meta-model in the database are shown in table 9.6, together with their corresponding meanings. The predicates have been deduced directly from the sets, relations and functions found in the formal definition of the metamodeling technique in chapter 3.
The table indicates that two sections of a metamodel are not defined: META does not support the specification of verification and decision rules, mainly because the two mechanisms for interpreting these rules have not been implemented in SHELL (see section 9.3.2), and META's reason for existence is an easy specification and verification of metamodels serving as input for SHELL.

<table>
<thead>
<tr>
<th>task(taskobjectid, taskname)</th>
</tr>
</thead>
<tbody>
<tr>
<td>taskobjectid identifies an occurrence of the class of tasks with a name equal to taskname</td>
</tr>
<tr>
<td>decision(taskobjectid, decisionname)</td>
</tr>
<tr>
<td>taskobjectid identifies an occurrence of the class of decisions with a name equal to decisionname</td>
</tr>
<tr>
<td>trigger(taskobjectid, taskobjectid)</td>
</tr>
<tr>
<td>an instance of the task or decision identified by taskobjectid triggers termination of new instance of the task or decision identified by taskobjectid</td>
</tr>
<tr>
<td>pers_off(taskobjectid, taskname)</td>
</tr>
<tr>
<td>the task or decision identified by taskobjectid is part of the decomposition of each task with name taskname</td>
</tr>
<tr>
<td>inst(taskobjectid, taskname)</td>
</tr>
<tr>
<td>a new instance of the task or decision identified by taskobjectid is put onto the agenda when an instance of a task with name taskname is activated</td>
</tr>
<tr>
<td>info_place(p laceid, p lacename, mode lobjectname)</td>
</tr>
<tr>
<td>placeid identifies an occurrence of the class of information places with a name equal to placename</td>
</tr>
<tr>
<td>info_place(p laceid, p lacename, mode lobjectname)</td>
</tr>
<tr>
<td>instances of information places with name placename may contain instances of the concept mode lobjectname</td>
</tr>
<tr>
<td>supertask(p laceid, taskname)</td>
</tr>
<tr>
<td>the information place identified by placeid is part of the decomposition of each task with name taskname</td>
</tr>
<tr>
<td>local manipulation(taskobjectid, placeid, manipulate)</td>
</tr>
<tr>
<td>an instance of the task identified by taskobjectid will locally manipulate an instance of the information place identified by placeid according to the type of manipulation as defined in manipulate</td>
</tr>
<tr>
<td>global manipulation(taskobjectid, placeid, manipulate)</td>
</tr>
<tr>
<td>an instance of the task identified by taskobjectid will globally manipulate an instance of an information place with name placename according to the type of manipulation as defined in manipulate</td>
</tr>
<tr>
<td>local extended role(taskobjectid, placeid, role name, role name)</td>
</tr>
<tr>
<td>the role identified by the combination of association mode lobjectname and role rolename restricts the set of concept instances in an instance of placeid that may be locally manipulated by an instance of taskobjectid (see further chapter 3)</td>
</tr>
<tr>
<td>global extended role(taskobjectid, placeid, role name, role name, mode lobjectname, ro lename)</td>
</tr>
<tr>
<td>the role identified by the combination of association mode lobjectname and role rolename restricts the set of concept instances in an instance of placename that may be globally manipulated by an instance of taskobjectid (see further chapter 3)</td>
</tr>
<tr>
<td>task scope(taskname, mode lobjectname)</td>
</tr>
<tr>
<td>each instance of each task with name taskname may manipulate instances of the concept or association mode lobjectname</td>
</tr>
<tr>
<td>basic procedure(taskname, basic type, mode lobjectname)</td>
</tr>
<tr>
<td>0 each instance of each task with name taskname will perform the basic procedure basic type with as parameter the concept or association mode lobjectname</td>
</tr>
<tr>
<td>concept(mode lobjectname)</td>
</tr>
<tr>
<td>the name mode lobjectname is used to denote a concept in the concept structure association(mode lobjectname)</td>
</tr>
<tr>
<td>the name mode lobjectname is used to denote an association in the concept structure role(mode lobjectname1, ro lename, mode lobjectname2, ro lename)</td>
</tr>
<tr>
<td>the role with rolename rolename and with name rolename of association mode lobjectname1 is played by the concept with as name mode lobjectname2 subtype(mode lobjectname1, mode lobjectname2)</td>
</tr>
<tr>
<td>the concept mode lobjectname1 is a subtype of the concept mode lobjectname2 objectification(mode lobjectname1, mode lobjectname2)</td>
</tr>
<tr>
<td>the association mode lobjectname1 is objectified into concept mode lobjectname2</td>
</tr>
</tbody>
</table>

Table 9.6 The predicates used for the Prolog database of META

The three major META modules will be described in more detail next. The features of the concept and task structure editors, and the consistency checker are explained in sections 9.3.1.2, 9.3.1.3 and 9.3.1.4, respectively. Other META modules include general Prolog code for windows, menus and list processing.

206
9.3.1.2 THE CONCEPT STRUCTURE EDITOR

![Diagram of the concept structure editor]

Figure 9.15 The concept structure editor

Figure 9.15 shows the user interface implemented in META. As mentioned in the introduction, we have avoided a major effort in implementing a graphical user interface because we preferred to concentrate on the tasks required and their functionalities within the editors. By specifying various meta-models (the results of the chapters 5 to 7) with the editors, and changing the required functionality according to our experience, we have arrived at the current META version. This prototyping approach led to the concept structure editor described below.

The concept and task structure editors have essentially been implemented with the following two principles in mind:

1. the user must be provided with a list of tasks he is expected to perform, and

2. the user must be shown a view of the underlying database relevant for the current task.
The above principles have been implemented in the concept structure editor by a menu displaying the available tasks in the top half of the screen, and by a few text windows in the bottom half. The user thus oversees part of the entire concept structure in the bottom half of the screen, namely the part related to the "current concept": its associations, subtypes (children), supertypes (parents), associated concepts, and possibly its corresponding objectified association are displayed. For example, in figure 9.3 we see the current concept "External flow" with two associations, one with "External party" and one with "Context". It also has two subtypes, "Stimulus" and "Response", and one supertype "Flow". The example is taken from figure 6.15.

We experienced that in the creation of a concept structure, the main activity consists of specifying the various associations, subtypes and supertypes of one concept after another. Because concepts are processed in succession, we have introduced the notion of the current concept. This implies that the activation of, for example, the task "Add association" only requires the selection or creation of one additional concept, and possibly the naming of the roles. Associations and specializations can be specified very efficiently in this manner.

The concept structure editor has following relevant tasks, whose functionality we will briefly describe.

- **Add concept:** A new concept is added to the concept structure. Its previous existence is checked, and if negative, it is added to the database and becomes the current concept.
- **Add association:** A new association is added to the current concept. The associated concept can be created during this task, or can be selected from the existing concept structure. An existence check is performed before the association is added to the database.
- **Add specialization:** A supertype or subtype for the current concept is defined. The supertype or subtype can be created or selected during the task. An existence check is performed before the specialization is added to the database.
- **Add objectification:** An association to be registered as being objectified into the current concept is selected. The task can only be performed if no other objectification exists.
- **Edit concept:** The name of the current concept is edited. The result must not be an empty string.
- **Select concept:** If a current concept exists, one of the associated concepts can be selected to become current. If no current concept exists, concepts may be selected from the entire concept structure to become current.
- **Delete concept:** The current concept is removed from the concept structure. Its associations, subtype relations, supertype relations and possibly its objectified association are also removed. If the concept was used in the task structure editor for the definition of task scopes, the type of information places, or parameters of basic procedures (see table 9.1), these facts are also removed.
- **Delete association:** One of the associations related to the current concept is removed. In a similar way as was done in the task "Delete concept", possible consequences for task scopes, basic procedures and intended roles are also handled.
- **Delete specialization:** A specialization relationship of the current concept is removed.
- **Delete objectification:** The fact that the current concept is an objectification of a given association is removed from the database.
- **Unselect concept:** The current concept is unselected. This task is useful for switching to entirely different parts of the concept structure.
The task structure editor has been developed by applying principles similar to those used for the concept structure editor, and we have thus again concentrated on the identification of the relevant tasks part of the editor, and on the database view presented to the user.

The task structure editor always handles a current task, for which the system shows:

- its decomposable subtasks,
- its basic subtasks,
- its decisions,
- its initial subtasks or subdecisions,
- the triggers between its subtasks and subdecisions,
- its information places,
- the local manipulations between subtasks and subplaces, and
- the global manipulations between subtasks and places higher in the task hierarchy.
For example, in figure 9.16 we find the current task "Orientation" with four decomposable subtasks: "Define organization and its primary process", "Determine external parties", "Explore corresponding external relationships", and "Specify organizational structure". The example is taken from figure 6.5. Table 9.6 shows that tasks, decisions and information places have unique identification numbers, also shown in figure 9.16. Figure 9.16 also demonstrates that the task "Define organization and its primary process" is the initial subtask. Triggers have been defined between the tasks "5", "6", "7" and "8". Three information places, "Known primary processes", "Organization to be analyzed", and "Known external parties" have been identified. The results of the task "Define organization and its primary process" are stored in the information places "Known primary processes" and "Organization to be analyzed", the results of "Determine external parties" in "Known external parties".

Many tasks have been identified in the task structure editor, because also integration aspects between a task and concept structure (see section 3.5) are handled by this editor (except verification and decision rules). The identified tasks have therefore been collected in closely related groups corresponding to the menu items on the left hand side of figure 9.16. All menu items, except "Decompose task", have submenu. Figure 9.16 shows the "Edit decision" submenu. The groups of closely related tasks are discussed briefly below.

**Edit task:** New subtasks can be added to the current task, possibly using existing names. Existing subtasks can be edited or deleted. Deletion is a complex task because all consequences for the decomposition hierarchy, triggers, initials, task scopes, information place manipulations, and basic procedures must be taken into account.

**Edit decision:** New decisions can be added to the current task, possibly using existing names. Existing subdecisions can be edited or deleted. Decision deletion implies the deletion of their triggers, and of the fact whether it is an initial decision.

**Edit initial:** Existing subtasks and subdecisions can be marked as initial items, or such facts can be removed.

**Edit trigger:** Triggers can be defined between two existing subtasks/subdecisions. Triggers can also be deleted.

**Edit information place:** New information places can be added to the current task, possibly using existing names. Existing subplaces can be edited or removed. Deletion implies deletion of manipulations and place types. Types corresponding to subplaces can be specified (see table 9.1).

**Edit local manipulation:** Between existing subtasks and subplaces, local manipulations of the type "output", "consume" or "refer" can be defined. Intended roles may be added to local manipulations. Existing local manipulations can also be removed.

**Edit global manipulation:** A function similar to "Edit local manipulation" is available, differing in the sense that the information places must have been defined higher in the task hierarchy. Concepts and associations can be added to or removed from the scope of the current task. A different view on the database has been defined for this task group, so that the task scope is presented to the user (see figure 9.17). Both the concepts and the associations part of the scope of the current task are shown. Besides simple additions and removals, two specific tasks have been defined to simplify the specification of task scopes. In the task "Define task scope by inheritance", the intersection of the concepts and associations of the supertasks of the current task is defined as the scope of the current task. In the task "Define task scope by deduction", the union of the concepts and associations of the subtasks of the current task is defined as the scope of the current task.
9.3.4 THE CONSISTENCY CHECKER

The consistency checker is the last META module to be discussed. Most properties of properly designed meta-models have been covered in chapter 3 (except those identified for basic procedures in section 9.2.2.3). The consistency checker verifies these properties for meta-models set up by both editors.

The meta-modelling technique distinguishes between a-priori and a-posteriori verification rules, and META obviously does likewise. Some properties of chapter 3 cannot be violated due to the chosen implementation of the task structure editor, and can therefore be considered as a-priori rules, such as property 3.4.2.4, demanding that triggers may only exist between two task objects of the same supertask. Other properties must be verified explicitly, e.g. property 3.2.2.4 forbidding cycles in subtype networks. These properties can be regarded as a-
posteriori rules for the two editors. The consistency checker explicitly verifies only this type of properties. Table 9.7 specifies the checks performed by the consistency checker.

<table>
<thead>
<tr>
<th>Check for:</th>
<th>Warning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pater familias rule</td>
<td>Concept 4 has two pater familias, namely 4 and 4</td>
</tr>
<tr>
<td>Subtype cycles</td>
<td>Concept 4 and concept 4 are each other's ancestors</td>
</tr>
<tr>
<td>Objectified subtypes</td>
<td>Objectification 4 is a subtype of 4</td>
</tr>
<tr>
<td>Objectification cycles</td>
<td>Cyclic objectification structure exists for concept 4</td>
</tr>
<tr>
<td>Existence overall task</td>
<td>There is not exactly one overall task</td>
</tr>
<tr>
<td>Cycles in decomposition</td>
<td>Task 4 is involved in a cyclic decomposition</td>
</tr>
<tr>
<td>Number of subtaskobjected tasks</td>
<td>Task 4 has only one subtaskobject</td>
</tr>
<tr>
<td>Decisions connected to tasks</td>
<td>Decision 4 is not connected to a task</td>
</tr>
<tr>
<td>Inclusion of decisions in cyclic task structures</td>
<td>Task 4 is connected to itself without any decisions</td>
</tr>
<tr>
<td>Inclusion of tasks in cyclic task structures</td>
<td>Decision 4 is connected to itself without any tasks</td>
</tr>
<tr>
<td>Free decisions not part of a task</td>
<td>Decision 4 is not part of a task</td>
</tr>
<tr>
<td>Connectivity of initials</td>
<td>Initial 4 is reachable from initial 4</td>
</tr>
<tr>
<td>Reachability of non-initials</td>
<td>Taskobject 4 can not be reached by an initial</td>
</tr>
<tr>
<td>The existence of choices between the same task scopes</td>
<td>Decision 4 has two triggers to tasks both named 4</td>
</tr>
<tr>
<td>Existence of task scopes</td>
<td>Task 4 has no task scope</td>
</tr>
<tr>
<td>Subset constraints in task scopes</td>
<td>Model object 4 is not part of task 4</td>
</tr>
<tr>
<td>The task scope of the overall task</td>
<td>Model object 4 is not included in the overall task</td>
</tr>
<tr>
<td>The union constraint over task scopes</td>
<td>Task scope of 4 is not the union of its subtask scopes</td>
</tr>
<tr>
<td>The existence of supertypes in task scopes</td>
<td>Supertype 4 should be in task scope of 4</td>
</tr>
<tr>
<td>Objectifications in task scopes</td>
<td>Association 4 should occur in task scope of 4</td>
</tr>
<tr>
<td>Objectifications in task scopes</td>
<td>Concept 4 should occur in task scope of 4</td>
</tr>
<tr>
<td>Existence of basic procedures</td>
<td>Basic task 4 has no corresponding basic procedure</td>
</tr>
<tr>
<td>Possibility to manipulate the type of a basic procedure</td>
<td>Basic task 4 is not allowed to manipulate 4</td>
</tr>
<tr>
<td>Appropriateness of an output from a basic procedure</td>
<td>Basic task 4 cannot have output</td>
</tr>
<tr>
<td>Correct input places for a select by set procedure</td>
<td>Information place 4 cannot be input for basic task 4</td>
</tr>
<tr>
<td>Existence of information place types</td>
<td>Information place 4 has no corresponding type</td>
</tr>
<tr>
<td>Hierarchy of globally manipulated information places</td>
<td>Task 4 has incorrect global manipulation with place 4</td>
</tr>
<tr>
<td>Existence of two information places within one task</td>
<td>Information place 4 occurs twice in task 4</td>
</tr>
<tr>
<td>Correctness of global manipulations</td>
<td>Task 4 is not allowed to globally manipulate 4</td>
</tr>
<tr>
<td>Double global manipulations by ancestors and children</td>
<td>The intended role 4 for place 4 in task 4 is incorrect</td>
</tr>
<tr>
<td>Usage of information places</td>
<td>Information place 4 is manipulated twice by ancestors</td>
</tr>
<tr>
<td>Usage of information places</td>
<td>Information place 4 is not input for a task</td>
</tr>
<tr>
<td>Consume and refer relationships of a task</td>
<td>Task 4 both refers to and consumes information place 4</td>
</tr>
</tbody>
</table>

Table 9.7 The checks performed by the consistency checker

The verification rules have been implemented in Prolog in a form of identifying conditions violating these (the rule negation). The predicate inconsistent (warning) has been defined, and each verification rule has resulted in one clause for this predicate. For example, the check for the pater familias rule has been implemented as follows:

\[
\text{inconsistent(Warning)} :- \\
\quad \text{message("Checking pater familias rule"),} \\
\quad \text{subtype(C,)}, \\
\quad \text{pater_familias(CL,C),} \\
\quad \text{pater_familias(CZ,C),} \\
\quad \text{CL @ CZ,} \\
\quad \text{format(Warning,"Concept 4 has two pater_familias: 4 and 4",C,CL,CZ).} \\
\quad \text{4 find inconsistency} \\
\quad \text{4 feedback to meta-analyst} \\
\quad \text{4 find a subtype C} \\
\quad \text{4 find 1st pater familias of C} \\
\quad \text{4 find 2nd pater familias of C} \\
\quad \text{4 check difference between both} \\
\quad \text{4 construct warning message}
\]

The auxiliary predicate \text{pater_familias(concept1,concept2)} has been defined in the following way:

\[
\text{ancestor(C,C,):-} \\
\quad \text{concept(C).} \\
\quad \text{4 each C is an ancestor of itself} \\
\quad \text{4 at least if it is a concept}
\]

212
The properties and auxiliary predicates defined in chapter 3 have thus been translated into Prolog. Having established this transformation, inconsistencies are identified by the following Prolog statement:

\texttt{findall(\text{Error}, \text{inconsistent(\text{Error})}, \text{Errorlist})}

### 9.3.2 IMPLEMENTATION OF THE INTERPRETATION MECHANISM

SHELL, the prototype for the interpretation mechanism, includes the agenda, basic procedure execution, information place creation and adaptation, and history adaptation mechanisms (see section 9.2). The two others, the verification and derivation mechanism and the decision making mechanism, have not been implemented because their validities have been demonstrated by the META consistency checker and by the prototype described in Huizinga (1990), respectively. Besides the five mechanisms mentioned, SHELL offers facilities for loading meta-models, and for saving and loading modelling sessions, see figure 9.18.
9.3.2.1 THE SHELL DATABASES

The SHELL modules use two databases. The first one is for a meta-model loaded from file, the second registers all facts of the modelling session taking place. The first database is structured in the same way as the one used for META (see table 9.6). The predicates used in the second database are collected in table 9.8.

| Table 9.8 The predicates used for the Prolog modelling session database of SHELL |
|---------------------------------|---------------------------------|
| method(methodname)              | the metamodel of methodname is used during this modelling session |
| taskobject_instance(instanceId) | instanceId identifies a task object instance |
| class_of(instanceId, taskobjectId) | Task object instance or information place instance instanceId is an instance of the task object or information place taskobjectId |
| current(instanceId)             | task object instance instanceId is currently being performed |
| waiting(instanceId)             | task object instance instanceId is waiting to be selected |
| active(instanceId)              | task instance instanceId still has current, active or waiting subtask object instances |
| initiated(instanceId, taskobjectId) | task object instance instanceId has initiated an instance of task object taskobjectId |
| realized(instanceId, instanceId) | task object instance instanceId has initiated the task object instance instanceId |
| decomposed(instanceId1, instanceId2) | task object instance instanceId1 is part of task instance instanceId2 |
| last(instanceId)                | task object instance instanceId terminated most recently |
| history(instanceId1, instanceId2) | task object instance instanceId1 terminated directly before task object instance instanceId2 |
| mr_per_on_agenda(instanceId, x) | x task object instances have been put on the agenda as part of task object instance instanceId |
| mr_passive(instanceId, x)       | x task object instances have become passive as part of task object instance instanceId |
| mr_triggered(instanceId, x)     | x task object instances have been triggered by task object instance instanceId |
| concept_instance(instanceId, instanceName) | instanceId identifies a concept instance with name instanceName |
| association_instance(instanceId) | instanceId identifies an association instance |
| role(instanceId, x, instanceId2) | the x-th role of association instance instanceId1 is played by concept instance instanceId2 |
| subtype_instance(instanceId1, instanceId2, instanceId3) | concept instance instanceId1 is a subtype of concept instance instanceId2 |
| object_instance(instanceId, instanceId1, instanceId2) | association instance instanceId1 is objectified into concept instance instanceId2 |
| class_of(instanceId, mode_objectName) | mode_object instance instanceId is an instance of the mode_object with name mode_objectName |
| application_mode(instanceId)    | mode_object instance instanceId is part of the current application mode |
| info_place_instance(instanceId)  | instanceId identifies an information place instance |
| task_place(instanceId, instanceId2) | task instance instanceId1 contains information place instance instanceId2 |
| context(instanceId1, instanceId2) | information place instance instanceId1 contains mode_object instance instanceId2 |
| created(instanceId1, instanceId2) | task instance instanceId1 has created mode_object instance instanceId2 |
| deleted(instanceId1, instanceId2) | task instance instanceId1 has deleted mode_object instance instanceId2 |
| chosen(instanceId1, instanceId2) | task instance instanceId1 has selected mode_object instance instanceId2 |
| classified(instanceId1, instanceId2) | task instance instanceId1 has further classified mode_object instance instanceId2 |
| added(instanceId1, instanceId2, instanceId3) | task instance instanceId1 has added to information place instance instanceId2 the concept instance instanceId3 |
| consumed(instanceId1, instanceId2, instanceId3) | task instance instanceId1 has removed from information place instance instanceId2 the concept instance instanceId3 |
Most predicates, in fact, represent instantiations of the predicates defined in table 9.6. Other predicates indicate the current status of the agenda, such as current, waiting and active, or the current status of the application model and the information places, such as application_model and contents. Various predicates originate directly from the PrT nets of section 9.2.2.1.

9.3.2.2 THE SHELL AGENDA MECHANISM

As explained in section 9.2.2, the agenda mechanism is at the centre of the interpretation mechanism, initiating the six others.

The basic implementation principle

The agenda mechanism has been implemented by transforming the transitions in the formal PrT net into Prolog rules. For example, the transition "initiate task" in figure 9.3 has been translated into the following Prolog code:

```
transition(“initiate task”) :-
    object_to_be_initiated(N),
    class_of(N, T),
    task(T, Name),
    nr_of_initials(Thame, 0),
    retract(object_to_be_initiated(N)), 1,
    asserta(nr_to_be_put_on_agenda(N, 0)).
```

% this clause performs the transition “initiate task”
% if task instance N is in the state “to be initiated”, and
% the class of N is equal to T, and
% the name of T is Thame, and
% the number of initials of Thame is 0, then
% task instance N is removed from place “to be initiated”,
% and 0 task object instances have to be put on the agenda.

The transition "remove waiting triggered by aborted" of figure 9.10 results in:

```
transition(“remove waiting triggered by aborted”) :-
    object_aborted(N),
    nr_triggered(N, R),
    waiting(L),
    realised(N, L),
    initiated(N, K),
    class_of(L, K),
    nr_passives(L, T),
    nr_put_on_agenda(L, T),
    nr_triggered(L, T),
    T = 0,
    decomposed(L, Y),
    decomposed(N, Y),
    nr_put_on_agenda(Y, Z),
    Z1 = Z - 1,
    R1 = R - 1,
    retract(nr_triggered(N, R)),
    asserta(nr_triggered(N, R1)),
    retract(nr_put_on_agenda(L, T)),
    retract(nr_passives(L, T)),
    retract(nr_triggered(L, T)),
    retract(realised(N, L)),
    retract(initiated(N, K)),
    retract(class_of(L, K)),
    retract(task_object_instance(L)),
    retract(waiting(L)),
    retract(decomposed(L, Y)),
    retract(nr_put_on_agenda(Y, Z)),
    asserta(nr_put_on_agenda(Y, Z1)).
```

% perform the transition “remove waiting ...”
% if task instance N is in the state “abort”, and
% N has triggered R instances, and
% task object instance L is waiting, and
% L is realized by N, and
% N initiated an instance of class K with
% L belonging to this class K, and
% L has T passive subtask object instances, and
% L has T subtask object instances put on the agenda
% L has triggered T task object instances, with
% T equal to 0,
% furthermore L is part of Y
% and N is part of Y with
% Y having Z subtask object instances on the agenda,
% THEN Z1 is Z - 1, and
% R1 is R - 1, and
% N has triggered only R1 task object instances,
% and
% all data about task object instance L is removed
% and
% Y has put only Z1 instances on the agenda.
The essence of the agenda mechanism is expressed in the simple Prolog rule:

\[
\text{agenda:-}
\begin{align*}
\text{repeat}, & \\
\text{transition}(T), & \\
T = \text{"end"}, & \\
\end{align*}
\]

4 the agenda is
4 a repeat cycle
4 in which in each cycle a transition is performed
4 until the transition "end" has been fired.

This rule implies that the goal "transition(T)" must be satisfied as long as T="end" fails, which is a consequence of the Prolog backtracking mechanism. The transition "end" fires if no other transition can fire, or the information engineer presses the ESCAPE key.

**User interface extensions**

The user interface implementation has demanded a few extensions of the basic principle described above. We have identified three occasions in the agenda mechanism where interaction with the information engineer is required:

1. Interaction when the agenda is in the neutral state

While the agenda is in what we call a neutral state, the information engineer should be able to select tasks and decisions from the agenda, to abort the current task, or to transfer to other parallel tasks. We have therefore introduced the extra transition "select", displaying the system status and the available actions. This user interface is shown in figure 9.19.

The user interface has been set up according to the principles already introduced in the modelling knowledge editor: the user is shown (1) the available tasks, and (2) a view of the current state of the database. The available tasks in SHELL are obviously determined by the task structure, the current agenda state, and the general features of the agenda mechanism. For example, in figure 9.19 the information engineer may:

- select the waiting task instance of the task "Explore corresponding external relationships" by selecting this task from the menu,
- quit the modelling session by pressing <ESCAPE>,
- abort the current task instance of the task "Orientation" by pressing <F7>,
- abort the last subtask object instance of the current task instance, being an instance of the task "Determine external parties" here, by pressing <F8>, and
- transfer to the parent task instance of the current task instance, being an instance of the task "The construction of an information architecture" here, by pressing <F10>.
Figure 9.19 The user interface of SHELL

The view of the current state of the database displayed to the information engineer depends on the current task instance, for which the following items are shown:

- the waiting subtask object instances
- the active subtask object instances
- the passive subtask object instances
- the parent task instances (up till the overall task instance)
- the contents of the input information place instances
- the concept instances created by the subtask instances
- the association instances created by the subtask instances

(2) Interaction at selection time

When a decision instance is selected by the information engineer, it becomes current and its available options are displayed. A choice can be made or the decision can be aborted. This type of interaction has been implemented as part of the transition "make a decision" (see figure 9.5).
(3) Interaction at termination time

When a current task instance terminates, a basic procedure for basic task instances, or a confirmation for the termination of non-basic task instances is initiated. This user interaction has been implemented in the basic task execution mechanism, initiated in the transition "terminate task" (see figure 9.4).

The agenda mechanism has been developed by prototyping. Various iterations have taken place between the PrT net of section 9.2.2.1 and the Prolog implementation. The user interface has also changed several times.

9.3.2.3 THE SHELL BASIC PROCEDURE EXECUTION MECHANISM

The basic procedure execution mechanism performs basic procedures for all basic task instances. It also supports the termination of non-basic task instances by asking for a confirmation, and subsequently starts the information place adaptation mechanism for these non-basic task instances. The main section of the basic task execution mechanism has been implemented by the predicate `perform_basic(task_instance, basic_procedure, model_object)`, defining that the basic procedure `basic_procedure` with parameter `model_object` should be performed for task instance `task_instance`. Some clauses for this predicate will be presented to illustrate the implementation. They support the basic procedures `create-concept` - and `select-one-by-set`.

```
perform_basic(N, "create concept", MObject) :-
    format(Message, "Fill in name of %" , N, MObject),
    message(Message),
    cursorform([7, 7]),
    lineinput(5, 2, 48, screencolor, framecolor, MObject, "", CName),
    cursorform([20, 20]),
    CName = "", 1,
    new_instance(0, M),
    asserta(concept_instance(M, CName)),
    asserta(class_of(T, MObject)),
    asserta(application_model(M)),
    asserta(created(M, N)),
    asserta(choice(1)),
    adapt_info_places.(M).
```

```
perform_basic(N, "select one by set", MObject) :-
    halfrow(1, M),
    PartNr = HalfRow - 3,
    inside(M, InsideFC),
    format(Message, "Select instance of % to be selected", MObject),
    message(Message),
    findall(Place, direct_input(Place, MObject, Places),
            findall(Inst, correct(Inst, MObject, Places), Instances),
            get_name_list(Instances, InstNames),
```

% perform create concept procedure
% bottom line message construction
% show bottom line message
% show cursor
% ask for concept name using fieldeditor
% hide cursor
% check on not empty string
% create unique identifier for instance
% add new concept instance to database
% store its concept class
% add instance to application model
% store rel. between task and concept
% create return-code for agenda mechanism
% start information place adaptation mechanism
% handle exception: empty string or ESC
% give warning of exception
% create return-code for agenda mechanism
% perform select one by set procedure
% receive screen height
% take part of screen height
% receive screen width
% construct bottom-line message
% show bottom-line message
% find direct input information places
% find intersection of place contents
% find names of instances in intersection
Two predicates, *direct_input* and *correct*, have been used in the basic procedure *select-one-by-set*. They guide the selection of the relevant input information place instances, and the creation of the intersection of the contents of these input information place instances. The following code represents these predicates.

```prolog
direct_input(PI,II,C) :-
  input_instance(II,PI),
  class_of(PI,P),
  info_place(P,PHame),
  place_type(PHame,Type),
  ancestor(C,Type).

input_instance(II,PI) :-
  consume_instance(II,PI),
  refer_instance(II,PI).

consume_instance(II,PI) :-
  class_of(II,T),
  local_manipulation(T,P,"consume"),
  class_of(PI,P),
  decomposed(II,S),
  task_place(S,PI).

refer_instance(II,PI) :-
  class_of(II,T),
  local_manipulation(T,P,"refer"),
  class_of(PI,P),
  decomposed(II,S),
  task_place(S,PI).

direct_parent_instance(SI,II,P) :-
  decomposed(II,UI),
  decomposed(UI,S),
  task_place(S,PI),
  class_of(PI,P),
  info_place(K,P).
```

Sec. 9.3

PROTOTYPE OF AN MSS ENVIRONMENT

- show selection menu with names
- remove selection menu
- check on ESC (Choice=0): exception
- select chosen instance
- store rel. between task and concept
- create return-code for agenda mechanism
- start adapt information place mechanism
- handle exception: nothing to select
- give warning of exception
- create return-code for agenda mechanism

---

219
A large set of auxiliary predicates has effectively supported efficient coding of the basic procedures. The auxiliary predicates have been re-used in several basic procedures. The identification of the required auxiliary predicates has mainly been based on the formalization of basic procedures as described in Wijers et al. (1990).

9.3.2.4 THE SHELL INFORMATION PLACE CREATION MECHANISM

The information place creation mechanism is initiated in the transition "activate task". The mechanism creates instances for the information places part of the activated task instance. The mechanism has been implemented by the following clauses:

9.3.2.5 THE SHELL INFORMATION PLACE ADAPTATION MECHANISM

The information place adaptation mechanism is initiated in the basic task execution mechanism (see section 9.3.2.3). The implementation of this mechanism
Sec. 9.3  PROTOTYPE OF AN MSS ENVIRONMENT

supports the seven types of adaptations identified in section 9.2.2.3. The predicate adapt_infoplaces(task_instance) implements the mechanism. Some of the clauses for adapt_infoplaces are:

adapt_infoplaces(N) :-
    class_of(N,T),
    task(T,TName),
    basic_procedure(TName,"select one by set",C),
    consume_instance(N,PI),
    class_of(PI,P),
    info_place(P,PName),
    place_type(PName,C),
    chosen(N,C),
    retract(contents(PI,CI)),
    asserta(consumed(N,PI,CI)),
    fail.

adap_infoplaces(N) :-
    chosen(N,C),
    output_instance(N,PI),
    asserta(contents(PI,CI)),
    asserta(addicted(N,PI,CI)),
    fail.

adapt_infoplaces(N) :-
    deleted(N,CI),
    input_instance(N,PI),
    asserta(contents(PI,CI)),
    retract(contents(PI,CI)),
    asserta(consumed(N,PI,CI)),
    fail.

adapt_infoplaces(_) :-1.

By applying the fail predicate at the end of each clause, except the last one, all possible adaptations to information places are identified.

9.3.2.6 THE SHELL HISTORY ADAPTATION MECHANISM

The last mechanism implemented in SHELL, the history adaptation mechanism, follows from the possibility to abort a task object instance in the agenda mechanism. It is initiated by the transition "make aborted waiting". The mechanism was implemented as follows:

adapt_model_history(TI) :-
    remove_places(TI),
    remove_created(TI),
    remove_deleted(TI),
    retractall(chosen(TI,_)),
    retractall(classified(TI,_)),
    remove_added(TI),
    remove_consumed(TI).

The various predicates mentioned in this clause undertake the actions mentioned in the comment statements.
We have now completed the description of the prototype implementation of the meta-model editor and the interpretation mechanism. Both programs take up to 5000 lines of Prolog code, split up into 15 small modules.

9.4 APPLICATION OF THE PROTOTYPE MSS ENVIRONMENT

We shall now briefly review the experience gained with META and SHELL. As mentioned in the introduction, the prototype development concentrated on achieving automatic interpretation and execution of correct meta-models. The feasibility of this objective has been tested by simulating and replaying the actual modelling processes as observed in the three experiments.

The functionality of META has changed considerably during prototype construction. Features such as current task and current concept have been introduced after a few initial META runs, improving the functionality considerably. The restrictions remaining in META can only be overcome by introducing a graphical user interface, which we will not consider here.

META has been applied for the creation of various meta-models, particularly the ones of chapters 5 to 7. The ultimate meta-models as registered in META are extensions to the conceptual meta-models described in chapters 5 to 7. The main reason is that we had to decompose most tasks to the level of basic procedures as implemented in the basic procedure execution mechanism of SHELL. The meta-models set up with META are "implementations" of the conceptual meta-models resulting from the experiments.

SHELL has been applied using the meta-models set up by META. The agenda mechanism has been tested in several SHELL modelling sessions, as well as the functionality of the basic procedures, and the valid usage of information places. We conclude that SHELL is a prototype with the intended functionality, demonstrating the automatic interpretability of meta-models. Automated modelling support has been realized in the fields of process guidance, model construction, model verification and model derivation, see table 9.9.

Particular restrictions concerning SHELL can be found in its user interface. Potential user interface enhancements have been identified, the most important one being the implementation of a graphical user interface. Other user interface improvements can be achieved by hiding the making of certain formal decisions and the execution of tasks which can be performed automatically, for the information engineer.
Sec. 9.4 APPLICATION OF THE PROTOTYPE MSS ENVIRONMENT

<table>
<thead>
<tr>
<th>Process guidance</th>
<th>Model construction</th>
<th>Model verification</th>
<th>Model derivation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The agenda mechanism</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>The basic procedure execution mechanism</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>The information place creation mechanism</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>The information place adaptation mechanism</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>The history adaptation mechanism</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>The verification and derivation mechanism</td>
<td>x  x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>The decision making mechanism</td>
<td>x</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 9.9 The contribution of each mechanism to the required modelling support

Summarizing, the Prolog prototypes have demonstrated the viability of the MSS environment functionality, characterized by simple specification and verification of meta-models, and by automatic interpretation of these meta-models resulting in actual modelling support. Support is provided according to a way of working and modelling, formally described in a meta-model. And so, we conclude that the third hypothesis of our study (see section 2.10) cannot be rejected.

9.5 CONCLUSIONS AND FUTURE RESEARCH

In the last section of this thesis we return to our main objective related to improved modelling support for information engineers in information systems development.

Based on our findings we can conclude that a thorough and detailed understanding of the field of modelling to be supported is absolutely beneficial to the level of modelling support realizable. The studies performed with the experienced information planners have resulted in a very differentiated view of the approaches feasible to the construction of an information architecture. Adequate support for these information engineers implies individual support tuned to their way of working and modelling. In this chapter we have shown how to realize such support.

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. However, not all results have
to be transferable and the strategy applied by the information engineer can also
be considered as an individual responsibility. Therefore 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. In terms of the MSS environment we can state that the
application of the MSS environment at an organization-wide level will ask for
powerful meta-model management.

The meta-modelling technique has proven itself as an adequate technique for the
representation of a way of working and modelling. Constructed meta-models
however will probably be subjected to changes. Future research will have to look
at evolutionary development of modelling knowledge formalized in meta-models,
and thus to evolutionary MSSs. This might even imply that an information
engineer becomes self-responsible for changes to his MSS and underlying meta-
model. A hypothesis for future research is that the meta-modelling technique
offers sufficient means to an information engineer to change the meta-models
embedded in his MSS himself.

The acquisition approach has been directed at experienced practitioners in
specific established methods. We do not consider this approach to be equally
effective for the development of automated support tools for new methods as
no experienced practitioners are available for these methods. A useful application
of the meta-modelling technique and MSS-environment can be expected also for
these methods but another approach is needed for the construction of meta-
models describing parts of these innovative methods.

A final remark concerns the trade-off of between support and no support. In this
study we have focussed on the relationship between the level of modelling
support realizable and the available understanding of the field of modelling to
be supported, and we have demonstrated that a detailed understanding
contributes to the level of support achievable. Not every part of an information
systems development process however is equally profitable to be analysed to the
level of detail we have done in this thesis. We conclude by stating that future
application of the results achieved in this thesis may lead to insights in this
economical question.
APPENDIX A

A RESERVATION ORGANIZATION

The organization used in this case for experienced information engineers is a foundation called Dutch Reservations Centre (NRC), employing about 25 people. The organization is keen on knowing how to apply information technology in their business. The main activity is hotel room reservation for foreign clients, but theatre, congress, and tourist reservations are also offered.

The organization chart is shown in figure A.1, emphasizing the clear distinction between group and individual reservations.

![Organization Chart]

Figure A.1 NRC organization chart

The organization handles about 30,000 hotel reservations a year, mainly for weekend tourists and business people. Reservation services are free of charge for customers, but hotels and other accommodation suppliers pay a commission. The NRC acts as an intermediary between foreigners requiring accommodation in the Netherlands and suppliers who can make this available. More details about the organization can be found in the case scenario, documented in Wijers (1988b).

The main reason for selecting this foundation as an appropriate case for information engineers is the following: the organization is characterized by its complex primary process. It is an extremely dynamic organization handling a variety of events and processing large amounts of different types of data.
REFERENCES


BREUKER, J.A. AND B. WIELINGA, *Techniques for knowledge elicitation and analysis*, Report 1.5 Esprit Project 12, Department of Social Science Informatics, University of Amsterdam, Amsterdam, The Netherlands, 1984.


HOFSTEDE A.H.M. AND Th.P. van der Weide, Formalisation of Techniques: Chopping down the methodology jungle, Report 90-26, Department of Informatics, University of Nijmegen, Nijmegen, The Netherlands, 1990.


Winkels, R., J. Sandberg and J. Breuker, Coaching strategies and tactics of IHSs, Report UAM/EUROHELP/06, Department of social science informatics, University of Amsterdam, Amsterdam, The Netherlands, 1986.


AUTHOR INDEX

Aguero, U. 8, 227
Andersen, N.E. 1, 21, 227
Anjewierden, A. 65, 227
Antill, L. 2, 236
Assche, F.J.M. van 1, 233
Avison, D.E. 1, 2, 22, 23, 25, 227, 236
Aytom, P. 59, 62, 63, 236
Baker, G.J. 23, 232
Ballay, J.M. 7, 8, 19, 227
Bemelmans, T.M.A. 1, 2, 17, 23, 227
Benner, K.M. 4, 5, 227
Benyon, D. 3, 6, 14, 15, 227
Bhabuta, L. 23, 25, 232, 233
Black, W.J. 20, 234
Blake, R.R. 148, 227
Blank, J. 17, 23, 227
Blokdijk, A. 1, 227
Blokdijk, P. 1, 227
Bolesian 60, 227
Bommel, P. van 16, 227
Boom, J. 4, 231
Borland 204, 227
Bots, P.W.G. 6, 7, 17, 39, 180, 227
Bouzeghoub, M. 4, 227
Bratko, I. 204, 227
Bredeweg, B. 66, 233
Breuker, J.A. 59, 60, 61, 62, 64, 228, 236
Brinkkemper, S. 1, 13, 228
Bubenko jr., J.A. 1, 2, 3, 24, 25, 27, 228
Buchanan, B.G. 204, 234
Budgen, D. 4, 5, 228
Burgess, K. 2, 229
Burkholder, L. 205, 228
Burlon, R. 4, 233
Butler Cox 1, 3, 228
Cameron, J.R. 32, 42, 228
Cap Gemini Pandata 152, 228
Cardile, B. 4, 233
Carnap, R. 37, 228
Champion, R.E.M. 4, 231
Chand, D.R. 4, 236
Checkland, P.B. 2, 228
Chen, M. 3, 5, 26, 27, 32, 228
Chua, T.S. 4, 234
Clocksin, W.F. 203, 228
Cohen, B. 28, 229
Conti, M. 4, 233
Dasgupta, S. 8, 227
Davis, G.B. 1, 229
Davoodi, M. 66, 233
DCE 77, 229
DeMarco, T. 113, 117, 140, 229
Demetrovics, J. 3, 231
Dickson, G.W. 7, 8, 235
Diederich, J. 65, 229
Dignum, F. 4, 231
Dort, H.E. van 3, 236
Eisenstadt, M. 62, 65, 232
Ericsson, K.A. 59, 63, 64, 229
Essink, L.J.B. 1, 229
Falkenberg, E.D. 4, 27, 229
Fazlollahi, B. 1, 229
Feigenbaum, E.A. 10, 229
Fisher, A.S. 24, 229
Fitzgerald, G. 1, 2, 22, 23, 25, 227, 232, 236
Flaatten, P.O. 2, 229
Floyd, C. 3, 22, 229
Gaag, L.C. van der 51, 231
Gane, C. 113, 117, 229
Gardarin, G. 4, 227
Genrich, H.J. 186, 229
Gillingham, R.W. 205, 228
Gordon, J. 205, 228
Grabowski, M. 59, 230
Gray, P. 37, 230
Guida, G. 59, 60, 230
Gustafson, R. 205, 228
Hagelstein, J. 1, 233
Halpin, T.A. 33, 232
Harandi, M.T. 4, 231
Harel, D. 16, 230
Hart, A. 59, 60, 62, 230
AUTHOR INDEX

Harwood, W.T. 28, 229
Hatley, D.J. 113, 230
Hayes-Roth, F. 60, 230
Heijes, H. 2, 145, 154, 230, 236
Hernadi, A. 3, 231
Hindle, K. 23, 232
Hoc, J.M. 64, 230
Hofstede, A.H.M. ter 16, 37, 46, 49, 51, 186, 200, 202, 220, 227, 230, 236
Huizinga, G. 51, 213, 230
Jackson, M.A. 32, 34, 40, 42, 230
Jackson, M.I. 28, 229
Jonge, W. de 1, 235
Kappel, G. 186, 230
Karimi, J. 4, 230
Kato, M. 205, 228
Kempen, H. van 4, 229
Kensing, F. 1, 14, 21, 22, 227, 230
Kersten, M.L. 4, 231
Kidd, A.L. 59, 60, 231
Knuth, E. 3, 231
Konsynski, B.R. 4, 230
Kowalski, R. 37, 231
Kramer, J. 4, 5, 231
Krijger, M.J. 17, 23, 227
Lakatos, I. 11, 231
Ledderhof, H.J.A. 117, 231
Lee, P.T. 4, 234
Leeuw, A.C.J. de 15, 21, 231
Lenat, D.B. 60, 230
Leplat, J. 64, 230
Linster, M. 65, 229
Lith, P. van 77, 231
Lockemann, P.C. 3, 231
Loucopoulos, P. 4, 20, 231, 234
Lubars, M.D. 4, 231
Lucas, P.J.F. 51, 231
Lundin, J. 1, 21, 227
MacDonald, I.G. 1, 2, 77, 79, 231, 232, 233
Maddison, R.N. 23, 232
Marashi, M. 4, 5, 228
Martin, J. 2, 15, 25, 27, 77, 232
Mathiasson, L. 1, 21, 22, 227, 232
Mayr, H.C. 3, 231
McAllister, A.J. 5, 27, 234
McClure, C. 3, 15, 24, 25, 27, 28, 232
McCubrey, D.J. 2, 229
McMenamin, S. 113, 117, 140, 232
Meersman, R. 35, 234
Mellish, C.S. 203, 228
Mellor, S.J. 113, 235
Metais, E. 4, 227
Minker, J. 203, 232
Motta, E. 62, 65, 232
Mouton, J.S. 148, 227
Munk-Madsen, A. 1, 21, 227
Newell, A. 17, 64, 232
Ng, K. 4, 5, 231
NGGO 80, 117, 232
Nielen, G.C. 99, 232
Nijssen, G.M. 33, 232
Nimpen, N. 4, 229
Nisbett, R.E. 63, 232
Nolt, J. 37, 233
Nunamaker jr., J.F. 3, 5, 26, 27, 32, 228
Nurminen, M.I. 24, 233
O'Riordan, P.D. 2, 229
Olle, T.W. 1, 25, 233
Olson, M.H. 1, 229
Oosterom, N.E. van 37, 46, 49, 51, 200, 202, 220, 236
Palmer, J. 113, 117, 140, 232
Pietri, F. 4, 233
Pirbhai, I.A. 113, 230
Potts, C. 3, 4, 5, 231, 233
Proix, C. 4, 233
Puncello, P.P. 4, 233
Rajan, T. 62, 65, 232
Rasbech, M. 1, 21, 227
Rock-Evans, R. 77, 78, 233
Rohaty, D. 37, 233

238
AUTHOR INDEX

Rolland, C.  1, 4, 233
Sandberg, J.  64, 236
Sarson, T.  113, 117, 229
Schaapherder, F.E.S.  184, 233
Schreiber, G.  66, 233
Schrefl, M.  186, 230
Seligmann, P.S.  13, 22, 23, 24, 78, 80, 117, 233
Shafer, D.G.  205, 228
Shortliffe, E.H.  204, 234
Shubik, M.  15, 234
Simon, H.A.  17, 59, 63, 64, 229, 232
Skidmore, S.  3, 6, 14, 15, 227
Slatter, P.E.  59, 62, 63, 234
Søgaard, P.  1, 21, 227
Sol, H.G.  1, 2, 3, 6, 7, 13, 17, 21, 22, 23, 24, 61, 78, 80, 117, 233, 234, 235, 236
Song, J.H.T.  23, 232
Sorensen, P.G.  5, 27, 234
Stephens, M.  4, 234
Stokes, N.  23, 232
Stomph-Blessing, L.T.M.  7, 234
Sutcliffe, A.G.  20, 234
Tan, K.P.  4, 234
Tanniru, M.R.  1, 229
Tasso, C.  59, 60, 230
Torrigiani, P.  4, 233
Townsend, C.  205, 228
Tremblay, J.  5, 27, 234
Troyer, O. de  35, 234
Unisys  143, 144, 234, 235
Verhoeef, T.F.  80, 116, 117, 233, 235
Verlinden, P.  35, 234
Verrijn-Stuart, A.A.  1, 25, 233
Vitalari, N.P.  7, 8, 235
Vonk, R.  3, 24, 26, 28, 235
Vroom, V.H.  148, 235
Ward, P.T.  113, 235
Waterman, D.A.  60, 230
Weber, E.S.  3, 26, 228
Weelden, J.A.  61, 66, 235
Weide, Th.P. van der  16, 186, 227, 230
Weigand, H.  4, 231
Welke, R.J.  3, 33, 235
Whitehead, K.  4, 5, 231, 234
Wielinga, B.  59, 60, 61, 62, 66, 228, 233
Wieringa, R.  1, 235
Wijers, G.M.  2, 3, 13, 22, 23, 24, 37, 46, 49, 51, 67, 68, 78, 80, 117, 179, 200, 202, 220, 225, 233, 235, 236
Wilson, B.  15, 16, 23, 236
Wilson, T.D.  63, 232
Winkels, R.  64, 236
Wintraecken, J.J.V.R.  33, 236
Wohed, R.  4, 5, 236
Wood-Harper, A.T.  2, 23, 236
Wright, G.  59, 62, 63, 236
Yadav, S.B.  4, 236
Yetton, P.W.  148, 235
Yourdun, E.  3, 113, 115, 116, 117, 236

239
SUBJECT INDEX

4GL 24, 143

A
abort facility 194
activity analysis 155
activity/data matrix 164
adequate representation 172
agenda 185
agenda mechanism 185, 215
aggregation 153
analytical capabilities 15
application knowledge base 183
application level 26, 31, 179
application model 37, 184
association 32, 35, 206
homogeneous 48
association instance 200
automated system 127
automated tools 3, 10, 180

B
basic concept manipulation 72
basic procedure 32, 49, 199
basic procedure execution
mechanism 199, 218
basic task 49, 72
behaviour model 115
BFD 156, 161
black box 152
blackboard system 189
BLUES 30, 80, 117
bottom-up 155
bounded rationality 7
business activity 159
business engineering 143
business flow diagram see BFD
business function 159
business process 159
control 159
operational 159
query 159
signal 159
strategic 159
tactic 159

C
cardinality 131
CASE 2, 25, 29
cash cow 154
certainty factor model 51, 204
choice 39
classify-concept 201
client 86
cognitive limitations 7
cognitive psychology 64
coherence 8, 18, 19
communibility 172
competitor 87, 150
compiler 24
complex concept 174
concept 2, 32, 35, 206
concept instance 200
concept structure 32, 35
concept structure editor 185, 207
conceptual model 60, 65
conceptualization 59, 72, 179
condition 51
composite 51
simple 51
consistency 3
consistency checker 185, 211
constraints 37
consume 47
context diagram 16, 20, 125
create-association 200
create-concept 200, 218
current concept 208
current situation 140
current task 209
customer 150
customized 9

d
D2S2 77
SUBJECT INDEX

data 152
  external 152
  input 152
  internal 164
  output 152
data analysis 62, 155
data dictionary 24, 117
data elicitation 62
data model 131
  preliminary 131
data oriented 117
database management system 24
dataflow diagram 16, 19, 20, 75, 117, 121
  logical 137
  physical 124
  preliminary 126
datastore 127
dead dog 154
debugger 24
decision 8, 17, 20, 39, 43, 50, 193, 206
decision instance 185, 193
decision making mechanism 204
decision outcome 50, 193
  argument for 70
decision rule 20, 32, 51, 204
decomposition 39
decomposition diagram 75
delete-association 201
delete-concept 201
department 120, 127
dependency model 79
derivation rules 203
development process 1, 5
diagram 16
dialogue 129
dialogue flow diagram 144
divide and conquer 80, 171
domain knowledge 4, 7
dynamic interpretation 28
dynamic model see model

E
  editor 24
  effective support 2
  elicitation 59, 67, 178
  employee 127
  entity 90, 131
    associative 103
    conceptual 90
    concrete 90
    external 90
    financial 90
    permanent 103
    primary process 90
  entity life cycle 79
  entity relationship diagram 75,
    77, 79, 90, 117, 121, 131, 166
    global 90
  entity structure diagram 33
  environmental model 115
  ESI 143
  ESM 143
  ESP 143
  essential model 114
  event 92
    incoming 92
    outgoing 92
  event-driven 110, 140
  event list 20
  event partitioning 113, 117
  evolutionary development 224
  Excelelator 117
  expectation about organization 86
  experience 5
  experiment 75
  expert 59
  expressive power 37, 172
  external party 86, 119, 150
    offering 119
    requesting 119
  external relationship 119
  external studies 143, 151
SUBJECT INDEX

F
first order predicate calculus 37, 203
flexibility in modelling 172, 176
flow 92, 123
data driven 127
external 123, 127
financial 123, 127
information 123, 127
internal 123, 127
normal 127
primary process 123, 127
support 123, 127
supporting 92
time driven 127
triggering 92
formal definition of
ancestor 35
child 44
concept structure 35
connected 44
decision_path 44
direct_parent 55
global_manipulation 52
include 52
initial 43
label 43
local_manipulation 52
meta-model 52
objectification_of 35
objectified_into 35
part_of 43
pater_familias 36
place_name 52
relevant_parent 54
role 35
roles 53
subtype 35
supertask 53
task structure 43
task_path 44
task_scope 53
trigger 43
type 53
formal specification 16
fragment 65, 180
framework 13
free model see model
function hierarchy 97
function logic diagram 79
function model 79, 92
function oriented 120
functional decomposition 20, 156
fuzziness 2

G
government 150
graphical notation 170
group 65
guidance 4

H
heuristic production rules 204
hidden assumption 22
history 20, 183
history adaptation mechanism 203, 221
homogeneous association see association
Horn clauses 203

I
IE 77
IEF 80
IEW 30, 80, 145
implementation model 114
improvement project 143
inclusion 8, 18, 19
incompleteness 2
individual interpretation 2
information architecture 30, 75
information engineer 1, 2, 183
practising 3, 7, 9
information exchange 19, 177
information management 100

243
information model  155, 166  
information place  32, 47, 50
  global  50
  local  50
information place adaptation
  mechanism  202, 220
information place creation
  mechanism  201, 220
information strategy planning  77
inheritance  33
initial item  39
initiation  189
input-process-output diagram see IPO
integration  100, 134
intelligent assistant  5
intended role  48
internal studies  144, 151
interpretation  59, 70, 179
interpretation mechanism  28, 183, 185, 213
interview  63
  directed  63
  focussed  63
  structured  63
  unstructured  63
interviewing  63
introspection protocol  64
invariant  19
IPO  156, 160
IRAAD  77
iteration  39

J
JSD  32

K
KBS  60
knowledge acquisition  5, 11, 59
  methods  59
knowledge based system  60
knowledge elicitation methods  62
knowledge engineer  60, 180
L
language  15
level of
  coherence see coherence
  consistency see consistency
  inclusion see inclusion
  precision see precision
LINC  145
link  65

M
management  20
management oriented  26
management support  27
managerial level  14
managerial style  148
manipulation relationship  103, 136, 165
MAPPER  145
market analysis  145
marking  188
  initial  190
mathematical model  16
matrix  16, 79
measurement  149
META  205
meta-knowledge base  183
meta-level  5, 26, 31, 179
meta-model  51
meta-model editor  183, 184, 205
meta-model management  224
meta-modelling technique  32
meta-system  10, 27, 183
method  1, 214
method knowledge  4, 29
methodology  1
mission  148
model  6
  conceptual  60, 65
    dynamic  16
    free  15
mathematical 16
mental 7, 170
pictorial 15
simulation 16
structured 16
verbal 15
model 6
construction 27, 222
derivation 28, 222
verification 28, 222
model preference 171
model structure 8
modelling 6
modelling behaviour 7, 9
modelling concept 17
modelling environment 7, 18, 20
modelling knowledge 9, 10, 29
acquisition approach 30, 59
detailed 169
engineer 183
representation technique 30
modelling language 15
modelling process 7
modelling support 10
modelling support system see MSS
modelling task see task
modelling technique 16, 171
MSS 11, 183
MSS environment 11, 29, 183
architecture 183, 184
functionality 183
prototype 204

N
Nassi-Shneidermann diagram 16
NIAM 16, 33
non-modelling task see task
nothing 50, 193

O
objectification 33, 174
objectified association 33
objective 85, 149
observational study 64, 67
operational level 14
optional task 39
organization chart 148
organizational culture 148
organizational profile 148
organizational structure 119, 120
orientation 82, 117, 147
output 47
outside-in approach 109

P
parallelism 39, 195
Pareto 167
participant 7, 17
partner 150
PASCAL 16
passive 191, 193
personnel management 100
philosophy 15
physical dataflow diagram see
dataflow diagram
physical model 124
pictorial model see model
place 186
planning and control 100
portfolio analysis 154
postcondition 19
practice 2, 172
precision 8, 18, 19
predicate 186
preparation 59, 67, 178
primary process 84, 100, 119
problem situation 85
problem solving 6, 9
process 92, 127
co-ordinating 92
creating 92
data driven 92
externally triggered 92
internally triggered 92
logical 135
process 92, 127
  multi-functional 138
reporting 92
retrieving 92
time driven 92
transforming 92
process entity matrix 100
process flow 92
  financial 92
  primary 92
  secondary 92
process guidance 27, 222
process logic diagram 77
process oriented 6, 26, 117, 121, 140, 167
processor 127
product 84, 150
product development 99
product hierarchy 84
product oriented 3, 6, 26, 120
product portfolio 144
productivity 1
programming language 16
project management workbench 24
Prolog 204
Prolog predicates 206, 214
  active 214
  added 214
  application_model 214
  association_instance 214
  basic_procedure 206
  chosen 214
  class_of 214
  class_ofs 214
  classified 214
  concept_instance 214
  consumed 214
  contents 214
  created 214
  current 214
  decomposed 214
  deleted 214

  global_intended_role 206
  global_manipulation 206
  history 214
  info_place 206
  info_place_instance 214
  initial 206
  initiated 214
  last 214
  local_intended_role 206
  local_manipulation 206
  nr_passives 214
  nr_put_on_agenda 214
  nr_triggered 214
  objectification 206
  objectification_instance 214
  part_of 206
  place_type 206
  realized 214
  role 206
  role_instance 214
  subtype 206
  subtype_instance 214
  supertask 206
  task_place 214
  task_scope 206
  taskobject_instance 214
  trigger 206
  waiting 214

  Promod 117
  protocol setting 68
  prototype 16
  prototypical organization function 98
  prototyping 28, 143
  PrT-net 186, 187
  PrT-net places 191, 192, 194, 195, 198
    # passive 191
    # put on agenda 191
    # to be put on agenda 191
    # to be triggered 192
    # triggered 191
    active 191

246
class_of 191
current 191
decision made 194
decomposed 191
history 191, 192
initiated 191
last 192
last to be removed for sub 198
new_inst_number 191
overall abort 195
realized 191
subtask to be aborted 198
task object aborted 195
task object to be initiated 191
task terminated 192
waiting 191
psychological methods 64

Q
quality 1, 2, 10

R
rapid prototyping 60
real time system 113
refer-to 47, 49
    explicit 49
    implicit 49
relationship 90, 131
    associative 103
    instance-of 103
    recursive 103
    structural 103
relationship with organization 87
reminder 63
report and screen generator 24
repository 26
representation 8, 18, 20
research programme 11
response 123
role 33
role label 90, 131

S
scenario 1
scope 19, 46
screen layout 144
SDW 30, 80, 117
select-all-by-context 200
select-all-by-set 200
select-one-by-context 200
select-one-by-set 200, 218
self-activity 111, 149, 167
self-maintenance 99
sequence 39
service 84, 150
service co-operator 150
shareholder 87
SHELL 205, 213
simulation model 16
specialization 33, 174
specification formalism 16
star 154
state transition 186
stimulus 117, 123
stimulus-response mechanism 114
strategy 2, 6, 17, 171
structure chart 16
structured development 61
structured english 16
structured model 15
structured technique 16
structured text 16
subprocess 123
    financial 123
    primary 123
subtask 17, 39
supplier 86, 150
support
    automated 25
    effective 2, 25
    individualized 181
    intelligent 3
    methodical 9
support
   modelling 10
support environment 11
supporting tools 60
switch 194
synchronization 39
syntactically correct 37

T
   table 16
   task 2, 8, 17, 39, 43, 206
      implicit modelling 70
      modelling 18, 70
      non-modelling 18, 70
   task instance 185, 191
   task object 43
   task object instance 189
      active 185, 189
      current 185, 189
      waiting 185, 189
   task order 17, 39
   task scope 32, 176
   task scope editor 211
   task structure 32, 39, 43
   task structure editor 184, 209
   Teamwork 117
   technique 11
   temporary repository 47
   text analysis 65, 70
      concept oriented 70, 71
         task oriented 70
   text based model 65
   text object 65
   theory level 31, 32
   think aloud protocol 63, 178
      concurrent 64, 67
      retrospective 64
      selective 64
      simulation by teletype 64, 67
   time dependency 177
   time-intensive 180
   top-down 113, 155
   transcript 70
   transformation 3, 4
   transition 186
   trigger 39
   trigger analysis 131
   Turbo Prolog 204

U
   uncertainty 51
   understanding 10, 13
      detailed 169
   user interface 183, 184
   user involvement 171

V
   variety between experts 172
   verbal data 61, 65, 179
   verbal model 15
   verbalization 63
   verification 3, 4
   verification and derivation
      mechanism 203
   verification rule language 32
   verification rules 19, 32, 37, 47, 203
      a-posteriori 32, 47
      a-priori 32, 47

W
   way of
      controlling 13, 20
      modelling 13, 15
      supporting 13, 24
      thinking 13, 21
      working 13, 17
   wild cat 154
   word processor 24

Y
   YOURDON 20, 113
   Yourdon A/D Toolkit 117
SAMENVATTING

PROBLEEMSTELLING


Ervaren ontwikkelaars zijn in de praktijk tegen deze methodisch zwakheden aangelopen en hebben hiervoor oplossingen ontwikkeld. Hun opgebouwde ervaring houdt een uitbreiding, verfijning en verandering in ten opzichte van officieel bekende beschrijvingen van methoden en technieken in de vorm van boeken en cursusmateriaal. Deze gedetailleerde praktische modellerkennis vormt een waardevolle kennisbron voor geautomatiseerde hulpmiddelen. In dit proefschrift staat deze kennis van de ervaren informatiesysteemontwikkelaar uit de praktijk centraal.

In het proefschrift wordt beoogd te komen tot een verbeterde geautomatiseerde modellerondersteuning voor de informatiesysteemontwikkelaar. Een belangrijke ingrediënt daarvoor is goede kennis over de modeller- en werkwijze zoals door deze ontwikkelaar toegepast. Bij een gebrek aan kennis over het te ondersteunen proces blijven geautomatiseerde hulpmiddelen grotendeels beperkt tot documentatie- en verificatiehulpmiddelen, terwijl bij een verbeterd inzicht in versterkte mate ondersteuning gerealiseerd kan worden: (1) op het gebied van strategie en procesbegeleiding, (2) bij het maken van strategiekeuzen, (3) in het opstellen en modificeren van modellen, (4) bij het verifiëren van modellen afhankelijk van de uit te voeren taak, en (5) in het automatisch afleiden van nieuwe modelstructuren.

Een goede aansluiting van geautomatiseerde hulpmiddelen bij het te ondersteunen proces is eveneens essentieel voor het effectief gebruik van deze hulpmiddelen en ligt in het verlengde van kennis over dit proces. Een goede aansluiting kan bewerkstelligd worden door beschrijvingen van toegepaste modeller- en werkwijzen expliciet op te nemen in modellerondersteunende systemen (MSS-en) waardoor deze als aanpasbaar en instelbaar zijn te beschouwen en waardoor hun functionaliteit bepaald wordt door expliciet opgenomen modellerkennis.

De belangrijkste probleemgebieden die bij het komen tot een verbeterde ondersteuning voor de systeemontwikkelaar dus dienen te worden onderzocht, betreffen kennisvergaring over het te ondersteunen proces en het expliciet gebruik van de vergaarde kennis bij het aanbieden van geautomatiseerde
SAMENVATTING

ondersteuning. Daartoe worden de volgende onderzoeksvragen binnen dit proefschrift onderzocht:

1. een techniek te ontwikkelen waarin modelleerkennis vastgelegd kan worden met als uitgangspunt dat een modelleer- en werkwijze gespecificeerd worden en de onderlinge verbanden hiertussen,
2. een aanpak te formuleren waarmee gedetailleerde modelleerkennis van ervaren ontwikkelaars uit de praktijk verkregen kan worden, en
3. een architectuur en prototype te ontwikkelen van een MSS-omgeving waarin modelleerkennis kan worden vastgelegd en waarin een interpretatiemechanisme ondersteuning kan bieden middels een geautomatiseerde interpretatie van gespecificeerde modelleerkennis.

HET REPRESENTEREN VAN MODELLERKENNIS

De techniek voor het vastleggen van modelleerkennis biedt een begrippenapparaat waarmee een modelleerwijze en een werkwijze beschreven kunnen worden alsmede een divers aantal verbanden tussen beide wijzen.


Een werkwijze geeft aan welke taken mogelijkerwijs uitgevoerd kunnen worden tijdens een modelleerproces en hoe deze taken qua decompositie en tijdsvolgorde met elkaar samenhangen. Met betrekking tot tijdsvolgorde tussen taken zijn de volgende aspecten van belang: parallelisme tussen taken, sequentie van taken, iteratie over taken, alternatieve taken, optionele taken en synchronisatie van taken. Dit is gerealiseerd door de introductie van de begrippen taak, beslissing, trigger, decompositie en initieel item. Een beschrijving van een werkwijze in deze termen wordt een taakstructuur genoemd.

Verbanden tussen modelleer- en werkwijze zijn onder andere gebaseerd op de zogeheten omvattendheid, nauwkeurigheid en samenhang van taken. De omvattendheid van een taak is te bepalen vanuit de doelstelling van de taak en
SAMENVATTING

in het bijzonder vanuit het soort modellen en modeleerconcepten waarmee de taak te maken heeft. Voor elke taak wordt een scope gedefinieerd door een relevante verzameling concepten en associaties uit de conceptstructuur aan te wijzen. De nauwkeurigheid die binnen een taak in ogenschouw genomen moet worden heeft zich in de metamodellingstechniek vertaald in a-priori en a-posteriori verificatieregels die aan een taak gehangen worden. Het eerste type regels geeft aan waaraan geconstrueerde modellen voor en gedurende de taak moeten voldoen, het laatste type geeft aan waaraan deze modellen moeten voldoen bij beëindiging van de taak. Verificatieregels worden op de conceptstructuur gedefinieerd in de vorm van eerste orde predicaatenrekening regels. Samenhang tussen taken wordt gerealiseerd door de mogelijkheid informatieuitwisseling tussen taken te definiëren middels informatieplaatsen. Taken kunnen hier resultaten inzetten, uithalen of naar de inhoud van deze plaatsen refereren.

Voor taken die niet verder gedecomponeerd worden (basistaken) dient een basisprocedure opgenomen te worden die aangeeft hoe deze taak uitgevoerd kan worden. Beslissingen kunnen voorzien worden van beslissingsregels die uitspraken doen over de waarschijnlijkheid van bepaalde uitkomsten afhankelijk van specifieke condities. Beslissingsregels worden uitgedrukt in heuristische productieregels.

Een correcte vastlegging van modeleerkennis in termen van conceptstructuur, taakstructuur en genoemde onderlinge verbanden wordt een metamodel genoemd; de ontwikkelde techniek voor de vastlegging van modeleerkennis noemen we de metamodellingstechniek.

HET VERKRIJGEN VAN MODELEERKENNIS

Bij de aanpak voor het verkrijgen van modeleerkennis is het van belang dat het resultaat gedetailleerde en praktische modeleerkennis bevat die gebaseerd is op kennis van een ervaren systeemontwikkelaar. Ervaren mensen uit de praktijk dienen in de aanpak dan ook de primaire kennisbron te zijn, mogelijk aangevuld met achtergrondliteratuur. Deze focus vinden we ook in algemene kennisacquisitietheorieën en methoden voor de ontwikkeling van kennisystemen en daartoe is de ontwikkelde acquisitieaanpak hierop ook sterk gebaseerd.

In de acquisitieaanpak worden vier taken onderkend, te weten: (1) voorbereiding, (2) elicitation, (3) interpretatie, en (4) conceptualisatie. Tijdens de voorbereiding wordt het probleemgebied afgebakend en de betrokken expert voorbereid op de te volgen werkwijze. Bovendien worden de benodigde technische en organisatorische voorbereidingen getroffen. Tijdens de elicitation wordt de te onderzoeken
SAMENVATTING

modelleertaak daadwerkelijk uitgevoerd door de expert aan de hand van een casus. Tijdens een elicitatiesessie denkt de expert hardop na en communiceert hij met de gebruiker(s) door middel van een computer- en videooverdracht. De experimentele opzet tijdens een elicitatiesessie is gericht op het verkrijgen van een zo goed mogelijk hardopdenkprotocol. Alle verbalisaties en verrichte communicatie wordt vastgelegd in een transcript. Deze verbale data is invoer voor de interpretatietauf.


Het tekstmodel is een tussenstap tussen de verbale data en het uiteindelijke conceptuele metamodel. Het conceptuele metamodel wordt opgesteld tijdens de conceptualisatie. Tijdens de conceptualisatie worden de tekstobjecten omgezet naar begrippen zoals voorgeschreven door de metamodelleringstechnieken. Omdat gedurende de interpretatie tekstobjecten al gegroepeerd worden als taken, beslissingen en concepten is een eerste opzet van taak- en conceptstructuur snel mogelijk. Deze structuren worden vervolgens geïntegreerd en consistent gemaakt volgens het grote aantal consistentieregels dat geldt voor een correct metamodel.

De acquisitieaanpak beveelt een iteratieve aanpak tussen elicitatie, interpretatie en conceptualisatie aan. In elke taak is een duidelijke betrokkenheid van de modelleerexpert gedefinieerd in de vorm hardopdenksessies, gestructureerde en ongestructureerde interviews, en reviews.

DE TOEPASBAARHEID

Om de metamodelleringstechnieken en de acquisitieaanpak te toetsen, is de aanpak toegepast voor drie ervaren informatieplanners met een duidelijk verschillende methodische achtergrond, te weten D2S2, YOURDON en ESM. In elk experiment is een hardopdenksessie gehouden met als casus de constructie van een
SAMENVATTING

informatiearchitectuur voor een hotelreserveringsorganisatie. De verbale data van
zo'n sessie is geïnterpreteerd en geconceptualiseerd tot een conceptueel
metamodel van de aanpak zoals toegepast door de ervaren informatieplanner.
Hieronder wordt kort ingegaan op elk experiment.

1. Het D2S2 experiment

Het D2S2 experiment heeft goed inzicht opgeleverd over hoe een informatie-
architectuur met behulp van D2S2 technieken als entiteit-relatie diagrammen,
functieafhankelijkheidsdiagrammen en functiehierarchieën opgesteld kan
worden. In vogelvlucht kan de geëliciteerde aanpak gekarakteriseerd worden
door een modellering die start met de buitenkant van de organisatie
alvorens de interne structuur in ogenschouw te nemen, die zeer kritisch is
ten opzichte van de bestaande bedrijfsvoering, en die primair gericht is op
de bedrijfstoekomst. Entiteitenmodellering vormt de basistechniek in de
aanpak en kent een zeer verfijnde bijbehorende conceptstructuur. Functie-
modellen worden allereerst opgezet door de afhandeling van elke externe
gebeurtenis in kaart te brengen en vast te leggen in een functie-
afhankelijkheidsdiagram. Op basis van deze gebeurtenis gedreven analyse
wordt een functiehierarchie opgesteld die sterk beïnvloed wordt door het
gebruik van een standaard verzameling prototypische bedrijfsfuncties. Er is
een groot aantal regels gevonden die verbanden aangeven tussen de
verschillende modelleretechnieken. Als laatste punt vermelden we dat
opleiding en zelfactiviteit van gebruikers als zeer belangrijk wordt geacht.

2. Het YOURDON experiment

Het YOURDON experiment heeft geresulteerd in een gedetailleerde
vaatlegging van een aanpak voor de constructie van een informatie-
architectuur gebaseerd op het gebruik van YOURDON technieken en principes.
De aanpak wordt gekarakteriseerd door als uitgangspunt de huidige situatie
te nemen waarbij een duidelijke voorkeur geldt voor een procesgerichte
aanpak met als argument dat procesbeschrijvingen herkenbaarder zijn voor
gebruikers. Het "fysieke" procesmodel wordt volgens het principe van event-
partitioning opgesteld hetgeen inhoudt dat voor elke externe gebeurtenis de
afhandeling ervan in een fysiek dataflow diagram wordt vastgelegd waarna
deze diagrammen geïntegreerd, geëssentialiseerd en opnieuw ingedeeld
worden in nieuwe procesclusters, leidend tot logische dataflow diagrammen.
De conceptstructuur die is opgesteld voor de dataflow diagrammen toont een
groot aantal verfijnde concepten en regels die gebruikt worden bij het
opstellen van deze diagrammen. Het gebruik van entiteit-relatie diagrammen
is volledig geïntegreerd met het gebruik van dataflow diagrammen.
3. Het ESM experiment

De aanpak zoals naar voren gekomen tijdens het ESM experiment wordt gekenmerkt door een flexibele strategie waarin diverse keuzen en optionele paden naar voren komen. De expert kent eigen voorkeuren waarvan echter afgeweken wordt bij specifieke gebruikerskarakteristieken of tijdsbeperkingen. Naast de meer gebruikelijke interne studie in de vorm van functie-decompositie, zogeheten business flow diagrammen en ruwe entiteit-relatie diagrammen, wordt ook een uitgebreide externe studie uitgevoerd in de vorm van een black box beschrijving, een productanalyse en een portfolio matrix. Het belangrijkste model is uiteindelijk de functionele decompositie waaraan de expert een groot aantal concepten en regels heeft toegevoegd die de constructie ervan vereenvoudigen en standaardiseren. Voor een informatieplanning is volledigheid niet noodzakelijk en regelmatig zijn er in de strategie momenten waarin een afweging gemaakt wordt of het in het onderhavige geval noodzakelijk is bepaalde analyses uit te voeren en zo ja, hoe uitgebreid. Gedurende het gehele proces wordt de zelfwerkzaamheid van de organisatie als uitermate belangrijk geacht.

Op basis van de conceptuele metamodellen die de experimenten hebben opgeleverd waarvan hierboven kort enkele hoofdzaken staan beschreven, stellen we dat de ontwikkelde metamodelleringstechniek op een adequate manier modelleerkennis kan vastleggen. De techniek heeft een goede uitdrukkingskracht, is flexibel in gebruik, ondersteunt een effectieve communicatie met de expert en kent sterke interne consistentecontroles. De acquisitieaanpak leidt tot een goed en gedetailleerd inzicht in het bestudeerde modelleerproces zowel qua gevolgde strategie als qua gebruikte modelleerconcepten. De aanpak is tijdsintensief en het beheersbaar houden van de grote hoeveelheid verbale data, tekstmodellen en metamodellen leidt tot veel werk. Het introduceren van geautomatiseerde hulpmiddelen zal waarschijnlijk zeer bijdragen aan de efficiency van de aanpak.

DE ONTWIKKELING VAN EEN MSS-OMGEVING

De derde onderzoeksvraag van dit proefschrift betreft de architectuur, de functionaliteit en de prototype implementatie van een MSS-omgeving waarin modelleerkennis kan worden vastgelegd en waarin een interpretatiemechanisme door expliciete interpretatie van deze kennis komt tot geautomatiseerde ondersteuning van een ontwikkelaar. Een CASE-tool dat volgens deze geschetste opzet werkt wordt wel een metasysteem genoemd en in het bijzonder deze metasysteemaakt aanpak maakt het mogelijk tot aanpasbare en uitbreidbare geautomatiseerde hulpmiddelen te komen.
SAMENVATTING

De architectuur van een MSS-omgeving kent twee belangrijke componenten: (1) de metamodeleditor en (2) het interpretatiemechanisme. Deze zijn toegankelijk via een gestandaardiseerde gebruikersinterface voor twee soorten gebruikers. De eerste soort gebruiker is de informatiesysteemontwikkelaar die een metamodel laadt en vervolgens een modelleersessie opstart (of eventueel een eerder bewaarde modelleersessie laadt en opstart). Deze gebruiker maakt gebruik van het interpretatiemechanisme. De andere gebruiker, die we hier de MSS-ontwikkelaar noemen, gebruikt de metamodeleditor om metamodels volgens de ontwikkelde metamodelleringstechniek te specificeren en op consistentie te controleren.

De metamodeleditor kent drie modules, te weten een taakstructuureditor, een conceptstructuureditor en een consistentiechecker. In de taakstructuureditor wordt de taakstructuur gespecificeerd alsmede alles wat verband legt met de conceptstructuur zoals bijvoorbeeld het definiëren van informatieplaatsen en scopes. De conceptstructuureditor wordt gebruikt voor de vastlegging van een conceptstructuur. De consistentiechecker verifieert of een ontwikkeld metamodel voldoet aan alle regels die gelden voor een correct metamodel.

Het interpretatiemechanisme biedt ondersteuning op de volgende gebieden:

1. bij het aanbieden van mogelijk te volgen strategieën door het bijhouden van een agenda van uitgevoerde, actieve en nog uit te voeren taken en beslissingen,
2. bij keuze tussen mogelijk uit te voeren taken door het op verzoek laten "vuren" van beslissingsregels,
3. bij het manipuleren van modellen door het gebruik van een set van standaard procedures,
4. bij het verifiëren van modellen afhankelijk van de uitgevoerde taak door op het juiste moment de taakafhankelijke a-priori en a-posteriori regels te checken,
5. bij het afleiden van nieuwe modelstructuren door het doorlopen van de afleidingsregels die gelden bij de uit te voeren taak,
6. bij het maken van selecties en het doorgeven van resultaten tussen taken door middel van selectie procedures en het gebruik van informatieplaatsen,
7. bij het heroverwegen van gedaan werk door de mogelijkheid te bieden terug te stappen in de historie en daarbij voortdurend de inhoud van de agenda, de modellen en de informatieplaatsen bij te werken.

De genoemde ondersteuning is gerealiseerd door een zevental mechanismen te ontwikkelen welke tezamen het interpretatiemechanisme vormen.
SAMENVATTING

De architectuur en functionele beschrijving zijn uitgewerkt in twee prototypes ontwikkeld in Turbo Prolog. Het prototype geheten META implementeert de metamodeleditor; het prototype SHELL realiseert het interpretatiemechanisme. De prototypes kennen een tekstuele gebruikersinterface. Met META zijn de conceptuele metamodellen van de experimenten ingevoerd waartoe het noodzakelijk was deze metamodellen te verfijnen tot het niveau waarop de standaardprocedures van het interpretatiemechanisme werken. Vanuit dit oogpunt gezien is er dan ook een daadwerkelijk verschil tussen een conceptueel metamodel zoals dat uit de experimenten komt en een metamodel zoals vastgelegd en bruikbaar in de MSS-omgeving. Dit laatste model is een verfijning van het eerste tot op geïmplementeerd procedureniveau. Met SHELL zijn de modelleersessies zoals waargenomen tijdens de experimenten nagespeeld. Op basis van deze testen kan gesteld worden dat het ontwikkelde interpretatiemechanisme inderdaad in staat is genoemde ondersteuning te bieden en dat een metasysteem aanpak bij de ontwikkeling van modelleerondersteunende systemen realiseerbaar is en duidelijk op meer gebieden ondersteuning kan bieden dan huidige gemailiseerde hulpmiddelen. Hiermee is de laatste onderzoeksvraag beantwoord.
CURRICULUM VITAE


257