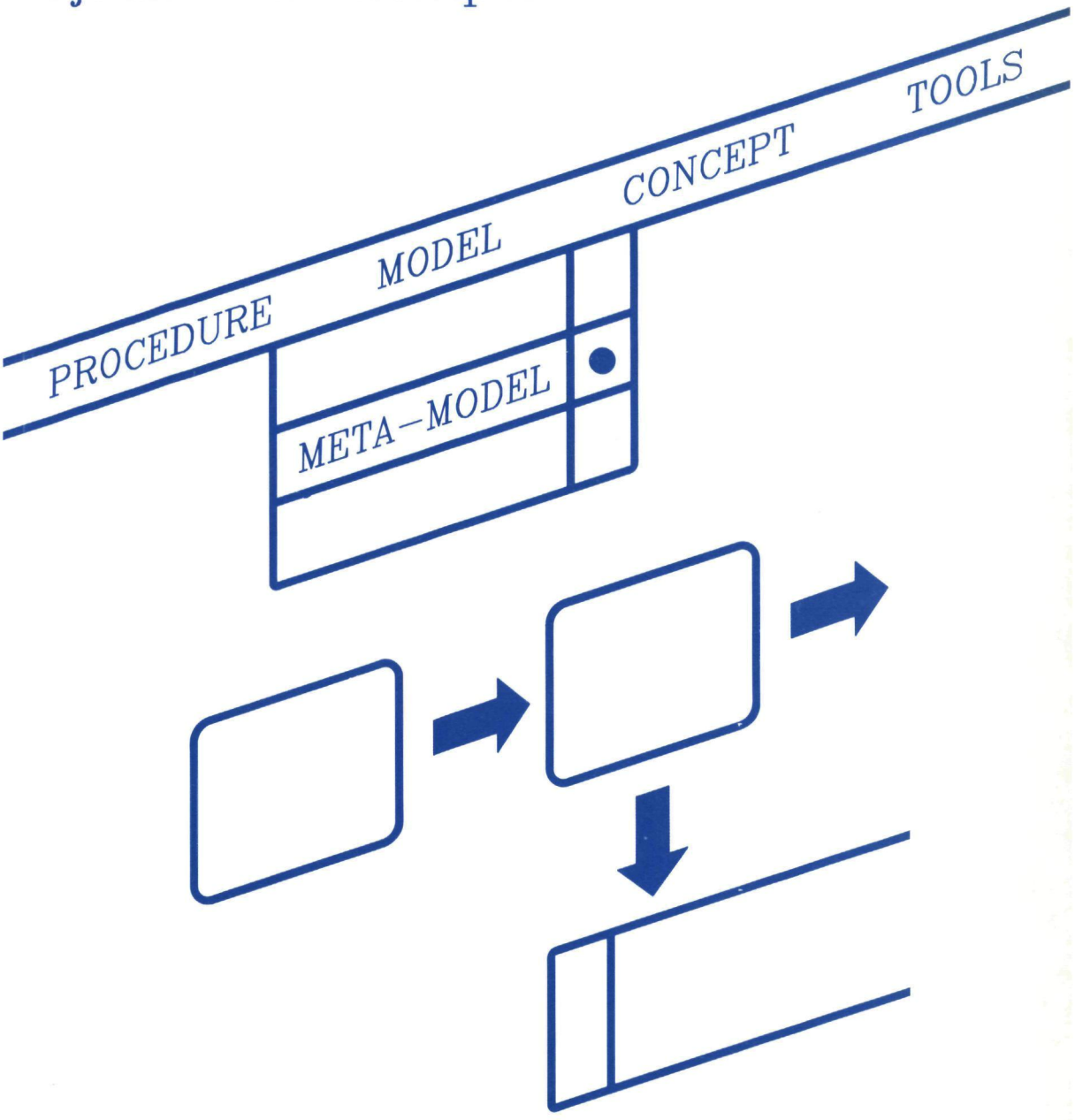
FORMALISATION OF INFORMATION SYSTEMS MODELLING

Sjaak Brinkkemper



Formalisation of Information Systems Modelling

Formalisation of Information Systems Modelling

een wetenschappelijke proeve op het gebied van de
Wiskunde en Informatica.

Proefschrift

ter verkrijging van de graad van doctor aan de
Katholieke Universiteit te Nijmegen,
volgens besluit van het college van decanen in het openbaar te verdedigen
op maandag 18 juni 1990, des namiddags te 3.30 uur

door

Jacobus Nicolaas Brinkkemper

geboren op 18 januari 1958 te Monnickendam.

Thesis Publishers

Promotores: Prof.dr. E.D. Falkenberg

Prof.dr. A.A. Verrijn Stuart, Rijksuniversiteit Leiden

CONTENTS

1. INTRODUCTION

2. MODELLING TECHNIQUES

3. ACTIVITY MODELLING

4. DATA MODELLING

5. TASK MODELLING

PREFACE

Modelling of information systems is a hardly touched research area. Of course, there exist hundreds or thousands of modelling techniques, or perhaps we should say notations, variants and extensions, that have been discussed in scientific articles and of which some are used in the daily software engineering practice. A practice that can be characterised as intuitive manipulation of modelling symbols.

But how should models be created properly? What steps are needed to come up with a complete process model? What input is required for data modelling: e.g. natural language descriptions or sample forms? How can abstraction levels be distinguished in processes? At which stage are the models of the various types related to each other? When can parts of the model be validated by the informants? Which rules have to be satisfied at which modelling steps?

The practical application of a modelling technique to some concrete problem often leads to all sorts of questions dealing with the modelling process. These are the questions that we want to address in this work. A fundamental motivation for us was the belief that a system of modelling symbols should always be accompanied by a description of the procedure that has to be followed during modelling. This gives rise to a lot more questions and issues, some which we will consider in this study. We will pay attention to the background of modelling and the reasoning about modelling by means of meta-modelling. This is elaborated in some modelling procedures and in a discussion on the support of modelling by tools.

Allthough there is just one name on the book's cover, many people contributed to its contents. First of all, I am very thankful to Marijke, Paul, Judith and Mirjam for their patiently bearing of the restrictions that this research laid on our family life. Your support and encouragement during the years made it possible to write this thesis. I thank Eckhard Falkenberg for introducing me to this research area, one that I have experienced as challenging and creative. His guidance enabled me to develop my own style of research. I also thank Alex Verrijn Stuart for the discussions of several issues of this research. I owe them and my colleagues, in particular Theo van der Weide, Peter Bruza, Mark McLoughlin, Arthur ter Hofstede, Denis Verhoef and Gerard Wijers, thanks for their constructive remarks and proof reading of the manuscript. I was very lucky that a lot of students of the department of Information Systems were willing to invest their time and energy in parts of my research. Among others, thanks go to Pauline van Boven, Mieke van der Linden, Nico Brand, Hein Bouman, Marco Knots, Arthur ter Hofstede, Mark de Lange, Ronald Looman, Carolien Koesen, Marianne Geurts, Ilse van der Kamp, Miep Berbers, Jacques

Beeker and Luc Schouten. Also I would like to thank the researchers and staff of the Department of Informatics of the University of Nijmegen, of the Software Engineering Research Centre and the consultants and staff of Moret Ernst and Young, management consultants, for the discussions, facilities and assistance. Finally, I thank my father and my brother, Dick and Nico Brinkkemper for the design and realisation of the cover of this book, and Eric Janssens for his assistance.

Getting ideas is rather simple during research, but having the opportunity, the assistance and the support to work them out is not possible without the cooperation of many, many people. The availability of all these talents is a gift for which I am thankful.

Nijmegen, April 1990 Sjaak Brinkkemper

1 INTRODUCTION

1.1 INFORMATION SYSTEMS MODELLING

Software is becoming more and more complex to be developed. New technological advancements and increasing user requirements demand large numbers of designers and programmers, good cooperation, and experienced project management for effective and efficient software development. Despite the resources employed in the software development process, the malfunctioning of automated systems and projects running out of budget are regular news items nowadays. The software crisis is a frequent subject of the news media.

Will this also be the case in the future? Certainly not! Either the software crisis will turn out to be an eternal phenomenon and will therefore lose its attraction as a news item, or the improved methods and techniques for software development will solve the crisis by increasing the quality of the development process and of the developed software systems.

This thesis is aimed to be a contribution to the last option. Methods and techniques for software development are a young research area and the feedback from experience in practice is limited. Research on all kinds of aspects of software development has to be performed and its results have to be incorporated in improved guidelines and procedures.

1.1.1 Information systems development

We start with an informal introduction to our research objective. We will restrict ourselves to the development of a particular kind of software systems, namely information systems. More than 80% of computerised applications belong to this category. Information systems are possibly automated systems in organisations, that aim at improving the knowledge of the members of these organisations and at improving the communication between their members. The information in such a system supports the members of the organisations in performing their tasks.

In order to provide an organisation with an automated information system a development project is usually set up and carried out according to a method. In such a project the relevant phenomena are recognised, modelled and incorporated in computer programs. Methodical information systems development is mostly performed in stages such as scope definition, analysis, design, construction and use.

During development there are two general problem solving techniques applied:

1. **conceptualisation**, that is the creation of a complete and formal system specification starting from informal rough specifications, and
2. **mapping**, that is the transformation of the specifications on a conceptual level (i.e. without presentation and implementation details) to the machine executable specifications.

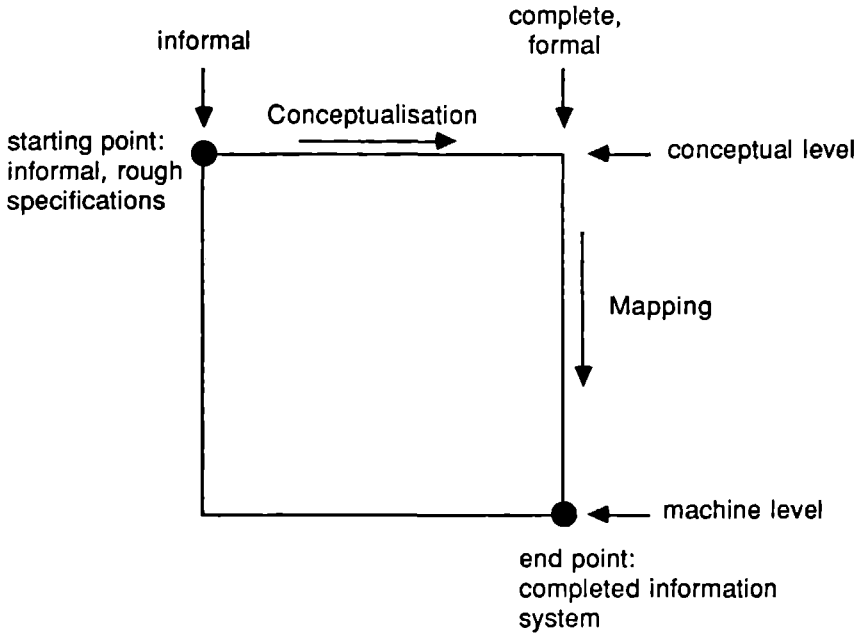


Figure 1.1 Conceptualisation and mapping

Development can then be depicted as a pathway in the two-dimensional area as in fig. 1.1 (after [Hesse 83]). The starting point of a development process is the informal rough specifications and the end point is the completed system, including all hardware and software. There are many possible pathways between these two points. In traditional software development, large parts of the conceptualisation process are mostly not performed on a conceptual level, but on the level of a programming language. As a consequence aspects of the conceptualisation process are mixed up with aspects of programming, which should be avoided [Tsichritzis 78] (conceptualisation and mapping will be defined more precise in section 1.4).

The recommended or ideal way is to separate conceptualisation and mapping activities completely, which in terms of fig. 1.1 means that first

the upper line is followed to the complete, conceptual and formal specifications and thereafter the right line which means the mapping of these specifications to the machine level via some transformation steps.

1.1.2 Modelling

Models are a commonly used development tool during conceptualisation as well as during mapping. Since organisations and their information systems are very complex, developers of information systems make use of manually or automatically created models that represent a certain part or aspect of the system. For example, there are process models, data models, data base models, interaction models. Variants of most model types are defined in relation to the various development stages, for which innumerable modelling notations have been proposed in the literature.

In the sciences models are used for many purposes. Models create a clear and structured view and help restrict to the relevant information. Theoretical explorations as well as practical development can be formulated by means of models. The use of the term model in science is therefore many-sided. Model, as we will consider it here, is best defined as done by [Apostel 60].

Definition 1.1
A system A is used as a *model* in order to obtain knowledge about a system B, where the system A is neither directly nor indirectly interacting with the system B.

During information systems development models denoted in schemas, diagrams, tables, algorithms, programs, etc., are used to describe parts of the information system. The activities and discussions of the developers are for a large part based on models. Moreover, since an information system is complex, all models of it are simplifying views, that are strongly interrelated and dependent on each other. A change in one model may cause a whole series of changes in other models.

The use of models in information systems development is not without problems. One of the motivations for this research was suggested by observations from practice that the quality of modelling processes as well as of modelling products are poor. The absence of instructions leads to subjective modelling, with related drawbacks as ambiguity and indistinctness in the models [Falkenberg 89c]. Research confirms these observations [De Brabander 84].
The observed basic problems were among others:
- Different models are made of the same system at hand.
- The modelling process takes too long.
- The models are not consistent with the systems they are supposed to model.
- The gathering of informants' specifications is hard.
- The modelling techniques are vague and unformalised.
- Experienced modellers are scarce and expensive.

In the active research area of information systems modelling much attention is given to addressing these problems. Most works deal with improved formal or informal techniques to model a certain aspect of an information system. The gathering of specifications regarding the system to be developed is also addressed quite frequently. The process of modelling and its background are however seldom focus of research. Most modelling techniques are introduced in literature by means of examples and some rules on the notation. The *modelling process*, that is the way in which models can be constructed, is lacking in general. See [Lyytinen 87] for an overview of recent information systems research, from which the above can be deduced.

1.1.3 Research objective

The problems concerning the use of models in information systems development, as far as they are related to the modelling process, gave rise to a variety of research activities, of which this thesis is the amalgamation. These research activities were motivated by one overall research objective, that is formulated as:

> *Make the modelling process of the techniques applied during information systems development more explicit.*

This objective covers many research issues. The three issues we found most important were:

a. What is the best way to construct a model, when a specific technique is applied?
b. What is the formal basis of a modelling technique?
c. How is one particular model related to the other models of an information system?

This thesis mainly addresses the construction of models (a). For some modelling techniques we will present ways to construct models. These descriptions of modelling processes are integrated with results concerning the formalisation of modelling techniques (b). The relationship between models (c) is discussed, where this turned out to be relevant.

The starting point for making the modelling techniques more explicit is the claim that the modelling process can be formalised far more than it is now. A mapping of a technique to a proper mathematical formalism is applied and the resulting theory is investigated. The concepts of the technique, its syntactic rules and its semantics are precisely described. On this basis the steps of a modelling process can be prescribed in a refined procedural way, which also directs, for instance, the determination of modelling alternatives. We call these refined modelling steps *modelling procedures*. The experience of modellers can be incorporated in these procedures, which makes their expertise available to more people. This and other consequences will be discussed in section 2.5.

The remainder of this chapter will be devoted to establish a proper framework for the discussions. In section 1.2 and 1.3 some terminology is

introduced concerning the various roles of persons involved in the development process, and also concerning the development means, such as methods, techniques, tools and notations. Models and modelling have a central role in this thesis and therefore their background in relation to information systems development is extensively discussed in section 1.4. Finally, in section 1.5 we present, in addition to an overview of this thesis, a positioning of the thesis in a widely known information systems research framework.

1.2 THE HUMAN ROLES IN INFORMATION SYSTEMS DEVELOPMENT

In order to establish some terminology regarding the various roles of people involved in software development projects we depict in fig. 1.2 a simple general process structure of such a development project. The notation in this figure is of the Channel-Agency modelling technique [Scheschonk 84] extended with the concept of agent, denoted in the lower half of the rectangles. This technique is discussed in chapter 4.1

The development process can be split into the activities of the developers, shown in the lower half of fig. 1.2, and those of the environment, depicted in the upper half. The corresponding roles in the environment are:

a. **Commissioning agent** has the responsibility of giving the task to develop a system. This task may be given to developers in the same organisation as the commissioning agent or to a specialised software development firm. The persons in this role are often also responsible for the assignment of the other roles in the environment.
b. **Informant** is any person specifying information about the organisational context, the hardware requirements and about the data to be stored in the information system. The misleading term user can then be avoided for this role, since not only future users specify this information, but also their managers, domain experts, etc.. The informants also validate the results of the work of the analysts and designers. We will come back to this role in section 2.4.
c. **Acceptor** has the responsibility of the approval of the complete intermediate requirement specifications and of the final operational system including all related documentation, user training, data conversion, etc..
d. **User** is any person who uses the system after it has been put into operation.

The roles of the developers are:

a. **Project manager** plans and manages all development activities and the related resources. So the project manager is therefore responsible for the assignment of the other development roles.
b. **Analyst** performs the analysis stage of the project in which, among other things, the system scope is defined and the current system or situation is described. The result of the analysis is passed on to the designers.

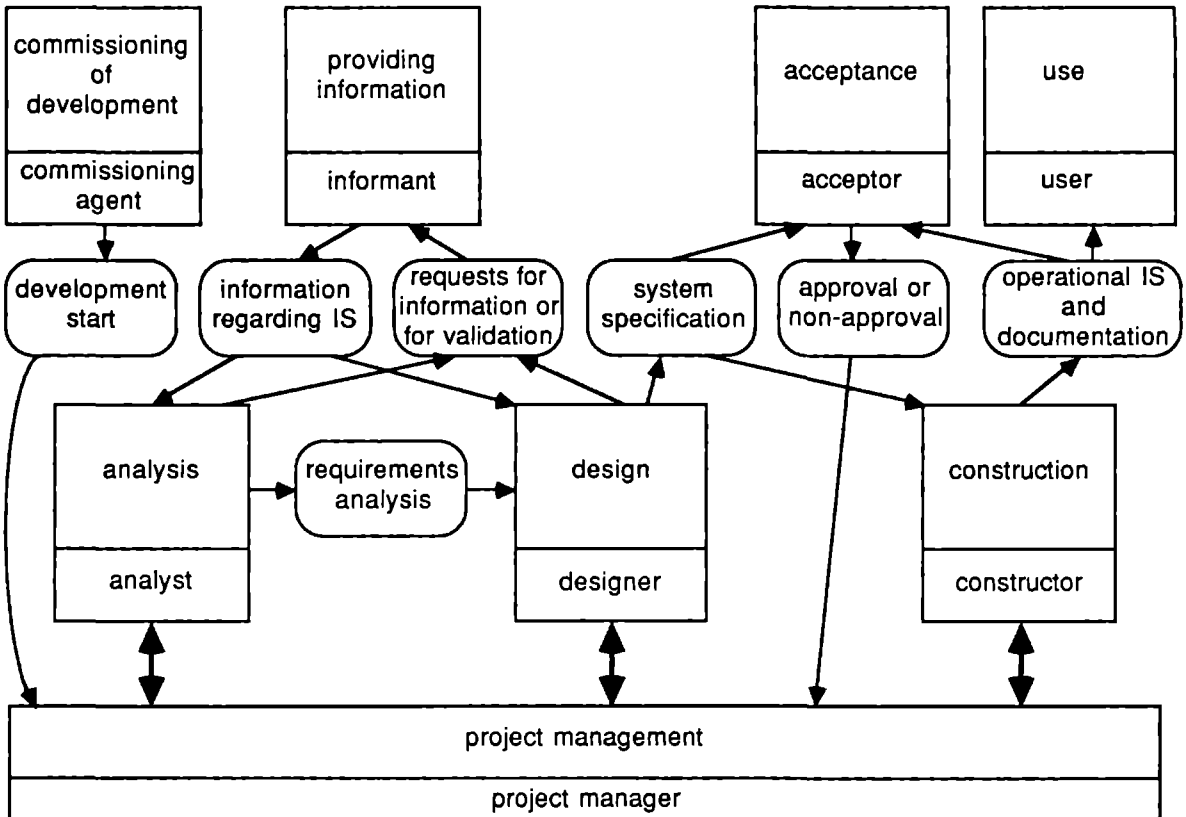


Figure 1.2 Process structure and cast of a development project

c. **Designer** developes the specifications of the new system. These specifications are formally approved by the acceptor and thereafter either adjusted or passed on to the constructors.
d. **Constructor** constructs and documents the information system on the basis of the requirements specification. The system is tested and installed in the user environment. The latter activities may be performed by different persons, but we do not consider this possibility in this thesis.

Similar roles are distinguished in [Olle 88b] and in [Verrijn Stuart 87]. Note, that the above assignment of roles is applicable to the development of systems of any kind. For example, the development of a new system is often commissioned by an organisation to a software firm and thus the roles are separated between those two parties. In contrast to this, the commissioning and development of standard software packages is performed by persons from the same organisation, except for the users who are the purchasers of the package. Note furthermore, that some roles may be played by the same person, such as the acceptor and the commisioning agent, or the analyst and the designer.

The current information processing system as well as the system to be constructed are to a great extent described by means of models during the analysis and design stage. Since we are dealing with modelling in this thesis and need not to distinguish between analysis and design, we use the general term of **modeller** for analysts and designers. The communication between the modellers and the informants is very important and intensive during development. The quality of this communication is reflected in the quality of the resulting system and influences the length of the development process. We will come back to this in section 2.4.

From fig. 1.2 it can also be seen that the design of the information system has resulted in a complete system specification and therefore no communication is needed between the constructors and the informants. Programming should be done on the basis of completed specifications and not have details left open that require any extra information. We discuss this in depth in chapter 5, where a specification technique is presented that has been developed in order to fulfil this requirement.

1.3 METHODS, TECHNIQUES AND TOOLS

The term technique has already been mentioned a number of times and will often be used together with the terms method and tool. In view of the differences between them, it is essential to define them and employ them accordingly. One should note that all terms are to be understood in the context of information systems development. We start with the terms method and methodology, that originate from the Greek word '$\mu\varepsilon\theta o\delta o\varsigma$' (methodos), which means a way to investigate something.

Definition 1.2

The *methodology of information systems development* is the systematic description, explanation and evaluation of all aspects of methodical information systems development.

Note from this definition, that there is just one methodology of information systems development and that all research activities in this field contribute to this methodology. However, some methodological schools can be distinguished: software engineering, database management, management informations systems and the infological approach among others. See [Iivari 89] for a thorough paradigmatic analysis of these schools. In the field of information systems there exist, in addition to the methodology of information systems development, the methodology of information planning, the methodology of assessment of information systems and the methodology of information systems maintenance [Bemelmans 87].

Definition 1.3

A *method* is an approach, based on a certain way of thinking, to carry out an information systems development process, consisting of directions and rules structured according to a systematic ordering of development activities and corresponding development products.
In case the method only prescribes activities restricted to what has to be done and how the systems development process should be controlled and managed, we call the method a *project management method*, whereas a *development method* prescribes also the way how activities have to be performed.

The methods prescribe the development of the information system in an ordered hierarchy of steps, which have to be performed by the analysts, designers, constructors and project managers, in order to deliver the parts of the system in a proper manner. The system can be constructed from these parts and brought into use by the organisation, which commissioned the development. The steps prescribe in general only the type of development products that have to be generated and not their creation procedure. For example, it is stated that a business activity hierarchy or a data model should be developed. In some methods the developers are free to choose the specification convention and format of these products, and in other methods it is precisely defined, which are to be used.

The number of methods for information systems development can not be determined any more, because of the countless variants and the tailoring to specific circumstances. Different methods can cover different parts of the system development. For example, ISAC [Lundeberg 80] and Information Engineering (IEM; [Brand 89ab], [Martin 88a]) pay explicit attention to data, processes, technology and organisation, whereas the System Development Methodology (SDM; [Turner 87]) focuses mainly at the management of system development. The first are therefore development methods, whereas the latter is a project management method.

A more elaborate definition of a method can be obtained from [Seligmann 89], where a framework for the description of information systems

development methods is introduced. This framework comprises the way of thinking (or philosophy), the way of modelling (the models to be constructed and their interrelationships), the way of organising (subdivided in the way of working, i.e. how to do the development, and the way of control, i.e. how to manage the development) and the way of supporting (the description techniques and the corresponding tools). The emphasis within a particular method on either the way of working or the way of control defines the distinction between a development method and a project management method.

Definition 1.4

A *technique* provides the description of the manner in which, and the notation with which a part of the development must take place. This incorporates the practical steps to follow when carrying out a development activity.

A *notation* is a system of symbols with a corresponding set of rules, which determine the correct application of the symbols. A notation is used to denote the results of a technique.

A *tool* is a, possibly automated, means to carry out a part of the development process. A tool may support a notation, a technique or even a method.

Examples of techniques are process decomposition, affinity analysis, change analysis, group interviewing, etc. Examples of notations are decomposition diagrams, association matrices, the symbolic system of the data modelling technique according to NIAM [Nijssen 89], and the way to set up an interview protocol description.

According to our definition, techniques can be applied in the steps of a method, where products of the type the technique delivers are required. For instance, a particular data modelling technique can be used to design a data model, or affinity analysis can be used to cluster entity types that are related to processes in an association matrix. In the case of a project management method it is the task of the project manager to select a number of well matching techniques with which the development can be carried out. In development methods techniques, or parts thereof, have been incorporated.

Although many notations for the various development products of an information systems development project have been proposed in the literature, the techniques to create such products are in general not given. Part of the formalisation of information systems modelling is therefore the detailing of the techniques in so-called modelling procedures. This aspect will be elaborated in section 2.3, where the concept of technique will be refined.

For most of the techniques, corresponding automated tools exist in various types and sizes. In section 6.1 we will distinguish some types of tools and their functionality. A formal way to relate the techniques of a method to the capabilities of a tool is presented in section 6.4.

1.4 MODELS

Models serve as means to obtain knowledge about a certain system. This it not only the case for information systems development, but nearly every branch of science makes use of models. In this section the origin of models and their scientific methodological background will be related to information systems modelling. The work of Bertels and Nauta [Bertels 69] and the thesis of Dietz [Dietz 87] serve as a starting point.

1.4.1 Types of models

The word model originates from the Latin '*modulus*', what was used for a measure in architecture. Via the Italian '*modello*', that stands for scale imitations of buildings, it came into the English language as the word model and as the word mould. Nowadays the word model is used in a large variety of contexts and has therefore various synonyms: schema, diagram, sketch, example, blueprint, paradigm, picture, pattern, prototype, etc.

The use of the term model, as we consider it here, is already given in definition 1.1. A model is a system, but systems exist in a variety of concrete or abstract forms. Assume, for simplicity, a system to be a collection of interrelated entities, then the classification of entities into concepts, concrete objects and symbols according to Ogden and Richards [Ogden 49] implies three types of systems:

- **Conceptual systems**: systems of which the entities are concepts, or, say, things in the minds of people. Examples are set theory and the periodical system of elements. Theoretical system can also be termed conceptual systems.
- **Concrete systems**: systems in which the entities are concrete objects. Examples of concrete systems are a car, a factory and a societal organisation. Real system and empirical system are synonyms for concrete system.
- **Symbolic systems**: systems of which the entities are uninterpreted syntactical symbols. Examples are all languages restricted to their syntax. In the sequel we will often refer to a symbolic system as notation (see def. 1.4).

Note that the motivation to consider an entity as either conceptual, concrete or symbolic is related to a certain point of view. In different points of view the same entity may play different roles. The demarcation, as far as we know, cannot be made strict. Another remark here is that the term conceptual, in the context of the three level architecture: conceptual, internal and external [Tsichritzis 78], is derived from the term as defined above. In this work both meanings are used and the context will make clear which is meant.

Definition 1.1 defines as well that a model is a system, that stands in a particular relation with another system. Given the three types of systems, this yields nine types of models. These are depicted in the model triangle of fig. 1.3.

We will not discuss here all nine types of models, but only the four that are relevant for this work, which are in bold font next to the non-dotted arrows. Customarily, we term a system of type A a model of type A, if it is used as a model for an other system. An extensive discussion of all types of models and associated examples can be found in [Bertels 69] and [Dietz 87].

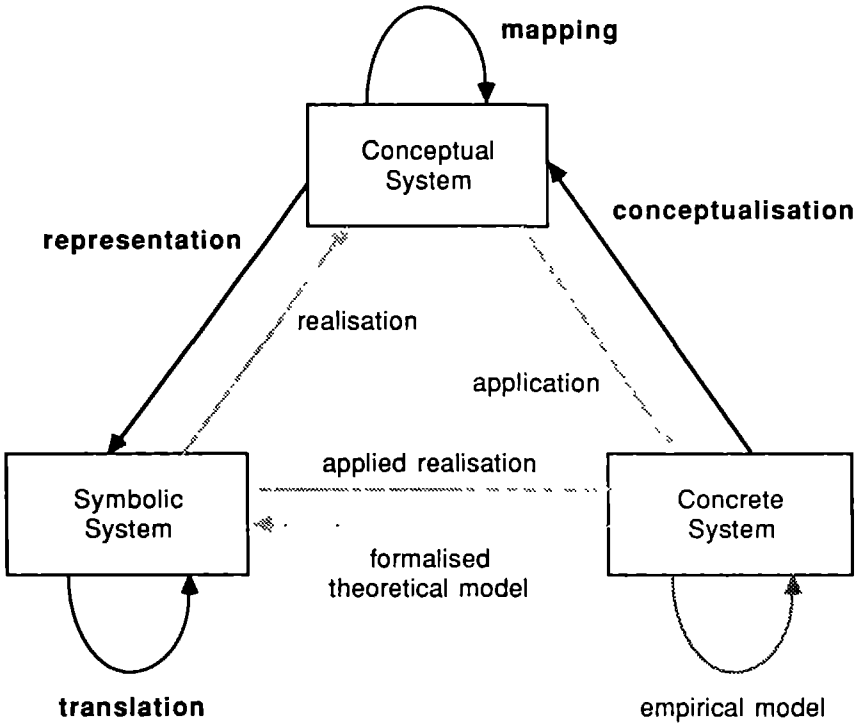


Figure 1.3 The model triangle

Firstly, a *conceptual* model of a *concrete* system is called a **conceptualisation**. Such a model may result after the observation of a concrete system, followed by the coupling of certain concrete entities to concepts and their mutual relationships. The construction of conceptualisation models is one of the main activities during the analysis phase. An example of a conceptualisation is a data model of the information stored in an information system.

A *conceptual* model of a *conceptual* system is called a **mapping**. An example is the transfer from the system of concepts of the Data Flow activity modelling technique [Gane 79] to the system of concepts of the ISAC activity modelling technique [Lundeberg 80]. Such a transfer is not just a

change of symbols, but requires an adaptation of the way of thinking about activities due to the differences in the perception of the concepts in the respective techniques.

A *symbolic* model of a *conceptual* system is called a **representation**. An example is the representation of an activity model into a symbolic system that is able to express activities, such as the Data Flow diagrams or the ISAC A-graphs. The representation is often assumed to be implicitly present. For instance the conceptualisation in section 1.1 assumes the representation of all models in suitable symbolic systems. In [Bertels 69] representation was called notation, but we prefer to use the latter term for a symbolic system.

Finally, a *symbolic* model of a *symbolic* system is called a **translation**. A translation often corresponds with a transfer between conceptual systems. This translation need not necessarily be isomorphic, i.e. one to one, but can be any morfism. An example is the translation of a model denoted in the symbolic system of the Data Flow activity modelling technique to a model in the symbolic system of the ISAC activity modelling technique. In the latter the symbol of for the notion of a data store is missing, because it is not a concept in ISAC, and therefore a translation should include a decision for each data store whether it should be seen as a message set or as an activity. See [Falkenberg 88] for a more elaborate discussion of the mapping between those systems of concepts and the corresponding systems of symbols.

In addition to these four types of models, we define a **formalisation** as a mapping onto a mathematical system of concepts with a corresponding representation. An example of a formalisation is the result of the mathematical description of a conceptual modelling system. We will show this for most of the modelling systems in the next chapters. For instance, the Conceptual Task Modelling technique is mapped onto a 12 tuple in the formal description of section 5.3.

Now that we have a clear notion of model, we use the word *modelling* for the process of setting up a model of any type as shown in fig. 1.3. A *modeller* is a person, who models. Modeller was already introduced for the generalisation of analyst and designer in section 1.2. The context will determine which is meant.

1.4.2 Models of information systems

The triangle of model types of the preceding section clarifies the use of models during the development of information systems and their study with the help of meta-modelling. The first aspect is shown in fig. 1.4 and discussed in this section. Meta-modelling is elaborated in chapter 2.

On either side of the development traject are the Universe of Discourse (UoD) as starting point and the information system as end point. The UoD is a concrete system observed or known by the informants, that has to be conceptualised and subsequently represented into a collection of models, such as a data model, a process model or a dialogue model. These models

are then mapped with a corresponding translation onto a collection of models of the internal (i.e. machine) level and the external (i.e. user interaction) level of the information system. This confirms that an information system is a symbolic system, which follows already from the fact that an information system is a data processing system.

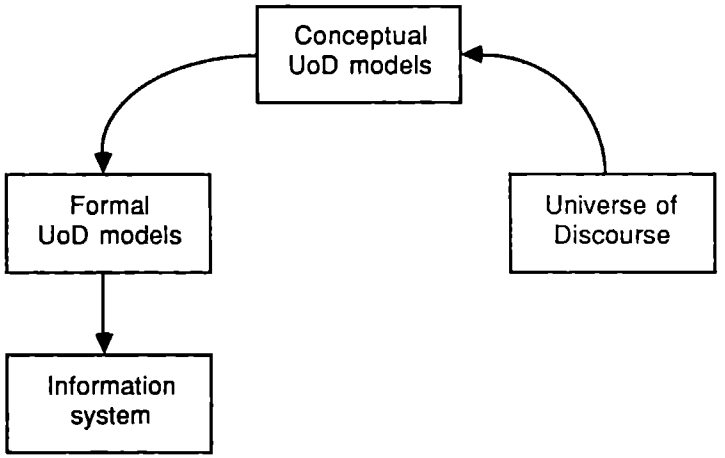


Figure 1.4 Information systems modelling

We now can propose improved definitions of both systems compared to those given in [ISO 82]. The definitions relate the scope of the system to a certain given *objective* of a person or an organisation in developing an information system for a particular UoD. The origin and format of this objective are postulated.

Definition 1.5

A *Universe of Discourse* is a system of concrete entities, which were, are or will be relevant with respect to a given objective.

An *information system* is a manual, partially automated, or fully automated system of symbolic entities, representing facts about concrete entities, that are recorded because of their relevance with respect to a given objective, and that can be updated, retrieved, and from which other facts can be derived.

Note that from another viewpoint an information system can be seen as a concrete system with hardware, software, input forms, output reports and human operators for instance. This is not considered in this work. Some authors, see for instance [Griethuysen 81], subdivide the UoD into a concrete system, called object system, and a conceptual system, called abstraction system. The abstraction system is then defined to consist of rules about the entities. We, however, consider these rules to be implicit in

the system of the UoD. In other words, we assume that conceptualisation makes them explicit.

1.4.3 Formal visuals

Many diagrammatic notations are used for representation purposes in the information modelling process. An overview of the most popular ones can be found in [Olle 82] and [Martin 88b]. Harel introduced in [Harel 88] the notion of *visual formalism* in the research area of modelling techniques. A modelling technique is a visual formalism, if it is based on a mathematical theory with a corresponding graphical notation. Harel advocated the replacement of the popular informal modelling techniques by new visualised formal techniques.

We claim, on the other hand, that the existing techniques can be extended with a rigourous definition of their semantics, which formalises the visual techniques. Our experience shows that the syntax and use of the techniques are already very structured and do not cause severe difficulties to be formalised. The formalism reveals mostly extra knowledge about the technique. The obtained *formal visuals* combine the advantage of simple generation, comprehension and communication by humans with the automatable manipulation, maintenance and analysis of models based on the formalism. The issue of chosing a suitable formalism will be discussed in sec. 2.3.

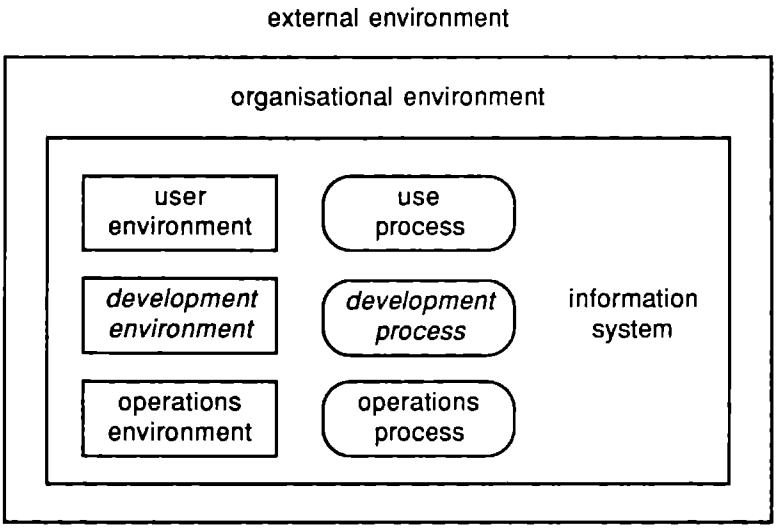


Figure 1.5 Information systems research framework

1.5 RESEARCH PERSPECTIVE AND OVERVIEW

1.5.1 Research framework

Research in the field of information systems is broad and comprises many different research approaches. An adequate information systems research framework was developed by Ives, Hamilton and Davis [Ives 80] by categorising a set of more than three hundred doctoral dissertations in this area. This framework can diagrammatically be depicted as in fig. 1.5.

The model distinguishes a user environment, a development environment and an operations environment (represented by squares), that determine the resources and constraints of the scope and form of information systems and their processes. The dynamic interaction of all components is modelled as the use process, the development process and the operations process (represented by ovals). The information system, its environments and its processes are surrounded by the organisational environment, which in its turn is embedded in the external environment.

This thesis falls within the components *development process*, which is defined as the selection and application of organisational resources that yield the information system, and the *information systems development environment*, which comprises development methods and techniques, design personnel and their characteristics, and the organisation and management of information systems development and maintenance. We aim at improving the parts of the development process that deal with modelling by using existing or new techniques with a formal basis.

In a survey paper of Lyytinen [Lyytinen 87] the framework of [Ives 80] is refined. The problem classes of the development process and the use process are related to the impact of new technology, alternative development process models, innovative project management, modelling improvements and theory development. The impact is assessed on the basis of an extensive survey of research results. Lyytinen states that many of the problems in information systems development are due to the poor, undisciplined and incomplete development practices.

This thesis aims to contribute to the knowledge about modelling of information systems in the formalistic as well as the functional way. Formalistic information system models are geared towards more abstract specification and support a more disciplined modelling process. Functional modelling is improved by the introduction of a new specification technique for task modelling (see ch. 5). Empirical evidence for these claims is however not yet available, which is a common shortcoming in information systems research ([Lyytinen 87], p. 36).

Another shortcoming in information systems research according to Lyytinen, from which this thesis definitely does not suffer, is the theoretical diversification and the rarity of cumulative research. The studied modelling techniques, which we will improve in the following chapters,

are well known and commonly applied in practice. The mathematics applied is mostly simple predicate calculus. Moreover, the new technique for task modelling builds on research of others, because it is composed from three existing techniques.

1.5.2 Overview

This thesis is organised as follows. The following chapter deals in detail with models and modelling in the information systems development process. The research technique of meta-modelling is explained and justified with respect to the general framework of modelling of section 1.4. Based on a discussion of various ways to formalise modelling, the reasoning about modelling is motivated. We introduce modelling procedures as stepwise guide-lines to perform the modelling process.

Modelling procedures are presented in the chapters 3, 4 and 5 for the modelling of events, activities, data and tasks. These parts have a more or less a similar structure. The models to be constructed, e.g. event model or task model, are discussed and related to their meta-models. A modelling procedure is given and illustrated with some examples. Theoretical consequences of the modelling constructs are finally related to the modelling procedures.

The last chapter deals with four subjects that are related to modelling and its support by automated tools. The terminology and functionality of these tools, popularly called CASE-tools, are discussed. The support of the interrelationships between models in tools is called **modelling transparency**. Various degrees of modelling transparency are defined. Many modelling tools are developed to support the development of information systems according to a specific method, wheras others aim to support several methods. We introduce the term **method companionship** for the relationship between tool and method. A formal procedure to derive method companionship using meta-models is presented and illustrated for the case of one specific method and tool. Finally, we discuss the construction of models in layers to overcome problems of complexity. This layered modelling turns out to facilitate a splitting of the formalisation into several parts.

A large part of this thesis has already been published and some publications served as a starting point for the research reported here. Some sections are extended abstracts of these publications. The precise references and their status with respect to this work are indicated at the beginning of each chapter.

2 MODELLING TECHNIQUES

2.1 FORMALISATION OF MODELLING

From the problems encountered in information modelling, as discussed in section 1.1, and from the necessity to obtain more insight and to put more rigour in the information modelling techniques applied in practice, we conclude that the modelling techniques themselves should be formalised. Maximal knowledge about the techniques should be made explicit in order to minimise the informal and heuristic application of the techniques and to justify an effective and efficient application resulting in models of high quality. Among other things, the inevitable subjectivity of modelling [Falkenberg 89c] should be restricted as much as possible.

Recall from section 1.4.1 that we consider formalisation to be the mapping of the system in a mathematical conceptual system and the corresponding representation. With regard to modelling techniques this means that we add formality to the informally defined technique. This therefore does not lead to a replacement of the technique by some nice mathematical formalism, which cannot be applied in practice by the average modeller. The application of techniques in practice gets a sound basis.

Our position towards formality is rather pragmatic. Mathematics offers a rich variety of theories for the formalisation of the structured problem areas of information systems. For our purposes we choose a formalism that fulfils the following requirements:

- The mathematical formalism should express the technique in an *adequate* manner, i.e. the intended meaning of the technique should be reflected in the mathematical constructs. Possible operations in the technique should have equivalents in the formalism.
- The mathematical formalism should be *generally known*. This is not as trivial as it looks, since this excludes the mapping onto an ad hoc formalism for which no elaborated theory is present. The task modelling technique CTM (see chapter 5) was therefore mapped onto the formalism of Predicate-transition nets, which has an extensive theoretical basis [Genrich 87], although some disadvantages of PrT-nets indicated the need for the development of a new theory.
- The formalism should be *simple* to apply. A good theory increases the knowledge about the technique in a simple way.

Examples of the choice of formalism made in this work are first order predicate calculus for the data modelling and the activity modelling technique (ch. 3 and 4), PrT-nets for CTM (ch. 5), and automaton theory for

dialogue modelling (sec. 6.5). More formalisms for modelling techniques are discussed in section 2.3.

The above requirements were inspired by the requirements on formal specification languages as formulated by Bergstra and Renardel de Lavalette [Bergstra 89]. There does not exist a generally accepted formulation of the requirements on formalisation techniques in our field. Such a formulation is not a trivial issue, because the research approaches vary considerably and deviate substantially from the approaches in research fields as formal specifications or programming languages. An overview of approaches for information systems can be found in [Falkenberg 89b].

How can formalisation of techniques be seen in the model triangle of fig. 1.3? Crucial to this is to treat the modelling technique as a concrete system that has to be modelled. This means that the system of concepts of a modelling technique is considered as a concrete system on an abstraction level higher than the application of modelling in the development of an information system. It can be seen as an instance to type abstraction, that frequently occurs in the science of specification techniques. Since modelling is our object of study, we call this meta-modelling. Meta-modelling as the superpositioning of model triangles is depicted in fig. 2.1.

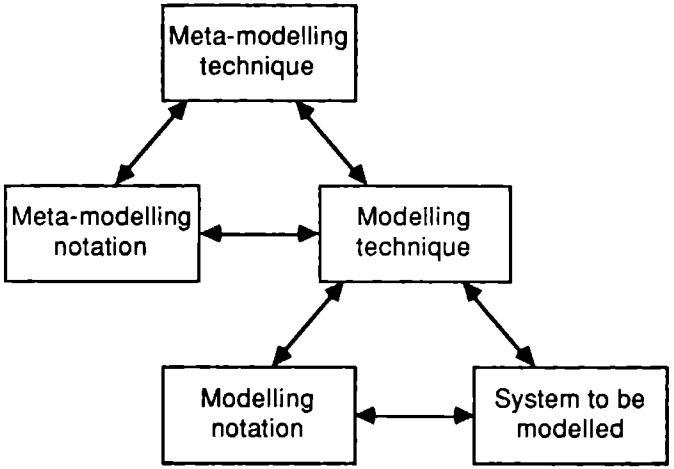


Figure 2.1 Meta-modelling

The modelling during ordinary information systems development corresponds in this figure with the lower model triangle and the meta-modelling with the upper model triangle. This leads to the following definition of meta-model and meta-modelling.

Definition 2.1
A *meta-model* is a conceptual model of a modelling technique.
Meta-modelling is the process of the conceptualisation of a modelling technique.

As an information system is not captured in one single model, meta-modelling has also to be applied in various perspectives using different techniques. The resulting models are classified according to the perspective. For example, there are meta-data models and meta-activity models.

Some research approaches in information systems can be explained in terms of combinations of the model triangle of fig. 1.3 by considering a system of a particular type to be of another type. Recall that meta-modelling was obtained by considering a conceptual system as a concrete system. An application that treated symbolic systems as concrete systems was reported in lexicographics, the science of dictionaries, see [Linden 88ab]. The superpositioning of several model triangles is only significant to, say, two or three times, which was already pointed out by Falkenberg [Falkenberg 83]. In [Brinkkemper 89b;90a] it is described how an extra meta-level was needed in the derivation of the support of a tool to a method (see also section 6.4). The models were therefore called meta-meta-models. To some extent the reasoning about meta-modelling in this chapter can be seen as meta-meta-modelling as well.

We will describe meta-modelling and its use in more detail in the remainder of this chapter. The various types of meta-modelling and their applications are discussed in the next section. In section 2.3 the conceptualisation and representation techniques that can be applied for meta-modelling are discussed. Modelling techniques can thereafter be improved by means of the so-called modelling procedures, which are based on the obtained formalisation. These modelling procedures and their quality requirements are considered in section 2.4. We conclude with a short discussion of the consequences of formalisation by means of meta-modelling and modelling procedures in the final section of this chapter.

2.2 META-MODELLING

2.2.1 Meta-activity models and meta-data models

As discussed in the previous chapter meta-modelling is the process of conceptualisation of a modelling technique. Restricting the scope of interest to a particular aspect or perspective requires meta-modelling to elicit and formalise a certain aspect of the knowledge about the given technique. The resulting meta-models provide deeper insight in the examined modelling technique since statements about the technique can be justified. The process of meta-modelling can be related to modelling as shown in fig. 2.2 (adapted from [Ter Hofstede 89b]).

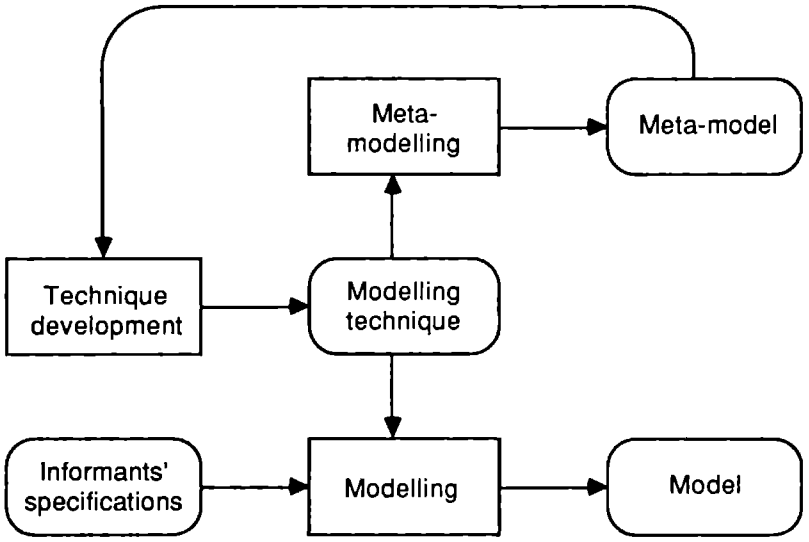


Figure 2.2 Meta-modelling versus modelling

This figure is an extension of figure 1.3 and shows input and output of both processes. We also show the evolutionary development of a modelling technique by means of the feedback of the meta-model to the technique development. This is one of the possible applications of meta-models.

A model of the steps of a technique is obtained after an analysis of the way the technique works, which provides us insight in the procedure of the technique. The result is therefore called a meta-activity model or a meta-process model. Well-known representation techniques for activities can be applied to denote the resultant model. The data of the techniques are exemplified in the products of the techniques. Data modelling of these products yields the meta-data model or the meta-information model. The precise relation between the meta-activity model and the meta-data model is captured in the parts of the meta-data model that correspond with the intermediate and final products of the technique. Obviously a lot of the knowledge about modelling is analogously valid for meta-modelling. Examples of meta-models are shown in figure 2.3 and 2.4, which are taken from [Brinkkemper 89d]. The notations of both models will be discussed in section 2.3.

We see in this meta-model the activity 'model functional architecture' (with nr. 6.2), that consists of three subactivities with the products 'functional decomposition', 'organisational unit data model' and 'enterprise information needs' as input, and the product 'functional design' as output. The example shows furthermore that the products may consist of

subproducts and the subactivities may result in intermediate products, that are not outputs of the activity.

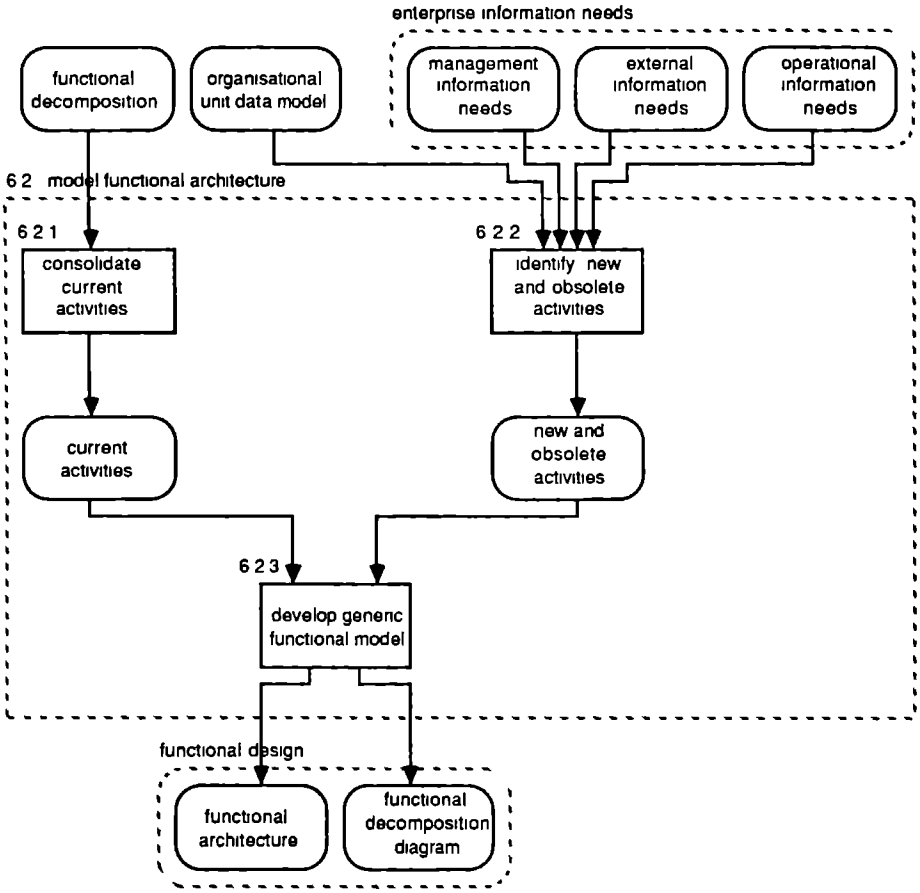


Figure 2.3 Example of a meta-activity model

In this meta-model the entity types 'organisational unit', 'function', 'process', and 'activity' can be distinguished. An example of a rule is that the relationship 'organisational unit' is-responsible-for 'function' is a one to many relationship, which means that one particular function falls under the responsibility of just one organisational unit, but one organisational unit may be responsible for more than one function.

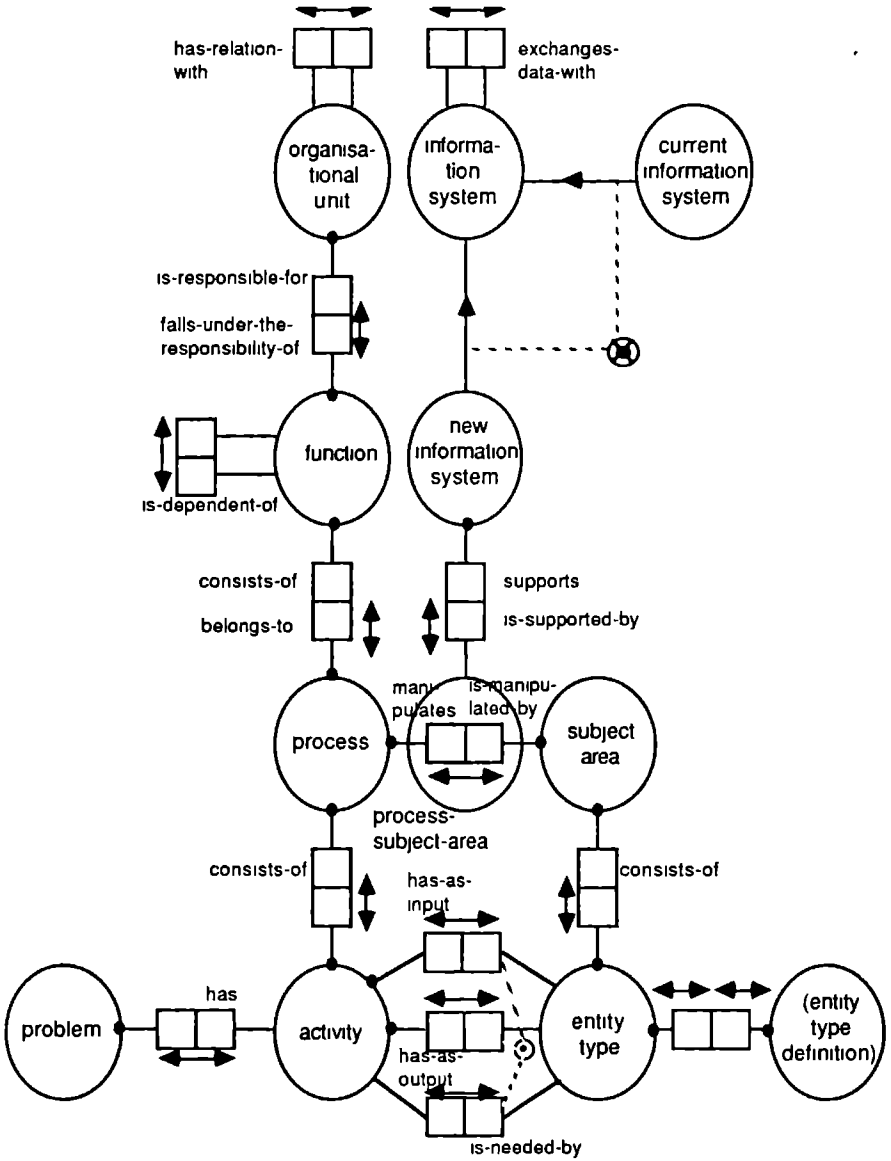


Figure 2.4 Example of a meta-data model.

2.2.2 Applications of meta-modelling

Meta-models can serve many purposes in the area of information systems research. We review here some of those applications reported in the literature.

a. The explicit and concise description of techniques, methods and tools is the most obvious benefit of meta-modelling. These kinds of meta-models are used in the following investigations.

- The selection of tools. The meta-models of some available tools for systems development were constructed to justify the selection of one of them [Vonk 88].
- The discussion of the software development process [Lehman 87], [Kokol 89]. Lehman introduced the notion of *Software Process Model*, in the debate about the various approaches for the software engineering process, such as the waterfall model and the spiral model. Meta-models were derived for the different approaches in order to focus the discussion [Wileden 86]. The software process model is a kind of meta-process model.
- The comparison of methods. The assessment of aspects of methods according to some framework can hardly be justified without some degree of uncertainty or individual interpretation. Based on the manuals of three different methods for information systems planning, the complete meta-models of the methods were constructed. The meta-models make it possible to decompose the methods into their elementary building blocks. These models simplified an extensive, complete and nearly objective comparison of the methods in terms of their steps and products [Brinkkemper 89d].
- The development of tools. Since tools can be considered as information systems for techniques, the data model for these systems, i.e. a meta-data model of the technique, serves for the development of the tool data base or dictionary. This was applied to a dialogue modelling technique in [Koesen 89] and to a workbench shell in [Ter Hofstede 89b].
- The determination of method companionship. This so-called mapping of the techniques of a information systems development method to a development support tool based on meta-models of both method and tool preceded the drafting of guide-lines for the use of the tool in the method [Brinkkemper 89b;90a].
- The formulation of methodological insights. Given the enormous variety of information systems development methods one is interested to know what is common to most methods from a perspective free of any particular method. This knowledge enables one to judge whether a particular method is complete for systems development and what is special to or particularly emphasized in a method. In [Olle 88b] a framework of information systems development methods is presented by means of generalised meta-data models, of which the task modelling part was reviewed in [Brinkkemper 89a].
- The assessment of techniques. The question whether a particular technique is suitable for practical use is in general answered after its application in some pilot projects. In [Godwin 89] an assessment method, called DMAT1, is presented, which uses the meta-modelling of the assessment method itself as input material for the assessment of

the analysis and communication facilities of several aspects of the technique.

b. The formalised reasoning about techniques. Mathematical meta-modelling techniques make the formal justification of statements about the technique possible. In [Ahituv 87] an axiomatic meta-model of information flow is presented in order to evaluate some properties of information theory. As mentioned before, the task modelling technique of CTM is based on Predicate-Transition nets [Brinkkemper 89ae], [Ter Hofstede 89a]. In chapter 5 we will discuss some of the corollaries and theorems, that are consequences of the properties of a well-formed CTM-net structure. Properties and consequences of the formalised dialogue modelling technique are given in [Koesen 89].

c. The comparison of the prescribed methods with the actual performance of the methods. The deviation within projects from the descriptions in manuals of the applied method were collected based on some reports of information systems planning projects [Brinkkemper 89d]. It turned out for this investigation that the reports followed the methods rather well, although the experience of the planners played an important role.

d. The description of the behaviour of experts. In order to improve the functionality of information modelling tools, research projects were set up in which some experts in information modelling were extensively observed during their work [Wijers 89], [Ter Hofstede 89b]. The tools may then be adapted to the strategy of the modellers, such that novices can imitate the experts' way of working and subsequently gain their own experience. Meta-modelling and protocol analysis were applied as knowledge representation techniques in these projects.

2.3 FORMALISATION TECHNIQUES

2.3.1 Categories of techniques

In principle, every information systems modelling technique can be used for meta-modelling. Some techniques have been developed purely for meta-modelling. We will review the techniques in the following sections on their suitability for meta-data modelling and meta-activity modelling. Parts of this section are based on [Ter Hofstede 89b].

In a similar way as in the science of programming languages, we can define the syntax and semantics of a modelling technique. The *syntax* (or grammar) of the technique defines the valid constructs of the technique. The *semantics* of the technique states the meaning of each syntactic construct. The meanings belong to a certain domain of values, that is chosen to be appropriate for the purpose of the technique. The semantics are then expressed by means of a function, called *valuation function*, that associates each construct of the technique with its value in the domain of values.

This gives rise to the distinction of three categories of techniques for meta-modelling:

1. **Formal techniques:** techniques of which the syntax and the semantics are rigourously defined. It may be a mathematical theory or a technique, that is mapped to a mathematical formalism. Examples are predicate calculus and formal specification languages, like VDM [Björner 78] and Z [Spivey 88]. Petri-nets [Reisig 85] and Predicate-Transition-nets [Genrich 87] are examples of formal graphical techniques with a precisely defined syntax and semantics.
2. **Structured techniques:** techniques of which the syntax is defined. This category consists, for a large part, of diagrammatic techniques for information systems modelling, that have precise rules that define which constructs are allowed and which are not. Examples are data-flow diagramming and most data-modelling notations.
   The valuation function of structured techniques can in general not be made explicit, because the domain of values is the real world, of which we assume that it cannot be formalised. The meaning of the constructs in these techniques is therefore in general based on a common understanding communicated by means of examples.
3. **Informal techniques:** techniques for which there is no complete set of rules to constrain the models created by the technique. Natural language and unstructured pictures are informal techniques.

The process of meta-modelling is more or less the same as the ordinary modelling process. The modelling concepts are treated as concrete entities (see section 2.1) and therefore those entities, their interrelationships, the procedure in which they are applied during the modelling process, and all related aspects are modelled in meta-models. In case a modelling procedure is known for a meta-modelling technique, we simply apply this procedure.

Lay-out aspects are not incorporated in meta-models, because this has to do with the representation of the modelling technique and nothing with the system of modelling concepts. This is especially important for graphical modelling techniques: shape, size, position, fonts, etc. are ignored. One has to be careful with links to other models. This is either due to the limitations of the representation device, and is thus not conceptual and therefore not to be incorporated in the meta-model, or this is due to the links between modelling components of one or two particular types and therefore have to be represented in the meta-model by an association between these types. For instance, the hierarchy of data flow models is caused by the fact that processes are decomposed in subprocesses. So the hierarchy is represented by the association of processes having subprocesses.

Meta-data modelling of a technique may be simplified by a reversed data transformation of the dictionary of an available tool for that technique. Since such a tool is capable to store models produced by that technique, the data base structure of the tool should reflect the concepts of this technique. Most tools are provided with documentation on the structure of their dictionary for import-export facilities, so the meta-modelling is straightforward. One should, however, take into account that this

dictionary does include lay-out details and that these should not be considered due to the motivations cited in the previous paragraph.

2.3.2 Meta-data modelling

Meta-data models capture the more static aspects of a method. The data that is recorded during the methodical development in the products of the method is modelled in terms of concepts, associations between those concepts and the rules that must hold for these concepts and associations.

Those concepts, associations and rules can be modelled in most data modelling techniques. The extent to which a data modelling technique is suitable for meta-data modelling is determined by the expressive power of the technique and the clarity of its models. The *elementary data modelling technique* according to NIAM [Nijssen 89], an extension of the *binary data model* [Verheijen 82], and the *Entity-Relationship modelling technique* [Chen 76] have been used for meta-data modelling in various projects: [Brinkkemper 89abde;90a], [Koesen 89], [Wijers 89]. The elementary data model is especially suitable for meta-data modelling because of the existence of a modelling procedure for it, the equal treatment of objects, the arbitrary degree of the relationships and the existence of a variety of constraints. An example of a meta-data model in this notation is shown in fig. 2.4. The Entity-Relationship technique proved to be useful in the case of a large meta-data model in a project for the determination of method companionship [Brinkkemper 89b;90a].

The notation of the meta-data models used in [Olle 88b] is too restrictive to represent detailed models. This technique seems only suitable for a global impression of a meta-model, because only binary relationships are possible and nesting of relationships is restricted via artificial cross-reference components. Also, only two out of the possible sixteen binary relationships are provided. For frameworks, such as [Olle 88b], a proper meta-modelling technique still has to be developed.

First order predicate calculus is a mathematical formalism suitable for meta-data modelling. An algebraic structure is defined consisting of sets, that represent the concepts of the modelled technique, and relations, that represent the associations between the concepts. Axioms formulate the rules of the technique. We will use this meta-data modelling technique in chapters 3, 4 and 5. The notational variant we use is illustrated in the following example of a part of a meta-data model for unmarked Petri-nets.

The **concepts** of this technique are *place*, *transition* and *arrow*:

S : set of places

T : set of transitions

A : set of arrows

There exist basically two **associations**: a place is input for a transition and a place is output for a transition:

predicate input **over** $S \times A \times T$

predicate output **over** $S \times A \times T$

In this notation *input(s,a,t)* means that the place *s* is input to the transition *t* via the arrow *a*, and similarly for *output(s,a,t)*. The basic predicates can not be defined in terms of other predicates, they are just assumed to be valid for certain values of the parameters.

Rules can be formulated in terms of the predicates and the logical operators. We call such rules **axioms**. An example is the axiom that each arrow connects a place to a transition as either input or output:

$$\forall a \in A\ \exists s \in S\ \exists t \in T\ [\text{input}(s,a,t) \vee \text{output}(s,a,t)] \qquad \text{(P1)}$$

For simplicity of reasoning we may introduce **auxiliary predicates**. For instance, we may call a place a source in the case it is not the output of any transition:

predicate source **over** S **as**

$$\text{source}(s) \equiv \neg\ \exists a \in A\ \exists t \in T\ [\text{output}(s,a,t)]$$

All quantifications are assumed to be over the correct set as the variable is used in the predicates and sometimes we leave the forall quantifier implicit. From P1 and the definition of source we may deduce the **corollary** that a source is always input to at least one transition:

$$\forall s \in S\ \text{source}(s) \Rightarrow \exists a \in A\ \exists t \in T : \text{input}(s,a,t) \qquad \text{(P2)}$$

Sometimes it may be convenient to introduce **functions** as a special type of predicate in case one parameter can have just one value for which the predicate is valid, where the other parameters may be arbitrary. We illustrate this with the predicate in_degree_p that stands for the number of incoming arrows of an transition. The set N stands for the set of natural numbers.

predicate in_degree_p **over** $T \times N$ **as**

$$\text{in\_degree}_p(t,n) \equiv$$

$$\exists a_1,a_2,\ldots,a_n\ [[i \neq j \Rightarrow a_i \neq a_j] \wedge [\exists s_1,s_2,\ldots,s_n\ [\forall i: \text{input}(s_i,a_i,t)]] \wedge$$

$$[\forall a \in A \setminus \{a_1,a_2,\ldots,a_n\} : \neg\ [\exists s : \text{input}(s,a,t)]]]$$

It can be shown that the predicate is a function from T to N, i.e.

$$\text{in\_degree}_p(t,n_1) \wedge \text{in\_degree}_p(t,n_2) \Rightarrow n_1 = n_2$$

We then may define the function in_degree_f as follows:

function in_degree_f **over** T **to** N **as**

$in\_degree_f(t) = n \Leftrightarrow in\_degree_p(t,n)$

which defines the value of the function in_degree_f equal to the unique value for which the predicate in_degree_p is true. The distiction between the predicate and the function by means of the subscript can then be dropped.

For the sake of simplicity we will directly define such predicates as functions. The axiom of the uniqueness of the function image is implicitly assumed to be valid. All necessary arithmetical or set theoretical operations are also assumed to be present and properly defined. We ought to remark that the notation $a_1,a_2,\ldots,a_n$ is not according to the rules for first order predicate calculus. This expression is, however, according to Hilbert's thesis [Barwise 77], expressible in first order logic, and we will therefore use the shorthand expression.

Other mathematical formalisms, that have been used as meta-data modelling, are *set theory* as used in [Ahituv 87] and *automaton theory* as used in [Koesen 89]. These theories are based on first order predicate calculus and are particularly useful for meta-modelling, since both have a broad theoretical basis. This way one has at one's disposal a standard terminology, proper definitions and a number of insights, that can be used for acquiring additional knowledge about the modelling technique.

We have experienced that the combination of NIAM and predicate calculus as meta-data modelling techniques is very effective. The concise and detailed notation of a meta-model in NIAM provides a clear insight in the modelling technique. Predicate calculus represents the rules of the technique in a simple mathematical formalism with a similar structure. The examples in the coming chapters will illustrate the combined use of the meta-data modelling techniques.

2.3.3 Meta-activity modelling

Meta-activity models capture the more dynamic aspects of a modelling technique by describing the way the technique produces its products. A technique to be used for meta-activity modelling should be able to express the concepts of system development activity, flow of information, and decisions as well as their inter-dependencies.

Activity modelling techniques are used for meta-activity modelling. The information process diagramming techniques, such as the *ISAC A-graphs* [Lundeberg 80], *Data Flow diagrams* [Gane 79] and *Channel-Agency nets* [Scheschonk 84], represent activities and information flows in a global way. The precise operational intention can not be described in these techniques, except by good naming. This may cause problems due to the ambiguity and informality of natural language. The mentioned techniques describe the activities and input and output data in a hierarchical bi-partite graph, so that different levels of detail can be modelled. A decomposition diagram of

the activities facilitates the acquisition of an overview of the meta-activity model. The meta-activity model of fig. 2.3 is denoted in the Channel-Agency formalism.

A technique that possesses the concept of decision is the *task analysis diagramming technique* as proposed by Bots in [Bots 89]. A task diagram consists of a decision structure and a task structure. A task is defined as a problem solving process and a decision as the choice phase of that problem solving process. This kind of diagram is suitable to describe modelling activities with a high degree of decision making, whereas the modelling of information and its processing can not be described in this technique. In [Wijers 89] this technique was therefore combined with NIAM for the meta-modelling of the information modelling process.

Mathematical formalisms are not available for meta-activity modelling, because action is not considered in mathematics. There are, however, several mathematically based formalisms for the description of computer processing. *Petri-nets* [Reisig 85], *Predicate-Transition nets* [Genrich 87] and variants thereof, such as *IML* [Richter 82] and *CTM* [Brinkkemper 89ae], [Ter Hofstede 89a], model activities by means of places and transitions. In the meta-model the places correspond with the states of the technique and the transitions correspond with the events. Dependencies between states and events are depicted by arrows. A disadvantage of these modelling techniques is that the models become very complex for medium to large systems. A strong point is the level of detail that can be modelled with these techniques. They will therefore only be used for modelling small parts of the modelling technique.

Two techniques were especially developed for representing modelling techniques and thus also for meta-modelling purposes. The *General Design Representation (GDR)* was proposed in [Lubars 89] as a set of primitive modelling constructs out of which activity modelling techniques, such as data flow diagrams and Petri-nets could be constructed. The GDR formalism was not intended to be used by modellers, but by constructors of modelling support tools.

In [Humphrey 89] *Entity Process Models* are proposed for software process modelling, i.e. meta-activity modelling. This technique focusses on the description of the states of the products during the system development process. This is a more behaviourally oriented approach of development modelling in contrast to the task oriented approaches of the aforementioned techniques. The representation is that of state charts of [Harel 87;88], and the technique is modelled in an unconstrained and in a constrained way. The latter is done to model the limitations of system development, such as the availability of personnel and hardware.

2.4 MODELLING PROCEDURES

2.4.1 Background and definition

Nowadays, the concrete application of a modelling technique turns out to involve a large amount of intuition and many unformalised heuristics. The modellers have, on the one hand the freedom to choose the way in which they perform their task, but this leads, on the other hand, to indistinct and different applications of a method, less structured activities and cumbersome progress discussions. There is a lack of precise description as to how these modelling processes should be carried out. The related drawbacks include: absence of instructions with complicated modelling, ambiguity in modelling, quality of modelling related to quality of modellers, protracted training.

We therefore claim that an information system design method should include a description of the precise steps involved in the performance of modelling. We do this by means of so-called *modelling procedures*. In these procedures the modelling steps are prescribed in a refined procedural or even algorithmic manner. The modeller should know at any time what information is required from the informants and how the model is derived from this information. Furthermore, these procedures should contain criteria which, at any time when modelling alternatives occur, define the appropriate alternative in the given situation. A well-known illustration from Entity-Relationship modelling is a criterion which determines whether a particular object should be modelled as an entity type, a relationship type or an attribute type.

This leads to a strongly directed and a more procedural or algorithmic type of modelling technique. Modelling procedures can make the expertise needed for the modelling process explicit. In this section we will define modelling procedures and the related concept of the informant's specification. Requirements on modelling procedures will be discussed in the next section and the set up of a procedure, i.e. a modelling procedure for modelling procedures, is treated in section 2.4.3.

The concept of modelling procedure is introduced in terms of the model triangle (fig. 1.3) of section 1.4 and we use its general terminology. As follows from our problem setting, we restrict modelling procedures to the conceptualisation and representation process. We must first discuss where the information about the Universe of Discourse, i.e. the concrete system, comes from.

Definition 2.2 An *informant* is any person that provides the modeller with information about the concrete system.
The information that is provided by the informant to the modellers is called *informant's specification*.

As already discussed in section 1.2 the informant can be any person that has knowledge about the concrete system: domain experts, future users,

managers. Even developers of the earlier stages, such as information system planners and analysts, may be informants to developers in later stages, because of their knowledge of the concrete system. The informant's specification may be of an arbitrary format and contents: spoken words during interviewing, notes, reports, books, etc. Important, however, is that the format and content is determined by the informants on the basis of their familiarity with it.

Definition 2.3 A *modelling procedure* is a stepwise prescription of the processes of conceptualisation in terms of a given system of modelling concepts, and the corresponding representation of a concrete system making use of informant's specifications and resulting in a model.

Most modelling techniques were introduced only with a specification of the representations of the concepts distinguished in the technique, or in other words an explanation of the notation that is to be used. A precise description of what kind of information is needed from the informants and how then a model is conceptualised from this information is in general lacking. We realise that for some modelling techniques, like decomposition diagrams, the modelling procedure is rather simple, but for the more complex techniques, like n-ary data modelling techniques, the procedure is far from trivial.

We therefore state precisely a definition of a modelling technique, which is a special type of technique as defined in section 1.3.

Definition 2.4 A *modelling technique* is a modelling procedure and a corresponding notation to carry out a certain type of modelling activity.

In the same way as Wirth formulated that programs are a combination of data and algorithms in [Wirth 76], we state that modelling techniques are a combination of notation and modelling procedure: *technique = notation + procedure*. In other words, a representation technique becomes a modelling technique when the modelling procedure is formulated.

The concept of modelling procedures was introduced in [Brinkkemper 88ab]. Before this, the necessity to combine a modelling representation technique with the conceptualisation guide-lines was already advocated by Nijssen in NIAM. This data modelling technique was the first to be combined with a modelling procedure [Vermeir 82]. In subsequent publications this modelling procedure was refined: [Wintraecken 85], [Falkenberg 87] and [Nijssen 89]. A procedure for the Entity-Relationship modelling technique was among others given in [Brinkkemper 88ab] and this will be extended in chapter 4 with a proper formalisation.

Normalisation, in the context of the relational model, is extensively covered in the literature, but most procedures for the construction of these tables, which mainly consist of the subsequent removal of normalisation anomalies, suffer from three drawbacks. First, the starting point is always a set of more or less arbitrary constructed tables with key definitions. No procedure is given how to come up with these tables. Secondly, the

normalisation is hardly preceded by a proper step to discover the unwanted dependencies. Only definitions and examples of normal forms and tables in non-x-normal form are given. Finally, the set of normal forms is a subset of the total set of constraints that may exist in a data model, such as optionality, cardinality and equality, so that normalisation is just a part of the complete story. In [Gillenson 87] and [Teorey 86] modelling procedures are given for relational tables, of which the latter starts with data modelling steps using the Entity-Relationship model and a subsequent transformation to tables.

For other notations for models of information systems no procedures are known, which are more than a set of unstructured heuristics.

2.4.2 Requirements for modelling procedures

Modelling procedures are part of methodical information systems development and therefore the requirements for development methods and techniques apply to some extent also to modelling procedures. Beside this, additional requirements can be formulated related to the specific nature of modelling. We propose a set of requirements, which will be discussed below. Some of the requirements are derived from [ISO 82], [Bemelmans 87] or [Falkenberg 89c].

Requirements for modelling procedures follow from the desire to strive after an optimal quality of information systems development. Quality of systems development is determined by two main requirements on techniques:

1. *Effectivity*
   A technique should contribute to the development process in such a way, that a good functioning information system results. Measurement of effectivity is difficult, since it is determined by the difference between the delivered system and the system as desired by the commissioning agent. However, general quality aspects for effective development can be derived from the techniques used.
2. *Efficiency*
   A technique should contribute to the development process with optimal use of resources as personnel and hardware. Measurement of efficiency is simple, because it measures the relative costs of the development process in terms of resource quantities.

Although we claim that modelling procedures contribute to the efficiency of systems development (see 2.5), we do not treat this aspect in detail, because it is beyond the scope of this work. The following requirements we impose on modelling procedures are all directed at effectiveness.

- *completeness*
  This and the next two requirements are aspects of the correctness of a modelling procedure. In [ISO 82] the 100% principle was formulated, stating that *all* relevant aspects of the Universe of Discourse should be described in the conceptual schema. For modelling procedures we derive from this principle two requirements: all types of concepts of a

modelling technique are treated in the procedure, and it should be assured that the informants specify complete information about the Universe of Discourse.

- *consistency*
  The guide-lines of a modelling procedure should not contradict each other and the procedure should result in consistent models. Tools may support the consistency rules by either preventing the addition of an inconsistent aspect to a model (pre-analysis, see section 6.1), or by executing a consistency verification operation after the model is finished (post analysis). An example of the first is the refusal to insert a name that is already used and an example of the latter is the balancing of a hierarchy of activity models.

- *accuracy*
  The modelling should result in a model in which the intended meanings of the informant's specifications are reflected. All statements about the Universe of Discourse that are incorporated in the model, should be true, or satisfied.

- *well defined products*
  The model resulting from a modelling procedure should be constructed in a clear, well defined way. This implies a precisely described starting point, proper steps to add aspects to the model and a final completion step. Well defined products facilitate the transfer of models to other development activities. The quality of the models produced by the corresponding modelling procedures are thus dependent on each other.

- *determinism*
  At each point where modelling alternatives appear, a rule should be given which determines the correct alternative in the given situation. In other words, for any aspect in an informant's specification it must be uniquely determinable to which concept of the modelling technique it corresponds and with which symbol it is represented in the model. For instance in Entity-Relationship modelling, it must be decidable whether a concrete object has to be modelled as an entity, an attribute or as a relationship.
  However, determinism is an objective which is very hard to achieve, because full formalisation is for most modelling techniques impossible. In the procedures we have developed, determination rules occur in some steps and the remaining steps were designed to generate no alternatives. This cannot be guaranteed, however, since it is impossible to test them in all possible cases. Further research on determinism in the design and application of the procedures is needed.

- *informant's specifications*
  It should be avoided that informants specify their information in terms of the modelling techniques, because this would lay the responsibility of the accurateness of the model with the informants, whereas this should be with the modeller. By preference the informant's specifications

should be taken from the Universe of Discourse, such as examples of the information processing.

- *relevance*
  The relevancc of the components of the model is required by the definition of the Universe of Discourse (see sec. 1.4) and the aforementioned 100% principle. It is the task of the modeller to ask for relevant informant specifications and to differentiate relevant and irrelevant information. The informant is responsible for the acquisition and specification of relevant information. The difference between what is supplied by the informant and what is needed by the modeller is a matter of their personal capabilities and their mutual cooperation.
  This requirement is a generalisation of the *Conceptualisation Principle* in [ISO 82], that stated that a conceptual model must only and exclusively include conceptually relevant aspects of the Universe of Discourse. In the same way one can state that an internal schema should only include internal aspects and an interaction model only interaction details. The objective of the development activity, for which the technique is employed, determines the class of relevant aspects of the UoD.

- *formalisability*
  The modelling steps should consist of as much as possible formalisable operations. Informal operations, also called heuristics, are in some modelling techniques unavoidable, since qualification of phenomena, i.e. relating concrete entities to conceptual entities, is inherent to modelling.
  The degree of formalisation contributes to an objective modelling process, in which different modellers modelling the same concrete system come to the same model. Furthermore, all formalisable operations can be implemented in a modelling support tool, which reduces the amount of work for the modellers. Subjectivity in modelling should be reduced to a minimum [Falkenberg 89c].

- *communicatable*
  The quality of the result of a modelling procedure is also related to the interaction between modellers and informants. The procedure should contain guide-lines for this communication. Furthermore, it should be taken into account that proper documentation is drafted during the modelling process. The development process and the subsequent maintenance stage benefit considerably from an effective transfer of knowledge by means of documentation.

- *reducing complexity*
  Most modelling techniques are equipped with constructions for reducing complexity, such as top-down decomposition and bottom-up integration. The modelling procedures should incorporate steps to set up models according to these constructions. An overall completion step must finish the modelling of all parts of such a construction. For example, a check on all rules on the hierarchy should be included in the case of a top-down decomposition technique.

- *stepwise*
  The modelling should be split into coherent steps, which are not too large and not too small. The communication with the informant or the completion of a particular aspect of the model may determine the splitting of the steps.

- *integrated*
  The development of an information system is divided into a set of highly interrelated activities, which is for a great part hidden in the models. These relationships among models should be made explicit and incorporated in the steps of the corresponding modelling procedures. The task modelling technique (see ch. 5) is related to activity modelling and data modelling, which is reflected in the task modelling procedure given in section 5.4.

2.4.3 The construction of modelling procedures

Since we are familiar with reasoning on a meta-level, we also make the construction process of a modelling procedure explicit. Obviously, structured and formalised meta-modelling are part of it. The experience of the construction of some modelling procedures is incorporated, but for a crystallised procedure it should be applied to itself. The requirements of the previous section are not all incorporated in order to avoid the repetition of text. The procedure consists of the following five steps.

1. Position technique in the development process

It should be made clear in which development activities and with what purpose the technique in question is applied. Moreover, the modelling technique requires input from some activities and the resulting model is in its turn required by other activities. These relationships are made explicit. The requirements on models completed by the procedure are formulated.

2. Design preliminary procedure

A rough preliminary procedure is set up using possibly available experience with the technique. The format and contents of the informant's specification are added. The steps should be coherent.

3. Construct meta-data models of the technique

First, a meta-data model is constructed using a structured technique, such as NIAM. Samples of completed models and of all intermediate information needed in the preliminary procedure are input for the meta-data modelling process, which can be performed according to its own guide-lines.

Next, this meta-data model is translated into a predicate calculus meta-data model. Entity types correspond with sets, relationships with predicates and all constraints with axioms. Try to construct a set of axioms that represents the intended meaning of the technique. Use auxiliary predicates whenever convenient. Theorems can be formulated and proved based on the axioms.

4. Construct meta-activity model of the preliminary procedure

A meta-activity model of the procedure is made, using for instance Channel-Agency nets. Each step corresponds to a process, substeps to subprocesses, and intermediate and final products to states. It should be possible to describe each product in terms of the meta-data model. Make adaptations in both meta-models where necessary.

5. Complete modelling procedure

Incompleteness of the procedure can be discovered in one of the following ways. The information flow through the procedure has to be controlled for missing and redundant information or processes. All concepts and associations between concepts in the meta-data model should be modelled in at least one step. All axioms and theorems of the meta-data model should be satisfied or explicitly verified by a step. The procedure can also be tested on small cases.

The steps of the procedure have to be adapted according to the outcome. It may turn out that extra steps are needed or that some should be combined. One should also check that the relationships with other techniques and development activities are proved in the procedure.

These adaptations must be reflected in the meta-models. Try to establish a meta-model with a high as possible level of formalisation and distinguish informal and formal processing explicitly. The formal part can be described in the predicate calculus model.

We did not assume in this procedure that a tool for the technique is available. If this is the case, the functionality of the tool influences the steps of the procedure. The formal descriptions of the processing should reflect the automated procedures of the tool.

2.5 CONSEQUENCES OF THE FORMALISATION

Modelling during informations systems development profits from a more formal approach. In this section we will discuss briefly the benefits of the formalisation of modelling techniques and procedures. We split the benefits in two categories: those due to the *explicit* description of the modelling process and those due to the *formalisation*. We finally address the issue of *quality assurance*.

The advantages due to the explicit description of modelling techniques are:

- **Improved modelling during development**
  Obviously, due to the availability of knowledge about the modelling process, intuitive and informal modelling will gradually disappear. Decisions, qualifications and the format and contents of informant's specifications will become well grounded, thereby abandonning the need for discussions about the application of the technique and reducing the dependency of the quality of the result on the quality of the modeller.
  Positioning of the modelling technique in the overall development process and the clarification of the rules for the transfer of models to and from activities will contribute to the management of software projects.

- **Better modeller - informant dialogue**
  The communication gap between the informants and the modellers can be bridged by formulating explicit instructions regarding the information that is needed from informants. The familiarity of the informants with the format and contents of these specifications, such as texts in natural language and samples taken from their environment, contributes to the mutual comprehension in this dialogue. The modellers should communicate with the users in terms of the informant's specifications and not in terms of arrows, boxes and bubbles. In addition, a better separation of concerns can be achieved by holding the modellers responsible for the modelling result and the informants for the informant's specification.

- **Effective utilisation of personnel**
  The approach leads to *uniform* modelling activities, which is especially important in larger projects and larger software companies. The steps of the modelling procedure can be shared among the analysts. Interviewing can, for instance, be separated from the modelling, provided that the interviewers generate complete and correct informant's specifications for the modellers. Also the assignment of personnel to specific tasks can be more effective. After a change of task the results of the modeller's predecessor can be used much more easily. Finally the straightforward guide-lines and manuals of the modelling procedures lead to more *effective training* of automation personnel. These aspects contribute to the efficiency of development projects.

Formalisation has the following consequences.

- **Conceptual clarity**
  The formulation of the concepts of a modelling technique in a mathematical formalism aims at realising the precise understanding of the concepts. Discussion about the properties of the concepts needs not to be based on their intuition, but can be based on the mathematical properties. Axioms and theorems can be formulated to prove statements about the concepts and procedures. Furthermore, if the modelling technique is mapped to a mathematical formalism that has a well

established theory, the consequences of this theory can be translated into extra knowledge of the modelling technique.

- **Automated support of modelling**
  Some of the steps turn out to be partially or even fully automatable due to the formal description of the modelling. This supports the modellers and can reduce the amount of work. When informant's specifications can be inserted into a tool and be analysed by linguistic programs that deliver the model, the modeller only has to check and direct the modelling process. A first approach to doing this is reported in [Falkenberg 88]. In the SOCRATES project, formalised expertise of modellers will be implemented in tools [Ter Hofstede 89b].

Quality is a topic that is difficult to discuss in a formal way. In the benefits mentioned above and in the fulfilment of the requirements of effective modelling procedures in section 2.4, we have indicated ways of achieving improved quality. We finally want to note that modelling according to some procedure is in a sense a form of *quality engineering*, i.e. the quality of the resulting model is assured by the performed steps, provided the steps are considered good. The latter may be due to the formalisation, because the modelling procedure becomes accepted among the modellers, or because the procedure is certified by a recognized institute. The models made using such a procedure get an implicit guarantee of quality.

All these aspects contribute to the effectivity and the efficiency expressed in the quality of both the modelling activities and the resulting model. Despite the improving functionality of tools, we claim that complete formalisation of modelling is not possible, which implies that modelling can never be performed without the intellectual activities of modellers.

3 ACTIVITY MODELLING

Activities are modelled during the analysis and design stage of a development project. In this chapter we deal with the analysis of the system at hand resulting in the descriptive activity model. The events that cause data flows from outside into the system or that occur at a regular predictable point of time, form the starting point of this analysis. The reactions to the events are modelled and incorporated in the global activity model of the system. The activities are then captured in detailed models which include both the data stores and decomposed data flows.
The design of the prescriptive activity model is performed by rearranging, extending or reducing the descriptive model of the analysis stage. This design will not be considered here.

Three diagramming techniques are used:
- a **context diagram** to depict the interaction of the system with other systems;
- a **decomposition diagram** for the reactions of events and for the hierarchical global activity model;
- a **data flow diagram**, in one of the well known conventions, to denote the details of the activities.

We will discuss in this chapter the background of the concepts of event and activity and their diagramming techniques. Modelling procedures are presented and some formal aspects of the techniques are discussed. Portions of this chapter has been published in [Brinkkemper 88ab].

Other authors may have other perceptions of the concepts of activity. In [Olle 88b], for example, activities are called business activities. In our system of concepts concerning activity modelling the concept of activity is used for modelling information processes at a global level in a system, e.g. organisation or information system. This will be discussed in relation to the concept of task in chapter 5.

3.1 EVENTS

3.1.1 Motivation and definition

The starting point for the discussion of the concepts of event and activity is that of the well known ISO report entitled 'Concepts and terminology for the conceptual schema and the information base' [ISO 82] and the extensions discussed in chapter 1. Recall from definition 1.5 that an information system (IS) is a system for recording and manipulating information. Several activities have to be identified during the analysis

phase in order provide support for this task. These activities can be twofold[1]:

1. **Reactions** to events in the environment or in the UoD. Certain occurrences in the environment or UoD are reported to the information system. The information system has a response, which we call the reaction. Examples: the processing of an incoming order; the answering of a telephone call using on-line computer consultation. The incoming of the order and the call are the events and the processing and the answering are the reactions.
2. **Control activities** in the information system. The users of the IS may wish to obtain certain data in order to control their activities as well as those of the IS itself. There may be triggers for such wishes, such as a managerial request or a possible error. These triggers are not, in most cases, explicit and are hard to formalise beforehand, whereas in the case of events such triggers need to be present.

Each control activity is of one of the following kinds:

- Retrieval or inspection of contents. Example: Display a list of withdrawals between two dates.
- Update of contents. Examples: Insertion of new stock item types; modification of information about a particular supplier.
- Correction of errors. We consider errors here to be an inconsistency of the data between the IS and the actual state of affairs in the UoD. Inconsistencies between the models of the IS (data model, process model, etc.) and the contents of the IS are assumed not to be present.

Both reactions and control activities offer starting points for the activity modelling in the analysis phase. First, for reactions we have events. Since an existing system as well as a future system will have recognizable points at which they communicate with the environment, knowledge about the events triggering the incoming and outgoing information can be collected. Secondly, we have for the control activities, some distinguished points in the system where one wishes to have control over certain data and processes. The persons responsible for the data and processes at these points will have to specify their needs for control, so that the appropriate control processes can be designed.

In contrast to data and activity modelling a common view on the concept of an event has not yet evolved. There are few methods that possess "event" as a modelling construct and those that do have diverse views on this concept.

First of all there is the Petri-net approach, of which [Antonellis 81], [Richter 82] and [Kung 86] are examples. In this approach an event is considered to be a change of conditions described by means of preconditions and post-conditions. The reaction to an event is essentially incorporated in the event itself. In REMORA [Rolland 82] an event is defined as anything that can happen at a given time. They initiate two kinds of associations: ascertain associations, which express the state changes of objects, and

---

1 The examples in this section are taken from the inventory control case described in [Olle 88a].

trigger associations, expressing that an event triggers one or more operations. Finally, in JSD [McNeile 86] event is not defined, but the concept of action is defined as an atomic event occurring at some point in time and considered to be instantaneous. In the philosophy of the JSD method entities perform and suffer actions. The behaviour of the entities in the UoD is expressed in actions, which can be ordered and constrained.

We follow the ISO-report [ISO 82] in the definition of an event.
Definition 3.1 An *event* is the fact that something has happened and is perceived in either the universe of discourse, the environment, or in the information system.

Note that we do not consider the reaction of the IS as being part of an event. Furthermore note that this definition is in the present perfect tense to express only that the happening of the event precedes its perception. It implies that current and future events are considered.

An event and its reaction can be visualised as in figure 3.1 (from [ISO82]). The report of the event, in any kind of representation, is transferred to the information system, which performs a pre-planned reaction.

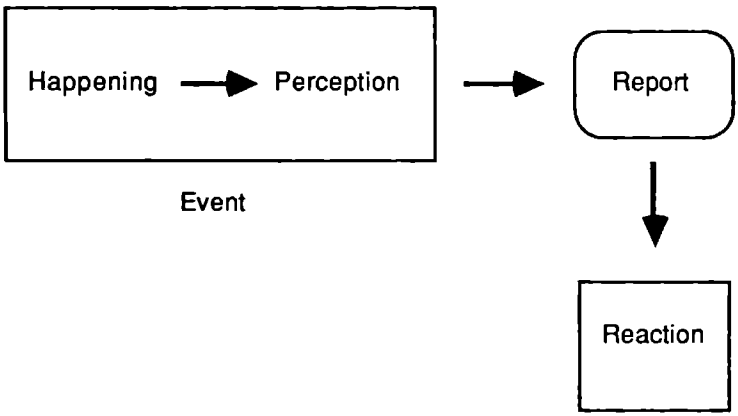


Fig. 3.1 Dependency between event and reaction

From figure 3.1 we can deduce that events have certain characteristics, which may be important to incorporate in the analysis process, namely
- event occurrence
- reporting of event
- type of event
- number of event occurrences
- frequency of event occurrences
- reaction to event
- control of event: internal or external with regard to the UoD

- stimulus of event: time driven or non time driven; controlled by signals: levels and semaphores
- reaction settlement of the event: manual control or automatic; immediate or to be fixed later

The first three, occurrence, reporting and type can be seen as the core characteristics, where the occurrence and type are analogous to entity and entity type: the incoming of purchase orders is an event type of which the incoming of a particular purchase order from the company 'Buy Ltd.' on 1st of September 1988 is an occurrence. The purchase order is its report.

The number and frequency of event occurrences, the control and the stimulus are characteristics which are important to record during the design stage, whereas the reaction and its settlement, being part of the information system, should be modelled fully in the analysis stage.

Since there is much confusion due to the large number of characteristics and the types of events already proposed in the literature, such as external event, triggering event, temporal event, business event, etc., we have developed a classification of events that provides a starting point for the modelling procedure. This classification is derived from explicit motivations as follow.

We consider three aspects of events: the impulse, the control and the settlement of the reaction.

First, the impulse is a kind of generic term for triggers due to reaching a certain point in time, reaching a particular limit value, or the evaluation of a condition. Examples are: it is Monday at 15.00h.; the re-order level of the stock item 'Paracetemol' has been passed. We consider impulses generated by a clock as special, since temporal events are a major and frequently occurring type of events. The regularity is, usually, known and we incorporate this in the modelling process.

Secondly, an event can be inside or outside the control of the Universe of Discourse. For instance the incoming of a purchase order is not under the control of the inventory control department, but the passing of a re-order level is. We represent this dichotomy by means of the notions of external and internal control respectively. This dichotomy of control is also motivated by the distinction of external observations from the environment and internal observations from the primary system in the Control Model of [Blumenthal 69].

Thirdly, in information systems we have two types of reaction settlement. Some requests have to be obeyed immediately, the so called commands. Typical of this kind are telephone calls for information and system start-up and shut down commands. Other requests can be answered at a moment which is found to be most convenient. The processing of the ordinary mail can be handled at a moment the person himself wishes. The regular preparation of all sorts of lists can also be done at a point in time when the system load is low.

Impulse and control / Reaction settlement	Time impulse		No time impulse	
	Internal	External	Internal	External
Immediate settlement	Scheduled Command (1)	External Temporal Command (2)	Internal Command (3)	External Command (4)
To be fixed later	Scheduled Request (5)	External Temporal Request (6)	Internal Request (7)	External Request (8)

Fig. 3.2 Classification of events

When we combine these three two-valued aspects we get 8 possible types of events. They are shown in figure 3.2.

We have given each type a name and a number for convenience. We call events which need immediate settlement "commands" (1,2,3,4), and events which do not need this we call "requests" (5,6,7,8). Some of these requests may in fact be offers, if they offer some new data to the system, but since these are implicitly accompanied by a call to enter the data, we also call them requests.

We now give an example of each of the above types of events taken from the inventory control case, where the time events are listed with a possible reaction of the information system:

1. On 31st of December at 12.00h. a list of the total inventory has to be created.
2. Each Monday around 12.00h. the financial department phones to check all open purchase orders.
3. A stock item passes the re-order level.
4. Someone from a branch-establishment phones for information on a particular stock item.
5. On every first Friday of the month the total value of the stock items with stock item type 'Capital expenses' has to be given.
6. Every second Wednesday of the month the kitchen supplies are delivered.
7. Create the statistics of the purchasing of stock items.
8. A delivery arrives.

We reduce this classification for two reasons as follows. The settlement of the reaction is modelled and realised in the design phase. The immediate settlement of commands can be handled by giving the reaction processes a high priority in the process schedule design. Information obtained about

the need for immediate reaction is put into the process description. Hence commands and requests are treated identically in the analysis phase.

Despite the fact that we know that there are external time driven events (2,6) we cannot fully anticipate these events, due to the fact that we do not have control over external processes. If such a process fails to deliver its input for the system, without letting the system know, the system cannot keep on waiting for it. Hence we treat them just like non time driven external events (4,8). The regularity of these events can however be used to plan the load of the system.

Three classes of events are left:
1. external events (2,4,6,8)
2. internal temporal events (1,5)
3. internal non-temporal events (3,7).

We can now give formal definitions of these types of events.

Definition 3.2 *External events* are events which are perceived in the environment. *Internal temporal events* are events which are perceived in the UoD or in the IS and are time driven. *Internal non-temporal events* are events which are perceived in the UoD or in the IS and do not have time as their impulse.

External events will be modelled in the external event analysis (3.2.1). Internal temporal events will be modelled in the internal temporal event analysis (3.2.2). Their scheduling is modelled in the design stage. Internal non-temporal events form the most difficult class of events to model, since there are in general a large number of internal stimuli that trigger processes: passing of levels, flags, etc.. We consider the perception and the reporting of this kind of event as independent processes and they will be modelled as such (see 3.4).

This concludes the motivation of the treatment of the concept of event. Starting from activities of an information system we distinguished aspects of an event and came to a classification. How does this classification relate to other classifications in the information systems literature? First, the presented classification resembles the classification of events of Lindgreen given in [Lindgreen 86]. External events correspond to externally initiated events, internal temporal events to period initiated events and internal non-temporal events to act initiated events. Control activities correspond to agent initiated events, since the latter lack a formalisable happening. Secondly, the ADISSA method [Shoval 88] distinguishes user event, time event, communication event, real-time event and internal event. The event types of this classification are not mutually disjoint. The user and the internal event of this method are called internal non-temporal events in our classification. The time event is generic for all events with a time impulse. A real-time event can be seen as a scheduled command or as a external temporal command. Finally, communication events are the same as the external non temporal events.

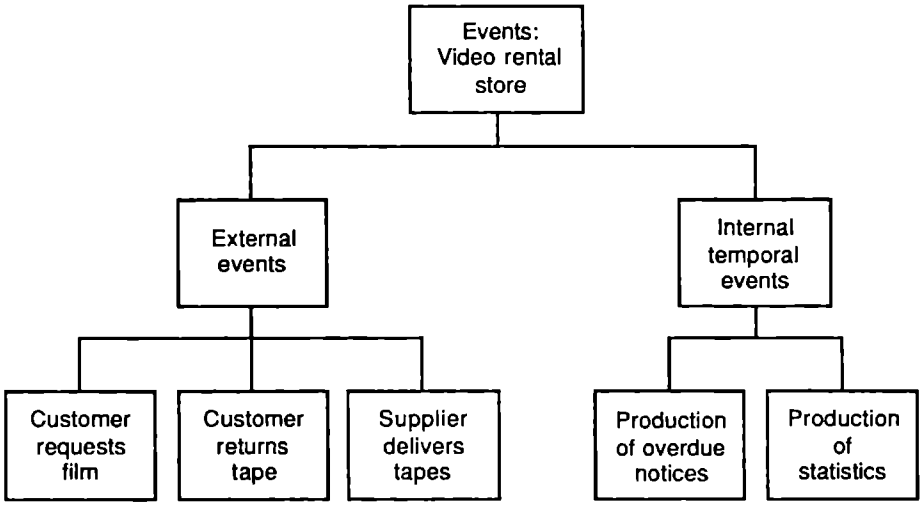


Fig. 3.3 Event decomposition diagram

3.1.2 Diagramming events

Since few methods possess the concept of event, there exists no commonly agreed way to model an event diagrammatically. We discuss two diagramming techniques inspired by the techniques of the Information Engineering method [Martin 88a], in which some aspects of events are separately modelled.

First, a decomposition diagram is used to gather the external and internal temporal events. An example is given in fig. 3.3

We have put in this diagram the events of the video rental store of appendix A. The use of a decomposition diagram is not so properly, because the first decomposition layer in the figure is used as a generalisation or subtyping layer, whereas the second layer is used as an instantiation layer. In general, decomposition diagrams should be used to decompose components of the same type. We nevertheless allow this misuse because event decomposition is only performed as an intermediate specification aid.

From the information about these events a context diagram for the system or organisation is constructed. A context diagram is a special type of data flow diagram. Examples are shown in the figures 6.3 and A.1. In a context diagram the interaction of the system at hand with other systems is depicted. The other systems are called external agents and are denoted by a shadowed box. The data flows of the system to and from the external agents are represented by arrows.

Some conventions may hold for this diagram:

- There is just one central system, on which the analysis activities are focussed. Refinement of the specification is modelled by decomposition during activity modelling in various levels.
- The interaction between the systems is modelled only with respect to their data exchange. This means that only data flows and no material flows may occur in the context diagram. Some flows may seem to be material flows but this is because of the data that is physically inseparable from the material. For instance the data flow 'Returned tape' in the context diagram of fig. A.1 refers to the name, tape number and other data that are printed on the box of the tape. This rule also ensures that the decomposition of the system contains only data flows.
- Data stores are not allowed in context diagrams, because the context diagram models the interaction of systems via data flows. A system never updates or retrieves a data store directly, but supplies update or retrieval requests to another system in which that data store is embedded.

If these conventions hold for the context diagram, then this diagram shows the interaction of the system with other systems via data exchange in its simplest form. The context diagram serves then for the precise determination of the scope of the system and hence also to establish the scope of the analysis stage. It is also a reference point for the modelling of the activities (see 3.4).

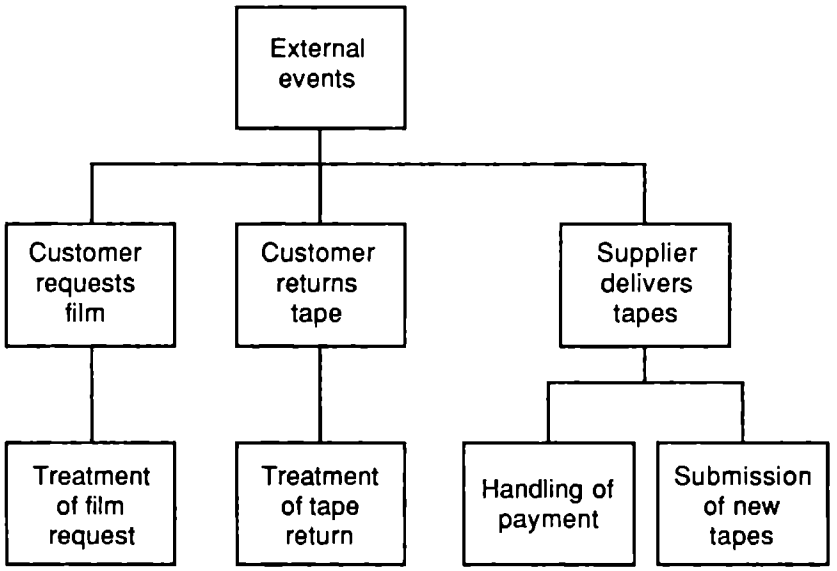


Fig. 3.4 Reactions to events

Finally, the reactions of the events are modelled as an extension to the available event decomposition diagram. The activities of a reaction are placed under the associated event in the diagram. Their sequence in the diagram is determined by the triggering sequence of the activities. An example is shown in fig. 3.4.

Alternatively, these reactions may be specified in a data flow diagram. The activities, being part of the reaction, and the intermediate data flows can then be modelled including the processing sequence of the activities. This is recommended for systems with a complex event structure, but for most administrative information systems the global decomposition diagram will be sufficient.

3.1.3 Meta-model of event modelling

The main aspects captured in the three aforementioned diagrams are modelled in the meta-model in fig. 3.5.

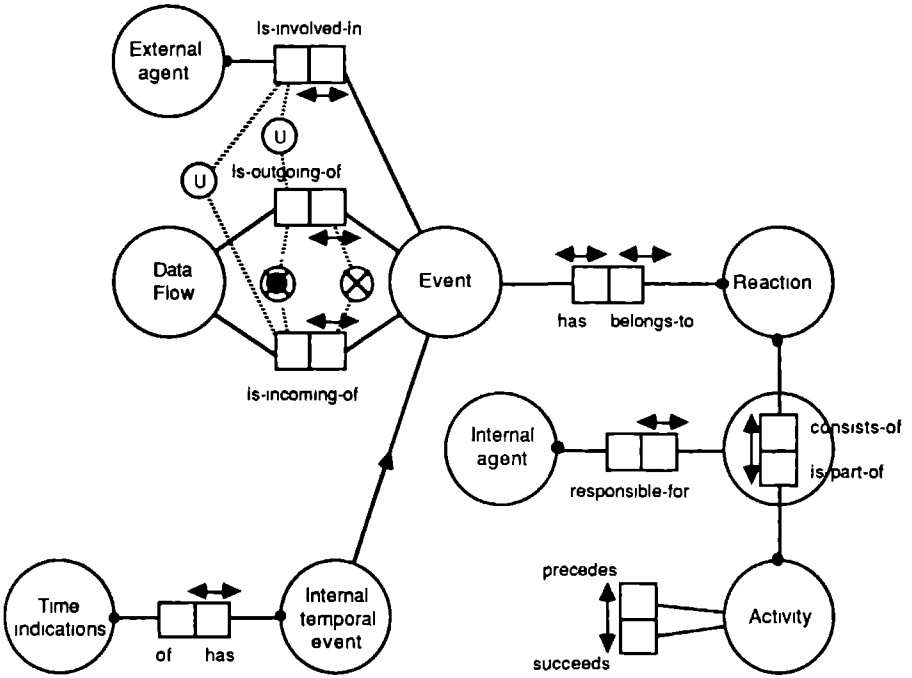


Figure 3.5 Meta-model of event modelling.

We see in this figure that an event has a unique reaction and a reaction belongs uniquely to an event. Parts of reactions may be similar to each

other or have the same name, but for the sake of a strict modelling procedure it is desirable to obey this one to one relationship. The reaction consists of a group of activities, of which the sequence is specified. In case an event occurs in the context diagram, which is true for all external events and not for all internal temporal events, the corresponding data flows and external agents are specified. Therefore no totality constraint is specified at these fact types. A time indication is recorded for internal temporal events.

Internal agents are responsible for the activities, that are part of a reaction. The internal agents are persons or organisational units of the system and could also be called subsystems. Internal agent is a central modelling component in activity modelling (see 3.3 and 3.4).

Some of the constraints are interesting. All data flows are either coming in or going out of the system and an event has either an incoming data flow or an outgoing data flow. The external agents have unique incoming and outgoing data flows. Furthermore an equality constraint exists that cannot be shown in the meta-model. The set of external events is the same as the set of events that have an incoming data flow.

In the meta-model the external event could as well be incorporated as a subtype of event and agent could be the supertype of internal agent and external agent, if special facts about these types were to be recorded. We decide not to do so for the sake of simplicity. Note furthermore, that some of the entity types of this meta-model occur in the meta-model of activity modelling (fig. 3.7).

3.2 EVENT MODELLING PROCEDURES

3.2.1 External events

The analysis of events is based on the analysis of information given by the informant. The informant must be instructed in the format and style of the data to be supplied. The modeller can then perform some steps in order to determine the required models. The modelling procedure consists of the following 5 steps.

Step 1: Gather required information.

In the system there exist subsystems, like an organisation is divided into organisational units, that are responsible for the reactions to the external events and that treat the incoming and outgoing reports of the events. These subsystems must be investigated for their interaction with the environment. An association matrix of entities versus processes available from a preceding information strategy planning phase may contain useful information. The processes that create entities should have some triggers, possibly external to the system. One or more external events should belong to those processes. If knowledge or documentation about these processes

already exists we use this, and if not, the appropriate persons in the system are interviewed to obtain the following information:

- Name of the event.
- Description of the event: the precise happening and the type of report.
- Concise description of the reaction.
- The subsystem responsible for the reaction. These are mostly organisational units and we call these units **internal agents**.
- The report of the event; preferably some samples of reports which can be used in the data modelling (see chapter 4).
- The external agent involved.

The description of the event should be in active voice, or else it should be transformed to it, with the system as the acting agent. Typical verbs in this kind of events are: receive, inform, obtain and answer. The event is then placed in a decomposition diagram as is shown in fig. 3.3.

Step 2: Analyse description

A linguistic analysis is performed on the description of the event. The subject is usually the person responsible for the reaction, the object describes the data flow involved, the verb clause describes the event happening or perception and the objects after the prepositions are the external agents.

Example:

```
    Store assistant receives returned tape from customer
```

gives

```
        Event:              Reception of returned tapes
        Internal agent:     Store assistant
        Report:             Returned tape
        External agent:     Customer
```

Note that this event is not described as: 'Customer gives returned tape to store assistant', because this is not from the viewpoint of the system and it is obvious that we do not have any control on the bringing of these tapes in this case. The information gathered in step 1 should already conform to the required format.

Step 3: Construct context diagram

The context diagram, as discussed in section 3.1.2, is created in accordance with the analysis of the preceding step. The external agent is placed in the diagram and a data flow labeled with the report name is drawn from the external agent to the system. In figure A.1 this is shown for the example together with data flows from other events of the decomposition diagram in fig. 3.3.

External events have just one external agent. When two types of reports from distinct external agents are involved, we split the event. When there is one type of report and distinct external agents, we either have two flows (and so two external agents) if the external agents occur individually in

other events, or else we have just one flow and one external agent, being the combination of the initial two.
Examples:

```
1. The Store administration receives empty order forms
from suppliers and video catalogues from possible
suppliers.
2. The Store administration receives video catalogues
from suppliers and possible suppliers.
```

In example 1 this event is split into the event of the receipt of empty order forms from suppliers and the event of the receipt of the video catalogues from possible suppliers. If in the UoD of example 2 the external agents suppliers and possible suppliers occur separately in different events, we have two external agents in the context diagram, both connected with the data flow 'video catalogues' to the system. Otherwise there is just one external agent, namely 'Suppliers and possible suppliers'.

Furthermore the exclusion and uniqueness constraints of the meta-model of event modelling (fig. 3.5) have to be ensured by choosing a unique combination of the names of the reports and the external agents.

Step 4: Model reactions

The reactions to the event are modelled in global terms using the description available from step 1. The starting point of the reaction is the receiving of the report by the internal agent. After this some activities are performed until processing stops in the system or until some kind of data object is send out of the system to some external agent. The activities are taken on some global abstraction layer and included in the decomposition diagram as shown in fig. 3.4. The simplicity of the video rental store organisation causes the simple reactions in this model.

Frequently the reaction splits into several sequences of activities, for instance the arrangement of the payments and the sending of the ordered goods. The sequence of the activities in the decomposition diagram should reflect their sequence of processing, where a linearisation of possibly simultaneous activities is employed. The internal agents responsible for the activities of the reaction may already be recorded.

Step 5: Add outgoing data flows to context diagram

Some reactions in the preceding step may give rise to data flows that go out of the system to external agents. These data flows reflect the interaction of the system with the environment as well and should therefore be modelled in the context diagram.

This is done by treating the reaction's final activity in the system, which has the outgoing data flow, as an event. The outgoing report of this event is denoted as a data flow and the receiving system as the external agent. Special care has to be taken that this step does not introduce external agents that are synonyms for others.

End of external event modelling procedure

Note that not all information gathered in this modelling procedure is represented in graphical models. Some information is recorded elsewhere to be used in subsequent analysis steps. For instance the information concerning internal agents will be modelled in the activity model (see 3.4). Fig. A.1 is the result of the modelling procedure on the video store case, in which the internal temporal events are taken into account as well. These will be discussed in the next subsection.

3.2.2 Internal temporal events

Internal temporal events are modelled analogously to the external events. The difference is that there is now a time indication instead of an external agent. So this can never result in an incoming data flow in the context diagram, but the event may lead to an outgoing data flow with a corresponding external agent. For the description of the procedure we give only the adaptations below and note that step 3 has no counterpart.

Step 1: Gather required information

The informants must specify which events are regular, e.g. each week, every first Wednesday of the month, and which events are related to processes by time intervals, e.g. 30 days after a purchase order has been sent and no delivery has arrived, a reminder notice is sent. Matrices from the information systems planning phase could be a starting point for this step as well. Similar information about the internal temporal events is gathered as is done for external events.

Step 2: Analyse description

The event happening, the internal agent, the report and the time indication are lifted out from the event description.
In the video store case of appendix A two examples of internal temporal events occur:

```
The store administration produces overdue notices every
day at 17.00h.
The store administration produces statistics of the video
rentals every last day of the month.
```

The analysis of the first example gives:

```
Event:              Production of overdue notices
Internal agent:     Store administration
Report:             Overdue notices
Time indication:    Every day at 17.00h.
```

Step 3: Model reactions

This step is the same as for external events. In general the reaction to an internal temporal event consists of few activities inside the system, because

the time trigger is mostly used to reactivate an interaction with an external agent. Typical examples are reminders for payments and renewals of memberships.

Step 4: Add outgoing data flows to context diagram

This step is also the same as for external events. Of the two examples only the event concerning the overdue notices has an outgoing data flow as is shown in fig. A.1. The production of statistics is done for internal purposes of the video rental store and so no outgoing data flow is in the context diagram.

End of internal temporal event modelling procedure

Note that the time indication of internal temporal events is not modelled in either the context diagram or the decomposition diagram. This information is recorded to be incorporated in the realisation of that part of the information system that supports the event in question.

At the point, when both external events and internal temporal events are modelled, it is required to check the completeness of the context diagram. By then, it should depict all interactions of the system with the environment, i.e. all incoming and outgoing data flows to and from external agents. All subsystems that interact with the environment should be verified in order to ensure that all communication with externals is incorporated in the context diagram. In this way one will also encounter internal non-temporal events that result in outgoing data flows. This type of events can be handled in the same way as internal temporal events.

One could think of various ways of automated support for these modelling procedures. For the steps that gather information, special dialogues and screens can be designed such that the input is directly given in the required format. Then the steps that transform this information can deduce a possible outcome, which is to be approved or modified by the analyst. The links with other models, such as the association matrix of processes and entities, could be employed to implement the completeness of the dependent information, as discussed above.

3.3 ACTIVITIES

We will not spend as much time on the clarification of the concept of activity as was done for event, because this concept has a generally accepted meaning and is employed in all kinds of diagramming conventions. We refer to section 5.1 for a discussion of three types of processes: activity, task, and operation.

3.3.1 Definition and diagramming of activities

Several diagramming techniques for activities have been proposed in the literature and some of them are widely accepted in practice. We mention SADT-diagrams [Ross 77], Data Flow diagrams according to the notation of Yourdon [Yourdon 79] or to the notation of Gane and Sarson [Gane 79], A-graphs of the ISAC method [Lundeberg 80] and Activity diagrams in the German variant of ISAC [Scheschonk 84].

From these techniques we observe that an activity in an activity model has the generic form shown in fig. 3.6.

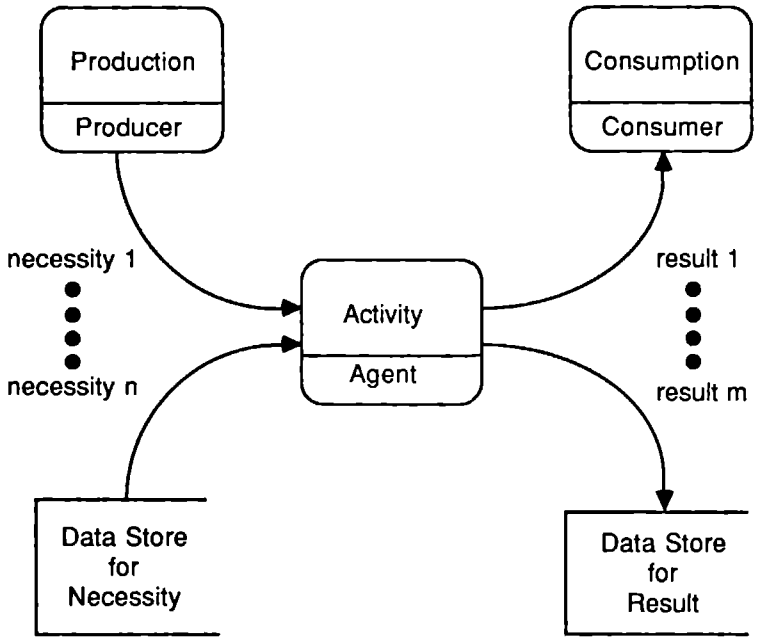


Figure 3.6 Generic form of activity

We define therefore the following components of an activity model.

Definition 3.3

a. *Activities - Data Flows*
An *activity* models a process that processes instances of data compounds (see def. 5.4). A data compound models a composition of entities or associations among entities (see def. 5.3). Data compounds are denoted as input or output of activities. Data flow, message set or state are other names for data compound in the mentioned techniques. In this chapter we will use the term *data flow* instead of data

compound. The well known decomposition rules hold for activities and data flows.

b. *Agents*
An *agent* is any kind of system which either performs activities or is responsible for them. Persons, organisational units or subsystems are possible types of agents. *Internal agents* are agents inside the Universe of Discourse and *external agents* are agents in the environment. Activities have just one agent.

c. *Necessities - Results*
All data flows that are input to an activity are called *necessities*. The data flows that are output of an activity are called *results*. There may be any number of necessities and results for an activity.

d. *Production - Consumption*
A *production activity* of a given activity is an activity that has as output a necessity for the given activity. A *consumption activity* of a given activity is an activity that has as input a result of the given activity.
An activity can have more than one production activity, as well as more than one consumption activity.

e. *Producer - Consumer*
A *producer* is the agent of a production activity. A *consumer* is the agent of a consumption activity.

f. *Data Stores*
A *data store* is an intermediate depository for information, necessary for or resulting from of an activity, and which is required to be accessible by various agents. A data store is introduced whenever this is explicitly stated or when necessities and results need to be kept for use by others.

From these ingredients, activity models are constructed. In general, activities are composed of subactivities, and therefore data flows may have subflows. Data stores may not have substores because the decomposition process is only performed on the activities. Proper decomposition rules, also referred to as balancing rules, apply for the subdivisions. Agents may be decomposed as well and it must be assured that the agent responsible for a subactivity is part of the agent of the corresponding super-activity. The modelling of activities in several decomposition layers results in hierarchy of activity models. We refer to this hierarchy of models as the *global activity model* and to any activity model, that is part of the global activity model, as a *local activity model*. The context diagram is the activity model at the top of the global activity model and in this way links the event modelling to the activity modelling.

In [Dijk 86] more components related to activities are distinguished next to the ones mentioned in definition 3.3: means, time, reason, condition, location, and aim. Although the information gathered by analysts incorporates data on all these components, we deliberately omit them because they are not modelled in the well known techniques. However, it may be important to record the data on these aspects as comments to the activity models for other development purposes.

The expressiveness of data flow diagrams is therefore restricted. Choice, triggering and iteration cannot be expressed. In chapter 5 we will introduce a new modelling technique, called the Conceptual Task Model, in which these aspects as well as the processing of data can be modelled. In this technique the activities of the bottom layer of the activity decomposition, the so-called tasks, are taken and extended with more detail, such as the input, output and intermediate data models.

In [Berdal 86] another extension to the data flow formalism was proposed in order to express the relations among input and output flows. This technique, called Process Port Modelling, extends the specification of the input and output side of an activity with a port that buffers the data. The occurrence of multiple data flows can then be combined, using an AND-port, or excluded, using an OR-port. Data flows are divided into three categories: signals, parameters and information objects. Process Port Modelling provides a formal description of the relations among flows and can therefore be seen as an extra layer on a data flow diagram. (for layered modelling see section 6.5)

3.3.2 Meta-model of activity modelling

In fig. 3.7 the meta-data model of activity modelling is shown. The introduction of the supertype 'Activator' for external agents, stores and activities makes bi-partite graphs of data flow diagrams. This simplifies the meta-model and the formalisation. We will see in section 3.5 that some extra rules hold for the diagrams. The rules depicted here include, among others, that a data flow may not loop (shown by means of the exclusion constraint on the combined roles) and that an activator is the source or the destination of a data flow (by means of the combined totality constraint).

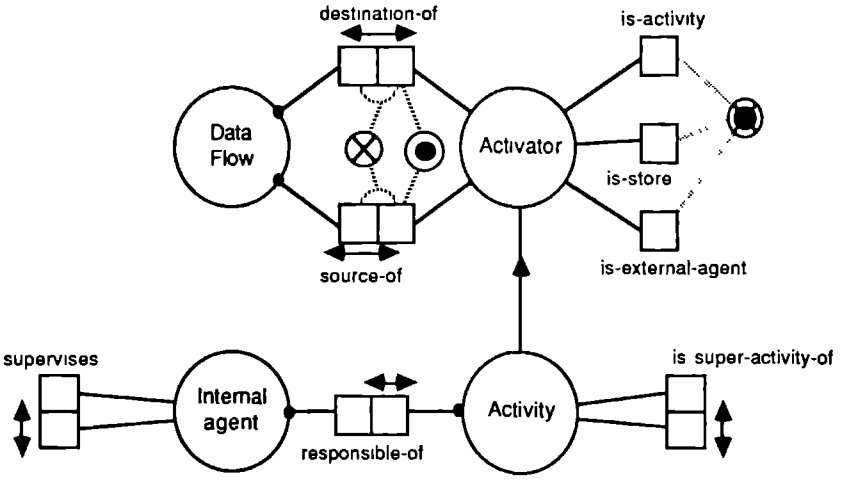


Figure 3.7 Meta-model of activity modelling.

In a plain data flow diagram a data flow has just one destination and one source. However, decomposition of activities may introduce the same data flow on several levels, hence these relationships are many to many.

The decomposition of activities is denoted by the homogeneous fact type 'Activity is-super-activity-of Activity'. Each activity has one internal agent, who is responsible for it. A hierarchy of internal agents, denoted by the fact type 'Internal agent supervises Internal agent', is derivable from the decomposition hierarchy of the activities. Since this hierarchy might be conflicting with the actual hierarchy in the organisation, its derivation serves as a check on the model and possibly as a indicator for a badly structured organisation.

3.4 ACTIVITY MODELLING PROCEDURE

The modelling procedure for activity models consists of eight steps. The procedure has to be applied several times recursively in order to construct with the complete global activity model.

Step 1: Gather required information

The form of informant's specifications for activity modelling are texts in natural language which specify the activities and the flow of information. From section 3.1 we summarise that there are three kinds of activities in an information system:
1. Reactions to events;
2. Control activities;
3. Perception and reporting of internal non-temporal events.

Activity modelling must make use of the results of the preceding planning and analysis steps of the development process, such as for example the event modelling procedure which gives the reactions. For all three kinds of activities, proper starting points for information gathering have to be distinguished.

Ad 1. The internal agents responsible for the reaction processes are asked to specify the whole reaction process in accordance with the list of events and the decomposition of the reaction, both of which have been modelled during event modelling (see 3.2).

Ad 2. The control activities are treated as follows. In the case that the analysis stage is preceded by a planning stage, we have a rough insight into the entity types and the business functions from the models of this stage. The responsible agents are asked to specify the entities, relationships and processes for which control is needed. In general, there must be control processes to check the whole content of the information system, except, perhaps, for very stable and well-known data.

The activities related to a control action can be simple, e.g. the updating of an entity or the printing of a list of entities on a screen, or more complex, e.g. a complete overview of the involvement of all entities in the processes of the system. A narrative description of the control action and the related activities is then set up by the informants. The analysts must take account of the fact, that control of the same information may be needed at several places in the organisation.

Ad 3. Internal non-temporal events are, in general, harder to discover. The analysts must be aware of this and as soon as they get the impression that certain processes are initiated due to internally created triggers, these triggers must be recorded on a special list. Also attribute names can suggest an association with the triggers involved. For instance the re-order level of stock items does so. Furthermore may the construction of the context diagram already have resulted into some internal non-temporal events, as discussed in section 3.2.2.

Step 2: Determine preliminary activities and flows

The texts available from step 1 have to be analysed for activities, agents, necessities, results and data stores. The most convenient way to do this is for each kind separately. First we will discuss the marking of each kind in the text and subsequently give an example.

2.a Activities
The information processing activities are marked first. The most simple way of doing this, is to mark all information processing predicates in the sentences. Transitive verbs are typical verbs indicating these activities, since the processing of the information by the system implies subprocesses with corresponding agents and objects. Examples of transitive verbs are create, design, produce, control, inform, warn, send, receive, transform, etc. Verbs not indicating processes are the auxiliary verbs and the intransitive verbs: be, have, can, must, exist, live, etc. The latter verbs are merely related to information, that is needed for data modelling.
The names given to the activities should be distinct from the names for the data flows. For instance the activity called 'production planning', that has as result the 'production planning', causes confusion. Use the imperative for activities for a clear distinction, in this case 'plan production'.
One has to take care in case the activities in the description do not process information but material goods. The material flow may only be modelled if there is data inseparable on the material and this data is processed by the activity. This has to be treated similarly as described for the context diagram (see 3.1.2).

2.b Agents
The agents are mostly the subjects or the by-phrases in the sentences. When modelling a current situation there has to be at least one agent for every activity scanned in the previous step. For a missing agent in a future situation it is very important to note this explicitly and plan to establish

one. Representatives of the agents can be interviewed to refine the descriptions of the activities.

2.c Necessities - Results
The sentences are scanned for the objects that are needed for an activity or produced by an activity to be modelled as necessities and results respectively. The analysts deliver for each activity zero, one or more necessities and zero, one or more different types of results.
Creative information processing activities with results and without necessities are seldom. They may appear when knowledge of agents is used to create the information. If this knowledge is not formalisable and will not be incorporated in the automated system, it is not explicitly modelled as an necessity. The same argument holds for activities with necessities but without results.

2.d Producer - Production - Consumer - Consumption
Most informants are better acquainted with the agents that perform the previous or the next step in a process than the precise activity of these agents. Either the agent, or the activity or both are marked in the text. This information will be needed for the composition of activities (see step 5).

2.e Data Stores
The necessities and results in an information processing system are sometimes kept in some depositories, either computerised or not, to enable the synchronisation of activities or the retrieval of data by several agents. Informants have to be instructed to specify this explicitly. Either the data goes directly from one activity to another or it is kept somewhere. In the latter case the depository is modelled as a data store.

The following is an example of a textual specification of some activities from the inventory control case described in [Olle 88a]. The activities are underlined, the agents boldly underlined, the necessities and results in italics, the producers or consumers are in bold type, and the data stores are in underlined italics.

> Each *delivery form* received is first checked by **the Stock department** with the matching *purchase orders*, that are kept in a *purchase order file*. It is possible that a supplier may combine several purchase orders in one delivery. When only a part of the purchase order has been delivered, the *purchase order* is marked as such and the delivered quantities of each stock item are recorded. When the complete *purchase order* has been delivered, this is also recorded.
> After this a *message* is given to **the stock control personel** to unpack and store the delivered goods. The *delivery form* is forwarded to **the financial department**.

Step 3: Establish naming conventions

In most organisations and therefore in most information systems development projects it is rare that there exists a common terminology, in which each term is uniquely defined. Synonyms, homonyms and multiple homonyms may occur quite often, especially the terminology of different subsystems may vary considerably.

After the previous step, in which objects have been named, the terminology of the system must be established rigorously by specifying precise definitions. In these definitions the distinction with closely related objects should be made explicit. This may lead to an adaptation of the terminology conventions of some parts of the system. Data modelling, that is usually performed after activity modelling, can profit from this established terminology or even define it more precisely.

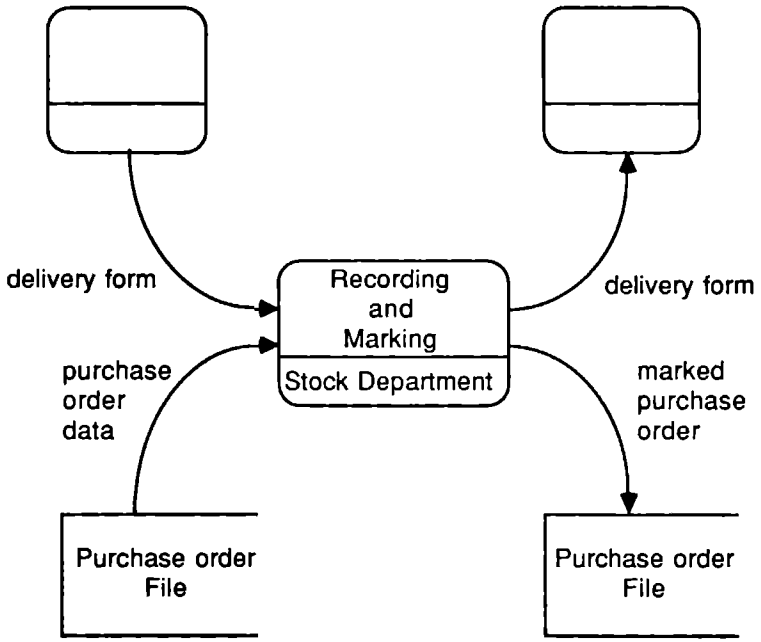


Figure 3.8 The activity Recording and Marking

Step 4: Preliminary construction of activity models

From the linguistic analysis of step 2 the template of fig. 3.6 can be filled in for each activity, although, of course, not all components exist. The partial data flow diagram for the activity "recording and marking" from the

example is shown in fig. 3.8, where only the information in the sentences 'When only a part ... is also recorded.' is represented.

The *completeness per activity* can be achieved by asking the informant to improve the textual specification with the lacking components in the model of the activity.

Step 5: Composition into data flow diagrams

We can now compose the preliminary models of each activity into one diagram by coupling the data flows. This implies that we have to investigate the activities for sequentiality and parallellism.

Two activities are *sequential* if any of the results of one activity is a necessity for the other activity. Hence the results determined in the previous step must be mapped onto necessities. Mostly this is implicit or trivial in process descriptions. For sequential activities of different agents we must map the triplet *(result, agent, consumer)* onto *(necessity, producer, agent)* in order to be sure that the activities match. In the example above this can be done with the triplet *(stock department, delivery form, financial department)*, when the description of the processing of the delivery forms of the financial department becomes available. The two activities are then connected with the data flow that stands for the result of the one activity and for the necessity for the other.

Two activities are *parallel* when none of the results and necessities is a direct or indirect necessity or result for the other activity. Or in other words, when there is no flow of information from the one to the other. There is not a special notation for parallel activities other than the lack of data flows between the activities in the model.

For a good model of the processes it is very important to have both types of relationship of activities analysed explicitly and validated by the informants. In the example the checking is sequentially before the recording and marking. The forwarding of the delivery form can be parallel to the giving of messages.

Having the model composed, this may possess gaps or data flows, that do not fit. The specification of the missing components by the informants will lead to the *completeness per diagram*. The proper balancing of the data flows and activities over the various levels of the hierarchy of activity models may as well give rise to adaptations. The resulting data flow diagram of the example is shown in figure 3.9.

Step 6: Combination into super-activities

The informants specify the activities starting at an arbitrary abstraction level. Some may start with a very detailed low level description, whereas others specify the activities at a more global, high level. It is hard to influence this, so we take this as given. We proceed then by constructing

the levels above the available activity models until the connection with the context diagram is made. This usually is a modeller's task, for which some information from informants is needed to draw the right boundaries between the various activities.

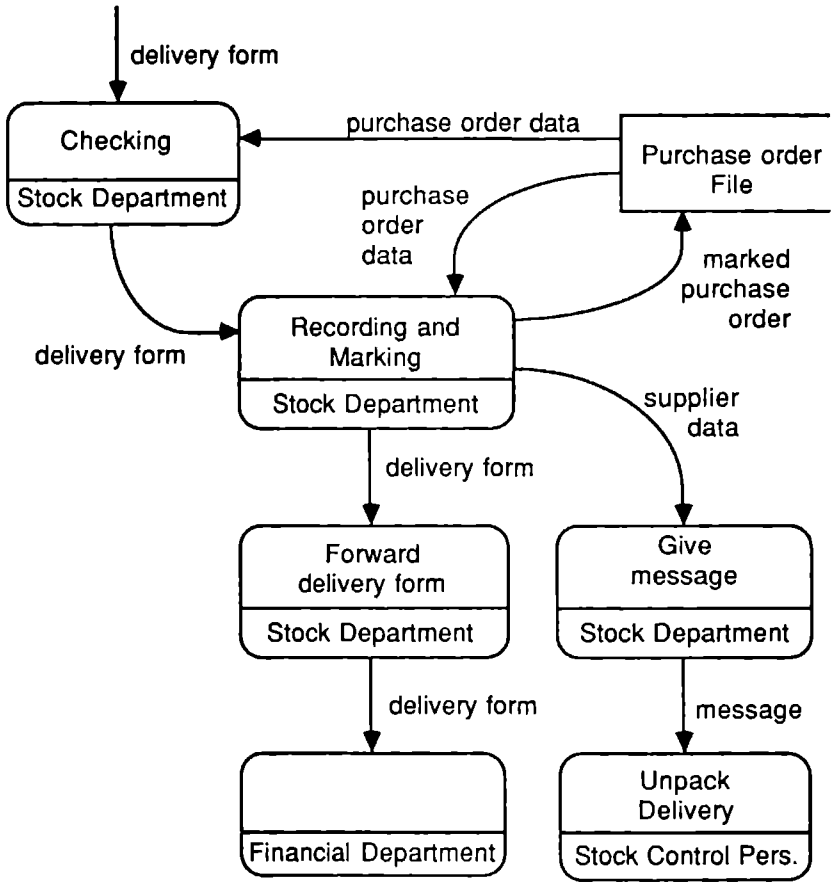


Figure 3.9 Data flow diagram of example

6.a. Grouping of activities

When a group of activities forms a natural unit, usually because the activities are performed by the same agent and/or are always performed together, we may introduce a super-activity that combines these activities. The results of this super-activity are the results of the activities of the group that are necessities of other activities outside the group. These results may be combined into one or more results of the super-activity. The same

applies for the necessities. Data flows that are intermediate, i.e. only result and necessity of activities that are combined in the super-activity, are not shown in the higher level. In figure 3.10 the super-activity is shown that results from the combination of four activities performed by the Stock Department in the activity model of fig. 3.9.

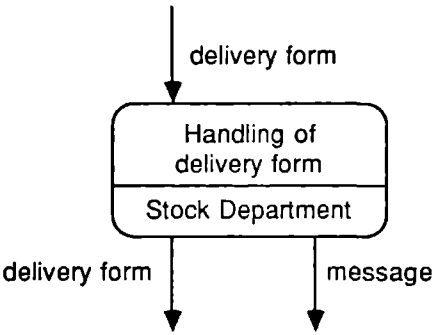


Figure 3.10 Super-activity of example

We see that the super-activity has one necessity and two results. The data store Purchase order file is hidden in the activities on the lower level. This is explained in the next substep.

6.b. Handling of data stores

A data store should be modelled in the decomposition of the highest activity in the activity hierarchy, for which it is functional. Only the subactivities of the activity in question can exchange data with the data store. In other words, the data store has only data flows to and from the subactivities. This leads to the hierarchy of activity models in which all retrieval and update processes of a particular data store occur in the sub-hierarchy of the activity in which the data store is modelled. A decomposition representation of the global activity model, in which the models with a data flow to a data store are marked, can be used to illustrate the fragmentation of the interaction with the stored data over the activities (see fig. 3.11).

It is also shown in figure 3.11, that not necessarily all sub-activities of an activity with a data store need to have data flows to that data store in their decomposition, but at least one of the subactivities should. This implies that there exists a path of activities from the highest activity in the hierarchy to a certain bottom activity, that form a chain of subactivities and all have flows to the data store.

Figure 3.11 Marked global activity decomposition

During the design phase the global activity model of the analysis phase is eventually reconfigured or new data stores may be introduced. For the global activity model of the newly designed system the same rules apply as for the model of the analysis phase as described above. If the data will become stored centrally then the data store will in general be in the decomposition of the context diagram, unless all but one subactivities of the context activity at the top of the global activity model do not exchange data with a store, which is rarely the case.

We remark, in concluding this step, that one should not take the context diagram as a starting point for informant's specification. This is because of the fact that the informants are then forced to think in terms of the decomposition. A better familiarity with the activities is achieved if the abstraction level of the informants is taken as starting point.

Step 7: Refinement in sub-activities

Hierarchical decomposition of activities can be done by asking the informant to refine a specific activity in a textual explanation to which the same modelling procedure can be applied. This is a kind of recursive application of the modelling procedure, that therefore requires a stop criterion. This is the so-called **sample criterion**. This means that an activity is decomposed as long as the input and output flows are data compounds, e.g. forms, letters, etc., of which samples have to be provided for the subsequent data modelling. Refined modelling of the activities at the bottom level of the global activity model, the so-called tasks, is done during task modelling. The sample criterion will be discussed extensively in section 5.1.

The refinement takes the activity with all its necessities and results as a starting point. The incomplete description of the recording and marking activity of the example obviously needs further clarification, unless it turns out that this would violate the sample criterion.

Step 8: Overal completion

The rules of the top-down decomposition of the activity hierarchy are incorporated in the previous steps. The result of these steps, the global activity model, should therefore obey all the decomposition rules and we already checked the completeness per activity and diagram in the steps 4 and 5. The complete decomposition may, however, be adapted to the obtained insights regarding the complete information processing. A global activity model, that is constructed by several people or that was based on specifications of different informants, will in general need some adaptations. Indications for adaptations are:

- the splitting of too complex activities in more subactivities and the combination of too simple activities in one activity;
- the reconfiguration of activities in order to have more or less the same abstraction level of processing on the levels of the hierarchy;
- the check that all data stores are properly filled and maintained.

End of activity modelling procedure

From this modelling procedure guide-lines for the informant's specification can be derived. If an informant knows the format of the specification and the concepts that must be included, this modelling can reduce expensive interviewing. Moreover, frequently occurring incompleteness or inconsistencies can be solved by returning the analysed text provided with comments about missing information.

Automated analysis tools for textual informant's specifications have been proposed in [Kersten 86] and [Cordes 89]. These approaches suffer from the combination of event, activity and data modelling in one complex analysis. Extensive case handling and error feedback to the user need to be incorporated in the automated procedures. A better support of modelling could be obtained from strictly separated analysis tools, that are fed with informant's specifications dedicated to just one aspect.

As discussed in section 3.3.1 informants often mix their specifications with aspects of activities that can not be modelled in an activity model. So we did not incorporate modelling steps for these aspects. However, in the case of conditional processing, experience shows that most informants only state the alternative corresponding to a correct outcome of the conditions. Complete specification requires that all choice alternatives are described fully. An indication of conditional processing is when activities are formulated by the informant with verbs like check, control and compare.

Beside this, we may not wish the specifications of the informant to be strictly devoted only to activity modelling. They normally may contain all

sorts of other kinds of information such as problems, data modelling information, etc.. This is not relevant to the modelling of activities, but must be kept for other development activities.

3.5 FORMALISATION OF ACTIVITY MODELLING

The formalisation of activity modelling is split up according to the four diagram techniques employed: entity decomposition diagram, global activity model, context diagram and local activity model. The latter two will be discussed as special types of activity models, which will be treated in the data flow variant.

The concepts of the techniques, that will be used in all parts of the formalisation, are the sets listed below. These sets are all mutually disjoint unless otherwise stated.

E : the set of events,
R : the set of reactions,
A : the set of activities,
F : the set of flows,
S : the set of stores,
X : the set of external agents,
I : the set of internal agents and
P : the set of activators.

Recall that the set of activators was introduced as the union of activities, stores and external agents. The disjointness of the three sets implies that an activator is either an activity, a store or an external agent.

$$P = A \cup S \cup X$$

Observe that we did not introduce a set for all of the concepts. This is because we will not formalise all aspects of activity modelling. A complete formalisation would involve a lot of trivialities and we prefer to restrict ourselves to the more interesting parts.

3.5.1 Rules for events

In section 3.1 and 3.2 we saw that various concepts are related to the concept of event. We formalise only some of these associations in basic predicates and the corresponding axioms.

predicate involved **over** $X \times E$

This predicate stands for the involvement of an external agent in an event. A given event may just have one external agent involved.

$$\forall e \in E\ \forall x_1, x_2 \in X\ [\text{involved}(x_1,e) \wedge \text{involved}(x_2,e) \Rightarrow x_1 = x_2] \quad (A1)$$

External agents need not all be involved in an event, since an external agent may have to be introduced as a result of outgoing data flow, that is the result of a reaction on an event.

The incoming and outgoing data flows (or reports) related to the events are formalised in the predicates *incoming* and *outgoing*.

predicate *incoming* **over** $F \times E$

predicate outgoing **over** $F \times E$

An event may have one incoming data flow and one outgoing data flow, but not both.

$$\forall e \in E\ \forall f_1, f_2 \in F\ [\text{incoming}(f_1,e) \wedge \text{incoming}(f_2,e) \Rightarrow f_1 = f_2] \qquad \text{(A2)}$$

$$\forall e \in E\ \forall f_1, f_2 \in F\ [\text{outgoing}(f_1,e) \wedge \text{outgoing}(f_2,e) \Rightarrow f_1 = f_2] \qquad \text{(A3)}$$

$$\forall e \in E\ [\exists f_1 \in F\ \text{incoming}(f_1,e) \Rightarrow \neg[\exists f_2 \in F\ \text{outgoing}(f_2,e)]] \qquad \text{(A4)}$$

Every data flow must either be incoming or outgoing, but may not be both.

$$\forall f \in F\ [[\exists e_1 \in E\ \text{incoming}(f,e_1) \wedge \neg[\exists e_2 \in E\ \text{outgoing}(f,e_2)]] \vee$$
$$[\neg[\exists e_1 \in E\ \text{incoming}(f,e_1)] \wedge \exists e_2 \in E\ \text{outgoing}(f,e_2)]] \qquad \text{(A5)}$$

The constraints that expresses the uniqueness of the combination of an incoming data flow and an external agent is formulated by the next axiom.

$$\forall x \in X\ \forall f \in F\ \forall e_1, e_2 \in E\ [[\text{involved}(x,e_1) \wedge \text{involved}(x,e_2) \wedge$$
$$\text{incoming}(f,e_1) \wedge \text{incoming}(f,e_2)] \Rightarrow e_1 = e_2] \qquad \text{(A6)}$$

Analogous for the outgoing data flow.

$$\forall x \in X\ \forall f \in F\ \forall e_1, e_2 \in E\ [[\text{involved}(x,e_1) \wedge \text{involved}(x,e_2) \wedge$$
$$\text{outgoing}(f,e_1) \wedge \text{outgoing}(f,e_2)] \Rightarrow e_1 = e_2] \qquad \text{(A7)}$$

It is not specified that an external agent may be involved in two different events where the same data flow is incoming in the one and outgoing in the other. If desired this may be added as an axiom. It may, however be possible that some information leaves an organisation just as it has entered with the same external agent involved. Because the organisation performs some activity with the information, otherwise it would not occur in the activity model, the outgoing data flow can be prefixed with an adjective that expresses the activity. An example is when some data is copied or derived from forms that are not altered during processing in an organisation and returned to the same external agent as who gave them. The forms come in and go out with the same information. If the incoming data flow is just

called 'forms', then the outgoing data flow could be called 'processed forms'.

We will return to the relation between external agents and data flows, when we discuss the context diagram in section 3.5.3.

For the relationship between the events, reactions, activities and internal agents we introduce again some basic predicates.

predicate belongs **over** $R \times E$

predicate consists **over** $R \times A$

predicate responsible **over** $I \times R \times A$

The one-to one relation between events and reaction is expressed as follows.

$$\forall e \in E\ \forall r_1, r_2 \in R\ [\text{belongs}(r_1,e) \wedge \text{belongs}(r_2,e) \Rightarrow r_1 = r_2] \qquad \text{(A8)}$$

$$\forall r \in R\ \forall e_1, e_2 \in E\ [\text{belongs}(r,e_1) \wedge \text{belongs}(r,e_2) \Rightarrow e_1 = e_2] \qquad \text{(A9)}$$

Furthermore, this predicate is not total in E. This means that there exists events, namely the internal non-temporal events, for which the reaction is not modelled in the event decomposition diagram. But the predicate is total in R, i.e. all reactions belong to an event.

$$\forall r \in R\ \exists e \in E\ [\text{belongs}(r,e)] \qquad \text{(A10)}$$

The reactions consist of one or more activities and an activity may be part of one or more reactions.

$$\forall r \in R\ \exists a \in A\ [\text{consists}(r,a)] \qquad \text{(A11)}$$

$$\forall a \in A\ \exists r \in R\ [\text{consists}(r,a)] \qquad \text{(A12)}$$

All internal agents are at least responsible for an activity that is part of a reaction.

$$\forall i \in I\ \exists r \in R\ \exists a \in A\ [\text{consists}(r,a) \wedge \text{responsible}(i,r,a)] \qquad \text{(A13)}$$

During event modelling it was not necessary to find all internal agents that are responsible for the activities of a reaction. But for each activity in a reaction there may be at most one.

$$\forall r \in R\ \forall a \in A\ \forall i_1, i_2 \in I\ [\text{consists}(r,a) \wedge \text{responsible}(i_1,r,a) \wedge \text{responsible}(i_2,r,a) \Rightarrow i_1 = i_2] \qquad \text{(A14)}$$

3.5.2 Rules for the data flow diagram

The rules for the data flow diagram were already discussed in a similar way in [Falkenberg 89a]. Some of the rules will be repeated here, whereas others are adapted to the special rules imposed on a context diagram and the decomposition of the global activity model.

The main predicates are *source*, that describes which activator is the starting point of a data flow, and *destination*, that describes its destination.

predicate source **over** $F \times P$

predicate destination **over** $F \times P$

Data flows may not be unconnected, so each data flow must have a source and a destination. Activators may not appear in the models without a data flow connected to them.

$$\forall f \in F\ \exists p \in P\ [\text{source}(f,p)] \qquad \text{(A15)}$$

$$\forall f \in F\ \exists p \in P\ [\text{destination}(f,p)] \qquad \text{(A16)}$$

$$\forall p \in P\ \exists f \in F\ [\text{source}(f,p) \vee \text{destination}(f,p)] \qquad \text{(A17)}$$

The data flow diagram may not contain activators which have data flows that are directly input to themselves. Such iteration loops need to be modelled in the decomposition of an activity.

$$\forall f \in F\ \forall p_1, p_2 \in P\ [\text{destination}(f,p_1) \wedge \text{source}(f,p_2) \Rightarrow p_1 \neq p_2] \qquad \text{(A18)}$$

In order to distinguish the three mutually disjoint kinds of activators: external agents, stores and activities, we need to define two of them and the third can be defined in terms of the other two, using an auxiliary predicate.

predicate ext_agent **over** P

predicate store **over** P

Stores are not external agents and vice versa.

$$\forall p \in P\ \neg[\text{ext\_agent}(p) \wedge \text{store}(p)] \qquad \text{(A19)}$$

Activities are defined by the auxiliary predicate *activity*.

predicate activity **over** P **as**

$$\text{activity}(p) \equiv \neg\text{ext\_agent}(p) \wedge \neg\text{store}(p)$$

The three subsets A, X and S of the set of activators P, that were defined in the beginning of this chapter, can be deined using these predicates. Their total and disjoint subdivision of the set P follows then from the definition of *activity* and A19.

$A = \{p \in P \mid activity(p)\}$
$X = \{p \in P \mid ext\_agent(p)\}$
$S = \{p \in P \mid store(p)\}$

The following axiom formulates that a data flow has always an activity as either source or destination.

$$\forall f \in F\ \forall p_1, p_2 \in P\ [source(f,p_1) \wedge destination(f,p_2) \Rightarrow p_1 \in A \vee p_2 \in A] \qquad (A20)$$

This axiom prevents three undesirable connections in data flow diagrams. External agents may not have a flow of data to another external agent nor to a store and stores may as well not have a data flow to another store. The first is not allowed because direct data flows between external agents are outside the scope of the activity model. The second is motivated in section 3.1 with the remark that external agents never access data stores directly, but that there is a system in which the data stores are embedded. Since data stores are considered to be passive activators it makes no sense to have data flows between them, which motivates the third prohibited connection.

The above axioms lead to the corollary that an external agent is always connected to an activity.

Corollary 3.1

$\forall p \in P\ [ext\_agent(p) \Rightarrow \exists p' \in P\ \exists f \in F$
$[[destination(f,p) \wedge source(f,p')] \vee [destination(f,p') \wedge source(f,p)]] \wedge$
$p' \in A]$

∇

Proof:

Let p be from P and ext_agent(p) valid, then A17 implies
$\exists f \in F\ [source(f,p) \vee destination(f,p)]$.
Suppose source(f,p) is valid, then according to A16
$\exists p' \in P\ [destination(f,p')]$.
Then $p' \in A$ follows, because the flow f connects the external agent p necessarily with an activity p' according to A20.
Suppose destination(f,p) is valid, then again according to A16
$\exists p' \in P\ [source(f,p')]$.
Then $p' \in A$ follows also, because the flow f connects the external agent p necessarily with an activity p' according to A20.

Δ

Analogously, data stores are always connected to activities.

Corollary 3.2

$\forall p \in P\ [store(p) \Rightarrow \exists p' \in P\ \exists f \in F$
$[[destination(f,p) \wedge source(f,p')] \vee [destination(f,p') \wedge source(f,p)]] \wedge$
$p' \in A]$

∇

3.5.3 Rules for the context diagram

The context diagram is, as we discussed in section 3.1.2, a special kind of data flow diagram, for which special rules exist. The context diagram is the activity model at the top of the global activity model and its constituents are therefore part of the sets X, A, F and S. We need the basic predicate *t-subact* for the definition of the context diagram. The properties of this predicate are discussed in sec. 3.5.4.

predicate t_subact **over** $A \times A$

Then *t_subact*(a_1,a_2) has to be read as the activity a_1 has the activity a_2 in its decomposition at some arbitrary level.

The context diagram is then defined by the sets X_C, A_C, S_C and F_C, where

$X_C = X$
$A_C = \{å\}$, with å such that $\forall p \in A \setminus \{å\}$ [t_subact(å,p)]
$S_C = \varnothing$
$F_C = \{f \in F \mid \exists p_1, p_2 \in X_C \cup A_C\ [source(f,p_1) \wedge destination(f,p_2)]\}$

In the next section we will define that there is just one such å in A that has that property, i.e.. the activity at the top of the hierarchy.

We see from the examples that a context diagram is star shaped. We actually are now in the position to proof that formally, where we have to assume that the context diagram is a proper data flow diagram and obeys therefore all the stated axioms. To simplify the reasoning we introduce the following set and auxiliary predicates.

$P_C = X_C \cup A_C$

predicate connected **over** $P_C \times P_C$ **as**

$connected(p_1,p_2) \equiv \exists f \in F_C\ [[source(f,p_1) \wedge destination(f,p_2)] \vee$
$[destination(f,p_1) \wedge source(f,p_2)]]$

predicate connect_all **over** P_C **as**

connect_all(p) ≡ $\forall p' \in P_C \setminus \{p\}$ [connected(p,p')]

P_C is then the set of activators in the context diagram. The predicate *connected* denotes that two activators in the context diagram are connected via a data flow. Observe that this predicate is symmetric. The predicate *connect_all* determines for a given activator whether it is connected to all other activators.

The star shapedness of a context diagram can now be proven in the following steps.

Lemma 3.3

The activator å is connected to all other activators in the context diagram, i.e.
connect_all(å)

∇

Proof:

We have $P_C \setminus \{å\} = X_C$
We then have to prove: $\forall p \in X_C$ [connected(å,p)]
Let p be arbitrary from X_C
then according to A17
$\exists f \in F_C$ [source(f,p) ∨ destination(f,p)]
Suppose source(f,p) is valid then from A16 we have
$\exists p' \in P_C$ [destination(f,p')]
Suppose then that $p' \in X_C$ then connected(p,p') is valid,
which is contradictory with Cor. 3.1, that says that two external agents
may not be connected.
Hence $p' \in P_C \setminus X_C = \{å\}$, so p' = å
and therefore connected(å,p)

The case in which destination(f,p) is valid is proven analogously.

Δ

Lemma 3.4

A context diagram with more than one external agent is not a double star, i.e.
$|X_C| > 1 \Rightarrow [\forall p \in P_C$ [connect_all(å) ∧ connect_all(p)] ⇒ p = å]

∇

Proof:

Suppose $|X_C| > 1$ and $p \in P_C$ such that $p \neq å$
Let p be an element from $P_C \setminus \{å\} = X_C$ and connect_all(p)
Then since $|X_C| > 1$ we have that there exists an $x \in X_C$

with $x \neq p$ and connected(p,x).
This is again in conflict with Cor. 3.1.
So $\neg$connected(p,x), hence $\neg$connect_all(p)
and p = å.

Δ

Lemma 3.5

All elements from P_C, except å, are exclusively connected with å in a context diagram, i.e.
$\forall p \in P_C \setminus \{å\}\ [\exists p' \in P_C\ \text{connected}(p,p') \Rightarrow p' = å]$

∇

Proof:

Let p arbitrary from $P_C \setminus \{å\} = X_C$ and connected(p,p')
If $p' \in X_C$ then we are in conflict with Cor. 3.1
So $p' \in P_C \setminus X_C = \{å\}$, so p' = å.

Δ

The theorem about the star shapeness of the context diagram is then formulated as follows.

Theorem 3.6

A context diagram is star shaped and if the context diagram has more than one external agent then the centre of the star is unique.

∇

Proof:

The centre of the star is å, which is connected to all other activators in the diagram according to lemma 3.3. If there is just one external agent
then the star shapedness is trivial, but has two centres.
Otherwise the uniqueness of the centre follows from lemma 3.4.
The exclusive connections of the activators with å is finally justified by lemma 3.5.

Δ

During modelling of context diagrams one has therefore only to assure that the formulated axioms are valid and then the star shapedness of the context diagram follows. Assume for instance that the axioms are implemented in a support tool for data flow diagramming and prohibit the entering of erroneous data, it is then impossible to enter a non-star shaped context diagram.

3.5.4 Rules for the decomposition of activities

The decomposition of activities in the global activity model needs also a proper formalisation to ensure all kind of well-formedness properties of the hierarchical decomposition. Parts of this formalisation were already discussed in [Falkenberg 89a].

The predicate *t_subact* was defined in the previous section, but its axioms were not. This predicate is irreflexive, not symmetric and transitive. The decomposition leads therefore to a strict partial order of the activities. Obviously the order is not total in general, since not for any pair of activities is valid that either the one appears in the decomposition of the other, or the other way round. We can not define this as an axiom, since trivial decompositions as for instance a chain of activities that each have one activity in their decomposition are totally ordered.

$$\forall a \in A\ [\neg t\_subact(a,a)] \qquad (A21)$$

$$\forall a_1,a_2 \in A\ [t\_subact(a_1,a_2) \Rightarrow \neg t\_subact(a_2,a_1)] \qquad (A22)$$

$$\forall a_1,a_2,a_3 \in A\ [t\_subact(a_1,a_2) \wedge t\_subact(a_2,a_3) \Rightarrow t\_subact(a_1,a_3)] \qquad (A23)$$

Based on this predicate the auxiliary predicate *subact* is defined.

predicate subact **over** $A \times A$ **as**

$subact\ (a_1,a_2) \equiv [t\_subact(a_1,a_2) \wedge \neg\exists a_3 \in A\ [t\_subact(a_1,a_3) \wedge t\_subact(a_3,a_2)]$

The hierarchical nature of the decomposition is then enforced by the axiom that tells that an activity appears in just one decomposition.

$$\forall a,a_1,a_2 \in A\ [t\_subact(a_1,a) \wedge t\_subact(a_2,a) \Rightarrow a_1 = a_2] \qquad (A24)$$

The hierarchy of activity models has exactly one top. We have already used this in the definition of the conetxt diagram in the previous section. We define this via an auxiliary predicate *top*.

predicate top **over** A **as**

$top(a) \equiv \neg\exists b \in A\ [t\_subact(b,a)]$

The axioms that enforce the existence and uniqueness of a top are then as follows.

$$\exists a \in A\ [top(a)] \qquad (A25)$$

$$\forall a_1,a_2 \in A\ [top(a_1) \wedge top(a_2) \Rightarrow a_1=a_2] \qquad \text{(A26)}$$

For convenience the auxiliary predicate *decomposed* is defined.

predicate decomposed **over** A **as**

$$decomposed(a) \equiv \exists b \in A\ [subact(a,b)]$$

Analogous to the decomposition of the activities the decomposition of the flows is formulated.

predicate t_subflow **over** $F \times F$

predicate subflow **over** $F \times F$ **as**

$$subflow\ (f_1,f_2) \equiv [t\_subflow(f_1,f_2) \wedge \neg\exists f_3 \in F\ [t\_subflow(f_1,f_3) \wedge t\_subflow(f_3,f_2)]$$

These predicates are to be read similarly as those for subactivities, i.e. *subflow* (f_1,f_2) means that the flow f_2 is a direct subflow of f_1. A lot of axioms can be formulated for these predicates but we restrict us to the one that says that subflows of flows inherit either the source or the destination. The exclusive or is notated with $\veebar$.

$$\forall f,g \in F\ \forall a,b \in A\ [source(f,a) \wedge destination(f,b) \wedge subflow(f,g) \Rightarrow source(g,a) \veebar destination(g,b)] \qquad \text{(A27)}$$

Also a lot of axioms can be formulated for the relation between activities and flows and their decompositions. We only give two of them, namely that activities, that have a decomposition, have flows that appear either in the decomposition or have a subflow that is connected to one activity in the decomposition.

$$\forall f \in F\ \forall a \in A\ [source(f,a) \wedge decomposed(a) \Rightarrow \exists b \in A\ [subact(a,b) \wedge [source(f,b) \vee \exists g \in F\ [subflow(f,g) \wedge source(g,b)]]]] \qquad \text{(A28)}$$

$$\forall f \in F\ \forall a \in A\ [destination(f,a) \wedge decomposed(a) \Rightarrow \exists b \in A\ [subact(a,b) \wedge [destination(f,b) \vee \exists g \in F\ [subflow(f,g) \wedge destination(g,b)]]]] \qquad \text{(A29)}$$

The theorem we are now able to formulate is already discussed in step 6.b of the activity modelling procedure of sec. 3.4. It deals with the interaction of activities with data stores and says that in the decomposition of an activity with a data store there exists a path of activities to the bottom layer

of the decomposition such that in each of these activities interact with the data store.

Theorem 3.7

All decomposable activities in the decomposition of an activity $\underline{a}$, that have a flow to a data store, have a subactivity that has a flow to the same store, i.e.

$\forall \underline{a},a \in A\ \forall f \in F\ \forall s \in S$
$[t\_subact(\underline{a},a) \wedge decomposed(a) \wedge source(f,a) \wedge destination(f,s) \Rightarrow$
$\exists b \in A\ [subact(a,b) \wedge \exists g \in F\ [source(g,b) \wedge destination(g,s)]]]$

∇

Proof:

According to A28 we have from source(f,a) $\wedge$ decomposed(a) that
$\exists b \in A\ [subact(a,b) \wedge [source(f,b) \vee \exists g \in F\ [subflow(f,g) \wedge source(g,b)]]]$

Suppose first that source(f,b) is valid.
then the theorem is satisfied taking g = f.

Suppose otherwise that we have a flow g, such that
subflow(f,g) $\wedge$ source(g,b)
Combine this with destination(f,s) $\wedge$ source(f,a) and we have according to A27 that destination(g,s) needs to be valid,
which proves the second alternative.

Δ

This theorem formulates a property on the input to data stores, and an analogous theorem can be formulated for the output of data stores. Similar theorems can also be formulated for the flows to and from external agents, because external agents are, like stores, not decomposable. It does not matter that the external agents appear only at the top level of the activity decomposition.

4 DATA MODELLING

4.1 DATA MODELLING PROCEDURES

Data modelling is one of the most frequently addressed topics in information systems research. The static nature of data in an information system turns out to be attractive to propose new notations, to introduce extra concepts and to investigate theoretical backgrounds. Over the years the data modelling scene has been enlivened with debates among the advocates and opponents of the major data modelling schools: hierarchical, network, relational, semantic and, recently, object-oriented. Beside the abundance of modelling techniques and variants thereof, this has resulted in important modelling fundamentals, such as the separation of conceptual, internal and external aspects, the availability of constructs for special kinds of concept interrelationships, and abstraction mechanisms such as modularity and derived schemas [Hull 87].

In this chapter we will deal with modelling procedures for semantic data modelling techniques[1]. The main data modelling techniques are the following. The first semantic data model was the binary modelling notation introduced in [Abrial 74]. The Entity-Relationship modelling technique (ER model) [Chen 76] is the most generally used data modelling technique and is implemented in various modelling tools. Functional relationships are the key concept of the Functional Data Model [Shipman 81] and are therefore suitable for a mathematical formalisation. The binary modelling technique of NIAM [Verheijen 82] and the elementary variant, that allows n-ary relationships [Nijssen 89], are based on a linguistic analysis of fact types. The IFO modelling technique [Abiteboul 87] was introduced to provide a theoretical framework for studying the semantic data modelling techniques.

As already described in section 2.4 the NIAM technique is accompanied by a modelling procedure. In [Wintraecken 85] a modelling procedure is given for the binary variant of NIAM, that consists of eleven steps. Further research lead to the development of modelling procedures for the elementary variant as presented in [Falkenberg 87] and [Nijssen 89]. The first procedure consists of the following steps.

1. Gathering of sample documents and removal of redundancy.

---

1 In the names of most data modelling techniques the term model is not used in the sense of the discussions in chapter 1, e.g. ER model. This misuse has, however, found its way into the standard data modelling terminology. We prefer to use the terms modelling technique and modelling notation in line with the definition of section 1.3.

2. Formulation of elementary sentences.
3. Provisional fact types.
4. Uniqueness constraints on fact types.
5. Check of splittability of fact types.
6. Determination of the structure of elementary fact types.
7. Redundancy check.
8. Integration of subschemas.
9. Determination of subtypes.
10. Identification of entities.
11. Determination of constraints.
12. Final quality checks.

Globally speaking, the relationships of the data model are established first, thereafter the entity types and finally the constraints. The procedure in [Nijssen 89] is to a large extent similar to that of [Falkenberg 87]. Some of the steps are more elaborated and can be qualified as *modelling subprocedures*, i.e. the steps contain a sequence of substeps to determine a particular aspect of the data model. Subtyping, for instance, is performed in nine substeps.

Modelling guide-lines for the ER modelling technique were given in [Hsu 85], [Teorey 86] and [Brinkkemper 88ab]. The first and second procedure use an extension of the original ER modelling concepts and are oriented towards the relational modelling theory with functional dependencies and all kinds of keys. In the third is a similar approach as in the NIAM procedures chosen. Here, as well, the starting point are sample documents. The three ER modelling concepts: entities, relationships and attributes are recognised on these samples and modelled together with cardinalities and identification structures.

In [Aerts 88] a modelling procedure for the Functional Data modelling technique was presented. Despite the complete formalisation of the representation technique the procedure is formulated in general terms and does not address the recognition of the modelling concepts in depth.

In this chapter we will elaborate the procedure that was first presented in [Brinkkemper 88ab]. The next section deals with the formalisation of the ER modelling concepts and proposes to use NIAM and ER together in software development projects. In section 4.3 the data modelling procedure is discussed and illustrated with output of a modelling support tool.

4.2 ENTITY-RELATIONSHIP MODELLING

4.2.1 Entity-Relationship modelling concepts

The intuitive nature of the ER modelling technique is appealing to both practitioners and researchers. However, the three key concepts of the technique entity type, relationship and attribute, were not defined formally, which gave rise to all kinds of different interpretations. Moreover, the ER

concepts were not capable of modelling subtyping and constraints, which led to various extensions.

We take a binary version of the original proposal in [Chen 76] as a starting point for the discussion and formalisation. An example of a simple data model using the ER notation is shown in fig. 4.1. If we compare this with the data model of the video store test case in the NIAM notation in fig. A.4, we see that the number of components is reduced: four entity types, three relationships and twelve attributes, but the degree of detail is decreased. The NIAM data model shows a lot more constraints that cannot be depicted by the ER notation. For instance, the restriction that the attribute rental charge of a rental is only filled in if the return date is filled in as well, and vice versa. This is modelled by an equality constraint in fig. A.4, but is lacking in fig. 4.1.

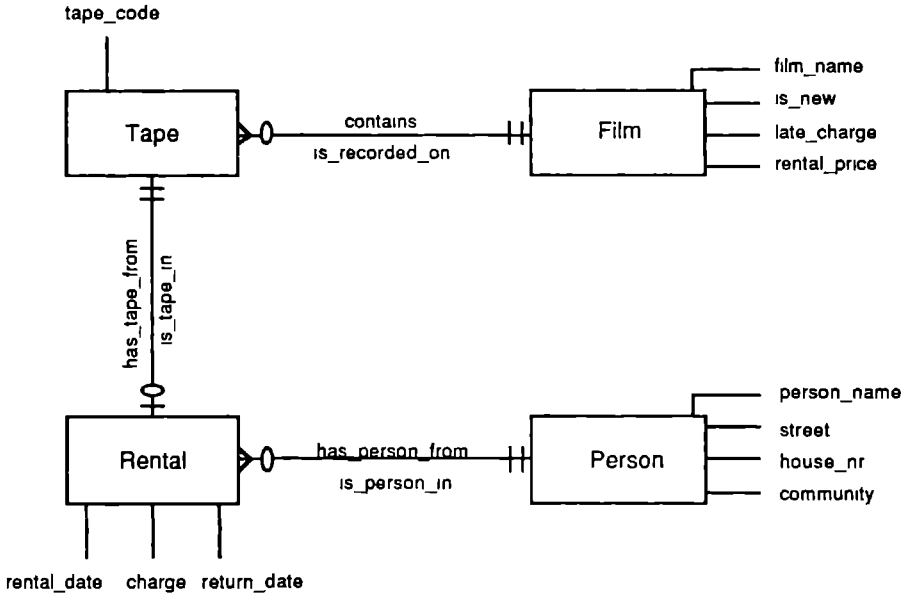


Figure 4.1 Example of ER data model

In this model we have left out the cardinalities for the attributes. The cardinalities on the relationships are in the so called crowfoot notation. A 0 or a I at the inner side denotes that the one entity type is optional or mandatory related to the other. A I or a > at the outside of a relationship denotes the maximal occurrence of one or many. Observe finally, that the entity type 'Rental' occurred in fig A.4 as an objectified entity type, but is an associative entity type in the ER data model.

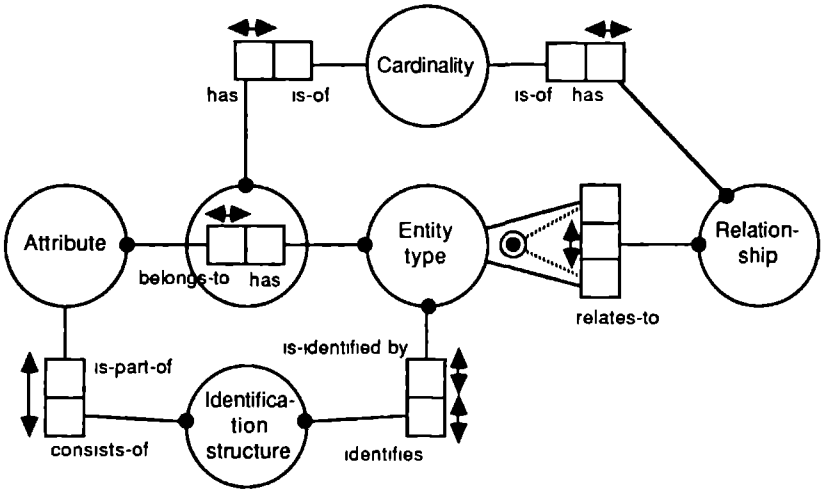


Figure 4.2 Meta-model of ER data modelling.

We discuss the concepts of the ER modelling technique with the help of its meta-model in fig. 4.2. Entity types model the "things" [Chen 76] or entities of the Universe of Discourse (see also def. 1.5). These entity types occur in all kinds of associations, that can be modelled by the relationships. Entity types are therefore related to each other in several binary relationships. This is modelled by a ternary relation to allow homogeneous relationships, i.e. relationships from an entity type to itself. Attributes model the values of properties of entities. Attributes belong to exactly one entity type and there may be several of them belonging to one entity type.

Relationships and the associations between attributes and entity types have minimal and maximal occurrence cardinalities. There are sixteen different combinations for cardinalities in the binary ER technique and we have depicted them in the meta-model as the type 'cardinalities' for sake of simplicity. More sophisticated relationship modelling techniques would require more complex representations for cardinalities. The associative entity type is not modelled as a special concept, because this is treated as an ordinary entity type. Their conceptualisation steps differ as will turn out from the modelling procedure (sec. 4.3).

Attributes are needed to identify entity types uniquely. As we will discuss in the modelling procedure, this identification structure can be a single attribute or a combination of attributes. It may even be possible in the latter case that attributes of other entity types, to which the underlying entity type is related, are needed for a proper identification. An example is the identification of the entity type 'Rental' in fig. 4.1. 'Rental' is identified by the attribute 'rental date' of itself and 'tape_code' of 'Tape'. This gives rise to a complex constraint on the meta-model: the set of attributes of the

identification structure of a particular entity types is a subset of the set of all attributes of this entity type possibly united with the set of attributes of the entity types to which the entity type is related. From a theoretical point of view the latter union may need to be repeated several times recursively, but this will rarely be the case during modelling in practice.

4.2.2 Formalisation of Entity-Relationship modelling

Different formalisations have been given for the Entity-Relationship modelling technique. The various approaches differ so considerably, that it is hard to compare their results. We mention [Parent 87] based on an ER algebra, [Gilmore 87] based on set theory, and [Simovic 89], that uses axiom systems. Next to these we put our predicate calculus approach. This differs to that respect that we consider only the ER model, its components and its well-formedness in the formalisation. The other approaches also relate the instances of the Universe of Discourse or of the data base to the models.

The basic concepts of the ER modelling technique, as discussed in the preceding subsection, are the followings sets.

E : the set of entity types,
R : the set of relationships,
A : the set of attributes
I : the set of identification structures and
C: the set of cardinalities.

All sets are mutually disjoint and the first three sets could be considered as subsets of a set of objects. Objects can be distinguished in a similar way as activators in the case of activity modelling (sec. 3.5). Properties of the objects, such as names, definitions and volumes, that are to be recorded for all three subsets can be related to the set of objects.

The participation of entity types in relationships is represented by the basic predicate *relate*.

predicate relate **over** $E \times R \times E$

This predicate realises the binary relationships by having the set E as argument twice. A relationship therefore always relates two entity types to each other. The two entity types of a relationship are uniquely determined by the relationship. Furthermore, it does not make sense that entity types do not participate in a relationship. These rules are formally described by the axioms D1, D2 and D3.

$$\forall r \in R\ \exists e_1, e_2 \in E\ [\text{relate}(e_1, r, e_2)] \qquad \text{(D1)}$$

$$\forall r \in R\ \forall e_1, e_2, e_3, e_4 \in E\ [\text{relate}(e_1, r, e_2) \wedge \text{relate}(e_3, r, e_4) \Rightarrow e_1 = e_3 \wedge e_2 = e_4] \qquad \text{(D2)}$$

$$\forall e_1 \in E\ \exists e_2 \in E\ \exists r \in R\ [\text{relate}(e_1, r, e_2) \vee \text{relate}(e_2, r, e_1)] \qquad \text{(D3)}$$

The entity types of a relationship may be the same in the case of an homogeneous relationship. We define this by means of an auxiliary predicate *homogeneous.*

predicate homogeneous **over** R **as**
homogeneous(r) $\equiv \exists e \in E$ [relate(e,r,e)]

All attributes belong to just one entity type and all entity types have at least one attribute. These rules are formulated in terms of the basic predicate *belong.*

predicate belong **over** $A \times E$

$\forall a \in A\ \exists e \in E$ [belong(a,e)] (D4)

$\forall e \in E\ \exists a \in A$ [belong(a,e)] (D5)

$\forall a \in A\ \forall e_1,e_2 \in E$ [belong(a,e_1) $\wedge$ belong(a,e_2) $\Rightarrow$ e_1=e_2] (D6)

We will not work out the predicates for the cardinalities of the relationships and of the associations of the attributes with the entity types, because this is similar to the other predicates. The identification of entity types is more interesting to elaborate, which is based on the basic predicates *identify* and *is_part.*

predicate identify **over** $I \times E$

predicate is_part **over** $A \times I$

The identification relation is one to one and total in both arguments.

$\forall i \in I\ \exists e \in E$ [identify(i,e)] (D7)

$\forall e \in E\ \exists i \in I$ [identify(i,e)] (D8)

$\forall i \in I\ \forall e_1,e_2 \in E$ [identify(i,e_1) $\wedge$ identify(i,e_2) $\Rightarrow$ e_1=e_2] (D9)

$\forall e \in E\ \forall i_1,i_2 \in I$ [identify(i_1,e) $\wedge$ identify(i_2,e) $\Rightarrow$ i_1=i_2] (D10)

The only rule for the *is_part* of predicate is that it is total for identification structures.

$\forall i \in I\ \exists a \in A$ [is_part(a,i)] (D11)

The combination of D8 and D11 implies that all entity types are identified by attributes. Observe, that entity types need not necessarily be identified by their own attributes. It is theoretically possible to construct a data model in

which the identification of one entity type requires all attributes of all entity types in the model.

The attributes that identify an entity type may then be described by the auxiliary predicate *is_part_idef*, that is a kind of join of the *identify* and *is_part* predicate.

predicate is_part_idef **over** A × E **as**
is_part_idef(a,e) ≡ ∃i∈ I [is_part(a,i) ∧ identify(i,e)]

If the cardinalities were made explicit, we could as well relate these to the identification structures. In tools that support the mapping of an ER data model to a relational data model, these rules are implemented so that the keys of the relational tables are automatically determined.

4.2.3 Synthesis of the ER and the NIAM modelling techniques

The ER data modelling technique and the NIAM data modelling technique have both a lot of similarities and differences. Instead of chosing the one in favour of the other, we could investigate their strong and weak points and formulate how both techniques can be used in conjunction with each other within software development projects. Subtyping and arbitrary degree of relationships are not discussed here, since these can be introduced in the ER technique in a simple way.

A strong point of NIAM is that the larger set of constraints enables modelling in more detail. Furthermore, all entity types are global, which simplifies query formulation in terms of the data model. Minor aspects of NIAM are the tendency to deliver large data models and consequently the loss of a global overview. In [Bouman 88ab] two shorthand notations of large similar parts of the data model had to be introduced to keep the overview of the model.

The disadvantages of the one are the advantages of the other and vice versa. The models in the ER notation are more concise and provide therefore an overview. The grouping of some attributes at an entity type gives a link to the grouping of the attributes in relational tables. This simplifies the transfer of data modelling to data base design. A weak point of the ER technique is the hiding of attributes in the entity type definition. This implies that constraints between attributes of different entity types cannot be specified and comparison queries between those attributes are more difficult to formulate.

Suppose now that we have two mappings at our disposal: a restriction mapping that maps a NIAM data model to an ER data model, and an extension mapping that maps data models the other way round. We can then combine the benefits of both techniques and overcome the disadvantages. We show this in figure 4.3.

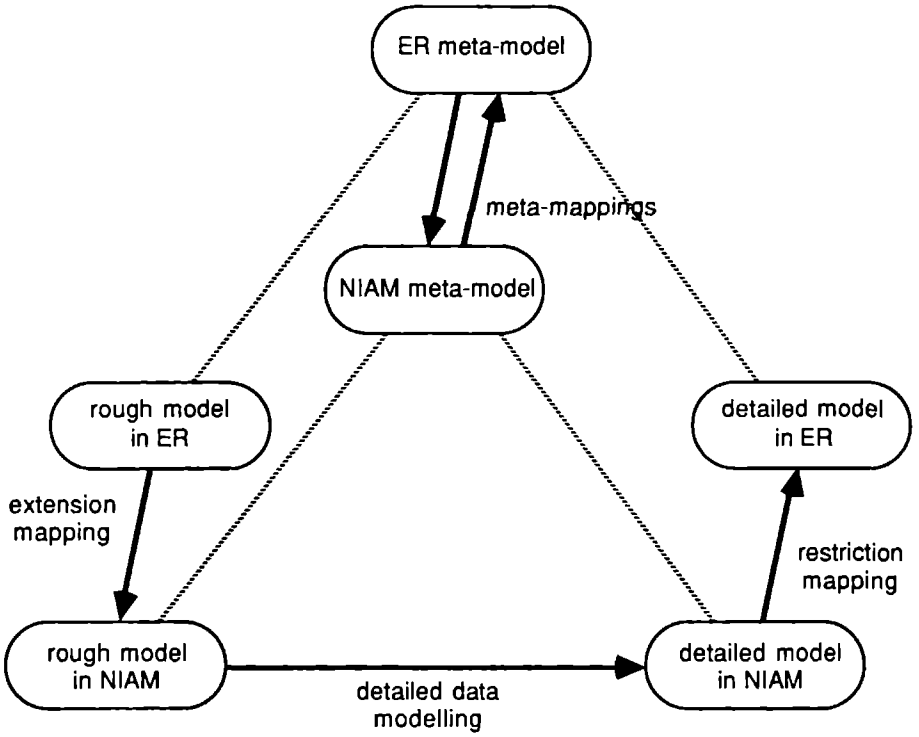


Figure 4.3 Mapping of data models.

We may then start with a rough, global ER data model with the main entity types and relationships. Such models come available for instance if development has been preceded by an information systems planning phase. This model is then mapped to the NIAM set of concepts, which will not cause too many problems due to the absence of attributes. Next, the data modelling is performed according to a NIAM modelling procedure. The resulting NIAM data model will in general contain more entity types and relationships than the model at the start, and contains a lot of constraints. This data model is then mapped to an ER data model, which hides a lot of the details. In the development activities thereafter one could then distinguish the activities that need the detailed NIAM model and the global ER model and use the models correspondingly.

The extension mapping and the restriction mapping should be formulated in terms of the meta-models of both modelling techniques. To a great extent these mappings can be performed automatically, except for the concepts of either technique that have no counterpart in the other technique and therefore need to be qualified and mapped by a modeller. For example should some of the entity types in the NIAM model be mapped to attributes

in the ER model, whereas others, the more important ones, should be mapped to entity types. Such qualifications are related to the objective of the system to be built and are difficult to support automatically.

4.3 ENTITY-RELATIONSHIP MODELLING PROCEDURE

Most information system development methods that prescribe the use of Entity-Relationship modelling techniques do not specify an ER data modelling procedure. We will present a modelling procedure here that consists of nine steps. This procedure is meant to be performed during the analysis stage of a development project. The procedure is applied to sample documents representing the data flows of the Universe of Discourse and results in partial data models. The complete data model diagram is created by integration of partial models, which is straightforward in the case of the ER technique.

We will illustrate the modelling procedure with the automated support of the Information Engineering Workbench (IEW, [IEW 88]), a tool developed for the support of the techniques prescribed by the Information Engineering method. The IEW supports among others Entity-Relationship modelling and data flow modelling. See section 6.4, [Martin 88a] or [Brand 89ab] for more information about IE or the IEW.

In the modelling procedure we use the term fact where we cannot make a distinction between associations between entities and associations of entities and attributes. The examples are again borrowed from [Olle 88a].

Step 1: Collection of samples

Information for data modelling is gathered by collecting representative samples of **all** flows at the bottom level of the hierarchy of activity models, or the global activity model, as explained in chapter 3. Since these flows are part of the flows on all higher levels, we are sure not to overlook any flow of information in the Universe of Discourse and so we can create a complete data model of the UoD. The flows of the external events bring most of the information into the organisation and are therefore contained in the collection.

The samples are mostly forms, tables, letters, reports or parts thereof, and excerpts from existing files and stores. The internal agents who can provide the samples are specified in the global activity model.

Step 2: Treatment of redundancy

Redundancy in an information system is tolerable, but must be controlled by the analysis and design phases. We start from the philosophy that during analysis all redundant information is left out and captured in rules. These rules can be recorded as comments via the detail screens. Efficiency and distribution reasons may then motivate the *controlled*

introduction of redundancy in the design phase. In this latter stage special redundancy managing application programs have to be designed.

There are four types of redundancy:

1. **Redundancy by repetition.**
   A fact is represented more than once in the samples. One occurrence is chosen to be the defining one and all others are considered to be repetitions. That it is a repetition is recorded in the description of the entities. The analyst will have to decide whether or not to incorporate the fact in the further analysis. Reasons to include it could be: better overview or better for the communication with the informant. Redundancy by repetition will mostly be excluded.
   Example:
   The price of a stock item on a purchase order is the same as the price of that stock item in the stock item file. The latter one is the basic one from which the others are copied.
2. **Redundancy by simple derivation.**
   Often totals, averages, minima and maxima of numeric fact types occur in samples. When the data to derive these quantities is present in the information system, these facts are left out.
   Example:
   On a purchase order the price, the quantity and the total price of an individual stock item are listed. The total price can be derived from the price and quantity and is not modelled.
3. **Redundancy by more intricate derivation.**
   This is the case when some facts have to be combined to derive another one in some prescribed way. This prescription must then be formulated and recorded as comment in a detail screen. It is up to the analyst to incorporate this fact for further analysis, or to leave it out. It should be marked as being redundant if it is incorporated.
   This type of redundancy is likely to occur in a cycle in a data model or when more than one entity type contains the same attribute (implicit cycle).
   Example:
   The relationship 'Stock item is ordered from Supplier' is derivable from the relationships: 'Stock item is on Purchase order line', 'Purchase order line is on Purchase order' and 'Purchase order is addressed to Supplier'. We decided to leave this out. (See fig. 4.9)
4. **Redundancy by partial derivation.**
   It quite often occurs that for a fact a derivation rule can be given, but that a small class of exceptions to this rule exists. These exceptions have to be recorded in the database anyway and we leave it up to the designers whether this relationship will be realised by means of the rule and its exceptions or that the rule will be ignored and the corresponding fact recorded for all entities. A remark and the rule should be placed in the description of the relationship.
   Example:
   An stock item can be purchased for a reduced price with a fixed reduction rate when the total of the order is

The XYX company

Stock on hand card

Supplier

Name: Rymans
Address: 23 South St. Walton
Phone: 234567
Nr.: 43

Stock items

Code	Description	Type	Price
3214	Xerox copying paper	S	10.05
1234	Parker fibre tip refill	S	0.45

Code: 3214 Quant. Date Time	Code: 1234 Quant. Date Time	Code: ..... Quant. Date Time
25 861003 17.30 10 880112 8.45	237 870304 9.40	

p.t.o

Figure 4.4 Sample of Stock on hand card

```
larger than a certain amount of money. But for some
customers a different rate applies due to their special
relationship with the firm.
```

All samples are scanned for redundancy. Special care has to be taken for redundancy that is spread over different samples. Modellers should in general discover the redundancy, while the informants can validate this and specify the derivation rules.

Step 3: Typing of objects

After the sample documents have been prepared for further modelling, we now have to type the different objects. Suppose we have the sample as shown in figure 4.4. This sample contains no redundant information. Samples are analysed by the following three substeps.

3.a Identify attributes

Attributes are the most elementary units of data and easy to recognise. They occur in a concrete form in the samples and often need to be chosen or filled in on a sample. In a whole set of samples of the same data flow an attribute never has just one value. Informants can be asked to mark the attributes by identifying all data that would have a possibly different value if another arbitrary sample had been taken.
For each attribute identified, an attribute type name is chosen which has to be unique within the sample. In forms and tables such as that of figure 4.4 these attribute type names are normally present (like name, address, code, quantity, etc.). On other samples such, as letters, they have to be introduced.

3.b Relate attributes to entity types.

All attributes identified in the preceding step belong to one entity type. For each attribute type this has to be named explicitly. Often this is very clear and already present on a sample. In the example in fig. 4.4 the entity types 'supplier' and stock item' are easy to recognize.
In cases where this is not so obvious, we do this using the sentence template:

"The *ET* has *ATN*."

At the position of ET the name of an entity type should be filled in and at that of ATN the name of an attribute type.

Examples:
The *Supplier* has *Address*
The *Stock item* has *Description*
The *Stock item of supplier* has *Quantity*

In this way the entity type 'Stock item of supplier' can be identified in the example. Note that now an *associative entity type* is introduced. Associative entity types occur when attributes say something about an association of two or more entity types. A more or less artificial entity

type is created by giving this association the status of an associative entity type.

3.c Relate the entity types.

In this substep we neglect all attributes, but consider only the entity types on the sample. For all entity types present in a sample relate the entity types in a sentence pair with the templates:

"*ET1* predicate *ET2*"
"*ET2* reverse predicate *ET1*"

At the position of ET1 and ET2 the names of two entity types have to be filled in and at the position of predicate and reverse predicate the names for the association of the two entity types in either direction. The predicates of these sentences will be used in the description of the relationships.

Example:
Supplier supplies *Stock item*
Stock item is supplied by *Supplier*

Associative entity types need to be treated specially. Because an associative entity type stands for an association of entity types to which attributes belong, such an entity type should match a sentence pair produced by relating two ordinary entity types. The above sentence pair obviously matches the associative entity type 'Stock item from supplier'. A standard treatment of associative entity types is obtained by filling the following sentence template pairs in for each entity type in an associative entity type (AET).

"*ET* is *ET* in *AET*"
"*AET* has *ET* from *ET*"

Examples:
Supplier is *Supplier* in *Stock item of supplier*
Stock item of supplier has *Supplier* from *Supplier*

A lot of the relationships created by this procedure might be redundant due to the fact that we try to establish all relationships. This redundancy can be treated using the arguments of step 2. Moreover, it may happen that some of the entity types can not be related in a binary way. The simplest solution is then to create associative entity types that render the relation and have binary relationships of the entity types to this newly created associative entity type.

The three substeps have given us the core ingredients of data modelling according to the ER technique in the sequence attributes, entity types and relationships. In the remainder of the procedure the cardinalities and other aspects of these components are modelled.

Step 4: Description of entity types and their attributes

The description and characteristics have to be formulated for all entity types and attributes of step 3. In the workbench there are special dialogues to handle this.

One starts with the entering of the name, purpose and definition of the entity type. Details such as volume and change of volume can be specified in comments. Figure 4.5 shows an example of adding comments in the IEW for the associative entity type 'Supplier of stock item'.

Thereafter the attribute types are entered with the cardinalities of the association with the entity type. The system derives the unique identification property from these cardinalities and places UI in the margin. To ensure global uniqueness of attribute type names all attribute type names are prefixed with the entity type name. Figure 4.6 shows the entering in progress of the attribute type 'delivery date' and its cardinalities to the entity type 'Purchase order' by the IEW. Figure 4.7 shows the final result for the entity type 'Supplier'. Note that the first figure shows the name of the context, i.e. the data flow in which this entity type occurs.

The cardinalities can be determined by the modeller from the samples available. If this is not straightforward the modeller can construct a sample population of the association at hand by making all combinations of a small set of values of the attributes and surrogates for the entity types. The informants should then cross out illegal combinations of elements in this population. One starts with a population according to a many-to-many relation. The informant may leave the population as it is or reduce it to a one-to-many, a many-to-one, or a one-to-one relationship.

Step 5: Description of relationships

The relationships' data has to be recorded. Cardinalities, the predicates from step 3 and comments can be specified. In figure 4.8 the one-to-many relationship 'Purchase order contains Purchase order line' is entered into the workbench. Again sample populations can be used to determine the cardinalities of the relationships.

Step 6: Establish naming conventions

For the same reasons as in the case of activity modelling (see sec. 3.4) one has to decide upon a standard terminology for the data elements. Synonyms and homonyms for data items may exist in the various subsystems of the UoD and have to be resolved. After the preceding two steps, in which attributes, entity types and relationships have been named, the terminology of the system must be established for the rest of the development project. The terminology definition may also profit the organisation for purposes other than information processing.

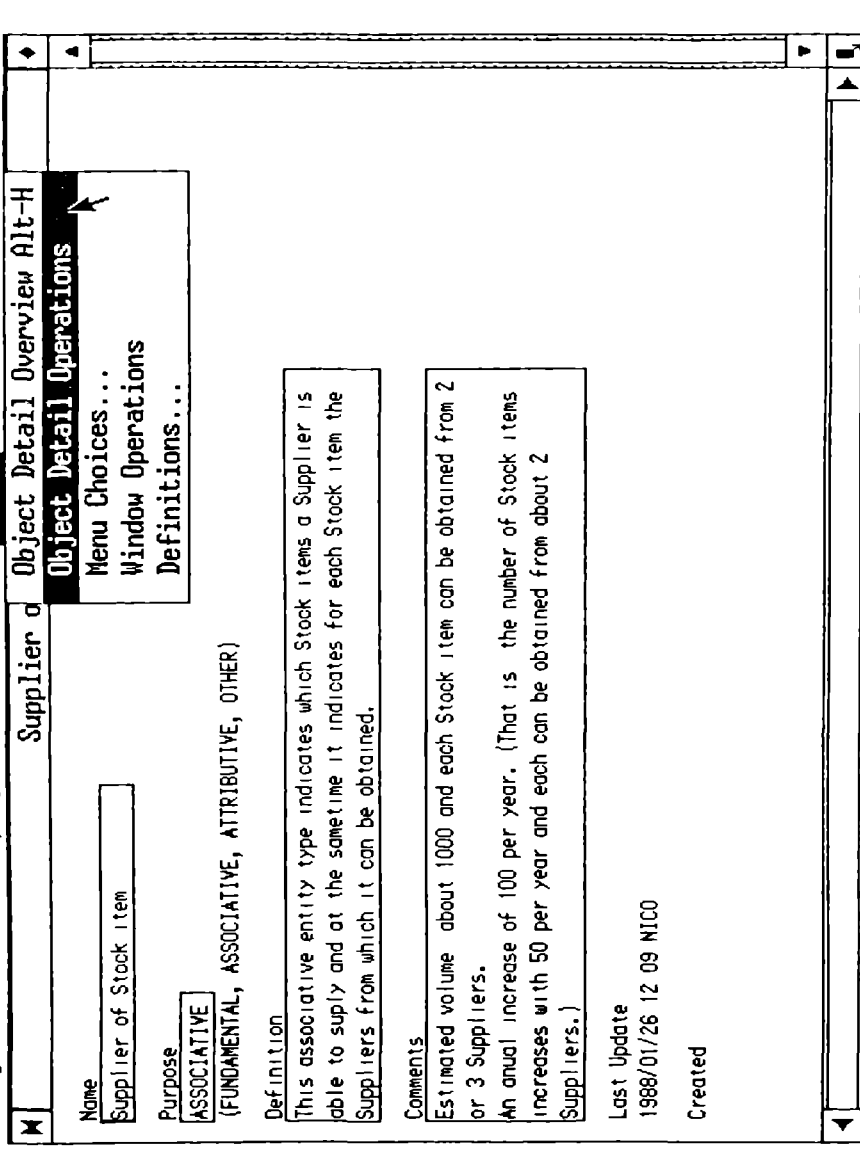


Figure 4.5 Entering of an entity type description

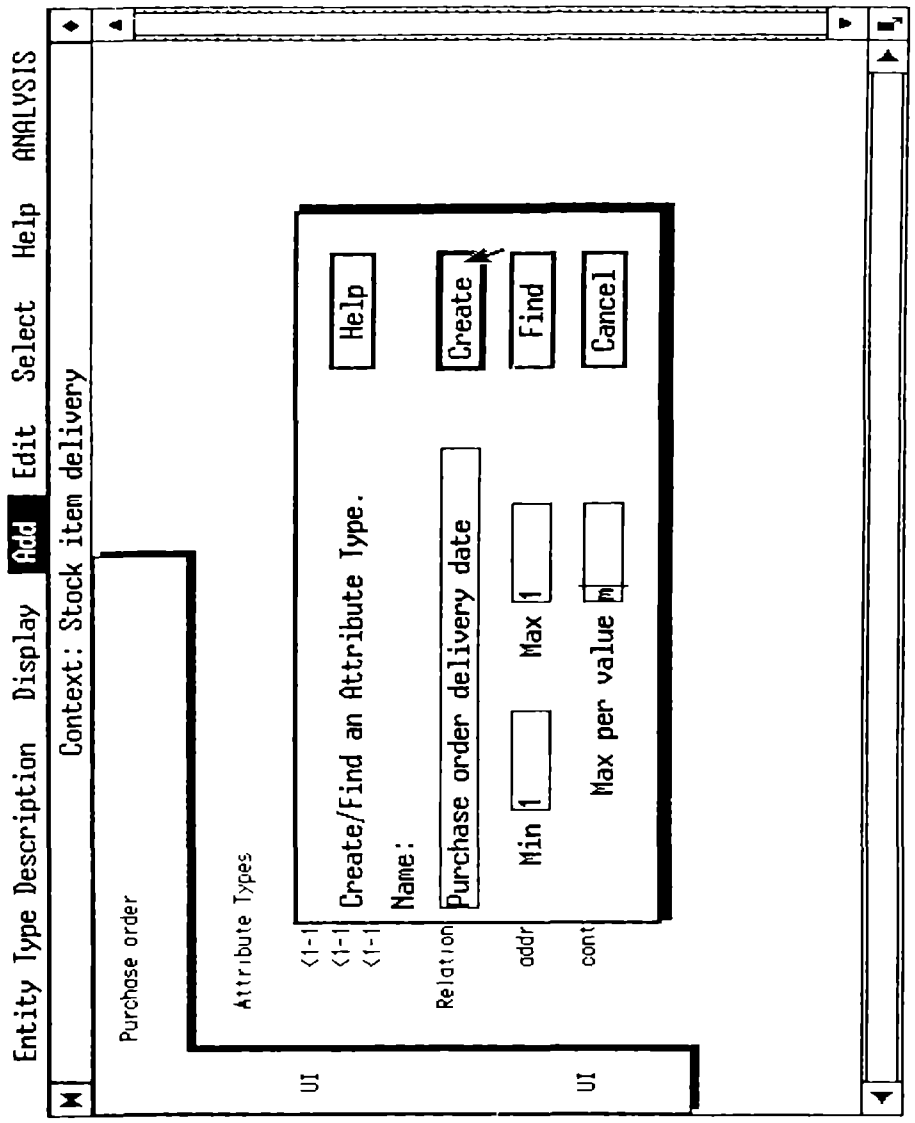


Figure 4.6 Addition of an attribute type

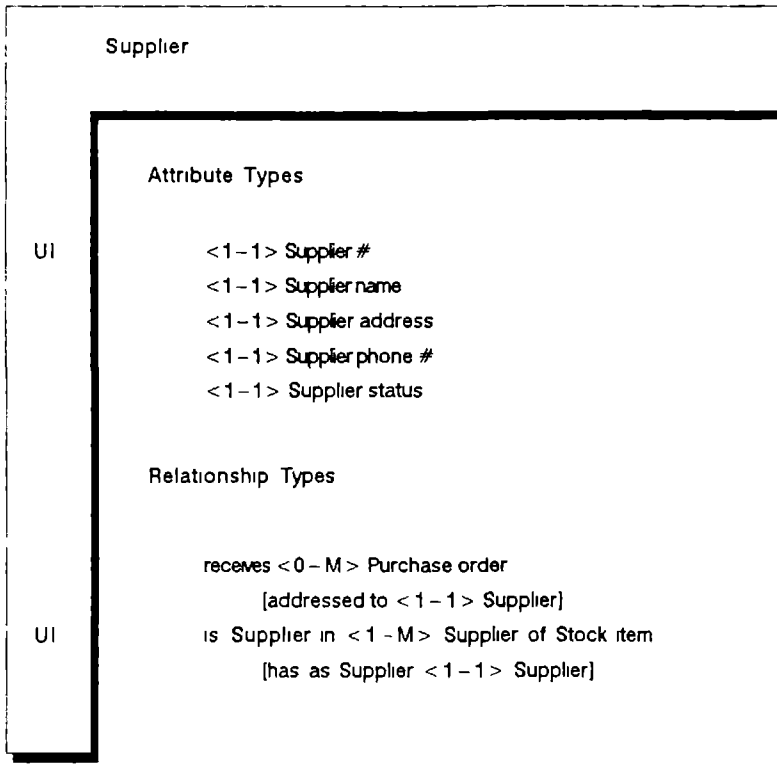


Figure 4.7 Entity type description

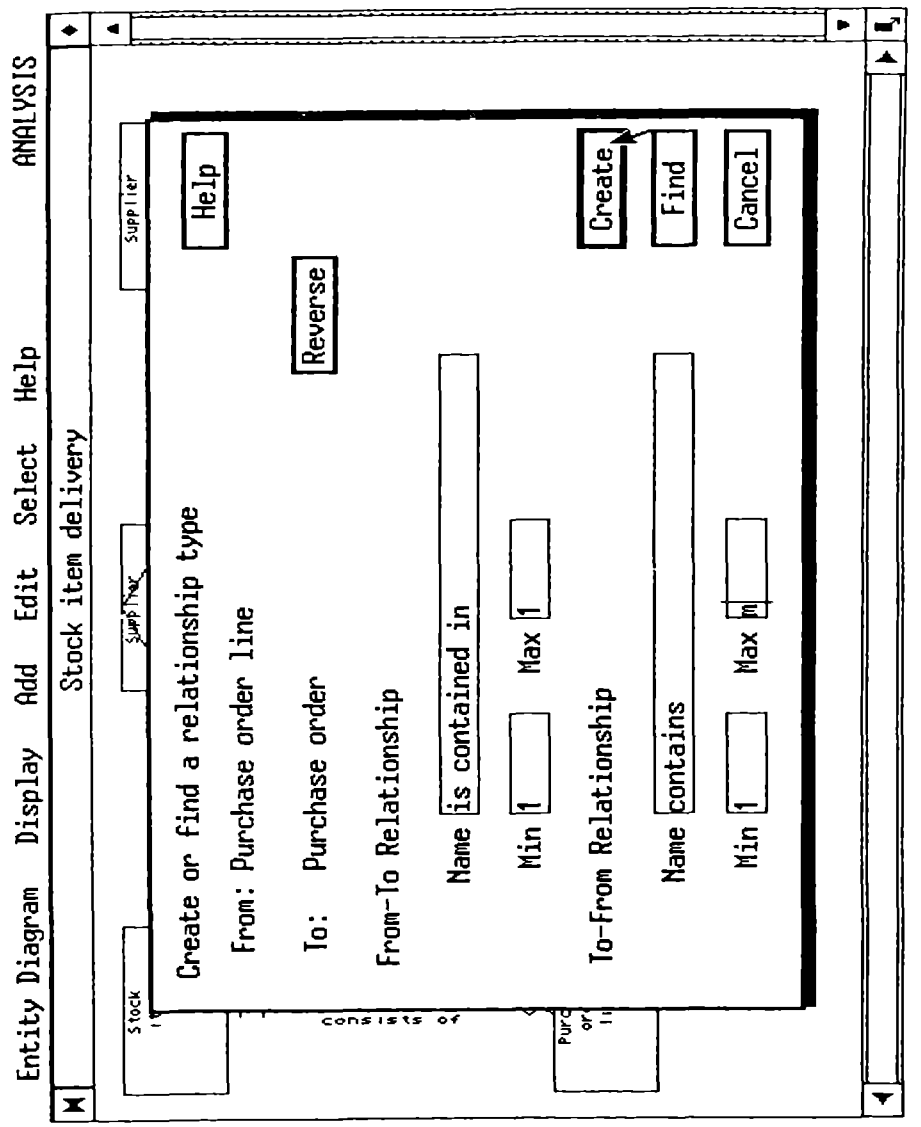


Figure 4.8 Creation of a relationship

Step 7: Identify entity types

Since all entity types, attributes, relationships and their cardinalities are determined and defined, identification of the entity types can be performed. Identification deals with the unique selection of an instance of an entity type by means of values of its attributes. In the data base literature the identification structure is also called key.

Entity types can be identified in three different ways:
1. Via a single attribute. This is already entered in the description of the entity type and is represented by a 'UI' (unique identifier) mark in the margin. If there are more than one possibility, one has to be chosen to be the primary key and the others are then alternate keys.
2. Via a combination of attributes. The combination of attributes has to be incorporated in the entity type description.
3. Via attributes of other entity types. Sometimes it is not possible to identify an entity type with its own attributes and therefore relationships to other entity types are also involved in the identification. The other entity types are already identified and their identification structure is included in that of the underlying entity type. This is also called foreign keys.

In the IEW the identification structures, called predicates, are incorporated in the entity type descriptions.

Step 8: Subtyping

A subtype is a class of entities for which some relationships or attributes are to be recorded. Some associations need not be recorded for all entities, which is expressed by a minimal cardinality of zero, but for a specific class of entities the association may be obligatory. This obligation can be modelled by introducing subtypes. For example the suppliers from which a special type of stock item, say capital expenses, is purchased have a special attribute: credit limit. We could then introduce the subtype 'Supplier of capital expenses' of the supertype 'Supplier' with the accessory attribute 'credit limit'. The subtype inherits all attributes from the supertype.

Subtypes were not included in the Entity-Relationship modelling technique, as it was proposed [Chen 76], but were in various extensions. The subtype relation can also be modelled by an ordinary relationship by way of the 'is a' predicate and one-to-one cardinalities, but this is restricted. As the current version of the IEW restricts itself to the original model, it does not support subtyping.

Step 9: Modelling of constraints

In the UoD there may be all kinds of constraints on relationships or entities. Relationships may be exclusive, equal, or subsets of each other. The values of attributes may have all kinds of restrictions and may as well

be related in certain ways to attributes of other entity types. Special symbols could express this in the data model.

Again this is not present in the original modelling technique. The current version of the workbench does not support these kinds of constraints. They must be specified in comments like all other special constraints.

End of data modelling procedure

We see that the procedure is relatively simple, which is due to the small number of concepts and associations of the ER modelling technique. In [Brinkkemper 88ab] was reported how this modelling procedure was performed on the Stock Control test case of [Olle 88a]. The completed data model is shown in fig. 4.9 and is small compared to real life cases. It contains seven entity types, of which one is associative, and seven relationships. The attributes, about thirty, are not shown.

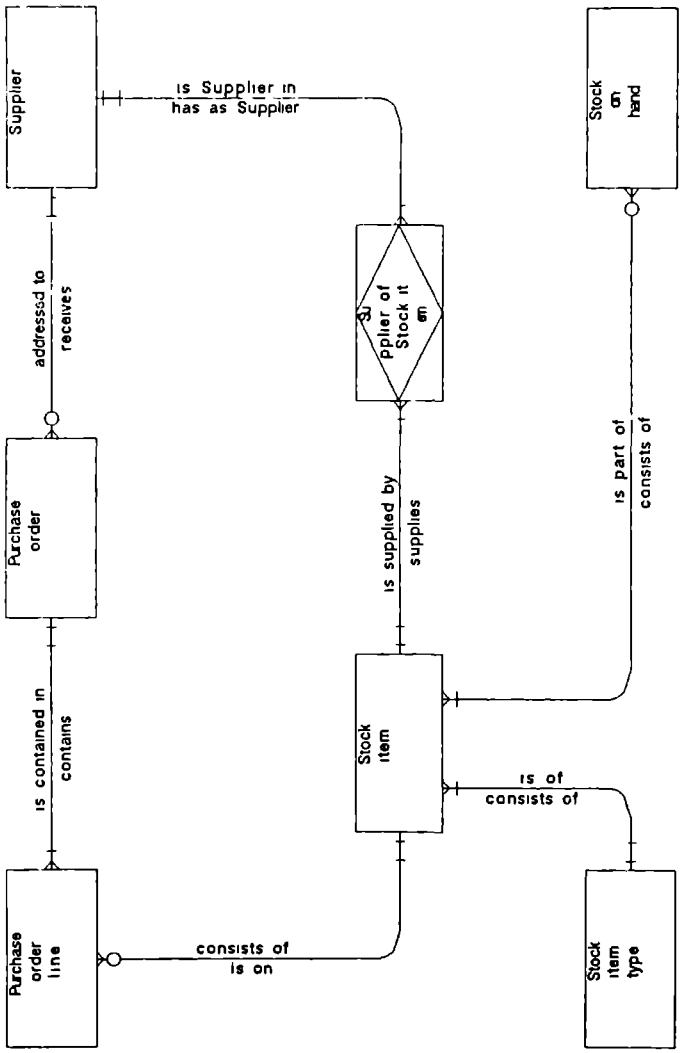


Figure 4.9 Completed data model

5 TASK MODELLING

In this chapter we will introduce a new specification technique for the modelling of tasks, called the Conceptual Task Model (CTM). The need for this technique, its formalisation, a modelling procedure and its use will be discussed in the following sections. Parts of this chapter have already been published in [Brinkkemper 89a], [Brinkkemper 89e] and [Ter Hofstede 89a].

5.1 TASK MODELLING CONCEPTS

In this section we will introduce the concept of a task, a component of an information system specification, for which proper specification techniques are still lacking. Requirements for task modelling are formulated and existing specification techniques are reviewed.

5.1.1 Process, activity, task, operation

In this section we address the proper definition of the terms task, activity, process, operation and related components, that occur during the requirements engineering of the process perspective of an information system. The formal definition of these terms is not yet possible due to the lack of a basic process unit, to which various types of process granularity can be linked.

In the context of business administration information systems the best we can do, is to propose definitions of the terms related to data processing based on recognizable units of data. In the definitions the term process is a basic notion and remains undefined. The latter is also one of the starting points of Process-Algebra [Bergstra 86]. The two concepts entity and Universe of Discourse, already defined in the ISO-report [ISO 82], serve as starting points as well, with some adaptations inspired by [Wieringa 89]. Furthermore, we assume that we can discuss and distinguish phenomena in the Universe of Discourse and its models.

Definition 5.1 An *entity* is any concrete or abstract thing, including associations among things. The *Universe of Discourse (UoD)* is the collection of possible entities, which currently are of interest.

Since we are interested in modelling, we need means to record the results of this activity, i.e. modelling concepts. We follow [Olle 88b] in the following definitions.

Definition 5.2 A *component* is a concept used for modelling phenomena in the UoD. A *data component* serves for modelling the data perspective and a

process component serves for modelling the process perspective. An *instance* of a component is a particular entity that is modelled by that component.

Various kinds of components can be defined due to the divergent purposes of the different types of modelling. We first introduce the conceptual data components, thus neither the user view of the datais considered, nor its implementation.

Definition 5.3 A *data element* is a data component for modelling entities. A *data compound* is a data component for modelling a composition of entities and associations among entities. A *data constituent* is a data component for modelling a simple part of the representation of an entity.

Examples of a data element are entity type, label type, relationship or attribute. Flows, messages sets and states are well known examples of data compounds. We only consider them in a meaningful context format, such as a form, table, list, standard letter, etc. Examples of data constituents are character, digit or pixel of a graphic.

These conceptual data components relate more or less to well known internal data components. A data compound relates to one or more records in a file (or tuples in a relational table), a data element relates to the contents of records or tuples, and data constituents to the bytes in an attribute of a record.

Three types of conceptual process components will be defined. The examples mentioned are from the test-case which can be found in Appendix A.

Definition 5.4 An *activity* models a process that processes instances of data compounds. An activity may be decomposed into other activities, in case the input and output remain data compounds. An activity has to be decomposed in tasks, in case the data compound has to be split into data elements for the precise specification.
Examples of activities are: 'Production of rental report', 'Treatment of film request'.

Definition 5.5 A *task* models a process that processes instances of data elements. A task belongs to the decomposition of an activity and may itself be decomposed in other tasks, in case the input and output of the task are still data elements. A task must be specified in operations, in case the processing refers to the representation of instances of data elements.
The same task may occur in more than one activity, provided the functionality of the task is exactly the same. Examples are: 'Film still available' in the activity 'Treatment of film request', 'Count number of film rentals' in the activity 'Production of rental report'.

Definition 5.6 An *operation* is a process that processes instances of data constituents. An operation belongs to the decomposition of a task.

An operation is not decomposed in the system design stage, but can be decomposed for its realisation in the construction stage, since in this stage machine dependencies are taken into consideration. The same operation may occur in more than one task, provided the process of the operation is exactly the same. Examples are: the comparison of two words being equal, calculating the sum of two numbers.

In the following we will, for ease of discussion, often drop the distinction between the UoD and its models.

5.1.2 The context of task modelling

Let us focus within the whole of IS development activities on the direct context of task modelling, as illustrated in fig. 5.1.

We see that task modelling can be performed after an activity model and a data model have been obtained. The resulting task model is then input to interaction modelling. In an ideal situation code is generated from these models.

We assume that the activities are modelled in a hierarchy of data-flow diagrams. The bottom level of the resulting activity model, defined by the sample criterion (see also 3.4), indicates the data-flows and data stores of which samples have to be provided by the informants. The data modelling starting from these samples yield a global conceptual data model by means of bottom up integration. The activities at the bottom level of the activity model can be further elaborated. This results in tasks that process data, which are specified in terms of the data model. The relationships of the various modelling activitiesis therefore to be expressed explicitly in cross-references among the models.

The same applies for the modelling of the user interaction. Dependent on the modelling conventions, it has to be determined from the activity model for which activities interaction is needed. The corresponding data-flows, data models and task models are the starting points for the modelling of the dialogue. The resulting interaction model should express the integration of the modelling activities. This will not be elaborated here, we refer to [Koesen 89].

In practice as well as in the literature on information systems specification techniques, we see, however, that models of different types are considered as separate and not as being an integrated view on the UoD.

Task modelling is performed during requirements engineering, but is it an analysis or a design activity?

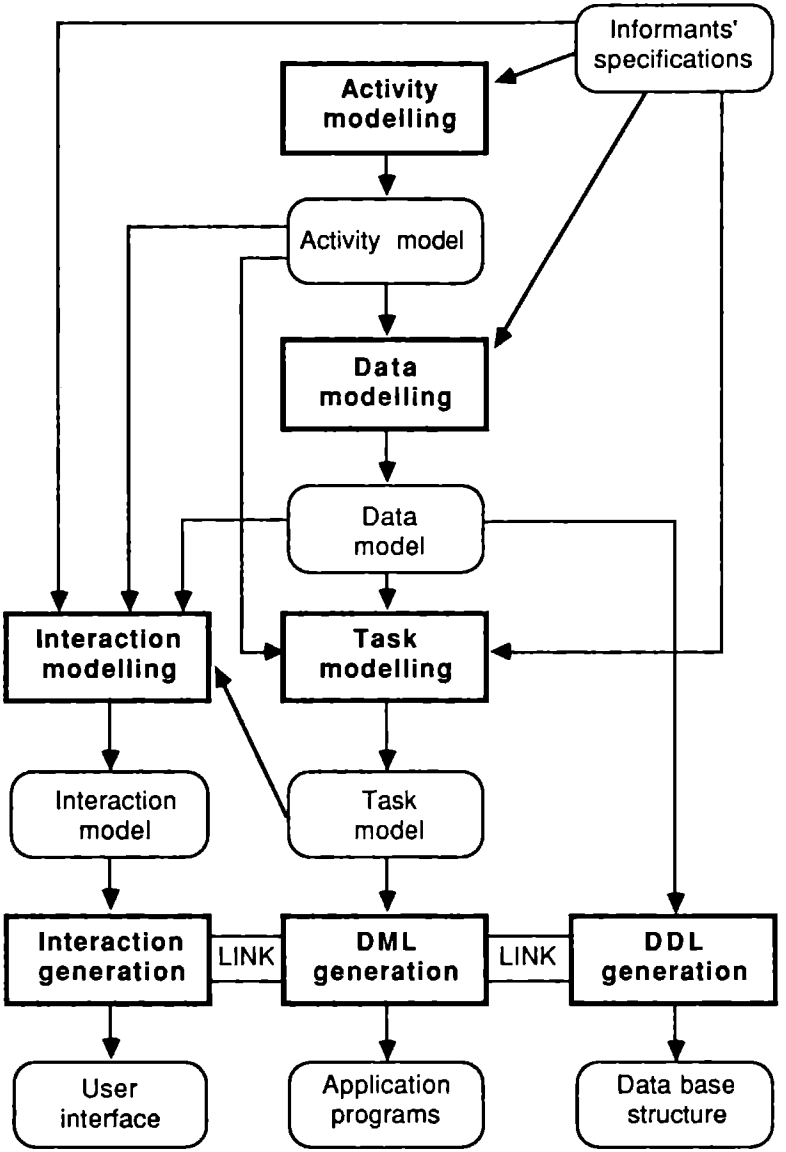


Figure 5.1 The context of task modelling

We claim it can be applied in both ways. The refined specification of activities may require input from the informants about the intricate data processing in the organisation. These existing processes have to be analysed thoroughly, before new process structures can be designed for

automation. It is a good approach to have the existing tasks and the future tasks both explicitly modelled. Differences can be reported, from which the detailed impact of an automated IS on the organisation can be derived. Since the refined specification of tasks is in general too detailed for the analysis stage and is not applicable to all activities, we propose to have this modelling done in the design stage.

The crucial problem is therefore in the specification of the tasks, the processes at the bottom level of the activity model. As observed, the specification activity for tasks is not properly integrated in IS development methods and there is a lack of suitable specification techniques. This chapter is intended to be a contribution to help overcome this lack.

Another non-trivial aspect is the demarkation between activity modelling and task modelling. We see from the definitions in 5.1.1, that the specification of the complete process perspective results in a process hierarchy, consisting of some levels of activities at the top, thereunder some levels with tasks and finally one bottom level consisting of operations. This decomposition is more or less analogous to the decomposition of the data perspective, going from the union of all data compounds at the top, decomposed until simple data compounds transfer in data elements, which in their turn are decomposed into data constituents. We call this phenomenon the **decomposition analogy**. This analogy may serve as a feedback mechanism, when one has to verify both models.

Furthermore, these somewhat pragmatic definitions serve as guide-lines to determine when the decomposition of activities or tasks should stop.

For the decomposition of activities we call the stop criterion the **sample criterion**. Since in the analysis stage only global modelling is of interest, the decomposition of activities may continue when the input and output flows are still data compounds, and should stop otherwise. The informants are asked to provide the analysts with samples of the input and output, such as forms, lists and tables, preferably filled in with some significant population. These samples are in turn to be utilised for the data modelling phase. This is discussed in chapters 3 and 4. This criterion is also motivated by the lack of modelling capabilities for triggering, decisions, iterations, etc., in the specification conventions for activities, like data-flow diagrams and A-graphs (see also 5.1.4).

The modelling of tasks during requirements engineering starts at the level where the modelling of activities has stopped. An objective of this stage is to construct a complete design of the IS, which implies the absence of communication with the informants in the stages thereafter. In the construction stage, the programmers must be able to do their job with the design products of requirements engineering as the only input. This leads to the **complete design criterion**, which specifies that the decomposition during the modelling of tasks should stop when the tasks at the lowest level of the decomposition are all completely specified as operations, which can be straightforwardly programmed.

5.1.3 Requirements of task modelling techniques

There exist numerous specification techniques for processes, whcih could be employed for the modelling of tasks. According to the discussion in the previous section, we impose on such a specification technique the following requirements:

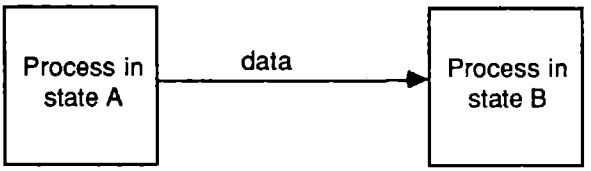
1. The technique should enable fluent transfer between the phases and steps. Cross-references between models should be explicit.
   Since the task model, like all models of system development, is highly related to other models, these relations should be formulated and their recording should be incorporated in the development activities. This gives rise to the need for automated support of the relationships among models. This support is called modelling transparency, which is further discussed in section 6.3.
2. The technique should be complete with respect to the specification of control flow, i.e. triggers, decisions, dynamic constraints and iteration.
   In fact, it should be possible to specify all kinds of control flow, which can be expressed in a target programming language. However, we restrict ourselves to business information systems, for which, for instance, the power of recursion is not required.
3. The technique should produce unambiguous models that can be input to code generation or programming in a straightforward manner.
   As discussed in the preceding section, it is important that the data manipulation part of the system can be generated effectively.
4. The technique should have a sound formal theoretical basis to enable the verification of theoretical statements and the formulation of rules about models that underlie all sorts of validation analysis.
   This also includes the communication regarding the models. Formality is the only means to ensure reliable communication. Analogous to Petri-nets [Reisig 85], it should be possible to derive or check certain invariants of the models.
5. The technique should be diagrammatic and enactable in order to ensure fast comprehension of the models during all kinds of written and verbal communication.
   Although it is often claimed that pictorial specification is informal , we state that it is possible to define such a diagrammatic technique in conjunction with a formal mathematical framework. Diagrams allowthe possibility of simulation and can also be populated to provide insight in the modelled task. In [Green 82] various diagrammatic specification techniques were assessed with respect to their suitability for fast and correct comprehensibility.
6. The technique should be complete with respect to data manipulation: retrieval of (derived) data as well as updates of the data.
   Tasks process data, hence various operations required for this should be included.

Similar and additional requirements are formulated in [Dubois 85] and in [Schiel 89]. It should be noted, that we deliberately did not include the requirements of other types of models, since we do not intend to define a new technique which comprises all kinds of specification. We restrict

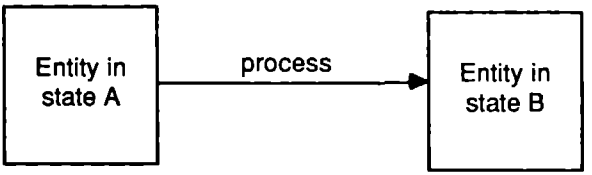
ourselves to the specification of tasks. The fulfilment of the above requirements by existing techniques is addressed in the next section.

5.1.4 Existing task modelling techniques

Generally speaking, process modelling techniques model state transitions. Dependent on the point of view two types of task modelling techniques can be defined, that are dual to each other (see fig. 5.2).



Data processing techniques



Entity life cycle techniques

Figure 5.2 Duality of types of task modelling techniques

a. **Data processing techniques.** Processes operate on input data to produce output data. This processing changes the state of the process. These techniques can best be applied by using conventional types of programming languages and data base management systems. Examples are data-flow diagrams and the CTM.

b. **Entity life cycle techniques.** For each entity type, all the different states are related diagrammatically. Processes indicate the change of states. These techniques are best suited for object-oriented data base management systems. Process algebra is a typical example.

Many techniques have been proposed for process specification, albeit that not all of them were especially intended to be used for task modelling. We have reviewed these techniques on their applicability for task modelling by

assessing them in relation to the requirements formulated in section 5.1.3. These techniques do not satisfy many of the requirements.

We first review the techniques, that we classify as data processing techniques. To start with, it is clear that specification techniques for global activity modelling, such as *Data-Flow diagrams* [Gane 79], are not capable to model tasks, because they are not formal and do not express data-manipulation (req. 4 and 6).

Action diagrams, discussed in [Martin 85], is a kind of pseudo-coding technique and hence lacks a formal framework. That is due to the fact that the language to be used is not formally described (req. 4).

ACM/PCM [Brodie 82] supports the modelling of tasks by using so-called transactions in conjunction with pre-conditions and post-conditions. Only database updates can be modelled. Not only for this reason is ACM/PCM too restrictive, but also because of the strong formulation of processes in terms of the data model (req. 6).

RIDL, as it was introduced as a conceptual database language [Meersman 82], lacks the relationships with other modelling techniques and is not diagrammatic. Moreover, RIDL cannot express triggers (req. 1,2 & 5).

REMORA [Rolland 82] meets the requirements quite well. It possesses only diagrammatic features for data and behaviour. Processes are specified in the language ISDEL, a relational language. Furthermore the data elements (c-objects) are required to be relational tables in third normal form, which restricts the output language of the application program generation (req. 3).

IML [Richter 82] is comparable to the approach of CTM and will be discussed in section 5.2. The diagrammatic data modelling technique is hard to grasp (req. 5).

The technique of *Structure Charts* [Yourdon 79] is also used to model processes, but is geared towards the modularisation of the resulting software. All kinds of task constructs can not be modelled. In addition, the technique is not complete with respect to data manipulation (req. 2 & 6).

In [Buhr 85] the *Ada Structure Graph* (ASG) technique is introduced. This is geared towards the Ada language. The approach takes therefore the constructs of the language as a starting point and models only the control and data flow of the processes, not the data-manipulation (req. 6).

EXSPECT introduced in [van Hee 88], aims at executable specification of a complete information system and makes code generation superfluous. The tri-partition of processors, stores and triggers enable the specification of the process, the data and the behaviour perspectives of an information system. EXSPECT meets the requirements relatively well, although it is intended to be a stand-alone complete technique and has therefore no links with other

specification techniques (req. 1). Experimentation should prove the suitability of the associated mathematical specification technique.

We now turn our attention to the other class of techniques, the entity life cycle techniques. These are not so numerous as the data processing techniques.

Process algebra [Bergstra 86] and related specification techniques, such as *CSP* [Hoare 85] and *CCS* [Millner 80], deal mainly with the formalisation of the communication and synchronisation aspects of processes, and are not integrated with the modelling of processed data. They also lack a proper connection with the other modelling techniques (req. 1&3). In [Wieringa 88] Process Algebra is used to describe the dynamics of objects and has some data modelling capabilities incorporated in order to fulfil requirement 3. Despite its rigorous axiomatisation it lacks convenient data manipulation mechanisms to be appropriate for task modelling (req. 6).

The *Function Logic Models* [MacDonald 82] are another analysis technique for the life cycle of entities. It has no formal basis and the manipulation facilities are restricted due to the distinction made between the entities and attributes of the underlying Entity-Relationship model (req. 4&6).

One of the first techniques aiming at code generation is *JSD* [Jackson 83], but the integration with data modelling is rather restrictive (req. 1). Extensions to the technique in this direction have been proposed [Mees 86].

Finally, *Petri-nets* [Reisig 85] can be used both as a data processing technique as well as an entity life cycle technique. It also lacks the possibility of fluent transfer to other techniques, and in addition it is not intended to model data manipulation (req. 1&6).

The aforementioned techniques were chosen to be representative of the numerous techniques proposed in the literature of information systems development methods. We are of the conviction that more techniques will be proposed in future, which aim at the complete generation of the code for data manipulation properly linked with the user interaction and data definition.

5.2 THE CONCEPTUAL TASK MODELLING TECHNIQUE

Task specification, as we propose it here using the Conceptual Task Model (CTM), continues with the results of the global activity model and the completed data models (see fig. 5.1). The manipulation of the data is expressed in terms of the data model, for which we use NIAM [Nijssen 89] and RIDL [Meersman 82]. The work here is an elaboration of ideas inspired by the work of Genrich [Genrich 87], Kung and Sölvberg [Kung 86], Richter and Durchholz [Richter 82] and Reisig [Reisig 87].

This section consists of four parts. The first part defines the Conceptual Task Model (CTM) and states some of the motivations for this definition. In

the second part an example of a CTM-net is given. In the third part some of the properties of the CTM are discussed in terms of the meta-model of CTM.

5.2.1 Definition of the conceptual task model

Predicate/transition nets form the basis for the Conceptual Task Model. They were introduced by Genrich and others in a series of articles, starting with [Genrich 79], and at the moment concluded by [Genrich 87].

Predicate/transition nets

To simplify the presentation of CTM, we will first give an introduction to Predicate/transition nets (PrT-nets). In short, PrT-nets are interpreted, inscribed high-level Petri nets, where the inscriptions consist of variables for individuals (as opposed to the non-individual token of Petri nets) and truth-valued expressions, preferably in first-order predicate logic.

Predicates correspond with states of a system the PrT-net is supposed to model. The extension of the predicate is the set of things for which the predicate holds. Due to events occurring in the modelled system, these extensions may change; the predicates are in fact variable. Transitions model the consequences of events. They are linked to predicates, of which the extension may change due to the execution of the transition.

Graphically, predicates are depicted by circles and transitions by rectangles. A causal dependency between a predicate and a transition, in either direction, is denoted by means of an arrow connecting the predicate and the transition.

Arrows may be inscribed with linear combinations of tuples. Transitions may be annotated with so-called 'transition selectors', being truth-valued formulas which can be interpreted in a static structure, called the support of the net. The inscriptions of arrows indicate variables local to the transition to which they are attached. The variables normally appear in the transition selector.

An arrow leading to a predicate means that the predicate is augmented by a number of tuples of individuals as indicated by the inscription of the arrow. An arrow coming from a predicate means that the predicate is diminished by a number of tuples of individuals as indicated by the inscription of the arrow. A transition is 'enabled', that is changes can take place, if there exists a substitution for the inscriptions of the arrows attached to the transition, such that the transition selector evaluates to true and such that all the predicates involved can change in the manner indicated. A predicate can be diminished by a tuple if this tuple is in its extension. A predicate can be augmented by a tuple if this tuple is not in its extension.

Examples of the application of PrT-nets can be found in [Genrich 87]. PrT-nets are especially suited for the specification of distributed systems such as banking transaction systems and complex telecommunication systems. For a complete and formal treatment of PrT-nets refer to [Genrich 87].

CTM-nets

A CTM-net is a PrT-net where

- Instead of the formalism of first-order logical formulas and their structures, the conceptual data modelling language NIAM [Nijssen 89] in combination with the corresponding data manipulation language RIDL [Meersman 82] is used as a supporting structure. Functions and expressions, which can be seen as special kinds of RIDL functions, are interpreted in this structure. Note, however, that any combination of data modelling technique and data manipulation language would suit this purpose. For example, relational tables and SQL queries, or the Entity-Relationship model [Chen 76] and the language GORDAS [Elmasri 85] could be used.
- A distinction is made between task places and information places (from now on the terms 'place' and 'predicate' will be treated as synonyms). A conceptual schema in NIAM is related to both kinds of places. Each place of the PrT-net is either a task place or an information place. The conceptual schema of an information place determines the information structure of the tuples that can enter that place. The conceptual schema of a task place describes that part of the Universe of Discourse consisting of all the individuals of the tuples that can enter that place.
- An additional typing is related to each task place. When the arity of a task place P is n, a typing $<T_1,T_2,...,T_n>$ is associated with P such that for every tuple $<P_1,P_2,...,P_n>$ that can enter P we have that P_i is of type T_i (for all $1 \leq i \leq n$). The typing of a task place is a linear representation of the two-dimensional conceptual schema associated with that task place. This typing is necessary for the specification of parameters in the RIDL expressions.
- Arrows may not be labeled with linear combinations of tuples, but only with single tuples. Applications in business administration systems do in general not require complex data structures. Multiple copies of tuples are not necessary and combination of tuples can be modelled using extra predicates.

The first adaptation was already suggested by Genrich in [Genrich 87] and put into practice in a comparable way as done here by Richter and Durchholz in IML [Richter 82]. There are however some essential differences between IML and CTM. In IML the data modelling technique used, IMC, is less powerful than the data modelling technique used in CTM, i.e. NIAM. The data manipulation language used in IML, the Information Management Language (IML), is hard to understand and in our opinion not on a conceptual level, contrary to the data manipulation language RIDL used in the CTM. Furthermore, in IML aspects such as consistency and completeness analysis are hardly addressed.

The adaptations to the formalism of PrT-nets were made for the following reasons:

- During the information systems development process one should be able to transfer fluently from the substage of modelling of activities and data to the substage of task modelling. The activities are to be refined into the tasks and the data that is processed in the tasks should, as far as it will be recorded, have been specified during the data-analysis stage.
- The data being input for, output of, or intermediate in, the tasks should be specified as rigorously as is done in the data analysis phase. Preferably the same specification technique should be used. The data models in NIAM provide insight into the semantics of the data, which is processed in a task, and into its relationship to the global data model.
- The interaction with databases should be specified on a conceptual level and easy to express. RIDL fulfils these requirements.

Some more should be said about the distinction between task places and information places. Task places can be associated with certain stages of a task and contain the intermediate information and control available there, while information places contain the more stable information and always correspond to a data store of a data-flow diagram. In general, data at task places correspond with local data in application programs, whereas data at information places correspond with global data from the data base.

Finally, we have adopted three notational conventions. The first convention is that if we have n ($n > 1$) disjoint conditions ($C_1,C_2,...,C_n$), possibly combined with m ($m \geq 0$) other conditions ($Q_1,Q_2,...,Q_m$), then instead of having n separate transitions for each condition, we introduce one combined transition containing all conditions, as shown in fig. 5.3.

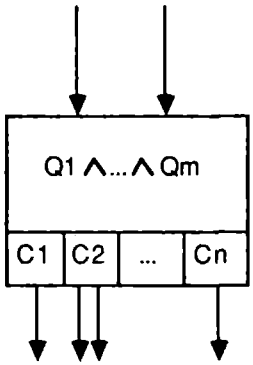


Figure 5.3 Combined transition for C1,C2,...,Cn

Output arrows coming from a transition containing condition C_i are now attached to the little box containing C_i inside the combined transition.The second and third notational conventions employed to reduce the number of arrows in a CTM-net are shown in fig. 5.4.

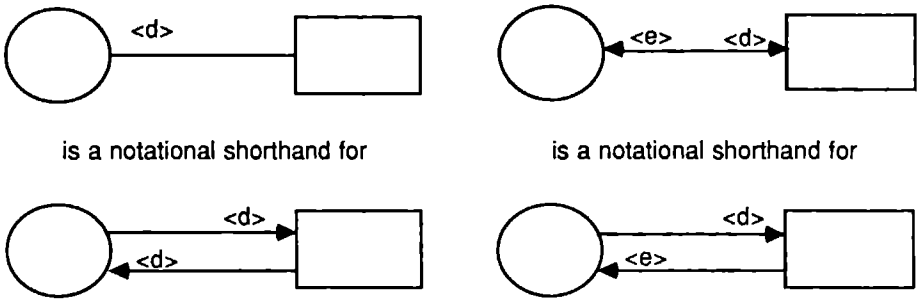


Figure 5.4 Double arrow convention for the CTM

5.2.2 Example: The Video store case

In order to present an example of a CTM-net, we will make use of the small case involving a Video store, which is given in appendix A.

Based upon the description of the task 'Treatment of film request' a CTM-net for this task can be constructed. This is shown in fig. 5.5. The corresponding data models of the information places "List of Rentals" and "Information concerning films and tapes" are depicted in fig. 5.6. These schemas are parts of the conceptual schema in fig. A.4. The RIDL functions used in the transitions of fig. 5.5. are shown in fig. 5.7. In the function headings the databases on which these functions operate are specified, whereas in the function definitions these relations are left implicit.

The identification of the entity types and role names in the conceptual schemas of the places are not always shown. In addition, a notational convention for derivable relations is adopted. The star which denotes a redundant relation is not put outside the whole relation, but inside the redundant roles belonging to that relation.

Whenever a film is requested, the task 'Treatment of film request' is activated. A film request can be considered as a pair <f,p>, consisting of a person 'p' and the film 'f' that this person wants to borrow. As a person can request more than one film and a film can be requested by more than one person (in the course of time), there is a many-to-many relation between the entity types 'Rentable film' and 'Person' in the data model of the task place 'Film request'.

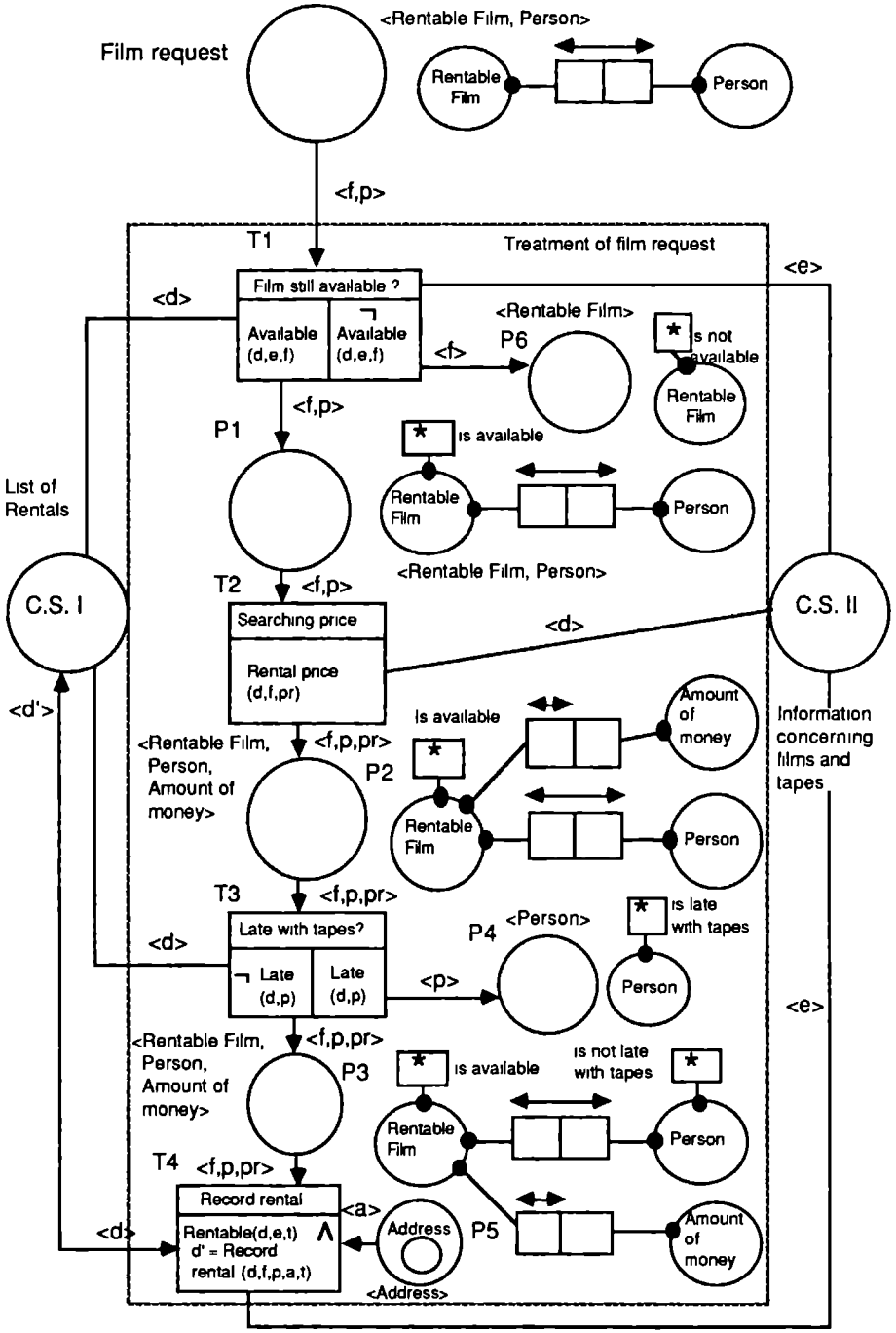


Figure 5.5 CTM for the task 'Treatment of film request'

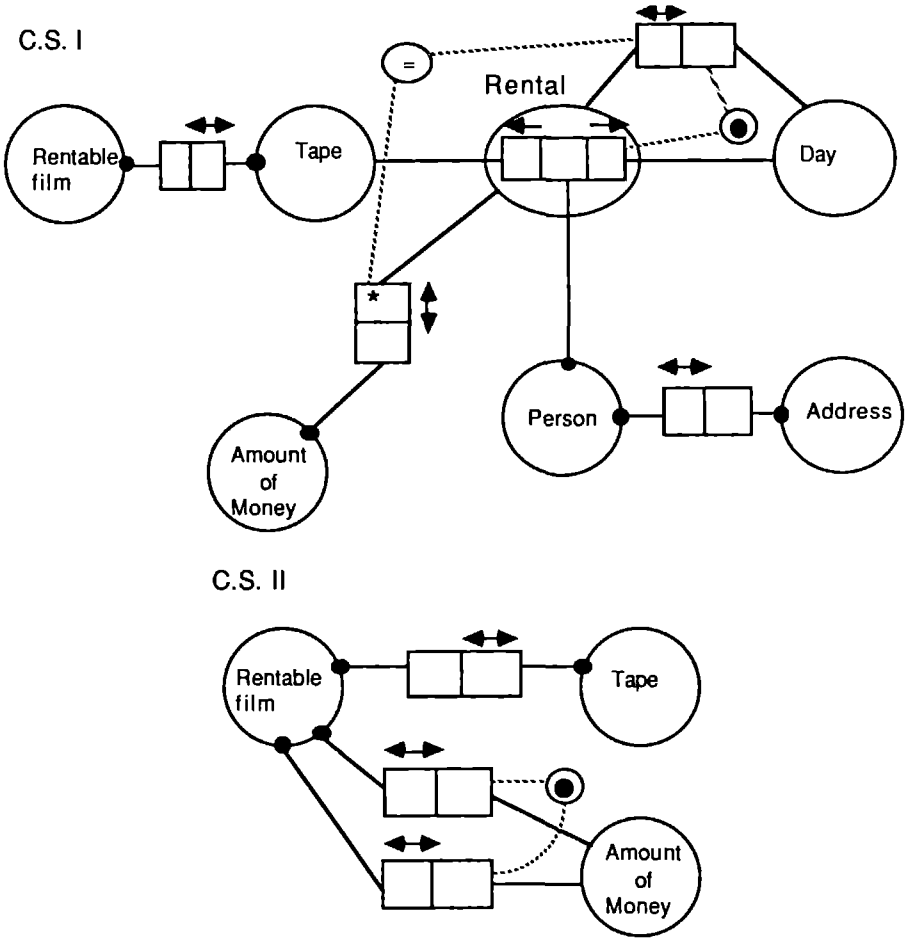


Figure 5.6 Conceptual schemas belonging to figure 5.5

```
FUNCTION Available (DATABASE a,b; RENTABLE FILM f)BOOL;
BEGIN
      NUMBER OF (Tape (contains Rentable-film f  AND
      OF Rental NOT has-return-date Day))
      IS LESS-THAN
      NUMBER OF (Tape contains Rentable-film f)
END;

FUNCTION Rental price (DATABASE a; RENTABLE FILM f,
                       AMOUNT OF MONEY pr)BOOL;
BEGIN
      pr = Amount-of-money is-rental-price-of Rentable film f)
END;

FUNCTION Late (DATABASE a; PERSON p)BOOL;
BEGIN
      NUMBER OF (Tape (OF Rental NOT has-return-date
      Day AND has-been-borrowed-by Person p
      on Day d WHERE now-d > 3))
      IS GREATER-THAN 0
END;

FUNCTION Rentable (DATABASE a,b; TAPE t)BOOL;
BEGIN
      t IN Tape contains Rentable-film f MINUS
      Tape OF Rental NOT has-return-date Day
END;

FUNCTION Record rental (DATABASE b; RENTABLE FILM f, PERSON p,
                        ADDRESS a, TAPE t)DATABASE;
BEGIN
      ADD Tape t has-been-borrowed-by Person p on Day now
      ADD Person p lives-at Address a
END.
```

Figure 5.7 RIDL-queries belonging to CTM-net of task 'Treatment of film request'

The first transition 'Film still available' of the CTM-net for 'Treatment of film request', tests whether the film the customer requests is available. A film is available if the number of currently rented tapes of that film is less than the total number of tapes of the film. The RIDL function 'Available' performs this comparison, using the information from the information places 'List of Rentals' and 'Information concerning films and tapes' and operating on the corresponding conceptual schemas.

Note also that in the conceptual schema of 'P1' the entity type 'Rentable Film' is now augmented with the unary fact type 'is available'. This fact type is derived from the contents of the database by the RIDL function. This

knowledge about the films is used in subsequent transitions. For example in the transition 'Record rental', the availability is not checked again.

If the requested film is not available, a tuple consisting of this film is placed in the task place 'P6'. In this case the task is finished and the user interface should handle the rest, for example, give a message to the user that the film is not available at the moment.

In the event that the film is available, the next transition 'Searching price' is enabled. In this transition, the rental price of the film is searched by the RIDL function 'Rental price'. This function operates on the conceptual schema of the information place 'Information concerning films and tapes', since this place contains the relation between a film and its rental price, and has as input parameter the requested film 'f' and as output parameter the rental price 'pr'. Since the output place 'P2' of this transition contains a three-tuple consisting of a film, its rental price and the person who wants to borrow that film, the conceptual schema of 'P2' is an augmented version of the conceptual schema of 'P1'. The relation between the entity types 'Rentable film' and 'Amount of money' denoting the rental price of a film, is added.

The transition 'Late with tapes' works analogous to transition 'Film still available'. A person is late with the return of a tape, if this tape has been borrowed more than three days ago.

The last transition of this task, 'Record rental', records the rental in the data store 'List of Rentals'. For this it not only needs the person and the requested film (note that the rental price is not used in this transition), but also the address of this person 'a'. Since this address may not be recorded in the list of rentals yet, we assume that this address is always asked for, as is specified in the description in appendix A. User interaction is necessary for this. The result of the interaction, the address of the person, is placed in task place 'P5'. Now the transition 'Record rental' is enabled. The RIDL function 'Rentable' selects an available tape containing the requested film, so in fact 't' is an output parameter of this function, and the function 'Record rental' actually records the rental by updating the list of rentals. Note that the function 'Record rental' yields a whole data base, i.e. the updated list of rentals, and that in the transition this result is assigned (when one looks at it operationally) to the new data base tuple that will be placed in 'List of Rentals'.

Note also that the input and output relations between tasks on the one hand and data-flows and data stores on the other hand as they exist in the data-flow diagram, remain the same in the CTM decomposition of these tasks, where we should keep in mind that data-flows become task places in the CTM-net and the before mentioned remark that data stores become information places. This property will be more formally addressed in section 5.3.

5.2.3 Properties of the CTM

In fig. 5.8 the meta-model for the CTM is shown in the form of a NIAM conceptual schema. This meta-model shows the links between the concepts of the CTM by means of the relationships and some of the properties of the CTM by means of the constraints. For example, every place has a conceptual schema, all task places have a typing and every place is of precisely one type. Other simple properties, such as the fact that a transition can have one or more output places and one or more input places, or the fact that there aren't any 'dangling' places, places that are neither input for a transition nor output of a transition, can also be deduced from the meta-model.

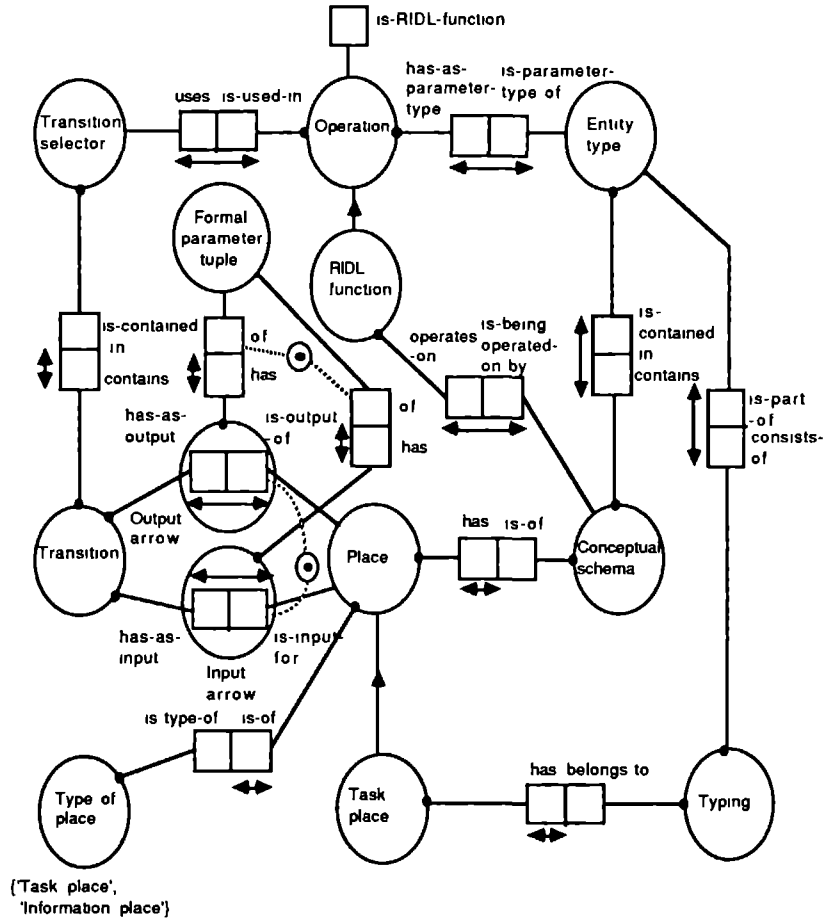


Figure 5 8 Meta-model for the CTM

The CTM, however, has a lot of additional properties, which can not be represented graphically. These properties can be formulated as constraints on meta-model in a formal way using RIDL, however we will not do so for reasons of clarity. Instead we shall give an informal formulation of each of these properties.

1. The typing of a task place is a flattened representation of the conceptual schema belonging to this task place. This was mentioned already in section 5.2.1. As an example consider the conceptual schema and the typing of the task place 'P2', <Rentable Film, Person, Amount of money>, in fig. 5.5.
2. Information places are never only input for, or only output of a transition. Information places always contain exactly one tuple. Database tuples are not dynamically created nor removed. An information place is therefore always connected to a transition by a double arrow. The contents of the database (tuple) on the other hand may, of course, vary. Again this can also be seen in fig. 5.5, where the information places 'List of Rentals' and 'Information concerning films and tapes' are never connected to a transition by a single arrow.
3. A task place is never input for more than one transition. Combined transitions, as e.g. in fig. 5.3, are considered as one transition. This has to do with the fact that we do not consider synchronisation aspects in our task model, but rather have a separate event model for dealing with these synchronisation aspects. The reason for this is that we want to distinguish between the control of tasks, the conditions under which tasks are initiated, and the functionality of tasks, i.e. the way they process data. This distinction between control behaviour on the one hand and functional behaviour on the other hand is also made in e.g. [Guyot 86] and [Hatley 87]. The event model should also prevent tasks from being initiated when they are already 'busy'.
4. The RIDL functions, used in a certain transition, operate on the conceptual schemas (and their population) of the information places input for this transition. As noticed before, this is done in a (somewhat) implicit way. RIDL functions never operate on the conceptual schemas of tasks places. Task places only provide for entity type parameters.
5. The entity types being a parameter type of a RIDL function always occur as entity types in the conceptual schemas on which this function operates. This derivable subset constraint is not shown in the meta-model, since parameters of a RIDL function always appear as instances of entity types, which by definition form part of the conceptual schema this function operates on. This can be seen in fig. 5.7.

The meta-model does not represent all the properties the CTM has, there are many more. The elaboration of some other properties will be discussed in chapter 5.3.

Constraints involving instances

In the CTM we can express dynamic aspects in a convenient way. The data models of both input and output specify the types of the data and their

constraints. RIDL is an easy to use language in conjunction with these models for the specification of the data processing.

There might however be constraints that deal with instances and can therefore not be expressed in the data models. These constraints are to be specified in RIDL using the IF statement and an appropriate predicate. The underlying transition should always be executed only if the predicate is satisfied.

There exist three types of constraints involving instances and we give an example of such a constraint in terms of the case of appendix A.

1. A constraint on the current state of the data. For example the rule that a Tape can be rented by a Person if that Person is not late with Tapes.
2. A constraint on the new state of the data. For example the rule that a new rental price may only be specified in entire dollars.
3. A constraint on both the current and new state of the data. For example the rule that the rental price may only increase if it is changed.

To check the constraints the current, and/or the new value of the data must be obtained and substituted in the constraint.

In [Prabhakaran 88] an extension to the NIAM technique was proposed to depict this kind of constraint graphically. The fact types of the global data model act as specification of the states of the data. The states are connected to events, that take place when the constraints are satisfied. This extension was not adopted in the CTM, because it appeared to be too complex. Furthermore, there might be more than one process in which a particular part of the data model is involved with different constraints. The specification of the constraints should therefore be coupled to the process models, and thus also to the task model. However, should the constraint get too complex, then a graphical Petri-net like notation may support understanding. The states of the data can be specified using RIDL statements in terms of the data model.

5.3 FORMALISATION

In this section first we will address four theoretical issues concerning the Conceptual Task Model. First, a formal definition of the CTM is given including the rules, that a correct CTM must satisfy. Thereafter, the relation between the task model, denoted in CTM, and the activity model, denoted in data flow diagrams, is formally defined. Finally, the computational power of the CTM and the correctness of conceptual schemas at task places are discussed.

5.3.1 Formal definition of the Conceptual Task Model

The formal definition of the Conceptual Task Model is a syntactic one. Semantical aspects will not be addressed since the precise operational meaning of a CTM-net is very hard to define. (It is already quite difficult to

define part of the semantics of RIDL, see [Bruza 89].) As mentioned before, part of the semantics of the CTM can be derived from the definition given in section 5.2 and the semantics of PrT-nets, NIAM and RIDL.

Definition 5.7

A CTM-net is a 12-tuple (Π, T, P, Σ, Ψ, Z, I, Φ, Λ, Θ, X, Ω), where

1. Π is a non-empty finite set of places,
2. T is a non-empty finite set of transitions (with transitions we do not mean combined transitions here),
3. P is a finite set of parameterised RIDL expressions,
4. Σ is a non-empty finite set of conceptual schemas,
5. Ψ is a finite set of linear typings (a linear typing is a tuple of arbitrary length consisting of entity types),
6. Z is a non-empty finite set of variables,
7. $I \subseteq \Pi$ is a set of information places; $\Gamma = \Pi/I$ (by definition) is the set of task places,
8. $\Phi \subseteq \Pi \times T \cup T \times \Pi$ is a non-empty set of arrows, denoting that a place is input for or output of a transition (the arrows to and from an information place are considered separately),
9. $\Lambda \in \wp\wp(Z)^{\Phi}$ is a function from the set of arrows Φ to the set $\wp\wp(Z)$ of tuples of arbitrary length of variables chosen from Z, denoting the labeling of the arrows with a tuple of variables,
10. $\Theta \in P^{T}$ is a function from the set of transitions T to the set of parameterised RIDL expressions P, denoting which RIDL query belongs to which transition,
11. $X \in \Sigma^{\Pi}$ is a function from the set of places Π to the set of conceptual schemas Σ, denoting which conceptual schema belongs to which place,
12. $\Omega \in \Psi^{\Gamma}$ is a function from the set of task places Γ to the set of linear typings Ψ, denoting which typing belongs to which task place.

Note that the elements of the 12-tuple correspond more or less with the meta-model of the CTM given in section 5.2.3. The sets of the 12-tuple correspond with entity types of the meta-model and the functions with the relationships. For the sake of simplicity a reduction was employed.

Examples of elements (shown in italics) of each constituent of the 12-tuple, that are borrowed from the CTM-net of fig. 5.5, are the following.

1. Π = {*Film request, P1, P2, P3, P4, P5, P6, Information concerning films and tapes, List of Rentals* }.
2. T = {*T1a, T1b, T2, T3a, T3b, T4*}. The combined transitions have been unfolded. With *T1a* we mean the transition containing the RIDL expression *Available(d,e,f)*, that results after unfolding T1.

3. The expressions in fig 5.7 are the elements of P.
4. The elements of Σ are the conceptual schema depicted next to each task place and the ones shown in fig. 5.6 for the information places. The conceptual schema next to the place *Film request*, is an example of an element of Σ. We will refer to this conceptual schema as *C1*.
5. Ψ = {*<Rentable Film, Person>, <Rentable Film>, <Rentable Film, Person, Amount of Money>, <Person>, <Address>*}.
6. Z = {*f, p, pr, a, d, d', e*}.
7. The set of information places I = {*Information concerning films and tapes, List of Rentals* } and the set of task places Γ = {*Film request, P1, P2, P3, P4, P5, P6*}
8. The arrow *(T1a, P1)*, going from transition *T1a* to place *P1*, indicating that *P1* is output of *T1a*, is an element of T × Π and an element of Φ. The arrow *(P1, T2)*, going from place *P1* to transition *T2* indicating that *P1* is input for *T2*, is an element of Π × T and also an element of Φ.
9. The ordered pair *((P1, T2), <f,p>)* is an element of Λ, which means that the arrow going from place *P1* to transition *T2* is annotated with tuple *<f,p>*.
10. The ordered pair *(T2, Rental price(d,p,pr))* is an element of Θ, which means that the RIDL expression *Rental price(d,p,pr)* belongs to transition *T2*.
11. The ordered pair *(Film request, C1)* is an element of X, which means that conceptual schema *C1* belongs to the place *Film request.*
12. The ordered pair *(Film request, < Rentable Film, Person>)* is an element of Ω, which means that the typing *<Rentable Film, Person>* belongs to the task place *Film Request.*

We now formulate all kinds of properties that a correct CTM-net in terms of the 12-tuple must satisfy. To formulate these properties some auxiliary functions and predicates must be introduced in an informal way.

- The function *entity* operates on a conceptual schema and yields the set of entity types occurring in that conceptual schema.
- The function *merge* operates on a set of conceptual schemas and yields the integrated union of these schemas.
- The predicate *part_of* defines a binary relation between conceptual schemas and is true if and only if the first conceptual schema is part of the second conceptual schema.
- The function *domain* operates on a RIDL expression and yields the domain (this is a conceptual schema) of that expression.
- The function *vars* operates on a parameterised RIDL expression and yields the set of parameters (i.e. a subset of Z) of that expression.
- *Type_in_expression (r, v, e)* is true if and only if the formal parameter *v* (from Z) has as type the entity type *e* in the RIDL expression *r*.
- The function *length* determines the number of constituents of a tuple of types or of variables.
- The selection of an element at a certain position in a tuple is denoted with a subscript.

The reason that these functions and predicates are not introduced in a formal way is the fact that they depend on a formalisation of RIDL and NIAM, which is not realised yet.

The following properties must hold for the 12-tuple:

Property 5.1 Information places are always input and output of the same transition:

$$\forall p \in I \; \forall t \in T \; [\; (p,t) \in \Phi \Leftrightarrow (t,p) \in \Phi \;]$$

Otherwise, an information place is never only input for, nor only output of a transition. This property is due to the PrT-net formalism. Information places always contain exactly one tuple, database tuples are not dynamically created nor removed. The omission of an arrow would imply the creation or deletion of the contents of a database as a whole. Therefore an information place is always connected to a transition by a double arrow. This is the reason for the notational convention in fig. 5.4. From this, the contents of the database can vary as is required.

Property 5.2 Every place is input for or output of a transition:

$$\forall p \in \Pi \; \exists t \in T \; [\; (p,t) \in \Phi \vee (t,p) \in \Phi \;]$$

There are no dangling places. In the meta-model of CTM (fig. 5.8) this property is reflected by the combined totality constraint.

Corollary 5.1 Every information place is input for at least one transition and also every information place is output of at least one transition:

$$\forall p \in I \;\; | \{ t \in T \mid (p,t) \in \Phi \} | \geq 1$$

$$\forall p \in I \;\; | \{ t \in T \mid (t,p) \in \Phi \} | \geq 1$$

This implies that any data store in the activity model is involved in at least one transition and so is the data in this data store. The set of information places of a particular CTM-net may be empty, which means that this task only performs operations that do not retrieve nor update stored data.

Property 5.3 All transitions have input and output:

$$\forall t \in T \; \exists p_1, p_2 \in \Pi \; [\; (p_1,t) \in \Phi \wedge (t,p_2) \in \Phi \;]$$

Every transition has at least one output place and one input place. This is shown in the meta-model with total roles on the two fact types Transition has-as-output/input Place.

On the basis of the specification so far, it is possible that a CTM-net may consist of two separate parts between which no exchange of information

takes place. Since this might occur due to erroneous specification, we forbid this via the following well-formedness rule.

Property 5.4 A CTM-net should not be splittable, that is the bi-partite non-directed graph $(\Pi \cup T, \Phi \cup \Phi^{-1})$ of a CTM-net should be *connected*:

$$\forall s, s' \in \Pi \cup T \; \exists s_1, s_2, \ldots, s_n \in \Pi \cup T \; [(s,s_1), (s_1,s_2), \ldots, (s_n,s') \in \Phi \cup \Phi^{-1}]$$

This means that all places and transitions of one task are connected to each other via a path of arrows along other places and transitions.

Property 5.5 Entity types of conceptual schemas and typings correspond:

$$\forall p \in \Gamma \; [\; \cup \Omega(p) = \text{entity}\,(X(p))\;]$$

In other words, the set (not multiset!) of entity types occurring in the typing of a task place is the same as the set of entity types occurring in the conceptual schema of that task place. This is an equality constraint on a cycle in the meta-model.

Property 5.6 Types of variables in expressions and in tuples correspond:

$$\forall t \in T \; \forall v \in Z \; \forall e \; [\; \text{type\_in\_expression}\,(\Theta(t), v, e) \Rightarrow$$

$$\forall p \in \Gamma \; \forall i \; [\; (((p,t) \in \Phi \wedge (\Lambda(p,t))_i = v\;) \Rightarrow (\Omega(p))_i = e\;) \wedge$$

$$(((t,p) \in \Phi \wedge (\Lambda(t,p))_i = v\;) \Rightarrow (\Omega(p))_i = e\;)]\;]$$

The type e of a formal parameter v, as can be derived from the typing of the task place t to which it belongs, should agree with the way this formal parameter is used (i.e. of which type it is supposed to be) in the RIDL expression of the transition $\Theta(t)$ in which it is a local variable.

Property 5.7 Parameters in a RIDL expression are either from the input or for the output:

$$\forall t \in T \; [\text{vars}(\Theta(t)) \subseteq \{\, w \in Z \mid \exists\, p \in \Pi \; [\; (t,p) \in \Phi \wedge w \in \cup \Lambda((t,p))]\} \cup$$

$$\{\, w \in Z \mid \exists\, p \in \Pi \; [\; (p,t) \in \Phi \wedge w \in \cup \Lambda((p,t))]\} \;]$$

The parameters in the RIDL expression $\Theta(t)$ of an arbitrary transition t are a subset of the union of the parameters of the input and output places. Note that this property is not derivable from property 5.6.

Property 5.8 The lengths of tuples correspond:

$$\forall p \in \Gamma \; \forall t \in T \; [\; ((p,t) \in \Phi \Rightarrow \text{length}\,(\Lambda(p,t)) = \text{length}\,(\Omega(p))) \wedge$$

$((t,p) \in \Phi \Rightarrow \text{length}(\Lambda(t,p)) = \text{length}(\Omega(p)))]$

The length of the tuples annotating arrows connected to a certain task place should be equal to the length of the typing of that task place.

Property 5.9 A tuple of an information place has length 1:

$$\forall p \in I\ \forall t \in T\ [\ (p,t) \in \Phi \Rightarrow (\text{length}(\Lambda(p,t)) = 1 \wedge \text{length}(\Lambda(t,p)) = 1)\ ]$$

The tuples annotating arrows connected to an information place always have length one, since these tuples stand for the contents of the data store as a whole.

Corollary 5.2 All tuples around a place have the same length:

$$\forall p \in \Pi\ \exists m\ \forall t \in T\ [\ ((p,t) \in \Phi \Rightarrow \text{length}(\Lambda(p,t)) = m) \wedge$$

$$((t,p) \in \Phi \Rightarrow \text{length}(\Lambda(t,p)) = m)\ ]$$

This corollary serves as a simple check on a CTM-net.

Property 5.10 The domain of a query is part of the domain of the input:

$$\forall t \in T\ [\text{part\_of}(\text{domain}(\Theta(t)), \text{merge}(\ \{\ c \in \Sigma\ |\ \exists\ p \in I\ [\ (p,t) \in \Phi \wedge X(p) = c\ ]\}\ ))\ ]$$

A RIDL expression in a transition should operate on the conceptual schemas of the information places connected to that transition, i.e. the domain of the RIDL expression $\Theta(t)$ of the transition t is part of the union of the conceptual schemas c of the information places p connected to t. This is a simple formulation of the type checking of queries and is derived from the more important rule that queries are formulated in terms of the data model. Of course more than property 5.10 can be said about the relationship between the query and the data model, but that is beyond the scope of this work. We only formulate the following simple corollary.

Corollary 5.3 The entity types in a RIDL expression are a subset of the entity types of the information places:

$$\forall t \in T\ [\text{entity}(\text{domain}(\Theta(t))) \subseteq \text{entity}(\text{merge}(\ \{c \in \Sigma |\ \exists\ p \in I\ [(p,t) \in \Phi \wedge X(p) = c]\}))]$$

Take the RIDL expression Available in fig. 5.7 as an example. This query has the types Tape, Rentable Film, Rental and Day as entity types, which occur all in the conceptual schemas in fig. 5.6.

Property 5.11 All information places are needed:

$$\forall p \in I\ \exists t \in T\ [((p,t) \in \Phi \vee (t,p) \in \Phi) \wedge \text{domain}(\Theta(t)) \cap X(p) \neq \varnothing]$$

No information place is defined without having a query defined on its domain. For all information places there exists a transition, to which it is input or output, and the domain of the query of that transition and the conceptual schema of the information place overlap.

Property 5.12 Output was input or has been derived:

$$\forall t \in T\ \forall o \in \Pi\ [\ (t,o) \in \Phi \Rightarrow$$

$$\cup \Lambda((t,o)) \subseteq \text{vars}(\Theta(t)) \cup \{\ w \in Z \mid \exists\ p \in \Pi\ [\ (p,t) \in \Phi \wedge w \in \cup \Lambda((p,t))]\}\ ]$$

Every variable occurring in a tuple annotating an output arrow of a certain transition must enter that transition (i.e. occur in a tuple annotating an arrow input for that transition), or occur in the transition selector of that transition.

There are more properties a correct CTM-net must have. We will mention two of these additional properties. We will not try to formalise them due to their complexity and the absence of a formalisation of RIDL and NIAM.

The first property is that the conceptual schemas at task places which are output of a certain transition must be derivable from the conceptual schemas at the places input for that transition and the transition selector of that transition. This is discussed more deeply in [Ter Hofstede 89a].

The second property is that there must exist an evaluation sequence for the parts of every transition selector, such that every variable occurring in a tuple annotating an output arrow of the corresponding transition can receive a value. In order to be able to check this, it must be indicated for every formal parameter of a procedure whether it is output of, or input for that procedure. A procedure is only allowed to be called if all its input parameters have a value.

In [Genrich 87] more properties of PrT-nets, such as safety, liveness and S-invariants are defined. Analogous reasoning would be possible for the CTM-net after introducing the marking of a net and the enabling of transitions. This analysis of the dynamics of a net is left for future research.

As a final remark we state that a CTM-net should contain the *complete* specification of a task. However, this requirement is not verifiable, since the completeness of a specification depends on the completeness of the informant's specification.

5.3.2 The relation between the models of activities and tasks

In this section the relationship between the activity model and the task model will be described formally. We will discuss this relationship between an extended version of activity models, in which a data model (conceptual schema) is related to every data flow, and CTM-nets.

The global activity model is assumed to consist of a hierarchy of local activity models with a correct refinement structure. Recall from definition 5.4 and 5.5, that activities may be decomposed into other activities and finally into tasks. Tasks appear at the bottom level of this activity model hierarchy (sometimes called the elementary net). Suppose now that the local activity models are denoted in data flow diagrams and let these be formally defined as follows, neglecting the subactivities and substates, but augmented with conceptual schemas related to every state.

Definition 5.8 A data-flow diagram is a 6-tuple (S, A, D, F, G, H), where
1. S is a non-empty finite set of states,
2. A is a non-empty finite set of activities,
3. $D \subseteq S$ is a set of data stores; $E = S \backslash D$ (by definition) is a set of flows,
4. $F \subseteq S \times A \cup A \times S$ is a non-empty set of arrows,
5. G is a non-empty finite set of conceptual schemas,
6. $H \in G^S$ is a function from the set of states to the set of conceptual schemas.

In chapter 3, some of the rules are given that data flow diagrams must fulfil. An example of such a rule is that every state has a source, which could formally be expressed as:

$$\forall s \in S \, \exists a \in A \, [(a,s) \in F]$$

Fig. A.2 shows an example of a data-flow diagram, which depicts the hierarchical decomposition of the activity *Activities of the Captain Video rental store* of fig. A.1. In fig. A.2 *Film request* is an example of a flow, *Treatment of film request* is an example of an activity and *List of rentals* is an example of a data store.

Let now d be a data flow diagram at the bottom level of the hierarchy, where $d = (S_d, A_d, D_d, F_d, G_d, H_d)$, then A_d is the set of tasks of this diagram. Furthermore, let a be a task of this diagram d, $a \in A_d$. We define $W_d(a)$ as the set of states, which are input for or output of the task a:

$$W_d(a) = \{s \in S_d \mid ((a,s) \in F_d \vee (s,a) \in F_d)\}$$

Let

$$C_a = (\Pi_a, T_a, P_a, \Sigma_a, \Psi_a, Z_a, I_a, \Phi_a, \Lambda_a, \Theta_a, X_a, \Omega_a)$$

represent the CTM-net for task a. For the definition of a CTM-net see section 5.3.1.

The relationship between the data-flow diagram d and the CTM-net C_a is expressed as an *injective* function f_a from the set of input and output states of task a in the data-flow diagram $W_d(a)$ to the set of places of task a in the CTM-net Π_a:

$f_a: W_{\mathcal{A}}(a) \rightarrow \Pi_a$

Normally activities, flows, places and transitions are named. We adopt the convention that s and $f_a(s)$ should also have the same name.

For this function f_a the following properties must hold:

Property 5.13 *Bijective data store mapping*

$f_a |_{D_{\mathcal{A}}} \rightarrow I_a$ is bijective

This property states that the restriction of f_a to $D_{\mathcal{A}}$ is a bijective mapping on I_a, i.e. every information place of the CTM-net C_a is the unique image of a data store in the data-flow diagram $\mathcal{A}$ which is input or output of the task a. Furthermore, all information places in I_a are images of data stores in $D_{\mathcal{A}}$ via the function f_a.

Corollary 5.4 No extra information places:

$\forall p \in I_a \; \exists s \in W_{\mathcal{A}}(a) \cap D_{\mathcal{A}} [\, f_a(s) = p]$

We repeat here the surjectivity part of the above property to indicate that no extra information places, other than those mapped from the original data flow diagram, may be introduced in the CTM-net of the task a.

Property 5.14 *Consistent input property*

$\forall\, s \in W_{\mathcal{A}}(a)\, [(s,a) \in F_{\mathcal{A}} \Leftrightarrow \exists\, u \in T_a\, [\, (f_a(s),u) \in \Phi_a]$

If a state s is input for task a in the data-flow diagram $\mathcal{A}$, then there exists a transition u in the CTM-net C_a, which has the corresponding place $f_a(s)$ as input. Conversely, if a place that is the image of a state s, is input for a transition u of the CTM-net C_a, then s must be input for task a in the data-flow diagram $\mathcal{A}$.

Property 5.15 *Consistent input from data stores property*

$\forall\, s \in W_{\mathcal{A}}(a)\, [\, (s \in D_{\mathcal{A}} \wedge (s,a) \in F_{\mathcal{A}}) \Rightarrow$

$\exists u \in T_a[(f_a(s),u) \in \Phi_a \wedge f_a(s) \in I_a \wedge (u,f_a(s)) \in \Phi_a \wedge \Lambda_a((f_a(s),u)) = \Lambda_a((u,f_a(s)))]]$

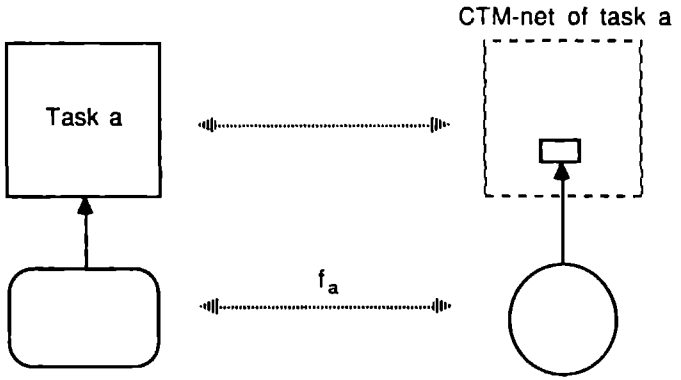
If a data store s is input for task a in the data-flow diagram $\mathcal{A}$, then there exists a transition u in the CTM-net C_a which is connected to the corresponding information place $f_a(s)$ by two arrows, one input arrow and one output arrow, with the same labeling. Note that the converse of the above implication follows from property 5.13 and 5.14.

Corollary 5.5 Information places are input for tasks:

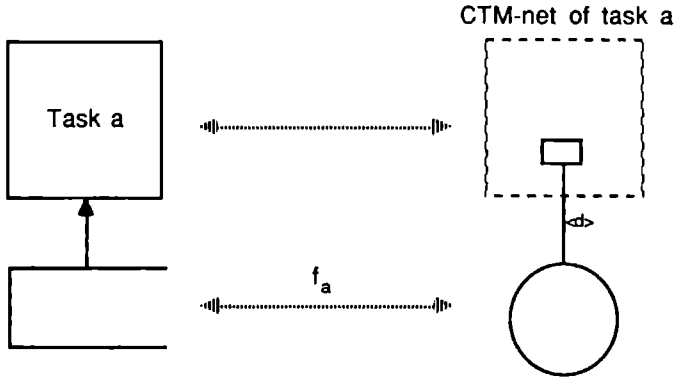
$$\forall s \in W_{\mathcal{A}}(a)\ \exists u \in T_a[((f_a(s),u) \in \Phi_a \wedge f_a(s) \in I_a) \Rightarrow (s \in D_{\mathcal{A}} \wedge (s,a) \in F_{\mathcal{A}})]$$

Note that if a place in a CTM-net is input as well as output of a certain transition and the arrow leaving that place to the transition has the same labeling as the arrow leaving the transition to that place, then this means that the contents of that place are only retrieved,and not changed by the transition.

In fig. 5.9 property 5.14 and 5.15 are depicted.



Property 5.14 Consistent input



Property 5.15 Consistent input from data stores

Figure 5.9 Consistent input properties

Output properties can be defined analogously to the above input properties.

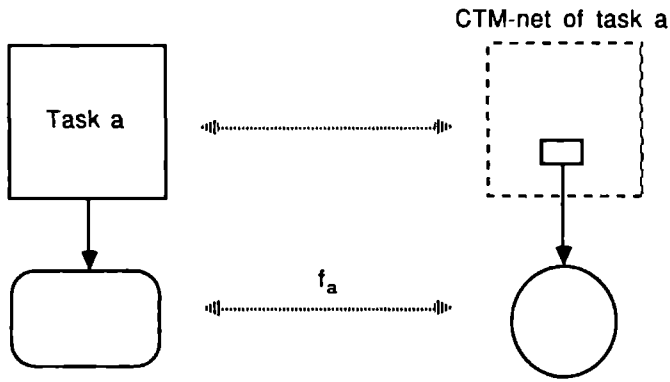
Property 5.16 *Consistent output property*

$$\forall s \in W_{\mathcal{A}}(a)\,[(a,s) \in F_{\mathcal{A}} \Leftrightarrow \exists u \in T_a\,[(u,f_a(s)) \in \Phi_a]$$

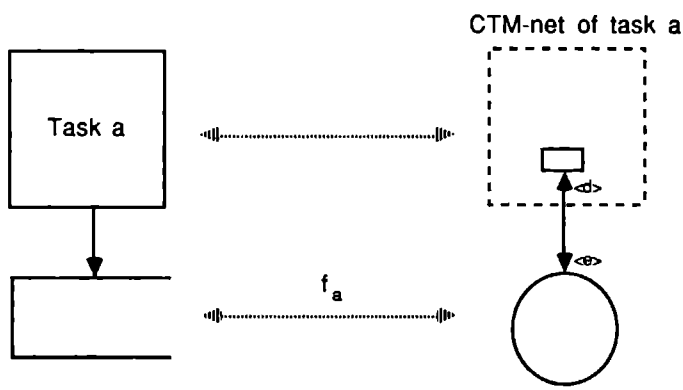
Property 5.17 *Consistent output from data stores property*

$$\forall s \in W_{\mathcal{A}}(a)\,[s \in D_{\mathcal{A}} \wedge (a,s) \in F_{\mathcal{A}} \Rightarrow$$

$$\exists u \in T_a[(f_a(s),u) \in \Phi_a \wedge f_a(s) \in I_a \wedge (u,f_a(s)) \in \Phi_a \wedge \Lambda_a((f_a(s),u)) \neq \Lambda_a((u,f_a(s)))]]$$



Property 5.16 Consistent output



Property 5.17 Consistent output from data stores

Figure 5.10 Consistent output properties

In fig. 5.10 the properties 5.16 and 5.17 are shown. The analogy of Corollary 5.5, that information places are output for tasks, is also valid.

Note also, that if a place in a CTM-net is input as well as output of a certain transition and the arrow leaving that place to the transition has a different labeling as the arrow leaving the transition to that place, then this means that the contents of that place can be updated by the transition.

We end with a property concerning the links of the states to the conceptual schemas.

Property 5.18 *Identical conceptual schemas property*

$$\forall s \in W_d(a) [H_d(s) = X(f_a(s))]$$

A state s in the data-flow diagram of task a must be associated to the same conceptual schema as its corresponding place $f_a(s)$ in the CTM-net.

The image set of the function f_a forms a subset of the set of places Π_a. When drawing a CTM-net, a dashed box is drawn around the set of places (with associated transitions that are not element of this subset. This box then represents the border between the actual description of the task and its input and output.

As an example, consider fig. 5.5. The places lying outside the dashed box correspond with flows and data stores input to, or output of, the task *Treatment of film request* in the data-flow diagram of fig. A.2. This correspondence is simple here since the flows and the data stores involved have the same names as their corresponding places in the CTM-net of fig. 5.5.
The flow *Film request* is input to the task *Treatment of film request* in the data-flow diagram. Its corresponding place in the CTM-net is an input place to transition *T1* (property 5.14). The data store *Information concerning films and tapes* is input to *Treatment of film request* and not output, therefore its corresponding information place (property 5.15) in the CTM-net is only connected to transitions in such a way that the input arrow has the same label as the output arrow (one must keep the notational abbreviation of fig. 5.4 in mind), implying that the contents of this information place cannot change by the execution of the task.
The data store *List of Rentals* is not only input for task *Treatment of film request* but also output. Its corresponding information place in the CTM-net is connected to transition *T1* by an input arrow and an output arrow with identical labels (property 5.15), and to transition *T4* by an input arrow and an output arrow with different labels (property 5.17).
Note that there are no information places added in the CTM-net of the task *Treatment of film request* and that the conceptual schemas of the information places of the task are specified in fig. 5.6, according to property 5.18.

We will finally present some theorems that are valid for the relation between the data flow diagrams and the CTM-nets. They are proved using

the properties of the CTM in the preceding section and the properties of the relation between the activity model and the task model.

Theorem 5.1 The images of any two states related to the task a in the data flow diagram are connected via a path in the CTM-net.

Proof: Suppose $s_1, s_2 \in W_d(a)$, then $f_a(s_1), f_a(s_2) \in \Pi_a$, by definition. According to property 5.4, there exists a path from $f_a(s_1)$ and $f_a(s_2)$. QED

Theorem 5.2 The domain of all queries of a task is specified in the conceptual schemas associated with the data stores of the task.

Proof: Let a be an arbitrary task and let T_a be the set of all transitions of a. Define $Q = \{q \in P_a \mid \exists t \in T_a\ [q = \Theta(t)]\}$. Q is then the set of all queries of the task a.
According to property 5.10 the domain of an arbitrary query $q \in Q$ is specified in conceptual schemas corresponding to information places p which are input for the transition t: $(p,t) \in \Phi_a$.
From property 5.13 we deduce that this p is the image of a data store s: $p = f_a(s)$, and according to property 5.18 the conceptual schema $H_d(s)$ of s is the same as the conceptual schema $X_a(p)$ of p. QED

Theorem 5.3 If all tasks of the global activity model are correctly specified in a CTM-net, then all data stores of the global activity model are used by queries in the tasks.

Proof: Recall that the global activity model is a tree structure of local activity models with a proper refinement structure. It is then sufficient to restrict ourselves to the local activity models at the bottom layer, i.e. the tasks. Therefore all data stores of the global activity model are connected to tasks according to definition 5.8.
All tasks are correctly specified in a CTM-net. From property 5.13 all data stores of an arbitrary task are one to one mapped to the information places of the CTM-net of that task. The information places are due to corollary 5.1 input and output for at least one transition, and, moreover, not specified without a query retrieving or updating it, because of property 5.11. So all data stores are used by the queries of the transition in the task. QED

Theorem 5.4 If the conceptual schemas of all data stores of the global activity model are specified and all tasks are correctly specified in a CTM-net, then the domains of all queries of the global activity model are specified.

Proof: As in the proof of the previous theorem, it is sufficient to restrict ourselves to the tasks at the bottom layer of the global activity model. All tasks are correctly specified in a CTM-net, then according to Theorem 5.2 the domain of all queries of this task is specified. Hence the domains of all queries of the global activity model are specified. QED

These theorems indicate some of the power of the CTM technique in conjunction with proper activity modelling and data modeling as discussed in the preceding chapters.

5.3.3 Computational power of the CTM

There are various approaches to capture the idea of computation. The class of the Turing computable functions is an example of such an approach. The principle that Turing machines are formal versions of algorithms and that no computational procedure will be considered an algorithm unless it can be presented as a Turing machine is known as Church's Thesis or the Church-Turing Thesis [Lewis 81]. If we can prove that in the CTM one can simulate any arbitrary Turing machine, we prove in fact that the CTM can compute any computable function. In [Ter Hofstede 89a] a CTM-net is presented that simulates an arbitrary Turing machine.

5.3.4 Correctness of conceptual schemas at task places

A conceptual schema of a task place describes that part of the Universe of Discourse of those individuals that can enter that particular task place. The conceptual schemas output of a certain transition must be derivable from the conceptual schemas at the input places for that transition and the RIDL expression belonging to that transition, as is more or less formulated in the properties 5.10 and 5.12.

It is possible to formulate more rules on the conceptual schemas of places being input or output to a transition. Based on the actual conceptual schema and the constraints therein, we can deduce that a particular combination of input and output schemas is invalid.

However, the reasoning on this subject is far from trivial, since in general the situation is much more complex. Places can be output of more than one transition, transitions can have several input place and schemas can change due to the RIDL expressions in the transitions and the schemas in the information places. For some examples and a more detailed discussion on correctness aspects of conceptual schemas at task places we refer to [Ter Hofstede 89a].

5.4 TASK MODELLING PROCEDURE

In the same style as in chapter 3 and 4, we will present a modelling procedure for CTM-nets. The procedure consists of 8 steps and has as input (according to the discussions in the preceding sections) a global activity model with the corresponding data model. The properties of section 5.3 are integrated in the steps to guarantee the correctness of the CTM-net.

The steps to construct a single CTM-net are:
1. Determine starting point
2. Determine preliminary transitions

3. Gather examples of data
4. Determine CTM-net structure
5. Model data at task places
6. Formulate RIDL-queries
7. Complete transitions
8. Check correctness properties

The description of the steps will be illustrated using a small example of a task in the video rental store of appendix A. A CTM-net will be constructed for the task that calculates the rental proceeds of a film.

Step 1. Determine starting point

The task, for which we have to construct a CTM-net, occurs in a local activity model at the bottom layer of the global activity model. Assuming that the local activity models are specified in some kind of data flow diagram, the task at hand has some states (i.e. data stores and data flows) as input and output.

1a. Specify input and output relations of data stores and flows to the task at hand.

The properties 5. 13 to 5.17 determine many aspects of this starting point. The data stores are to be mapped upon information places and the remaining input and output states become task places. All places can already be equipped with appropriate arrows for input and output.

1b. Collect the conceptual schemas that have been specified for all states appearing in step 1a.

The conceptual schemas relating to the states of the task have to be copied to the CTM-net specification according to property 5.18.

The part of the local activity model, in which the task 'Calculate rental proceeds' occurs together with all related states, is shown in fig. 5.11. The conceptual schemas of all states are given. Those of both data stores were already specified in fig. 5.6.

This model implies that we already have two task places and two information places. Both information places are only input to the task, so we may only formulate retrieval queries. We can already specify the boundary of the task as was indicated in section 5.3.2.

Step 2. Determine preliminary transitions

Based on the specified input and output, we must set up a preliminary structure of the data processing of the task. Dependent on the complexity and the domain knowledge required, informants have to be consulted to specify the data processing.

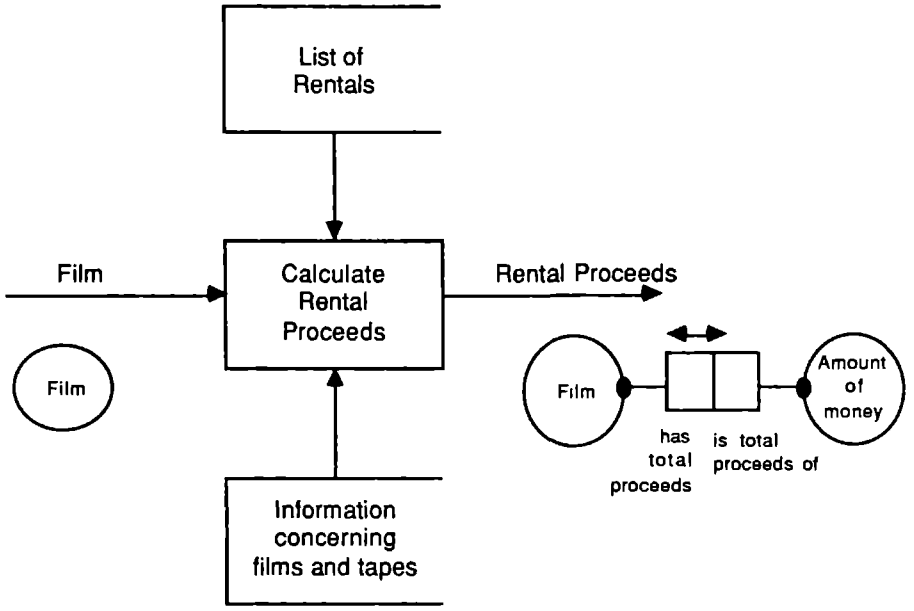


Fig. 5.11 Activity model containing task 'Calculate Rental Proceeds'

2a. Formulate the processing of the task in natural language

Be sure that this formulation is in terms of the activity model and the data model. Restrict the informants to the terminology set up in the preceding modelling activities. All descriptions of processing should be clear and unambiguous. It may be even worthwhile to set up a vocabulary of processing terms, that may be used, such as multiply, sum, set of, maximum, change, etc.

2b. Construct a preliminary CTM-net

The description obtained in step 2a is partitioned into elementary process steps. No strict guide-lines can be given for this partitioning, since the elementariness of a process is directly related to the possibilities of the data manipulation language, in this case RIDL. A single query should be the realisation of a elementary process. Also the calculation of intermediate results provides a partitioning structure.

Assume that in our example the following text was specified (in which, for simplicity the late charge is neglected):

```
The rental proceeds of a film is calculated by
multiplying the number of times the film was rented to a
person with the rental price of that film.
```

From this description we construct a preliminary CTM-net, that is shown in fig. 5.12. The intermediate task places are already drawn.

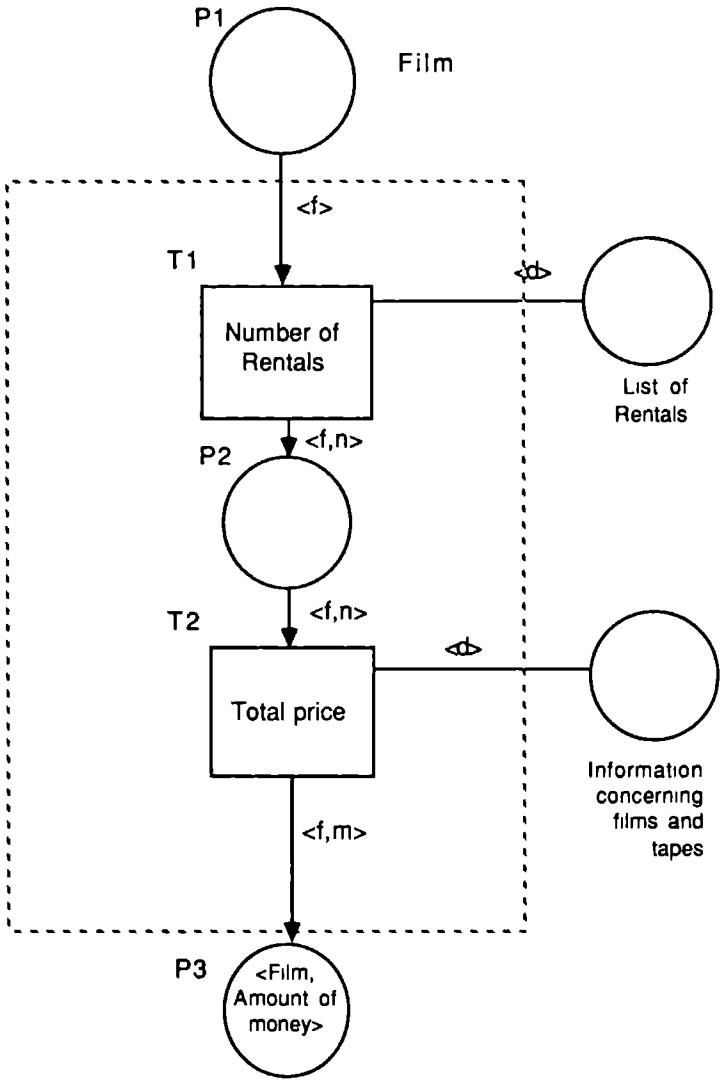


Figure 5.12 Preliminary CTM-net

Place P1

Film
Marathon Man
Goldfinger
The Godfather
Modern Times
Ghandi
The Great Dictator
Empire of the Sun

Place P2

Film	Number of Rentals
Marathon Man	6
Goldfinger	3
The Godfather	4
Modern Times	1
Ghandi	1
The Great Dictator	1
Empire of the Sun	

Place P3

Film	Rental proceeds
Marathon Man	$ 18,--
Goldfinger	$ 9,--
The Godfather	$ 16,--
Modern Times	$ 2,--
Ghandi	$ 5,--
The Great Dictator	$ 2,--
Empire of the Sun	$ 0,--

Figure 5.13 Examples of data

Step 3. Gather examples of data

We gather examples of data in order to construct models of it in the same way as in the modelling procedures for data in chapter 4.

3a. Extract examples

For every place in the preliminary CTM-net such examples have to be given. These may partially be available in samples that were input for

the preceding global data modelling and were determined by the sample criterion.

3b. Mark facts for possibilities of redundancy

Similar preparation steps are employed as for the global data modelling, except that redundancy is not deleted but marked. Redundancy at the level of tasks is important for the processing of the data. This processing must be completely specified.

The resulting examples of the preliminary CTM-net of fig. 5.12 are shown in fig. 5.13. Recall that the data of the information places is already modelled and the corresponding examples are shown in fig. A.3.

Step 4. Determine CTM-net structure

The processing will be provisionally formulated in order to establish the final structure of the CTM-net.

4a. Formulate data processing

For all transitions the way in which the data processing is performed is determined using the examples of the previous step. The transformation of the input to the output is formulated.

4b. Adapt preliminary CTM-net

Extra transitions and task places may be needed to cover processing that was not foreseen in step 2. In particular, the handling of erroneous or incomplete input of the user deserves attention. Most informant's specifications only consider processing under the assumption that everything goes well.

4c. Repeat steps 2 to 4 for extra transitions and places

For new transitions which resulted in the previous substep, the processing must be specified. This will probably be performed in terms of extra examples of data. This step results in the final CTM-net structure.

In our small example the informant forgot to specify that the rental proceed for new films was zero. This was seen from the fact that the input P1 of the preliminary CTM-net has as input a film, meaning possibly a film that is not rentable yet. The information place 'List of Rentals' only contains data of rentable films. This leads to an adaptation of the preliminar CTM-net, of which the extra transitions and places are shown in fig. 5.15.

Step 5. Model data at task places

The data of the individual entities that can enter a task place is modelled in a conceptual schema. This conceptual schema will be coupled to the task place.

5a. Construct conceptual schema

The construction of these conceptual schemas is performed in a similar way as the global data modelling. The intermediate data in a task is mostly not too complex, so this modelling is often simpler. Subtype hierarchies and complex constraints do not occur.

5b. Mark redundant roles

Due to the processing in the transitions certain facts are derived from others. The corresponding roles are marked.

The resulting conceptual schemas for the task places of our example are shown in fig. 5.15 next to the places P2, P3 and P4.

Step 6. Formulate RIDL-queries

This step consists of two substeps.

6a. Construct typings and parameter tuples.

According to the rules given in section 5.2 and 5.3 the tuples of the typing and of the parameters are determined. From the conceptual schemas of each task place a linearised typing is constructed. Corresponding parameter tuples with the same length as the typings are made (property 5.5 en 5.8). Information places receive, according to their function as input or output, a 1-tuple with a parameter name for the data store.

6b. Describe RIDL-queries

The RIDL-queries can be formulated in terms of the conceptual schemas of the information places using the parameters of the tuples of the input task places. The derived information is either assigned to the new state of a data store in case of an update query, or assigned to an output parameter in a tuple of an output task place in the case of a retrieval query.
Note that property 5.10 is directly guaranteed in this step. According to property 5.6 the types of the RIDL-queries and tuples should correspond. Furthermore, there should be no parameters in the RIDL-query, that are not mentioned in the input or output tuples (prop. 5.7).

The three RIDL-queries of the example are shown in fig. 5.14.

```
PREDICATE Is_new (DATABASE d; FILM f);
BEGIN
      f IS NOT IN Rentable-film
END;

PREDICATE Number (DATABASE d; RENTABLE FILM f, QUANTITY n);
BEGIN
      n = NUMBER-OF Rentals OF Tape contains Rentable-film f
END;

PREDICATE Price (DATABASE d; RENTABLE FILM f, QUANTITY n,
                  AMOUNT OF MONEY m);
BEGIN
      m = n * Amount-of-money is-rental-price-of Rentable-film f
END.
```

Figure 5.14 RIDL-queries of the rental proceeds task

Step 7. Complete transitions

So far, transitions have only be considered in relation with a single place. In this step all aspects of a transition have to be brought into a correct correspondence. Some transitions may be combined according to the convention in fig. 5.3 due to a disjoint condition. All types and names of parameters should be correct and unique. Information places are connected to a transition with pairs of arrows in reverse directions (prop. 5.1). Because of property 5.3 transition should also have input as well as output places.

Special attention should be paid to the further processing of data, which didn't fulfil the conditions for ordinary processing of the task, but implement the way of dealing with erroneous or incomplete user input. Finally, property 5.12 has to be checked, so that all output parameters are either the result of RIDL-queries or were already input parameters.

Step 8. Check correctness properties

Most of the checking of the properties of section 5.3 is integrated in the preceding steps, except for the properties that deal with the complete CTM-net.
First, property 5.2 does not allow superfluously specified places. All places should either be input or output or both, to tasks. Furthermore, the net may not be splittable. The case that this is inevitable, the data flow diagram of which the task is a part, should be revised. The task should be split into as many tasks as there are parts in the CTM-net, such that each task corresponds to one CTM-net. Finally, no information place should be specified without having a query defined on it according to property 5.11.

The final CTM-net of our example is shown in fig. 5.15.

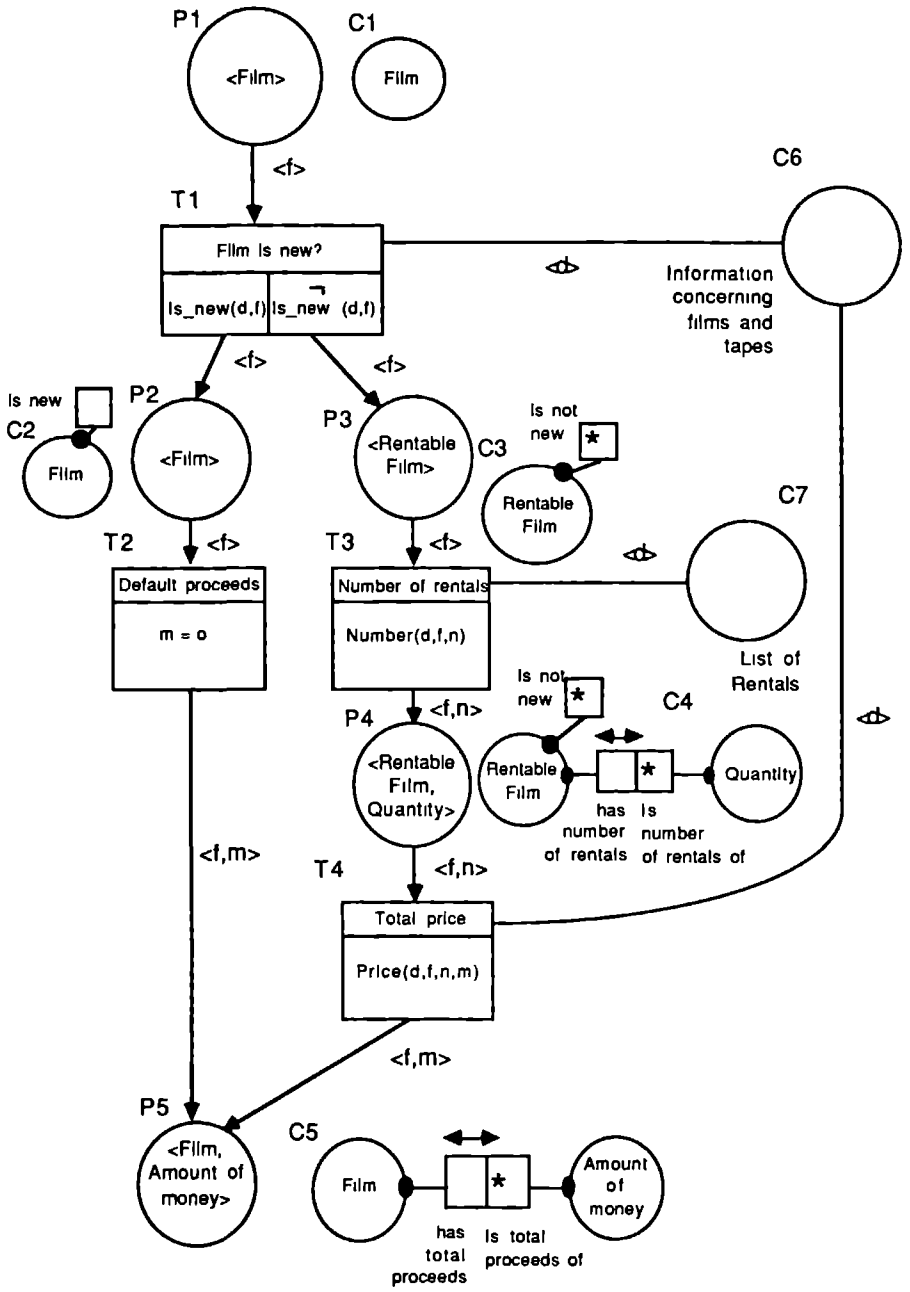


Figure 5.15 CTM-net for rental proceeds calculation

This concludes the modelling procedure for CTM-nets. Since the properties are integrated, it follows that for a net modelled in this way the theorems of the previous section hold. In this way statements about the quality of the modelling process and its products can be formulated with an explicit motiviation. We consider this the most valuable aspect of formalisation and modelling procedures.

It might occur, that the modelling of the CTM-net necessitates the alteration of the data flow diagram, to which the task at hand belongs. Extra data from data stores may be needed to specify the complete data processing, so extra input from information places must be specified. Even the conceptual schemas of the global data modelling might need changes, because it turned out during task specification that extra data has to be recorded. However, one should be very reluctant with regard to changes, since they are in conflict with earlier informant's specification. Reconsideration of this earlier specification process with the involved analists is required.

In [Reisig 87] properties of theoretical schedules to develop a PrT-net structure are discussed. This theory appears not to be applicable for the case of CTM-net as a special kind of PrT-net. The sample criterion implies that the decomposition of the processes is so detailed, that few transitions (about 10) are needed to describe the task. Reisig, on the other hand, considers PrT-nets with 100 to 1000 transitions, so that abstraction structures are necessary.

5.5 CONCLUSIONS AND FURTHER RESEARCH

5.5.1 Requirements assessment

This section assesses the requirements for task modelling techniques as formulated in 5.1.3, with respect to CTM. The six requirements on task modelling are fulfilled rather well by CTM. We assess them in the same sequence as in 5.1.3.

1. The transfer from activity and data modelling to task modelling is to be performed as discussed in 5.1.2. In the CTM explicit cross-references between the involved models are present via the components Entity type, Transition and Place. The CTM offers good opportunities to be transferred to interaction modelling and to event modelling.
2. The constructs of control flow, with the exception of process synchronisation aspects, can be expressed in the CTM. Simple process triggers can not be modelled in a CTM-net, but data triggers can.
3. The generation of code out of CTM-nets is not yet investigated thoroughly. The well known programming language constructs, like sequence, choice and iteration, can be generated straightforwardly, as is shown in [Brinkkemper 89e].

4. The PrT-net formalism underlying the CTM allows for the verification of theoretical statements. In section 5.3 this has been discussed extensively for the CTM.
5. The animation or population of a CTM-net is achieved by means of the ingredient modelling techniques. One the one hand the conceptual schemas in NIAM can be populated by instance facts and on the other hand the processing of tuples can be animated according to the so-called token game of the Petri-net formalism.
6. The completeness of data manipulation in RIDL ensures the complete data manipulation capabilities of CTM.

5.5.2 Use of the CTM

The CTM, as we propose it here, is an intermediate specification technique intended to be used for the modelling of the data-manipulation part of application programs. This role of go-between serves more purposes. For the main types of information systems development projects, new development, maintenance, enhancement and migration, there exist four options for development support with contemporary tools.

- Automatic system generation of the complete IS from the specifications of the analysis design stage.
- Incorporation of existing operational software or of standard packages.
- Reverse engineering of the current operational IS to regenerate their specifications.
- A bi-directional bridge, that combines of automatic system generation and reverse engineering.

Various IS specification techniques are applied in these approaches. In this section we will discuss the ways in which the Conceptual Task Model can be employed and what kind of tool support is needed.

System generation

We briefly discuss the possibilities for code generation for the CTM. As a target language for code generation we consider imperative languages like PASCAL, C, SQL, COBOL or ALGOL 68, which presupposes the existence of a RIDL compiler to that particular language. Some constructs of control flow in the CTM can be mapped straightforwardly onto constructs of the target language.

Sequential processing in the CTM is modelled by means of a directed chain of transitions and task places. This corresponds with the ordinary sequential statements in imperative programming languages. Task places between transitions do not have an equivalent construct in such a language.
Combined transitions (e.g. transition T1 in fig. 5.5) in the CTM represent conditional choices. The parts of a combined transition will contain disjoint and total conditions. In a programming language this corresponds with an IF-THEN-ELSE clause, in which all conditions and the corresponding actions from the CTM-net can be filled in.

Iteration appears as a cycle in the CTM-net, though not all cycles in a CTM-net correspond with iteration. The conditions which control the iteration, should not be handled as described above, but integrated in a pre-checked or post-checked loop.

Reverse Engineering

Reverse engineering in some (semi-)automatic way aims at the regeneration of the conceptual specifications from existing source code of a system. This is far from trivial. Semantic knowledge of the application is hidden in data structures, names of program variables, process structures and in all sorts of added commentary. Implementation choices and tricks complicate this even more.

The CTM allows the modelling of the data manipulation part of applications, in connection with a reverse engineering of the data definition part. Manual experiments indicate that reverse engineering with CTM is possible, but results in many diagrams. This is due to the fact that much world knowledge and domain knowledge is not represented in the code, but is made explicit in the CTM-net. If this knowledge was added to the code in some kind of language, the code would be of comparable complexity.

Simulation

As the CTM is based on Petri-nets, it is well suited for validation by means of simulation. A data tuple variant of the token game could be realised in a CTM-net tool. To accomplish this a compilation facility must be available for the simulation of the data modelling language NIAM and the data manipulation language RIDL. Though a RIDL compiler has not been implemented yet, there are no fundamental reasons why this cannot be done. The tool should ask for examples of the data at the information places and of the task places these provide the initial imput. With these examples an animation of the processing of the taks can be shown.

Workbench implementation

The CTM technique is not suited to the manual modelling of tasks in a realistic sized IS, due to the complexity of the resulting diagrams. Automated support of the technique in a tool, possibly combined with modelling techniques for activities, data and user interaction, is required. As discussed in section 5.3, the CTM is embedded in a sound framework. Properties and rules are formulated, on which all sorts of analysis of application models can be based.

In fig. 5.16 we show a proposal for a screen layout of a tool supporting the modelling of tasks using the CTM. The data models of the places and the RIDL-queries of the transition are shown in separate pop-up windows. When these windows are left out, a plain PrT-net remains.

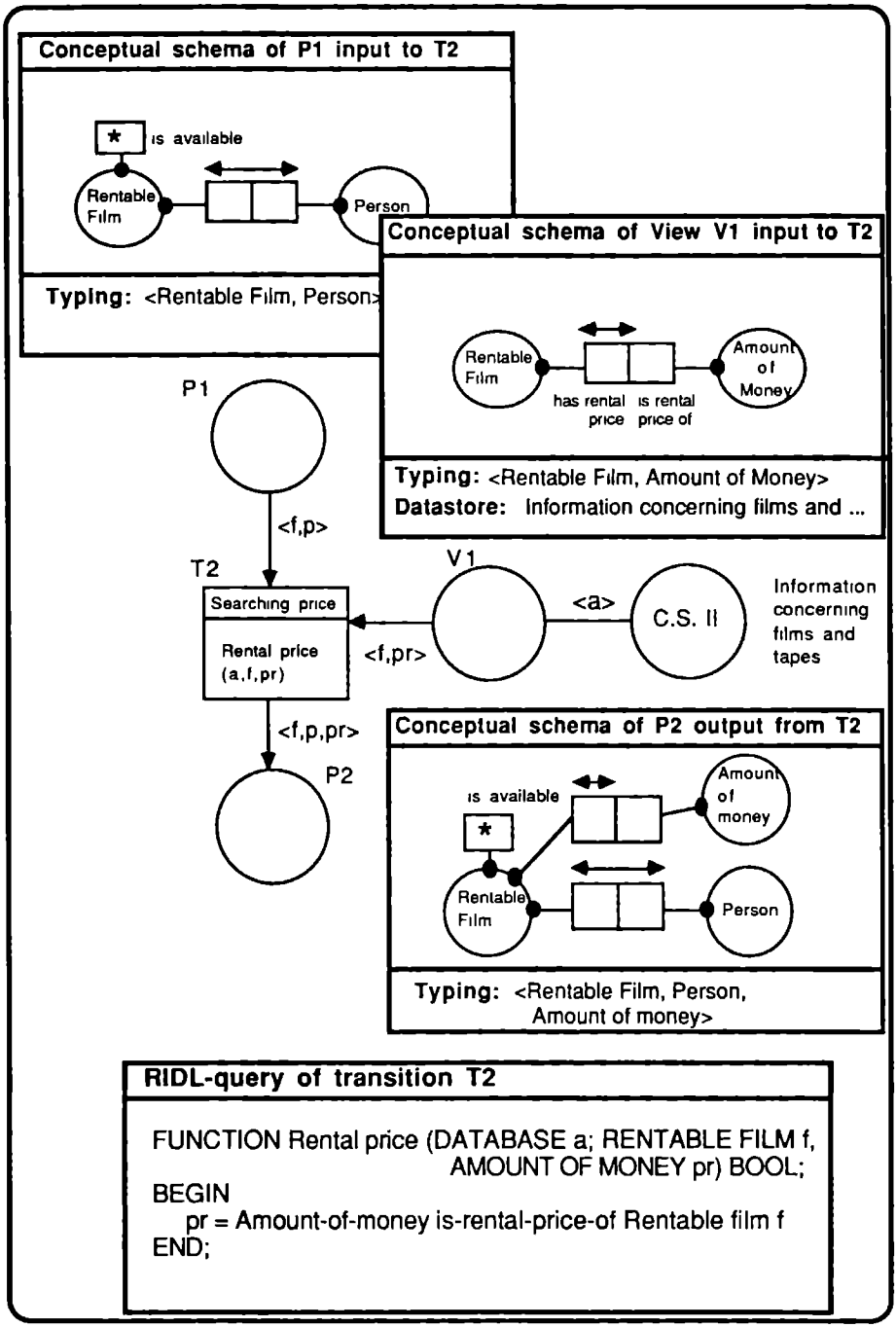


Figure 5.16 Screen layout of a CTM tool

The modelling of such a CTM may be performed as described in the modelling procedure in section 5.4 and therefore the tool may provide support for the preliminary task modelling, identification of individual transitions, modelling of data at the places, formulation of the RIDL-queries and the checks on the components. When the transitions in a task are known, they can be put in a preliminary schema, with some intermediate task places connecting them. These transitions can be modelled and analysed separately. Thereafter they can be integrated for global analysis of consistency, connectivity or for other purposes.

We propose in this section, however, a practical diagramming convention that deviates in two ways from the theoretical technique. This is necessary to improve the practical applicability. The discrepancies between the practical and the theoretical technique can be overcome by standard transformations, which can be derived from the descriptions below. The adaptations are the following.

First, we use data base views instead of database tuples. Since tasks need only a certain part of the data present in the data base, we define a view that models this part. This view is positioned on the arrow from the information place to the transition. An information place will be surrounded by such views. The conceptual schema of a view is a derivable part of the conceptual schema of the information place. Recall that the information places correspond with the data for which retrieval queries or update queries are formulated, whereas the data model at the task places stand for the parameters of the transition. Syntactical and semantical cross-checks of queries and parameters versus data models can be performed automatically.
All inputs and outputs of a transition are now specified by small conceptual schemas. An example of a view is shown in fig. 5.16 for the information place 'Information concerning films and tapes'. In the PrT formalism such views are not prohibited, but the strong relation of the data in the data store with that in the data view must be described completely. This is not practical, since database management systems offer very effective means to specify views.

Secondly, we propose that the tool implementation supports the decomposition of tasks. This decomposition obeys analogous rules to those for the decomposition of activities in data-flow diagrams. The conceptual schemas at the places may also be decomposed, but the decomposition must always satisfy the requirement that tasks process data elements (see the definitions in section 5.1).
In PrT-nets this is again not possible due to the unclear firing semantics of the decomposition, when data elements corresponding to more than one input or output place are optional.

In addition to this and to the discussions in the previous sections, we suggest three additional functionalities in a CTM-tool.

1. Support of modelling transparency. Because of the dependencies between the task models and the models of other types, such as activity models and global conceptual schemas, developers working with the

tool wish to be able to transfer directly from one type of model to the other via a dependency between the models. For example the transfer from a task model to the activity it belongs to. Some of the dependencies, along which transfer is desirable, are modelled in the CTM meta-model (fig. 5.8). See chapter 6 for a discussion of the modelling transparency functionality of workbenches and the various degrees of it.
2. Syntactic and semantic analysis of data models and queries. As already suggested above, the presence and the type of the data that are processed or created in a transition can be analysed and compared with the queries specified. Furthermore, the violation of the constraints can be pointed out.
3. Support of re-use. A support tool can compare the patterns of the data models or of the transitions with existing models and suggest to make use of them.

5.5.3 Further research on the CTM

In this chapter the CTM technique for task modelling in information systems development methods is introduced. This technique is intended to fill the gap between informal requirements engineering and program development. The CTM has some strong points. Fluent transfer between intermediate design results is supported due to the well-defined relation between activities and tasks and the integration with a data modelling technique. Tasks can be modelled on a conceptual level, thus enabling the analyst to abstract from particular machines and programming languages and their associated limitations. The CTM allows for code generation and the verification of all kinds of theoretical statements. Common constructs used in the processing of data elements as well as data manipulation and data retrieval can be expressed easily. Finally, in principle every computable function can be specified in the CTM.

We consider this technique as an innovative contribution which offers opportunities for new insights. Some aspects should be investigated more deeply, such as code generation, reverse engineering and a workbench implementation of the CTM. Feedback on the proposed technique from users of it in practice would be helpful to give direction to improvement or enhancement.
The theoretical framework can be extended by defining either the formal basis of NIAM and RIDL or additional auxiliary functions and properties. Extra theorems on the CTM-net structures may be formulated. The expressive power of CTM can be compared with other techniques, such as mentioned in section 5.1.4.
As discussed in section 5.3 it may be of interest to research further the dynamic properties of CTM-nets. As in similar approaches in Petri-net literature, it will be difficult to formulate general statements on the dynamics of a net, but it may turn out that about specific parts of a CTM-net more can be said. The recognition of frequent occuring parts is required for this. Statements on a complete net will then be based on the statements of its parts.

Finally, it has to be investigated whether the incorporation of views of information places and hierarchical decomposition in the theoretical model can be achieved. This requires an adaptation of the underlying PrT-net formalism or even a dedicated new high level net structure with a complete definition of its semantics.

6 MODELLING SUPPORT

The support of modelling with all kinds of tools is in the centre of interest today. Popular terms like CASE-tools, workbenches and IPSE are commonly used and misused in an area that completely lacks a scientific basis. The selection of tools and their proper introduction in system development are currently the most discussed topics. We refer to [Brand 89ab], [Dignum 86] and [McClure 86] for some aspects of these discussions. Overviews of the various tools and their capabilities are important for the proper selection of a tool. A detailed inventory of the tools available in the Netherlands was published in [NGGO 88]. An overview of contemporary research in the area of modelling support tools can be found in [Brinkkemper 90b].

The future development of modelling support tools will need a scientific approach. The age of the computerisation of diagramming techniques will end, users will require more complex support, and even that system development activities are taken over by tools. Based on the discussions in the preceding chapters, some aspects of modelling support will be introduced in this chapter. It is obvious that the prescription of modelling in the modelling procedures is to be build into tools. However, system development requires additional functionalities, for instance those that deal with the management of the models. We will discuss some of them in the following sections, based on earlier publications. We start with the definition of some terminology and functionalities of modelling support tools [Brand 89ab], [Brinkkemper 89b]. The incorporation of modelling procedures in system development methods will also be addressed [Brinkkemper 88ab]. Related to this is the degree of support that a tool may provide for a given method. The mapping of the techniques of a tool to the techniques in a method is called method companionship and will be discussed as well [Brinkkemper 89b]. In the analysis and design stage of a system development project, the models and their mutual relationships will become more and more complex. We present two ways of dealing with this problem: modelling transparency [Brinkkemper 89c] and layered modelling [Koesen 89].

6.1 TERMINOLOGY AND FUNCTIONALITY

6.1.1 Workbench, CASE-tool or IPSE

The terms method, technique and tool were already introduced in section 1.3. In methods the application of techniques is commonly prescribed. Automated tools supporting systems development exist in various types

and sizes. In the following definition we will distinguish some types of tools.

Definition 6.1

An *Integrated Project Support Environment* (IPSE, [Alvey 82], [NIIG 88]) is defined as a complete, integrated set of tools supporting all techniques that should be performed during system development.
Any tool which can be used for system development may be called a *CASE-tool* (Computer Aided Software Engineering).
If the use of a tool is not restricted to a single phase or activity, but offers possibilities for integrated modelling during different phases through a consistent interchange of development data, it is called an *I-CASE-tool* (Integrated CASE).
A *workbench* is a tool supporting one or more techniques in the stages of the development that precede programming, or a tool for project management. Depending on the particular stage a workbench supports, it may, for example, be called an analysis workbench or a design workbench.

[Case 86] distinguishes automated tools which support project management (life cycles, project control, PERT/CPM), programming (on-line editors, code generators, code optimisers) and the development process (data dictionary, encyclopedias, graphic and textual systems). The first and last mentioned tools are called workbenches. We refer to [McClure 86] and [Vonk 88] for a discussion of the components of a workbench. A list of tools with a description of their properties is given in [NGGO 88].

The support of all development activities by an IPSE is a goal that is still not realised in commercially available tools. The best development support tools can therefore be qualified as I-CASE-tools. Most CASE-tools, available today, have been constructed to be used within a particular method. Examples of these CASE-tools are the Information Engineering Workbench (IEW; [IEW 88]) and the System Development Workbench (SDW; [SDW 88]). The fact that a tool is designed for use within a particular method, does not preclude its use within other methods as well. Since the availability of tools, that fulfil certain quality requirements, is still rather limited, tools are frequently used within methods for which they were not designed. However, it is then not always clear when and how the tool should be used. To resolve this, guide-lines are developed to describe this relation. The following definitions are given to base the discussions on this problem in section 6.4. Since we restrict ourselves mainly to the analysis and design stages in which modelling is the main activity, we will henceforth use the term workbench instead of tool.

Definition 6.2

Method companionship of an information system development method and a workbench is the mapping of the required techniques and corresponding development products of the method to the offered techniques and corresponding reporting facilities of the workbench.
The workbench is then called the *method companion* of the method.

If a workbench is designed to be used within one particular method, we call this *initial method companionship.*
If a workbench is used within a method for which it was not designed and guide-lines have to be produced in order to make this relation more explicit, we call this *derived method companionship.*

The term method companionship originates from McClure [McClure 87], who introduced the following categories for CASE-tools: toolkits, workbenches and methodology companion. She defines a methodology companion as a tool that provides computerised assistance for a particular software development methodology. Examples of initial method companionship are the Information Engineering Workstation for use within Information Engineering, and the System Development Workbench for the use within the System Development Methodology. In section 6.4 we will discuss the determination of the derived method companionship of the Information Engineering Workstation for use within the System Development Methodology

Definition 6.3
The database of a tool or workbench, i.e. that part of a tool where all the data is stored, that is gathered by carrying out the technique in a project, is called an *information resource dictionary* or *dictionary* for short.

In some commercially available workbenches the dictionary is called the encyclopedia, because it contains more information about the objects than is recorded in data dictionaries of database management systems. The term project database is also used.

6.1.2 Workbench functionalities

Workbenches are becoming more advanced. In a workbench, various functional aspects can be distinguished, that are subject to rapid development. In [Vonk 88] the most important components of a workbench are described. We discuss only the following aspects, because of their importance in relation to the support of modelling.

Object significance and expansibility of the dictionary

The objects, that are part of a model and stored in the dictionary, can have either *syntactic significance* or *semantic significance.* Syntactic significance means that only the syntactic aspects of an object, such as its shape and position, are considered in the tool. For instance, a rectangle in a model only has significance as a rectangle. Semantic significance means that the object is treated in the context of the modelling technique with all the rules that apply to it. For instance, an entity type may occur in relationships and have attributes added to it.

A workbench may also allow users to extend the dictionary structure, which means that it is possible to add extra object types in the dictionary.

In this case the workbench is said to have a dynamic meta-data model. Workbenches that do not have this functionality are said to have a static meta-data model. The extension of the dictionary goes hand in hand with the addition of functions to the techniques supported by the workbench. A workbench that allows this so-called customisation is Excelerator [ITC 87]

Check facilities

A check facility or an analysis, for short, is an assessment of the status of (selected) objects, characteristics and associations by means of verification against rules. These rules are formulated in a technique and control its correct application. The following six types of analyses can be distinguished:

1. **Pre-analysis or post analysis**
- Pre-analyses are analyses carried out before data is entered into the dictionary. The data is then accepted or rejected and the dictionary is therefore always consistent with respect to the analysis criterion. For example an object may only be stored if its name is unique in the dictionary.
- Post-analyses are analyses carried out on the data in the dictionary or a part thereof. Temporary inconsistencies with respect to the assessed rule can occur. An example is data conservation analysis, which checks that all data that is output of a process is also input of that process.

2. **Syntactic analysis or semantic analysis**
- Syntactic analyses check only whether the objects are correctly entered in a diagram in syntactic terms, e.g. the end of an arrow must always be linked to another object, irrespective of the objects.
- Semantic analyses check whether the objects are semantically correctly entered in a diagram, e.g. two data files cannot have a direct link in a data flow diagram, or a data flow must begin and end in a valid source or destination, namely an external agent, a process or a data file

3. **Object analysis**
- The analysis is carried out on one selected object, e.g. the check whether an entity type is uniquely identified.
- The analysis is carried out on several selected objects or on all objects of a specific type. For example the check on a set of modules that the call sequence is non-recursive.

4. **Object type analysis**
- An analysis of an object type is carried out on a set of objects of a specific type. An example is the exception analysis, that checks for instance that all attributes of all processes are stored in the dictionary.
- The analysis of several object types covers objects of several object types. For instance an analysis that precedes the integration of data models and signalling whether homonyms or synonyms occur.

5. **Diagram analysis**
- The analysis takes place within one diagram. The check that no entity type occurs in a data model without participating in a relationship, is an example of this type of analysis.
- The analysis takes place in several diagrams of the same type. For example the analysis that the data flow diagrams obey the refinement rules.

6. **Diagram type analysis**
- The analysis assesses a rule formulated on one diagram type, e.g. the affinity analysis of an association diagram.
- The analysis assesses a rule formulated on several diagram types. For instance the association matrix of processes and entity types may be checked against the occurrence of the entity types in the data model of the processes.

These analysis dichotomies may occur in all kinds of combinations in the analyses built into a tool. For instance the check of the refinement structure of the data flow diagram hierarchy is a semantic post-analysis on several objects of several object types in several diagrams of one type. Note however, that of the four latter types of analysis only seven plausible combinations can be made.

Some analyses require that other analyses have already been carried out. Analyses are formulated in terms of rules on the models and those rules have all kinds of dependencies among each other. For instance, it makes no sense to perform a refinement analysis on the data flow diagram hierarchy without having performed all the correctness and completeness checks on the individual diagrams. The sequence of analyses and their mutual dependencies have to be determined carefully using the dependencies of the underlying rules. It may, however, be the case that some analyses are part of other analyses. The tool manual should specify whether the user should carry out the required analyses or whether these are part of a larger analysis.

Tasks

Finally, certain of the developer's tasks can be taken over by the workbench. We use the term task here for the introduction of changes to the dictionary on the basis of report and analysis interpretation. We distinguish three types:

1. Deterministic tasks: these are tasks where the outcome is unambiguously determined given the input for the task, so that no interaction is needed with the developer. The result of the task is obtained by transforming the products developed in other tasks. An example is the integration of two or more entity-relationship diagrams.
2. Interactive tasks: these are tasks where the workbench proposes a possible outcome and the developer can accept or modify it. The

developer constructs the desired product in an interaction with the workbench. The conversion of an entity-relationship diagram into a relational data base structure is an example.

3. Preparation tasks: this is a special kind of deterministic task that prepares a specific task using some information of another task. For instance in the activity modelling of a process that occurs in the decomposition of another process in a data flow diagram hierarchy, the data flows to and from the process in the parent diagram should also occur in the activity model of the process itself. A tool can assist in this by putting the flows in the diagram. The tool can automatically control the consistency of the hierarchy. Another example of a preparation task occurs in the task modelling procedure of the CTM in section 5.4. The first step of this procedure creates the starting point by extracting data flows and data models from the available activity model and data model.

Model status

In programming environments a lot of data is recorded about the status of a program: the author, the creation and last modification dates etc. Similarly, we propose to incorporate such functionalities in modelling support tools. The read, write and delete access of the developers for the individual models may be specified, as well as the status of the model with respect to the various analyses that can be applied to it: finished, checked, validated, etc.

Beside this, the models are strongly related to each other. Developers want to find out what models are already present and what still has to be done. Tools should therefore support some kind of overview of the models and their interrelationships combined with their status. For example, the hierarchy of data flow models can be shown in a decomposition diagram of the processes together with an indication of whether the corresponding data flow diagram and its data models are finished or not.

The functional aspects mentioned in this section are not found in all tools. The first tools on the market considered only syntax and not semantics, and they could only produce reports and not perform analyses. Most tools are now equipped with a variety of analyses and support all kind of tasks. In the future all kinds of enhancement of the functionality of tools will be introduced. The field of CASE-tools is young and its scientific research is still in its infancy. The price/performance trade-off will in the course of time determine the core set of functionalities for the support of modelling by tools.

6.2 THE INCORPORATION OF MODELLING PROCEDURES IN SYSTEM DEVELOPMENT METHODS

Information system development methods do not provide very detailed descriptions of the way in which the prescribed activities have to be

performed. This is quite obvious for methods that emphasize project management, but even system development methods mostly only prescribe the diagrammatic modelling tools that have to be used. The incorporation of the modelling procedures, as presented in the preceding chapters, turns out to be relatively straightforward for methods prescribing the development activities in a rather elaborated way. In case the tools and the development products that are input and output of the activities are known, the modelling procedures can even be integrated and adapted to fit the method precisely.

We will illustrate this for the case of the modelling procedures for the events (section 3.2), activities (3.4), data (4.3) and tasks (5.4) that will be incorporated in the Information Engineering Method (IEM) [Arthur Young 87]. Since these modelling procedures are for the analysis stage, we will consider only the Business Area Analysis (BAA) phase of the IEM. Some of the models that have been created using the Analysis Workstation of the Information Engineering Workbench [IEW 88] will be shown. The activities of the Information Engineering Methodology are applied to the Inventory Control and Purchasing System case given in [Olle 88a].Part of the analysis of the given test case had to be repeated in order to produce the following illustrations. It was therefore necessary to make some assumptions about the Universe of Discourse. This section is an adaptation of section 6 of [Brinkkemper 88ab].

The Business Area Analysis consists of 7 main activities:
1. Organise & Control Business Area Analysis Project
2. Define Business Area and Business Area Partitions
3. Model Existing Business Area Partition
4. Model Future Business Area Partition
5. Document Technical Requirements
6. Determine Business Area Implementation Approach
7. Perform Project Approval & Assessment Tasks

Here, we will present only a detailed activity plan of activity 3. The other activities are all beyond the scope of this section. The first and last activities are devoted to project management. The second is a short Information Strategy Planning study to be performed if the analysis has not been preceded by an extensive organisation wide information systems planning. The fourth is devoted to modelling the desired changes within the current Business Area Model and takes the models resulting from the third activity as a starting point. Finally, the fifth and sixth activities are concerned with hardware and software requirements and constraints such as volume, performance and security, and with the preparation of the subsequent System Design phase.

Activity 3: Model existing business area partition.

Activity 3 consists of 8 subactivities described in general terms in the IEM. We discuss an elaborated version developed mainly in practice and with the modelling procedures incorporated. From a collection of events a so-called Event Model is constructed, which is in turn the starting point for the

Business Area Partition Model. The latter model is elaborated and subsequently used for all kinds of related analysis of the business area, for example, of the existing software and of the available technical infrastructure.

3.1 Identify business events

The purpose of this subactivity is to analyse events within the Business Area. Business areas, coherent parts of the Universe of Discourse, have already been identified in the preceding activity. The behaviour of the information system in this area with respect to the environment is determined.

Subtasks:
- Identify events and external agents, using external event analysis (3.2) and internal temporal event analysis (3.2).
- Analyse the reaction of the events in a decomposition diagram, using activity analysis (3.4).
- Develop an Event Model by generating data flow diagrams (3.4) and entity relationship diagrams (4.5) of the decomposition.

The events from the inventory control case are:
External events:
- Receive Supplier catalogue
- Instructions for Purchase order generation
- Stock replenished
- Stock withdrawal
- Return of Stock items
- Deviation of Stock item withdrawals

Temporal events:
- Time to investigate suppliers
- Time to review Stock item types
- Suppliers delivery overdue

The reactions of the external events are shown in the decomposition diagram in figure 6.1. These activities express data processing in terms of concrete data compounds (see section 5.1) available in the organisation. Samples of these compounds are obtained that are input for data modelling. All resulting data models of the reactions can then be integrated to deliver the so-called Entity Model, shown in figure 6.2. The IEW provides the functionality to construct an integrated entity relationship model from the available entity relationship models in the dictionary.

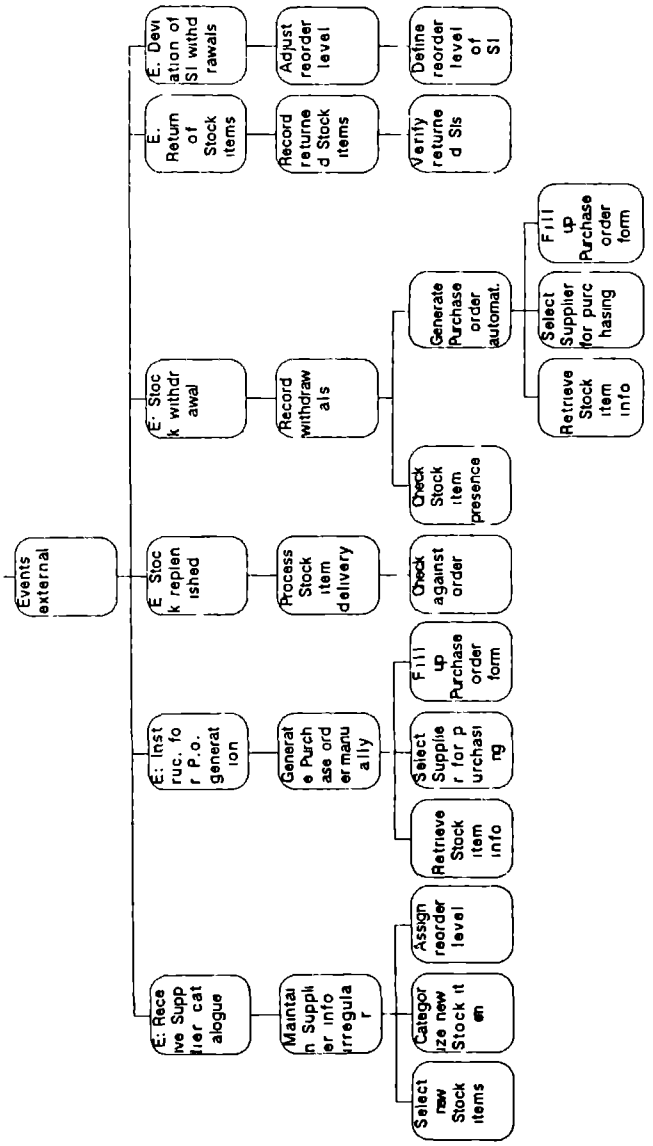


Figure 6.1 Process decomposition of external events

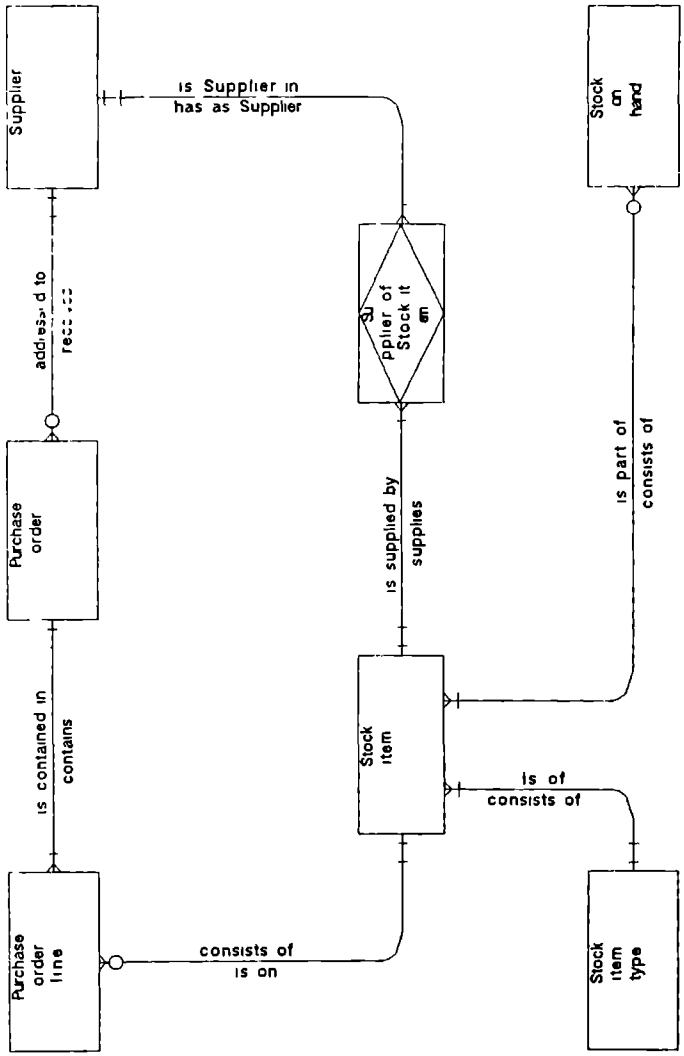


Figure 6.2 Integrated data model

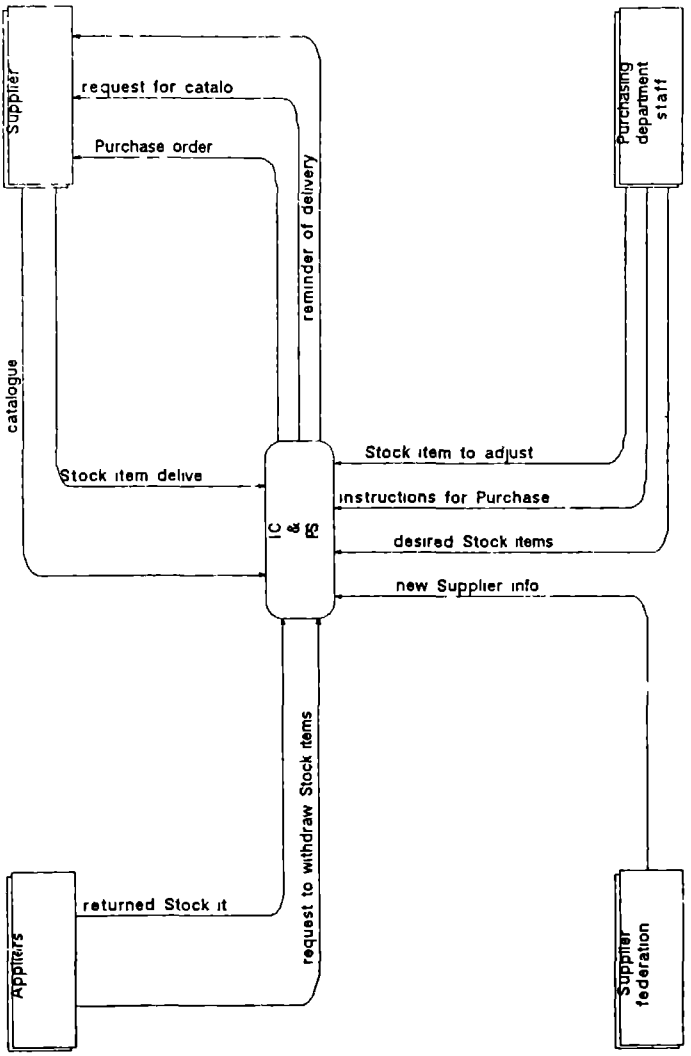


Figure 6.3 Context diagram

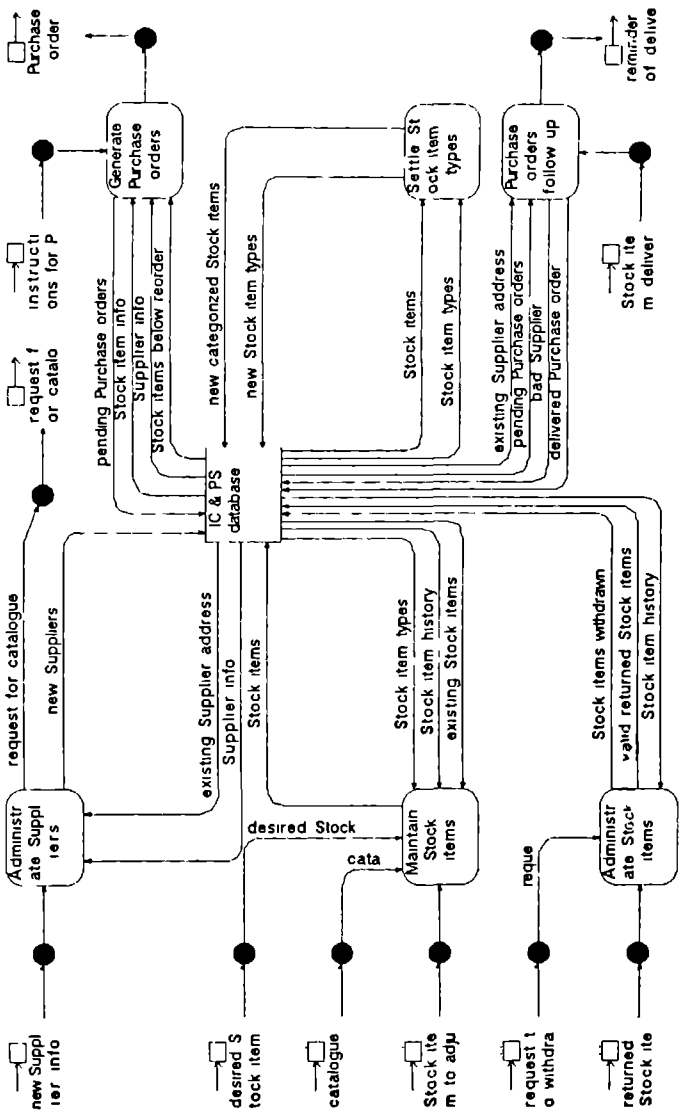


Figure 6 4Data flow diagram of context level

3.2 Determine the scope of the business area partition

This subactivity is performed to determine the externals and all flows of data within the scope of the business area partition. The constituent models of the Event Model of the previous task are input.

Subtasks:
- Construct a context diagram to represent the domain of the business area partition by including all data flows to and from external agents including the external agents of the Event Model.
- Add the Entity Model from the Event Model to the context diagram.
- Define the input and output flows from the business area partition in terms of attributes and flow expressions.

The developed context diagram is shown in figure 6.3. Observe that the modelling procedure for event analysis is split over subactivities 3.1 and 3.2.

3.3 Develop the current logical business area partition model

The goal of this subactivity is to describe the data and the data transformations as they are performed in the current business area partition. All data flow diagrams are created using the activity modelling procedure (3.4).

Subtasks:
- Take the lowest level activities from the Event Model and group similar activities together (so far as possible).
- Develop a decomposition diagram of the activity model of the partition by grouping the lowest level activities. Compose the activity model principally bottom up with a maximum of seven objects for each level.
- Add data flow diagrams for the activity model of the partition.
- Validate the business area partition model so far developed with informants.

The result of regrouping within the decomposition of activities can give rise to reorganisation in the business area. Event analysis will in any case highlight some of the flaws in the organisation. It is a good convention for the activity model that only the data flow diagram one level below the context level has a data store, provided that there is no distribution of the data base over various locations. This data store represents the central database (see figure 6.4).

3.4 Complete the current logical business area partition model

The purpose of this subactivity is to achieve a complete activity model and data model of the current data processing activities in the partition. All activity models were created using activity modelling (3.4) and all data models were created using entity-relationship modelling (4.3).

Subtasks:
- Detail the activity model to the level of tasks and develop a data flow diagram for the activity above these tasks.
- Develop a data model for each data flow.
- Develop a data model for each task using the data models of the input and output data flows.
- Specify the activity for each task by constructing action diagrams.
- Develop a data model for each activity in the activity model. This is a bottom up process using the entity relationship diagrams of the tasks and the data flows created in the previous subtasks (as mentioned before, a dedicated functionality of the Information Engineering Workbench supports this process). When you reach the context level compare the developed data model with the Entity Model, that is part of the Event Model (subactivity 3.2). Make adjustments if necessary.
- Perform a conservation analysis on the completed data model. This analysis checks whether the output data is consistent with the input data and is completely supported by the Information Engineering Workbench.

As discussed in section 5.1.4, the Action Diagram technique of [Martin 85] can be used in the same way as the Conceptual Task Model. In the Information Engineering Method the concept of task is called a functional primitive. It is defined as an activity known to the informant. This is a very vague definition and so we used the sample criterion to determine the abstraction level of the tasks. The modelling procedure of tasks for the case of action diagrams is similar to that of the CTM.

This subactivity completes the modelling of the Business Area Partition. For each object in the data model such details were recorded as its definition, the attributes with corresponding cardinalities, the relationships in which it participates, and development data. More data flow diagrams, entity relationship diagrams, decomposition diagrams, action diagrams, definitions, comments and other reports created for the inventory control case can be found in [Brinkkemper 88ab].

The remaining subactivities

The remaining subactivities are mostly concerned with correctness checks and preparations for System Design. We shall describe them briefly:

3.5 Analyse data files and systems
Associate the Business Area Model with the current automation environment to gain a better understanding of the strong and weak points of the current implementation.

3.6 Reevaluate alternative prepackage solution approaches
The reasons for the use of prepackage solutions may now become more evident because of knowledge gathered from the business area partition model.

3.7 Reconcile the current business area partition model with enterprise strategy

Determine the extent of changes needed for the current business area partition to satisfy the enterprise goals.

3.8 Reconcile the business area partition model with the business area information model
Ensure that the business area partition model follows the guide-lines of the baseline business area model. Compare the business area partition models with each other and update the baseline business area model.

Note that some models occur in the description of the activity 3 of the BAA that are specific for IEM, for example, the Event Model and the Business Area Partition Model. These models could better be called **model structures** since they are to be considered as a structure of models. For instance the Business Area Partition Model consists of a hierarchy of activity models with all related data models and task models. In IEM these model structures form the core of the modelling cycle, to which almost all development activities are related. In the above description the subactivities are formulated in terms of the model structures and their constituent models.

In the future, tools like the IEW may have functionalities to support the model structures implemented. For instance the addition of a model to a model structure, showing the status of the model structure (see 6.1.2), or the transfer of data from one model structure to the other. Brainstorming about the future of tools one could think of the following scenario:

1. Model structures are implemented in the tool according to the directions in the method. Deliverables of the method are defined in terms of model structures
2. Functions will be implemented to check the completeness of a model structure and to print a completed one.
3. The tool notices that a model structure is completed, or that a certain percentage of the models is still missing.
4. The tool lists which models are needed to complete a model structure, or even a listing of the activities of the method the developer has to perform in order to finish a model structure.
5. The tool asks for the missing data and completes the model structure. The developer combines and transforms them into a running information system.
6. The tool asks for all data and develops the information system.

Gradually more methodical knowledge is built into the tool. The method companionship increases until in the end the method is completely implemented in the tool, so one can say that the tool is the method. A lot of research and development on methods and tools is required to reach this ultimate goal.

6.3 MODELLING TRANSPARENCY

The many models resulting from a methodical system development process are intensely interrelated. The process model relates to views of the data model, the user interaction to certain processes of the process model, structure charts to particular data base tables, and so forth. CASE-tools supporting development activities should enable the management of the relationships among the models and the change-over from one modelling technique to another in a given context. Because of the resulting transparency of the models and the modelling techniques in relation to each other, we call this functionality of a workbench **modelling transparency**.

We use the term transparency to mean the absence of intermediate systems to operationalise the connections between two systems. The extent to which intermediate systems are present in tools is called the degree of the transparency. Apparent absence of intermediate systems makes the one model transparent to the other and is defined as a high degree of modelling transparency. The presence of many intermediate operations complicating the connections between models and is called a low degree of transparency.

6.3.1 Model dependencies

The dependencies among the models are due to relationships between the modelling objects, such as entity types, processes, modules, etc. These dependencies can be modelled using ordinary data-modelling techniques, resulting in a *meta-data model* (see sec. 2.2).

Furthermore, since the modelling activities are either in parallel or sequential, we make a distinction between *sequential dependencies* and *parallel dependencies*. These dependencies are defined in terms of models and objects. By a model, we mean an abstraction of a certain system described in a particular form of data representation, for instance, as prescribed by the diagrammatic techniques (entity-relationship diagram, data flow diagram, structure charts, etc.). Models can also be represented in tables, structured (pseudo) programming languages, etc.

- *Sequential dependency.*
  A model is said to be sequentially dependent on another already defined model, if objects within it need to be brought into agreement with data concerning these objects in the other model. This form of dependency is introduced because various aspects relating to the same objects from the Universe of Discourse are described in various models in successive activities of the method. Sequential dependency mostly relates to models of a different type.
  Assume, for instance, that the association matrix Entity types versus Processes has been definitely established, and that it indicates that the entity types "Supplier", "Order" and "Article" are needed for the "Ordering" process, then the entity-relationship model of this process must inevitably contain these entity types.

It is clear that, in principle, departure from these sequential dependencies is not allowed. Should a departure be necessary, however, the already established model should first be amended.
A special form of sequential dependency is related to so-called generative activities. These are activities with which another model is derived, and possibly enlarged, from an already available model. An example is the generation of a relational database structure from an entity-relationship model. It is needless to add that essential features in the database, such as for instance entity type, keys, optionality, etc., are never modified without first altering the relevant aspects in the corresponding entity-relationship model.

- *Parallel dependency.*
  Two models are dependent in parallel if they model aspects of a particular common collection of objects, are developed in parallel with each other and are more or less simultaneously established. This form of dependency can occur with models of the same type as well as with models of a different type. For example, a number of entity types can occur in the entity-relationship models of two different processes. The definitions of these entity types, including attributes and cardinalities, should be in agreement with each other or be brought into agreement. Since neither of the two models can prescribe anything for the other, any modifications should be well co-ordinated.

Because various models are created and maintained by means of workbenches, which in turn generate the progress documents, workbenches must be able to monitor these dependencies. Workbenches currently available support the placing and maintaining of dependencies to a very divergent degree. It would be desirable if the workbench were to provide direct support for the change-over from one modelling technique to another in a given context during the development process. Changes in a model should be processed in the dictionary, and could in that way be made immediately visible in all related models.
The *meta-activity model*, a process model of the modelling activities, should determine the type of dependency.

6.3.2 Degrees of modelling transparency

CASE-tools currently available support the placing and maintenance of dependencies to a very divergent degree. We distinguish four degrees of modelling transparency, defined on the basis of the functionality of a tool (see [Brand 89ab]).

0. Stand-alone workbenches with a non-accessible dictionary do not support modelling transparency. Developers are forced to recognise and monitor the dependencies between models themselves. We call this modelling transparency degree 0 (see figure 6.5-0).

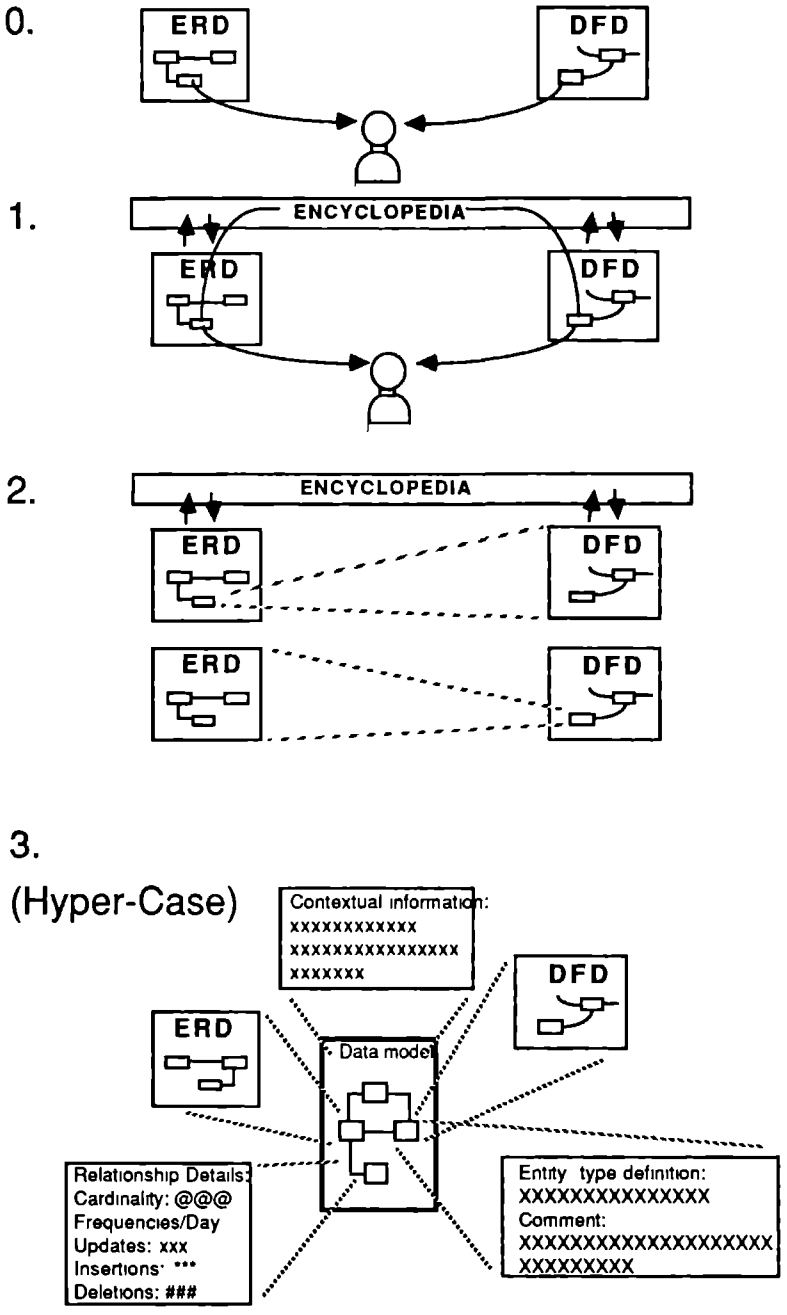


Fig. 6.5. The different degrees of modelling transparency

1. Workbenches with an accessible dictionary enable dependencies to be established. However, these links can only be shown by abandoning the one model, starting up the other workbench and displaying the other model with it. Because of this time-consuming process, developers will still manage many of the dependencies themselves. This is transparency degree 1 (see figure 6.5-1). There are groups of tools which have been matched to each other by the producer(s) and which therefore have this degree of transparency. In practice, this will only work properly if all parallel dependencies can be modelled in one tool and the change-over from the one to the other tool only concerns exclusively sequential dependencies.

2. A better form of modelling transparency is offered by workbenches providing direct support in switching from one model to the other and back (see figure 6.5-2). The best result is achieved if both models can be shown side by side on the screen and changes to the one model are immediately visible in the other. This is degree 2 of modelling transparency.

3. Transparency degree 3 is for workbenches offering the so-called hypertext functionality (see [Smith 88]) and we accordingly refer to this degree as "*Hyper-CASE*". Hyper-CASE means that the workbench user can place dependencies from each object or group of objects in a particular model to other models of all kinds, which are immediately accessible from these objects. This makes it possible to traverse the dependencies of the models in an arbitrary way (see figure 6.5-3). This is in contrast to workbenches with transparency degree 2, which permit only those traverses established in the workbench. Hyper-CASE offers flexibility to the user of the workbench, but can only be used by highly trained developers because of the difficulties in managing of unstructured dependencies.
   It is usually immediately apparent from their nature, between which model types dependencies can exist, as in the examples referred to above. By making the dependencies in an adjusted model subject to inquiry, the workbench simplifies the management of dependencies by the developer.

Considering this, we conclude that CASE-tools aiming at an integrated system development support should provide modelling transparency facilities. Structured system development projects with several analysts perform best with a meta-data model that is the same for everyone. One must therefore be very careful with dynamic dependencies and hyper-CASE. This may only be effective for highly qualified system developers or in one-person projects.

6.3.3 Modelling transparency in the IEW

The support of different modelling techniques within the constraints of the hardware is not a trivial issue and must be well motivated. We illustrate

the discussion of modelling transparency with its implementation in a currently available tool.
Modelling transparency degree 2 is implemented on a workstation of the IEW [IEW 88]. The users' interface permits the placing of several diagrams on the screen. Additions to and changes in, for instance, an association matrix can be shown direct in an entity-relationship diagram. Dependencies between models are predefined in accordance with the *Information Engineering Method* [Arthur Young 86]. Not all modelling techniques are available on all workstations. To manage dependencies between models created on various workstations, it is necessary to switch from one workstation to the other, therefore transparency degree 1. However, modelling techniques are grouped by workstation in such a way that dependencies between the various workstations are, in accordance with Information Engineering, exclusively sequential.

Sequential and parallel dependency occur in various forms in the IEW. For example, sequential dependency occurs in the IEW if action diagrams for the lowest level of processes are developed in the process model. The incoming and outgoing data flows of the process are set out in the action diagram. When further completing the action diagram, the data used by the process can be inserted, while a check is carried out to ensure that these data are actually input. An example of parallel dependency is the development of a process decomposition. A process decomposition can be shown in its entirety with the help of a decomposition diagram. A horizontal level of this decomposition can be shown in the form of a data flow diagram. The addition of a process within the decomposition diagram results immediately in the addition of the same process in the data flow diagram and vice versa. It is also possible with this dependency to link up a process to another process, after which one's attention is drawn to data flows which may possibly become lost.

6.4 DERIVATION OF SUPPORT

The problem of the derivation of guide-lines for the use of a tool within a method was already raised in section 6.1.1 at the introduction of the concept derived method companionship. For an efficient and effective system development, such a relation between a method and a tool should be established carefully. Derived method companionship can be determined using the meta-data models and meta-activity models of methods and tools. In order to formulate guide-lines for the use of the Information Engineering Workbench (IEW) within the System Development Methodology (SDM), we discuss the determination of the meta-data models and a formal derivation of the support.

6.4.1 Meta-modelling

The derivation of support guide-lines can be done in an informal way by experienced system designers, who have knowledge of both the method and the tool. This leads to the danger that the guide-lines are only applicable in

certain type of projects, and that insurmountable version problems arise when method or tool change. Therefore it is preferred that a formal derivation is performed, which has the advantage of objectivity.

The approach for this derivation is meta-modelling, which is extensively discussed in chapter 2. The meta-model of the method is obtained by analysing the development activities and the products, that result from these activities. This meta-model consists of a meta-activity model, which results from an analysis of the activities, and of a meta-data model, which results from an analysis of the products (see fig. 6.6).

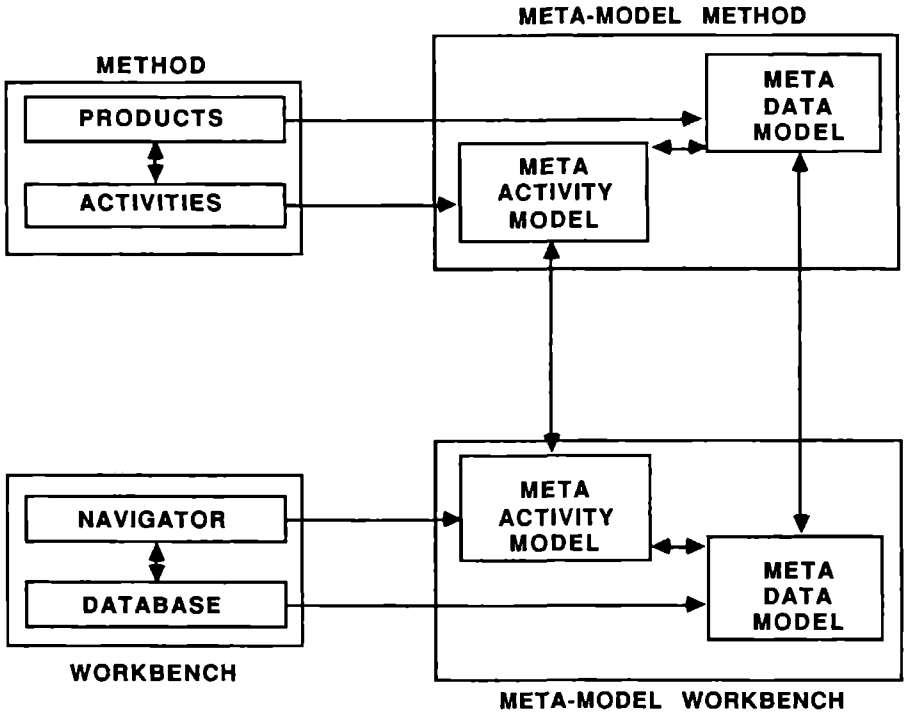


Figure 6.6 The derivation of method companionship

The same kind of analysis is performed for the tool, resulting in a meta-model of the tool. The structure of the central database used by this tool is modelled in the meta-data model. By analysing the navigator facility [Vonk 88], that prescribes the order in which modelling in the tool has to take place, the meta-activity model results. This navigator facility is at present seldom present in a tool.

When both meta-models have been created, a comparison has to take place to determine where the method is supported and where the tool is

supportive. This is depicted by the vertical arrows at the left side of figure 6.6. Here we only focus on the relation between the two meta-data models, because of the rudimentary structure of the meta-activity model of the tool.

The comparison of the meta-data models is to a great extent based on their structure, which in turn can be obtained by meta-modelling performed on the meta-data models, resulting in the *meta-meta*-data models. For every part of the meta-data model of the method we will look for a representative part of the meta-data model of the tool taking into account the structure of both parts.

The Method

We consider a derived method companionship for the System Development Methodology [Turner 87], or SDM for short. This is a method for the planning and control of an information system development project covering the whole system life cycle. SDM describes in detail which steps have to be taken in each stage of the development process, but no attention is paid to the way it has to be done. The SDM describes seven phases for top-down system design, implementation and maintenance.

Each phase consists of a number of activities which describe to a certain level of detail which steps have to be performed. The purpose of every SDM-activity is to create one or several SDM-documents, often by using one or more previously produced SDM-documents. For every SDM-document it is described which information should be included using several elementary data types, which we named SDM-elements. We considered only the phases from Information Systems Planning to Detailed System Design.

The meta-data model for the SDM is determined by analysing the products which should result from the activities, i.e. the SDM-documents. Doing this we get a small meta-data model for every SDM-document, which can be united to get the overall meta-data model for SDM. For a more detailed discussion of the derivation and the resulting meta-data model of SDM we refer to [Brinkkemper 89b]. The meta-meta-data model is created from the meta-data model. This model is given in figure 6.7.

The entities in the meta-meta-data model are the elementary data types or SDM-elements, which the method requires to occur in the SDM-documents. There are 3 kinds of SDM-elements:

1. **SDM-object type**: This represents an entity type of the meta-level, e.g. 'Process' and 'Relationship'. SDM contains about 350 object types.
2. **SDM-association type**: This represents a relationship type of the meta-level, e.g. 'Process -involves- Relationship'. An SDM-association type always involves two SDM-elements and is not recursive. SDM contains about 1000 association types.
3. **SDM-element**: This is either an SDM-object type or an SDM-association type. SDM-elements may appear as terms in a term hierarchy.

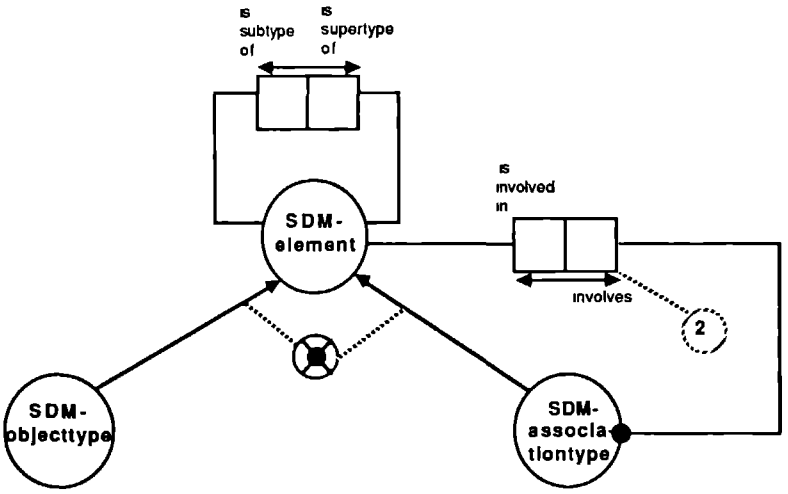


figure 6.7. The meta-meta-data-model of the SDM

The tool

The Information Engineering Workbench [IEW 88], or the IEW for short, is an initial method companion of the method Information Engineering (IEM) ([Martin 88a]). The IEM describes four phases for system development: Information Systems Planning (ISP), Business Area Analysis (BAA), System Design (DES) and Construction (CON). The IEW consists of four modules supporting these phases: the Planning Workstation for ISP, the Analysis Workstation for BAA, the Design Workstation for DES and finally the Construction Workstation for CON. For a more detailed discussion of this workbench, see [Brand 89ab].

Within every IEW-module several diagramming techniques are available reflecting the different techniques, that are prescribed by IEM. From every IEW-diagram the user has access to a part of the dictionary of the IEW, called encyclopedia. This encyclopedia has a fixed structure. We call the data types available in the encyclopedia IEW-elements. Furthermore there are IEW-reports, for reporting the contents of the encyclopedia, and IEW-analyses to check the contents of the encyclopedia or to perform certain development activities, such as the automatic generation of a relational database from an entity-relationship diagram.

The meta-data model of the IEW is that part of the meta-data model of IEM, that is implemented in the IEW. It therefore represents the structure of the encyclopedia. In the complete model there are exactly 32 object types and about 350 association types represented. The structure of the meta-data model of the IEW is revealed by the meta-meta-data model, shown in figure

6.8. This structure is needed to determine the derived method companionship of the IEW with the SDM.

The entities in this model are the elementary data types that can be distinguished in the encyclopedia of the IEW:

1. **IEW-object type**: This represents an entity type of the meta-level, e.g. 'Attribute type' and 'Data store'.
2. **IEW-association type**: This represents a relationship type of the meta-level and involves one or two IEW-object types, e.g. 'Entity type -is described by- Attribute' type.
3. **IEW-property type**: This represents an attribute type of the meta-level, e.g. 'Definition' and 'Comments'. A property type can be either an IEW-object type or an IEW-association type. There are about 100 property types in the IEW.
4. **IEW-property value type**: This represents a data type of the meta-level, such as 'Text' or 'Check mark'.

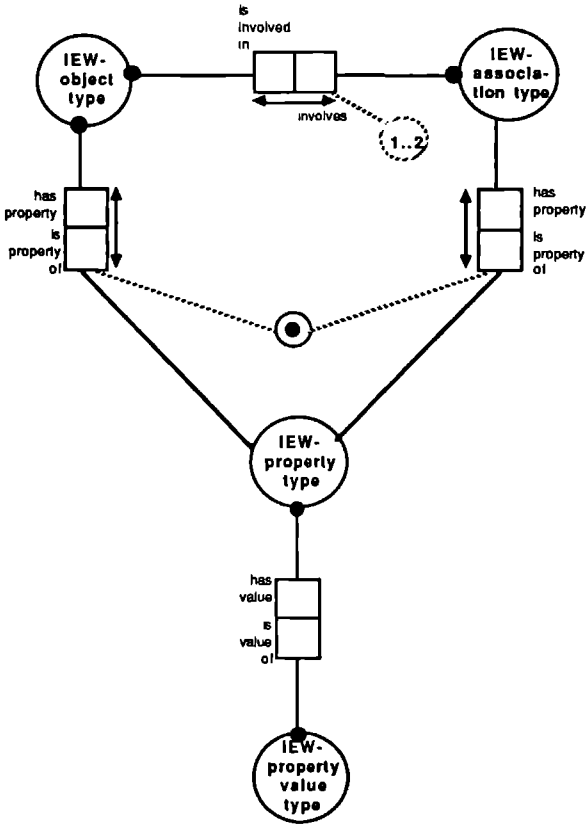


figure 6 8 The meta-meta-data-model of the iew

6.4.2 The derivation of method companionship

When both meta-data models are available, we are interested in the supported part of the method and the supportive part of the workbench. For finding the representation of a part of the method model we use the meta-meta-data models. The meta-meta-data model of the method shows us the type of the elements for which we are looking for a representation, while the meta-meta-data model of the workbench shows us all the possible types of representations. The constraints in these meta-meta-data models can be heuristics in the search process. Note also that it is possible that there are alternative representations in the workbench for certain parts of the method model.

For the SDM and the IEW this has been worked out using the models discussed in the preceding sections. The objective is to find the corresponding part of the IEW-model for the given SDM-model, which can be used for supporting SDM. This process is called the model projection M from the set SDM-MODEL, that consists of parts of the SDM meta-data model satisfying the constraints, to the set IEW-MODEL, that consists of parts of the IEW meta-data model also satisfying the constraints. M is described as a sequence of steps: the extension of the SDM-model, the projection of the extended SDM-model and the composition of the resulting IEW-models. This is illustrated in figure 6.9.

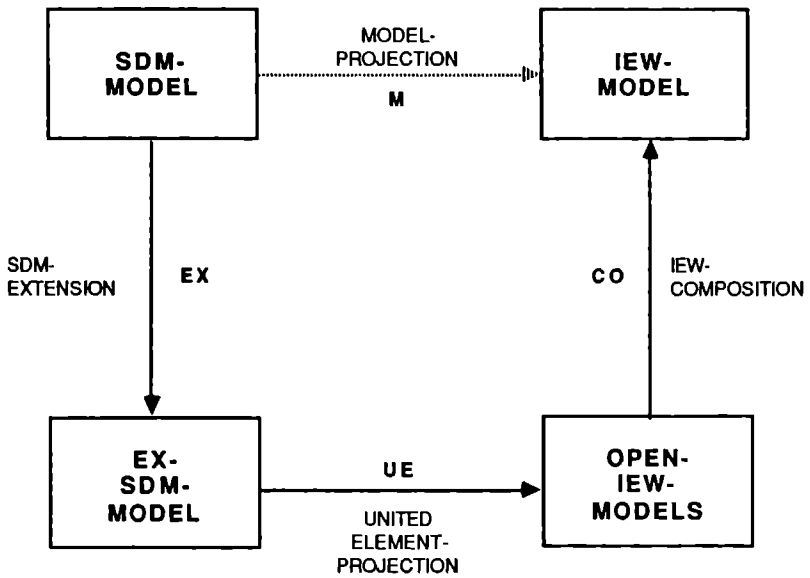


Figure 6.9. Model projection

$M : \text{SDM-MODEL} \rightarrow \text{IEW-MODEL}$

$M(x) = CO \circ UE \circ EX(x)$

The EX-function describes the extension of the SDM-model and is added to the projection because the method is not as detailed as the workbench with respect to the definition of data elements. For every part of the extended SDM-model, the united element projection UE checks whether there is a representation in the IEW. We make use of decomposition by considering the representation of a larger part as the combination of representations of the elements which are involved in it. Finally, the composition mapping CO combines the results into that part of the IEW-model, that supports SDM.

Of these three functions the UE-mapping is by far the most important. It is in its turn composed of a whole set of functions that project the various SDM-elements on the IEW-elements. For each of the eight combinations of the 2 entity types of the SDM meta-meta-data model and the four entity types of the IEW meta-meta-data model, projection functions are defined in terms of informally defined basic functions. A description of the basic functions needed for the derivation of method companionship for the case of SDM and IEW, can be found in [Brinkkemper 89b].

After performing the projection, guide-lines were drafted for using the workbench within the method. Based on the established derived method companionship, we also determined the diagramming tools in the IEW, that can be used for modelling within activities of SDM, the reporting and analysis functions in the IEW, that can be used within activities of SDM for retrieval of development information or for performing activities automatically, and the reports of the IEW, that are a part of reporting documents as prescribed in the SDM. The resulting guide-lines were published in [Lange 89].

From these investigations it turned out, that almost the full meta-data model of IEW supports SDM. This is not suprising, because system development processes will not vary that much in modelled data. The same result could be achieved for many other tools, although one should observe that tools differ greatly in the size and contents of the meta-data model. For the supported part of the meta-data model of the SDM, it appeared that the elements, whose data is scarcely updated during development and hence need little support, are mostly not supported in the workbench.

We conclude that the formal determination of derived method companionship using meta-modelling is a very useful approach to the structured formulation of guide-lines for the employment of a CASE-tool in a system development method. The process can be formulated explicitly and justified on the basis of the various meta-models.

Obviously further research in the field of meta-modelling and method companionship is needed. For instance, the derived method companionships of other methods and tools have to be determined, which

will probably lead to different meta-meta-models and hence to a different projection structure. Also feedback regarding the practical experiences with the guide-lines in projects is required. All in order to reach the ultimate goal of the improvement of the efficiency and effectiveness of system development.

6.5 LAYERED MODELLING

A lot of modelling techniques result in complex and busy diagrams. This hampers the development of correct models and the ease of comprehension by developers and informants. Petri-nets, NIAM data models, CTM-nets, among others, suffer from this deficiency. one way of handling complex models, without giving up the benefits of the modelling technique, is to split the model in several model layers. A model layer is a subset of a complete model restricted according to a particular aspect. For the specification of real-time systems, as suggested in [Hatley 87], such model layers were already introduced in the separate specification of the control flow and the data flow for common processes. In figure 6.10 the idea of layered modelling is shown for the case of dialogue modelling.

Layered modelling can be of great importance in relation to modelling procedures and their support tools. First, the support tools should provide the functionality to split a model into layers and have proper analysis checks defined on them. Secondly, the steps of a modelling procedure imply a splitting of the model over several layers. In most modelling procedures a specific aspect is treated in a single step. One could define a view on the model that only shows those aspects modelled in this particular step. The realisation of the modelling procedure with the layered modelling in a CASE-tool provides optimal development support.

Considering a layer as a model view seems relevant for various modelling techniques. Techniques in which standard or default components occur, or in which some distinguishable aspects exist in one model, are especially suited for layered modelling. In this section we will illustrate this with the layered modelling of dialogues. We will first discuss the modelling technique for dialogues and its relation to other modelling techniques. Thereafter the layered modelling is formalised.

6.5.1 Dialogue modelling

In the everyday practice of information system development, modelling of dialogues is an important design concern. The acceptance of information systems depends largely on the user interface. One of the most difficult problems of dialogue modelling is the complexity of dialogue models. To overcome this complexity we introduce layers of the model, in which aspects of a dialogue are modelled separately. The method applies augmented state transition diagrams as a specification technique. These state transition diagrams can be formally described to investigate their properties.

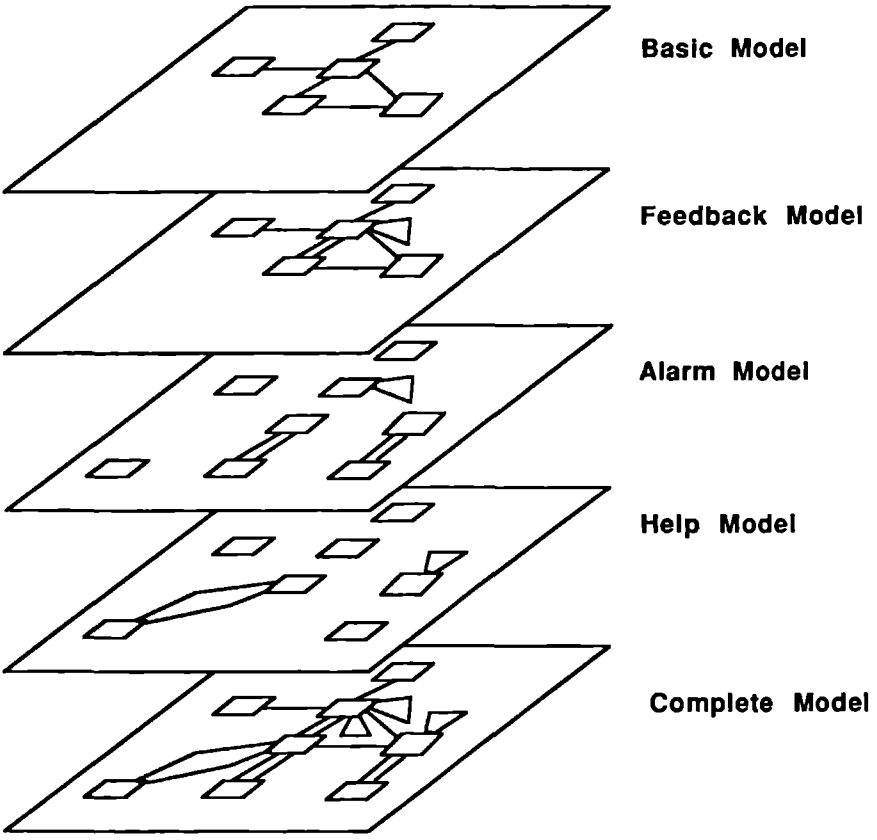


Figure 6.10 Layered Modelling of dialogues

After activity modelling and data modelling, processes at the bottom level of the global activity model (the so-called tasks, see section 5.1) can be classified as either interactive or non-interactive. An interactive task is a process for which dialogue between the system and the user is needed in order to exchange information processed by the task. For each interactive task a dialogue is modelled for which all input from the user, output to the user and all kinds of corresponding information must be described.

The overall stage of interaction specification results in models in which the various interactions between the system and the user, and the order of interaction are specified. The layout of the screens and messages is described as well. In dialogue modelling, certain system actions and conditions are identified but not elaborated. Based on the dialogue models, the user interface can be generated or programmed and fitted to the rest of the information system.

The specification technique for modelling dialogues we use here is the dialogue state transition diagram, which is an extension of the ordinary state transition diagram. A state is denoted by a rectangle and represents an initial state, a final state or an interaction point with the user. An interaction point with the user is a state of the system in which it will wait for input from the user. An example is shown in fig. 6.11.

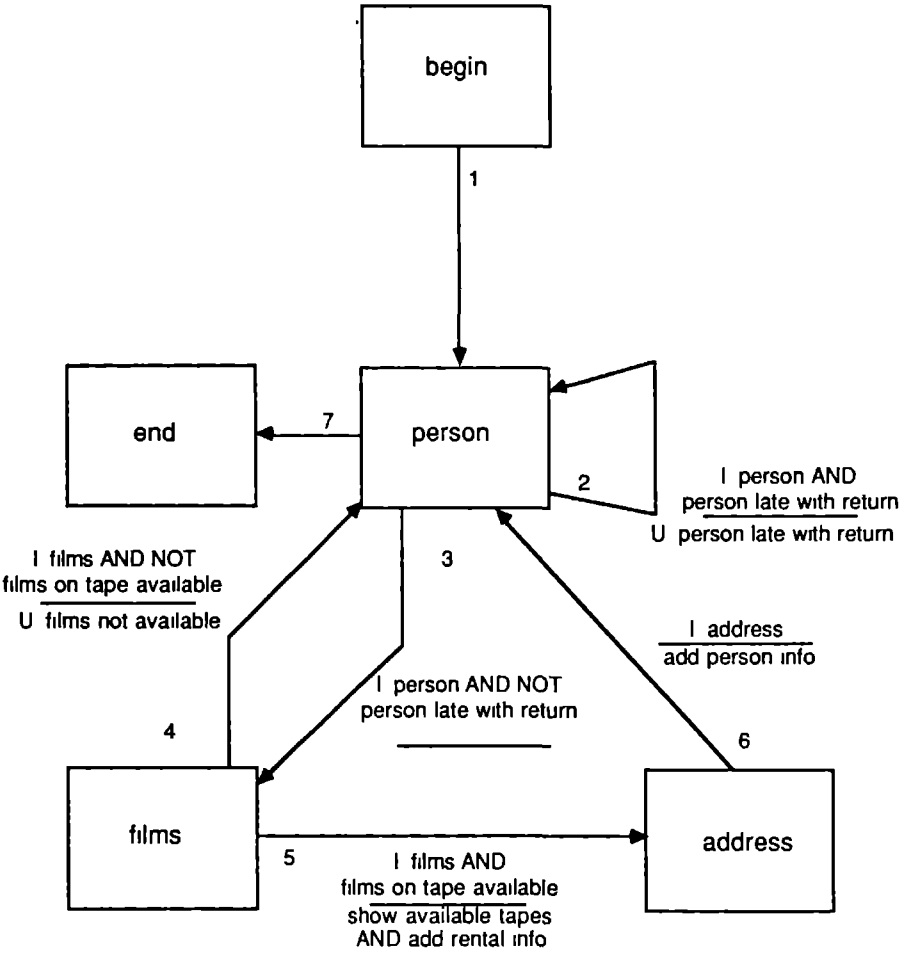


Figure 6 11 Example treatment of film request

Transitions are denoted by a directed arc and are labeled by input, conditions, actions and system output. A transition between two states occurs if the user has given a particular input and if the specified conditions are valid. As a result of a transition, the system performs the

prescribed actions and produces system output. Input and conditions are separated from actions and system output by a separator line. In addition, input is preceded by the I-symbol and system output is preceded by the U-symbol. Instead of an event, a transition can also be composed of a time limit. The transition with a time limit is made if no input is received from the user before the time limit runs out. Such transitions are called alarm transitions.

The example is part of the video rental case, as specified in appendix A. It is a simplified interactive task regarding the treatment of a film request in the video store. This task is interactive and its dialogue is depicted in the dialogue state transition diagram of figure 6.11.

For the modelling of dialogues, a modelling procedure can be given to support the designer in his task. Three kinds of messages are distinguished in dialogue modelling: basic messages, feedback messages and cancellations. A feedback message is a message in which the receiver of a message says how well he has understood the sender's message, where the sender can be either user or system. Cancellations are used to revoke a previously sent message. Basic messages are all messages between the user and the system, except feedback messages and cancellations.

A dialogue, then, is modelled in a number of steps. Each step treats a particular aspect of the interaction, for instance feedback or help. All aspects could be depicted in one model containing the specification of the complete dialogue. The complete dialogue would thus be described in one dialogue state transition diagram. This would, however, result in a very intricate and busy diagram.

We therefore elaborate the various aspects of a dialogue in different models. Feedback, for example, is modelled in the Feedback model, and help is modelled in the Help model. The various models can be seen as layers of the Complete model, which can be obtained through a union of the different models. The models we distinguish are:
- the Basic model,
- the Feedback model,
- the Help model,
- the Alarm model.

The Basic model shows the elementary dialogue, sufficient for ordinary interaction. The mutual understanding of exchanged messages is modelled in the Feedback model. The Help model specifies the assistance of the user with all kinds of help information. Finally, time limits are depicted in the Alarm model. The complete dialogue is then described in four dialogue state transition diagrams. Figure 6.10 illustrates the layered modelling of dialogues.

A complete and correct dialogue state transition diagram must meet certain requirements. For instance, only one state is the initial state. The initial state is the state from which the dialogue is started. If more than one initial state exists then ambiguity would be introduced. A transition

must begin at a state and end at a state because transitions always occur between states. These rules can be realised as checks in a dialogue modelling tool. The formalisation of these rules on the complete model or of a specific model layer is discussed in the next section.

6.5.2 Formalisation over layers

The modelling of dialogues is a technique of which both the representational concepts and the processing can be formalised. All kinds of rules, that a correct dialogue model must satisfy are formulated. In the tool, these rules can be implemented as pre-analysis checks and post-analysis checks (see 6.1.2). The meta-model of the dialogue state transition diagram enables us to depict the components of the technique, their interrelationships and the constraints. We will not discuss it here, as a discussion can be found in [Koesen 89].

The layered modelling of dialogues gives rise to rules for the complete model as well as for the separate layers. These more intricate rules cannot be expressed in a meta-model in a simple way. Moreover, the modelling of dialogues in a state transition diagram offers the possibility to specify the processing of a dialogue as an automaton, and hence the extension to prototyping tools more or less follows.

We will give just a brief outline of a formal mathematical model here.

A dialogue state transition diagram is an automaton $D = (S, I, C, A, U, T, s_i, s_f)$, where
S is the set of States,
I is the set of Inputs,
C is the set of Conditions,
A is the set of Actions,
U is the set of Outputs,
T is the set of Transitions, a subset of $((S \times I \times C) \times (A \times U \times S)$,
$s_i \in S$, the initial state, and
$s_f \in S$, the final state.

All sets are assumed to be finite. The sets A and C are sets containing both simple elements and combinations of elements. The meaning of a pair of triplets $((p, i, c), (a, u, q))$ is that the diagram D enters the State q, from the State p upon an Input i when a Condition c is fulfilled, performing the Action a and producing the Output u.

We define the set of States, for which a transition exists to a State s, as

$$T[\rightarrow \cdot\,] : S \rightarrow 2^S$$

$$T[\rightarrow s] = \{\, p \in S \mid \exists\, ((p, i, c), (a, u, s)) \in T \,\},$$

and the set of States, that can directly be reached from a State s, as

$$T[\cdot \rightarrow] : S \rightarrow 2^S$$

$$T[s\rightarrow] = \{ q \in S \mid \exists ((s, i, c), (a, u, q)) \in T \}.$$

In a similar way we define the transitive closures of these setsas : $T^+[\rightarrow s]$ the set of States from which s can be reached via one or more Transitions and $T^+[s\rightarrow]$ the set of States that can be reached from s via one or more Transitions.

Now we are able to define some rules for the complete model in terms of the above definitions.

1. There is one Initial State and the Initial State is that State which is the source of exactly one Transition and not a destination of any Transition:

$$\{ s \in S \mid |T[\rightarrow s]| = 0 \wedge |T[s\rightarrow]| = 1) = \{s_i\}$$

2. There is one Final State and the Final State is that State which is the destination of one or more Transitions and the source of no Transition:

$$\{ s \in S \mid |T[\rightarrow s]| \geq 1 \wedge |T[s\rightarrow]| = 0) = \{s_f\}$$

3. The Termination rule: the Final state has to be reachable from every State except the Final state itself.

$$T^+[\rightarrow s_f] = S \setminus \{s_f\}$$

In the same way the Reachability rule can be formulated: every State, except the Initial State itself, has to be reachable from the Initial State. Other rules formulate, among other things, the non-ambiguity of conditional choice and the completeness of input specification.

Similarly, an automaton can be defined and rules can be formulated for every model layer. We consider for example the Help model. Assume that S_B is the set of states of the Basic Model and S_H is the set of help states. In the Help model only those states of the Basic model appear that are connected to a single unique help state. For the Help Model, an automaton

$$D_H = (S_H \cup S_B, I_H, C_H, A_H, U_H, T_H)$$

can be defined, where I_H consists only of the help request and the help return, C_H contains only the trivial condition 'TRUE', A_H consists of help actions, U_H of help texts and T_H of the transitions from the states of the Basic model to those of the Help model and back. More specifically

$$T_H \subseteq ((S_B \times I_H \times C_H) \times (A_H \times U_H \times S_H)) \cup ((S_H \times I_H \times C_H) \times (A_H \times U_H \times S_B))$$

All sets in D_H are considered to be subsets of the corresponding set in D. For D_H in addition to the other rules the following two rules hold.

4. In each state of the basic model, help can be obtained.

$$\forall s \in S_B \; \exists i \in I_H,\ c \in C_H,\ a \in A_H,\ u \in U_H,\ s' \in S_H \; [\ ((s,i,c),(a,u,s')) \in T_H\ ]$$

5. From each help state we return to the state, where the help was requested for.

$$\forall s' \in S_H \wedge ((s,i,c),(a,u,s')) \in T_H \; \exists i' \in I_H,\ c' \in C_H,\ a' \in A_H,\ u' \in U_H$$
$$[\ ((s',i',c'),(a',u',s)) \in T_H\ ]$$

Other rules formulate the singularity and the uniqueness of the help state. From these rules various properties can be deduced, for example that the set of transitions from the basic model to the help model is a bijective relation.

In the same way rules for the other layers can be formulated. All rules provide insight in the complete model as well as in the separate layers. Most of the rules give rise to post-analysis checks to be built into a dialogue modelling support tool integrated with tools that support the remaining modelling activities of a system development method.

APPENDIX

THE VIDEO STORE BENCHMARK

In order to present examples of the discussed modelling techniques, we will make use of a small case involving a Video store. This case is on purpose kept small and simple as to avoid a high level of complexity and introduction of many irrelevant details. On the other hand, the case is large enough to show some interesting aspects of the presented technique. This case is an elaboration of the case presented in [Moriarty 87].

A.1 Activities of the Video store

The Video store receives tape requests and returned tapes from customers and new tapes from tape suppliers. Whenever necessary the Video store sends tape overdue notices to its customers.

Tape requests are taken care of by the store assistant. While treating these tape requests he uses information concerning films and tapes. He also uses and updates the list of rentals.

The store assistant also takes care of tape returns. For this task the store assistant again needs the information concerning films and tapes as well as the list of rentals. Here too the list of rentals is updated.

The other activities are the submission of rate changes and the submission of new tapes by the store management and the production of rental reports and the sending of tape overdue notices by the store administration.

The elaboration of this description of the activities of the Video store results in a context diagram (fig. A.1) and a data-flow diagram (fig. A.2). The context diagram depicts the relevant interactions of the Video store with the external agents, denoted by double rectangles, in its environment.

The data-flow diagram depicts the activities of the Video store in more detail. We used a variation for the notation of these diagrams, in which the internal agents are listed within the activities.

A.2 Information in the Video store case

As a starting point for the data modelling, samples, such as forms, letters and tables containing information used in the Video store are taken. A sample can be related to a data-flow or data store in the data-flow diagram

which resulted from the activity analysis. The sample tables are shown here in fig A.3. Explanation on the tables still to be extended.

In fig. A.4 the global conceptual schema resulting from the data modelling phase is shown in the notation of the elementary data modelling technique of NIAM. This global conceptual schema is constructed by means of a bottom-up integration, according to the activity decomposition, of all the schemas of the data-flows and data stores.

A.3 Elaboration of the task 'Treatment of film request'

In the following paragraph one of the activities of the data-flow diagram, namely the activity 'Treatment of film request', is described in more detail.

Film request are taken care of by the store assistant. Whenever a customer requests a film, the store assistant first checks whether there are still tapes, containing the requested film, available and then he searches, if necessary, for the rental price. Next he checks whether this customer is late with the return of other tapes. If so he may not borrow the tape requested. If this is not the case, he records this rental by updating the rental list. For this, the customer is asked for his address.

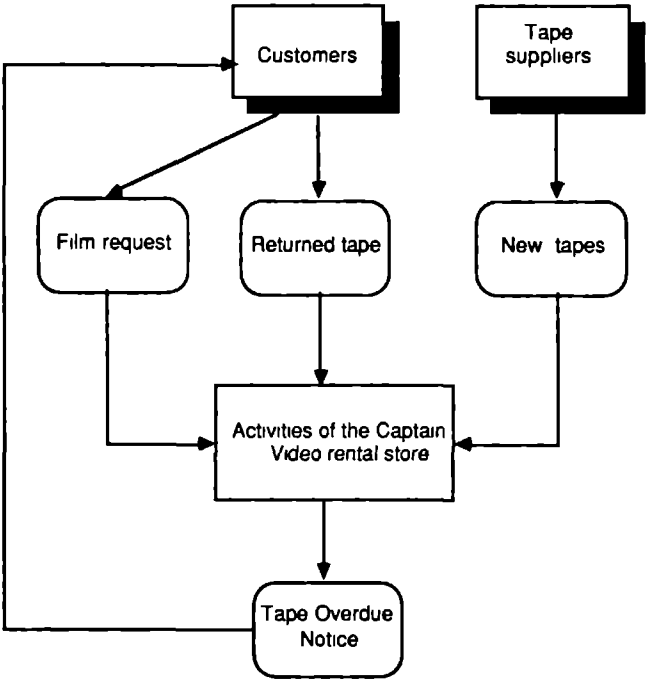


Figure A 1 Context diagram of the Video store case

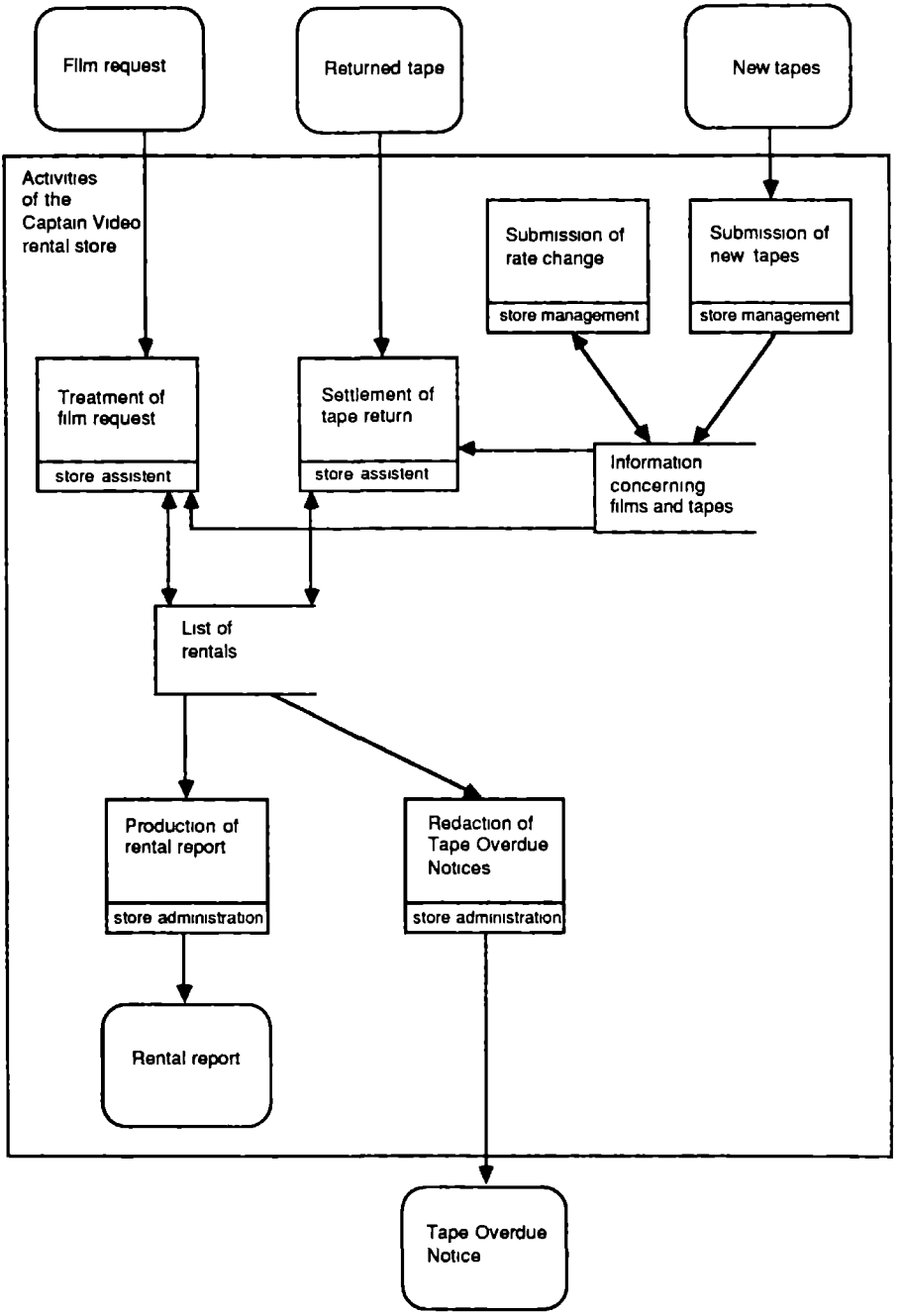


Figure A.2 Data-flow diagram of the Video store case

List of rentals:

Rented film	Tape	Person	Street and house-number	Community	Check-out date	Return date	Rental charge
Marathon Man	T301	F Smith	40 Church Road	Albany	18-2-1988	19-2-1988	$ 3,--
Goldfinger	T207	F Smith	40 Church Road	Albany	18-2-1988	19-2-1988	$ 3,--
The Godfather	T642	A Jones	5 Polstead Road	Prescott	19-2-1988		
Marathon Man	T305	A Jones	5 Polstead Road	Prescott	18-2-1988	22-2-1988	$ 4,--
Modern Times	T489	A. Jones	5 Polstead Road	Prescott	20-2-1988		
Ghandi	T500	W Brown	9 Oak Road	Albany	20-2-1988	25-2-1988	$11,--
Marathon Man	T301	F. Smith	40 Church Road	Albany	20-2-1988		

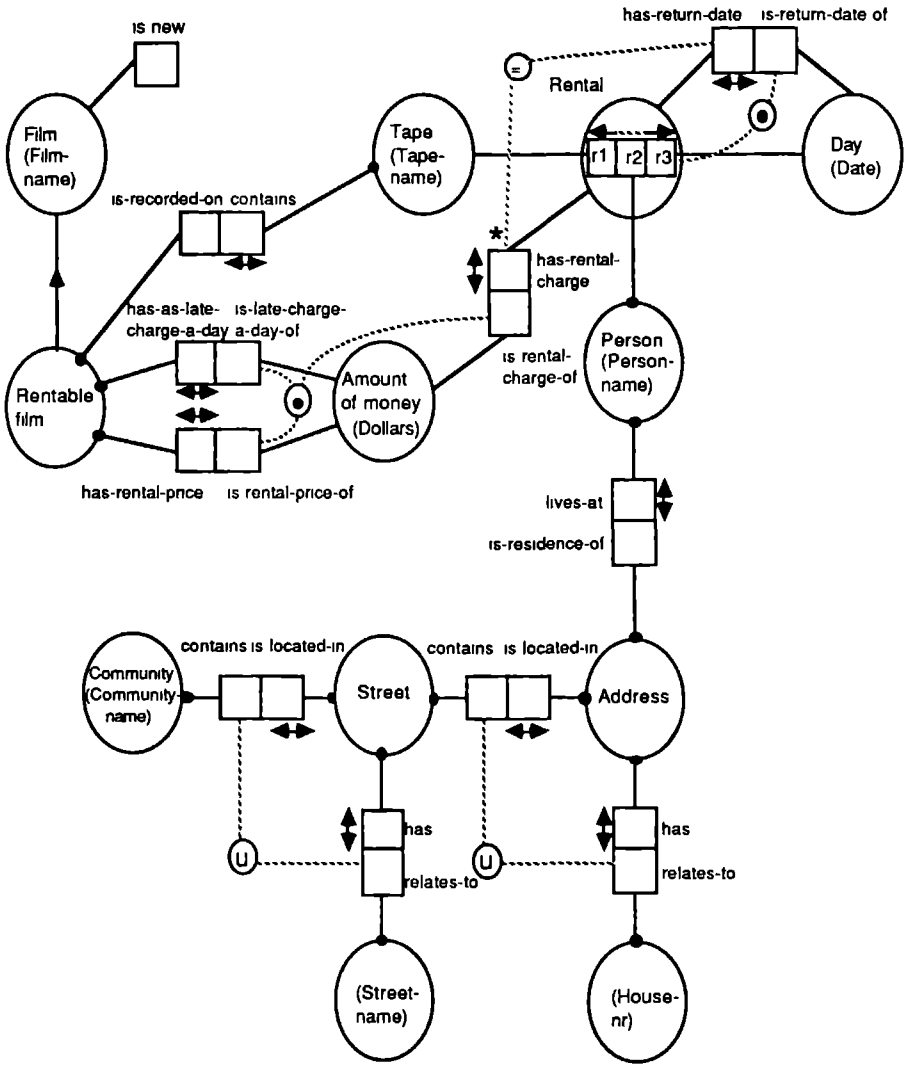
Information concerning films and tapes:

Film	Rental price	Late charge a day	Tapes
Marathon Man	$ 3,--	$ 1,--	T301 T305 T309
Goldfinger	$ 3,--	$ 1,--	T207 T208
The Godfather	$ 4,--	$ 2,--	T642 T666
Modern Times	$ 2,--	$ 1,--	T489
Ghandi	$ 5,--	$ 3,--	T500
The Great Dictator	$ 2,--	$ 1,--	T180 T181 T182
Empire of the Sun			

Rental report:

February 1988	
Film	Nr. of Rentals
Marathon Man	3
Goldfinger	1
The Godfather	1
Modern Times	1
Ghandi	1

Figure A.3 Sample tables for data modelling



A rentable film is a film which does not play the role 'is new'.

r1 = The Tape.. has been borrowed by Person.. on Day..
r2 = Person.. borrows on Day.. the Tape..
r3 = On Day.. Person.. borrows the Tape.

Figure A.4 Conceptual schemas

REFERENCES

[Abiteboul 87] Abiteboul, S and R Hull, IFO A Formal Semantic Database Model ACM Transactions on Database Systems, vol 12, nr 4, pp 525-565, 1987

[Abrial 74] Abrial, J R , Data semantics In Data Base Management, J W Klimbie and K.L Koffeman (Eds), North-Holland, Amsterdam, pp 1 60, 1974

[Aerts 88] Aerts, A T M and K M van Hee, A Tutorial for Data Modelling, Computing Science Notes 88/09, Technical University of Eindhoven, 1988

[Ahituv 87] Ahituv, N , A Metamodel of Information Flow A Tool to Support Information Systems Theory Communications of the ACM, vol 30, nr 9, pp 781-791, 1987

[Alvey 82] Alvey, A programme for advanced information technology, 1982

[Antonellis 81] Antonellis, V D and B Zonta, Modelling events in database applications design, In Proc of 7th Int Conf on VLDB, Cannes, France, pp 23-31, 1981

[Apostel 60] Apostel, L Towards the formal study of models in the non-formal sciences Synthese, vol 12, pp 125-161, 1960

[Arthur Young 87] Arthur Young Information Technology Group The Arthur Young practical guide to Information Engineering John Wiley & Sons, Inc , 1987

[Barwise 77] Barwise, J , An Introduction to First Order Logic In Handbook of Mathematical Logic, J Barwise (Ed), Studies in Logic and Foundations of Mathematics, vol 90, North-Holland, Amsterdam, 1977

[Bemelmans 87] Bemelmans, T M A , Business Information Systems and Automation (In Dutch Bestuurlijke informatiesystemen en automatisering) Stenfert Kroese, Leiden, 3rd revised edition, 1987

[Berdal 86] Berdal, S , S Carlsen, A. Sølvberg and R Andersen, Information System Behaviour Expressed through Process Port Analysis Unpublished manuscript, Division of Computer Science, The Norwegian Institute of Technology, University of Trondheim, Norway, 1986

[Bergstra 86] Bergstra, J A and J W Klop, Process Algebra specification and verification in bisimulation semantics In Mathematics and Computer Science II, CWI Monograph 4, M Hazewinkel, J K Lenstra and L G L T Meertens (Eds), North-Holland, Amsterdam, pp 61-94, 1986

[Bergstra 89] Bergstra, J A and G R Renardel de Lavalette The position of formal specifications in software technology (in Dutch De plaats van formele specificaties in de software technologie) Informatie, vol 31, nr 6, pp 480-494, June 1989

[Bertels 69] Bertels, K. and D Nauta, Introduction to the notion of model (In Dutch Inleiding tot het modelbegrip) De Haan, Bussum, 1969

[Bjorner 78] Bjorner, D and C B Jones (Eds), The Vienna development method the meta-language Lecture Notes in Computer Sciences, nr 61, Springer Verlag, 1978

[Blumenthal 69] S. Blumenthal, Management Information Systems: A framework for planning and development, Prentice-Hall, Englewood Cliffs, NJ, 1969.

[Bots 89] Bots, P.W.G., An Environment to Support Problem Solving, Doctoral dissertation, Delft University of Technology, 1989.

[Bouman 88a] Bouman, H.D., M.J.J. Knots, L.J.Th.O. van Erning and S. Brinkkemper, PACS in practice: The status of the PACS project at the St. Radboud University Hospital. Part B: A Digital Image Archive: Information Analysis and Development. Medical Informatics, vol. 13, nr. 13, pp. 265-278, 1988.

[Bouman 88b] Bouman, H.D., M.J.J. Knots, S. Brinkkemper and L.J.Th.O. van Erning, The development of a digital picture archive (in Dutch: De ontwikkeling van een digitaal beeldarchief). Informatie, vol. 30, nr. 10, pp. 796-801, 1988.

[Brand 89a] Brand, N.A., S. Brinkkemper and F.H.G.C. van der Steen, Information Engineering: Integration of method and workbench (in Dutch: Information Engineering: Integratie van methode en workbench). Informatie, vol. 31, nr. 2, pp. 95-103, February 1989.

[Brand 89b] Brand, N.A., S. Brinkkemper and F.H.G.C. van der Steen, Integration of an Information Systems Development Method and its Companion Workbench. Technical report 89-2, Department of Information Systems, University of Nijmegen, February 1989.

[Brinkkemper 88a] Brinkkemper, S., N.A. Brand and J. Moormann, Deterministic Modelling Procedures for Automated Analysis and Design Tools. In: T.W. Olle, A.A. Verrijn Stuart and L. Bhabuta (Eds.), Computerized Assistance during the Information Systems Life Cycle, Proceedings of the CRIS 88 conference, North-Holland, Amsterdam, pp. 117-160, 1988. Extended Version in Technical report nr. 88-10, Department of Information Systems, University of Nijmegen, May 1988.

[Brinkkemper 88b] Brinkkemper, S. and N.A. Brand, Deterministic modelling procedures for the automated analysis and design. Associated tools (in French: Procedure de modélisation deterministe pour l'analyse et la conception automatiques. Outils associés). In: proceedings of the 2ièmes journees Pratique des methodes et outils logiciels d'aide a la conception de systèmes d'information, H. Habrias (Ed.), Nantes, France, September 1988.

[Brinkkemper 89a] Brinkkemper, S. and A.H.M. ter Hofstede, The Modelling of Tasks at a Conceptual Level in Information Systems Development Methods. In: Workshop Proceedings for the CRIS Review workshop, G.M. Nijssen and S. Twine (Eds.), IFIP WG 8.1 meeting, Sesimbra, Portugal, June 1989. Extended version in Technical report nr. 88-18, Department of Information Systems, University of Nijmegen, December 1988.

[Brinkkemper 89b] Brinkkemper, S., M. de Lange, R. Looman and F.H.G.C. van der Steen, On the Derivation of Method Companionship by Meta-Modelling. In: Advance Working Papers, Third International Conference on Computer Aided Software Engineering, J. Jenkins (Ed.), Imperial College, London, UK, pp. 266-286, July 1989. Also in: Software Engineering Notes, journal of the Special Interest Group on Software Engineering of the ACM, vol. 15, nr. 1, January 1990.

[Brinkkemper 89c] Brinkkemper, S., The Essence and Support of Modelling Transparency, Position paper. In: Advance Working Papers, Third International Conference on Computer Aided Software Engineering, J. Jenkins (Ed.), Imperial College, London, UK, July 1989.

[Brinkkemper 89d] Brinkkemper, S , M Geurts, I van de Kamp and J Acohen, On a Formal Approach to the Methodology of Information Planning In Proceedings of the First Dutch Conference on Information Systems, R Maes (Ed), 1 - 2 November 1989, Amersfoort Submitted for publication

[Brinkkemper 89e] Brinkkemper, S and A H M ter Hofstede, The Conceptual Task Model a Specification Technique between Requirements Engineering and Program Development Technical Report nr 89 15, Department of Information Systems, University of Nijmegen, September 1989 In Proceedings of the CAiSE conference, Kista, Sweden, A Solvberg (Ed), Lecture Notes in Computer Science, Springer Verlag, May 1990

[Brinkkemper 90a] Brinkkemper, S , M de Lange, R Looman and F H G C van der Steen, The formal derivation of support of a system development method (in Dutch De formele afleiding van ondersteuning bij een systeemontwikkelingsmethode) Submitted to Informatie, 1990

[Brinkkemper 90b] Brinkkemper, S and G M Wijers (Eds), The next generation of CASE-tools, Proceedings of an international SERC workshop in Noordwijkerhout, Software Engineering Research Centre, Utrecht, April 1990

[Brodie 82] Brodie, M L and E Silva, Active and Passive Component Modelling ACM/PCM In T W Olle, H G Sol and A A Verrijn Stuart (Eds), Information Systems Design Methodologies A Comparative Review Proceedings of the CRIS 82 conference, North-Holland, Amsterdam, pp 41-92, 1982

[Bruza 89] Bruza, P D and Th P van der Weide, The Semantics of TRIDL, Technical Report, Department of Information Systems,, University of Nijmegen, 1989

[Buhr 85] Buhr, R J A , System Design with Ada, Prentice Hall, Englewood Cliffs, 1985

[Case 86] A F Case jr , Information Systems Development principles of computer aided software engineering, Prentice-Hall, 1986

[Chen 76] Chen, P P , The Entity-Relationship model - Toward a unified view of data ACM Transactions on Database Systems, vol 1, nr 1, pp 9-36, 1976

[Cordes 89] Cordes, D W and D L Carver, Evaluation method for user requirements documents Information and Software Technology, vol 31, nr 4, pp 181-188, 1989

[De Brabander 84] De Brabander, G and G Thiers, Successfull information system development in relation to situational factors which affect effective communication between MIS-users and EDP-specialists Management Science, vol 30, nr 2, pp 137-155, 1984

[Dietz 87] Dietz, J L G , Modelling and specification of information systems (in Dutch Modelleren en specificeren van informatiesystemen) Doctoral dissertation, Technical University of Eindhoven, Oktober 1987

[Dignum 86] Dignum, F , M Jacques and M L Kersten, Workbenches a design environment for information systems (In Dutch Workbenches een ontwerpomgeving voor informatiesystemen) Informatie, vol 28, pp 813-825, 1986

[Dijk 86] Dijk, A , Correspondences between information analysis and language analysis (in Dutch Overeenkomsten tussen informatieanalyse en taalanalyse) Informatie, vol 28, nr 1, pp 51-58, 1986

REFERENCES

[Dubois 85] Dubois, E., J. Hagelstein, E. Lahou, A. Rifaut and F. Williams, A Process Model for Requirements Engineering. In: ESPRIT '85: Status Report of Continuing Work, pp. 593-607, North-Holland, 1986.

[Elmasri 85] Elmasri, R., J. Weeldreyer and A. Hevner, The category concept: an extension to the entity-relationship model. Data and Knowledge Engineering, vol. 1, pp.75-116, 1985.

[Falkenberg 83] Falkenberg, E.D., Foundations of the Conceptual Schema Approach to Information Systems. In: Data Base Management: Theory and Applications, C.W.Holsapple and A.B.Whinston (Eds.), D. Reidel Publishing Company, pp. 3-17, 1983.

[Falkenberg 87] Falkenberg, E.D., Data Bases and Information Systems 1, Lecture Notes, University of Nijmegen, 1987.

[Falkenberg 88] Falkenberg, E.D., H. van Kempen and N. Mimpen, Knowledge-based Information Analysis Support. In: The Role of Artificial Intelligence in Databases and Informations Systems, Joint TC2-TC8 Working Conference, R. Meersman and C.H. Kung (Eds.), pp. 63-78, Guangzhou, China, July 1988.

[Falkenberg 89a] Falkenberg, E.D., R. van der Pols and Th.P. van der Weide, Understanding Process Structure Diagrams. In: Workshop Proceedings for the CRIS review workshop, G.M. Nijssen and S. Twine (Eds.), IFIP WG 8.1 meeting, Sesimbra, Portugal, June 1989.

[Falkenberg 89b] Falkenberg, E.D. and P. Lindgreen (Eds.), Proceedings of the Conference on Information Systems Concepts, North-Holland, Amsterdam, 1989.

[Falkenberg 89c] Falkenberg, E.D., Information Modelling - Subjective Forever, Proceedings Database dag, K.M. van Hee and T.M.A. Bemelmans (Eds.), Eindhoven, December 1989.

[Gane 79] Gane, C. and T. Sarson, Structured Systems Analysis: Tools and Techniques. Prentice Hall, Englewood Cliffs, 1979.

[Genrich 79] Genrich, H. and K.Lautenbach, The analysis of distributed systems by means of predicate/transition nets. Semantics of Concurrent Computation, G.Kahn (Ed.), Lecture Notes in Computer Sciences, nr.70, Springer Verlag, pp.123-146, 1979.

[Genrich 87] Genrich, H., Predicate/Transition Nets. In: Petri Nets: Central models and their properties, W.Brauer, W.Reisig and G.Rozenberg (Eds.), Lecture Notes in Computer Sciences, nr. 1254, Springer Verlag, pp. 207-247, 1987.

[Gillenson 87] Gillenson, M.L., The Duality of Database Structures and Design Techniques. Communications of the ACM, vol. 30, nr. 12, pp. 1056-1065, 1987.

[Gilmore 87] Gilmore, P.C., A Foundation for the Entity-Relationship Approach: How and Why. In: Proceedings of the Sixth International Conference on the Entity-Relationship Approach, S. March (Ed.), North-Holland, 1987.

[Godwin 89] Godwin, A.N., J.W. Gleeson and D. Gwillian, An Assessment of the IDEF Notations as Descriptive Tools. Information Systems, vol. 14, nr. 1, pp. 13-28, 1989.

[Green 82] Green, T.R.G., Pictures of programs and other processes, or how to do things with lines. Behaviour and Information Technology, vol. 1, nr. 1, pp. 3-36, 1982.

[Griethuysen 81] Griethuysen, J.J. van, Fundamentals and terminology of the conceptual schema (in Dutch: Grondslagen en terminologie van het conceptuele schema). Informatie, vol. 23, nr. 7/8, pp. 422-512, 1981.

[Guyot 86] Guyot, J., Un modèle de traitement pour les bases de données: un formalisme pour la conception, la validation et l'exécution de la spécification d'une application. Thèse de la Faculté des Sciences, Concept Moderne/Editions, Genève, 1986.

[Harel 88] Harel, D., On Visual Formalisms. Communications of the ACM, vol. 31, nr. 5, pp. 514-530, 1988.

[Hatley 87] Hatley, D.J. and I.A. Pirbhai, Strategies for Real-Time System Specification. Dorset House Publishing Co., New York, 1987.

[Hesse 83] Hesse, W., Methods and Tools for Software Development - a Walk through the Technology Landscape. In: Colloquium Programmeeromgevingen, J. Heering and P. Klint (Eds.), MC Syllabus 30, Mathematical Centre, 1983.

[Hoare 85] Hoare, C.A.R., Communicating Sequential Processes, Prentice Hall, 1985.

[Hsu 85] Hsu, C., Structured Database System Analysis and Design through Entity Relationship Approach. In : Fourth International Conference on Entity-Relationship Approach: The Use of ER Concept in Knowledge Representation, P.P. Chen (Ed.), pp. 56-63, North-Holland - IEEE, 1985.

[Hull 87] Hull, R. and R. King, Semantic Database Modelling: Survey, Applications and Research Issues. ACM Computing Surveys, vol. 19, nr. 3, pp. 201-260, 1987.

[Humphrey 89] Humphrey, W.S. and M.I. Kellner, Software Process Modelling: Principles of Entity Process Models. In: Proceedings of the 11th International Conference on Software Engineering, IEEE, Pittsburgh, May 1989.

[IEW 88] KnowledgeWare, Information Engineering Workbench ESP, comprising: Planning, Analysis and Design Workstation Guide, 1988.

[Iivari 89] Iivari, J., Contemporary Schools of Information System Development: a paradigmatic analysis. Unpublished manuscript, Institute of Information Processing, University of Oulu, Finland, 1989.

[ISO 82] van Griethuysen, J.J. (Ed.), Concepts and terminology for the conceptual schema and the information base. Report ISO TC97/SC5/WG3 - N695, 1982.

[ITC 87] Index Technology Corporation, Excelerator Customizer Manual, 1987.

[Ives 80] Ives, B., S. Hamilton and G.B. Davis, A Framework for Research in Computer-Based Management Information Systems. Management Science, vol. 26, nr. 9, pp. 910-934, 1980.

[Jackson 83] Jackson, M., System Development. Prentice Hall, Englewood Cliffs, 1983.

[Kersten 86] Kersten, M.L., H. Weigand, F. Dignum and J. Boom, A Conceptual Modelling Expert System. In: Proceedings of the Fifth International Conference on Entity-Relationship Approach, S. Spaccapietra (Ed.), Dijon, France, pp. 275-288, November 1986.

[Koesen 89] Koesen, C.A.M., S. Brinkkemper and H.E. Keus, The Layered Modelling of Dialogues and its Support Workbench. In: Advance Working Papers, Third International Conference on Computer Aided Software Engineering, J. Jenkins (Ed.), Imperial College, London, UK, pp. 87-107, July 1989. Submitted to: Information and Software Technology.

[Kokol 89] Kokol, P., Formalization of the information system development process using meta-models. ACM SIGSOFT Software Engineering Notes, vol. 14, nr. 5, pp. 118-123, 1989.

[Kung 86] Kung, C.H. and A. Sölvberg, Activity Modeling and Behavior Modeling. In: T.W. Olle, H.G. Sol and A.A. Verrijn Stuart (Eds.), Information System Design Methodologies: Improving the Practice, Proceedings of the CRIS 86 conference, North-Holland, Amsterdam, pp. 145-171, 1986.

[Lange 89] Lange, M de, The Use of the Workbench IEW within SDM: Information Systems Planning, Definition Study, System Design, Detailed System Design (In Dutch). Moret Advies, 1989.

[Lehman 87] Lehman, M.M., Process models, Process Programs, Programming Support. In: Proceedings of the 9th International Conference on Software Engineering, IEEE, pp. 14-16, 1987.

[Lewis 81] Lewis, H.R. and C.H. Papadimitriou, Elements of the theory of Computation,. Prentice Hall, 1981.

[Linden 88a] Linden, E.-J. v.d., K. de Smedt, M. v.d. Linden, P. van Boven and S. Brinkkemper, The representation of lexical objects. In: Proceedings of the BUDALEX '88 conference, T. Magay (Ed.), Budapest, September 1988.

[Linden 88b] Linden, E.-J. v.d., K de Smedt, M. v.d. Linden, P. van Boven and S. Brinkkemper, The representation of lexical objects (in Dutch: De representatie van lexicale objecten). Rapport TNO Instituut voor Toegepaste Informatica, Kenmerk 88 ITI B 21, May 1988.

[Lindgreen 86] Lindgreen, P., Entities from a systems point of view. In: Proceedings of the 5th International Conference on Entity Relationship Approach, S. Spaccapietra (Ed.), pp. 3-15, November 1986.

[Lubars 89] Lubars, M., General Design Representation. MCC report, STP-066-89, MCC, Austin, Texas, 1989.

[Lundeberg 80] Lundeberg, M., G. Goldkuhl and A. Nilsson, Information Systems Development - A Systematic Approach. Prentice Hall, Englewood Cliffs, 1980.

[Lyytinen 87] Lyytinen, K., Different Perspectives on Information Systems: Problems and Solutions. ACM Computing Surveys, vol. 19, nr.1, pp. 5-46, 1987.

[MacDonald 82] MacDonald, I.G. and I.R. Palmer, System Development in a Shared Data Environment. In: T.W. Olle, H.G. Sol and A.A. Verrijn Stuart (Eds.), Information Systems Design Methodologies: A Comparative Review, Proceedings of the CRIS 82 conference, North-Holland, Amsterdam, pp. 235-283, 1982.

[Martin 85] Martin, J. and C.L. McClure, Action Diagrams. Prentice Hall, Englewood Cliffs, 1985.

[Martin 88a] Martin, J., Information Engineering. volume 1, 2 and 3, Savant Research studies, Lancashire, 1988.

[Martin 88b] Martin, J. and C.L. McClure, Structured Techniques: the Basis for CASE. Revised Edition, Prentice Hall, Englewood Cliffs, 1988.

[McClure 86] McClure, C.L., Taking a closer look at software workstations - the newest productivity tools. Arthur Young International, 1986.

[McClure 87] McClure, C.L., Computer-Aided Software Engineering, System Development, vol. 7, nr. 11, November 1987.

[McClure 89] McClure, C.L., CASE is Software Automation, Prentice Hall, Englewood Cliffs, 1989.

[McNeile 86] McNeile, A.T., Jackson System Development (JSD). In: T.W. Olle, H.G. Sol and A.A. Verrijn Stuart (Eds.), Information System Design Methodologies: Improving the Practice, Proceedings of the CRIS 86 conference, North-Holland, Amsterdam, pp.225-246, 1986.

[Meersman 82] Meersman, R., The RIDL Conceptual Language,. Research Report ICIAS, Brussels, 1982.

[Mees 86] Mees, M. and F. Put, Extending a dynamic modelling method using data modelling capabilities: the case of JSD. In: Proceedings of the Fifth International Conference on Entity-Relationship Approach, S. Spaccapietra (Ed.), Dijon, France, pp. 37-56, November 1986.

[Millner 80] Millner, R., A Calculus of Communicating Systems, Lecture Notes in Computer Sciences, nr. 29, Springer Verlag, 1980.

[Moriarty 88] Moriarty, T., Which is the "right" data model for a given problem. In: Proceedings of the Sixth International Conference on the Entity-Relationship Approach, S. March (Ed.), pp. 17-18, North-Holland, Amsterdam, 1988.

[NGGO 88] Nederlandse Gebruikersgroep van Gestructureerde Ontwikkelingsmethoden, Werkgroep TOOLS, Computerised support of structured development methods: an inventory of tools (in Dutch: Computer ondersteuning van gestructureerde ontwikkelingsmethoden: een inventarisatie van tools). 1988.

[NIIG 88] Nederlands Ipse Interest Group, IPSE, a Survey of problems and definition IPSE (In Dutch: IPSE, Inventarisatie van problemen en definitie IPSE). 1988.

[Nijssen 89] Nijssen, G.M. and T.A. Halpin, Conceptual Schema and Relational Database Design: a Fact-Based Approach, Prentice Hall, 1989.

[Ogden 49] Ogden, C.K. and I.A. Richards, The meaning of meaning. 10th edition, Routledge and Kegan Paul, 1949.

[Olle 82] Olle, T.W., H.G. Sol and A.A. Verrijn Stuart (Eds.), Information System Design Methodologies: A Comparative Review. Proceedings of the CRIS 82 conference, North-Holland, Amsterdam, 1982.

[Olle 88a] T.W. Olle, Business Analysis and System Design specifications for an inventory control and purchasing system. Appendix A in: T.W. Olle, A.A. Verrijn Stuart and L. Bhabuta (Eds.), Computerized Assistance during the Information Systems Life Cycle, Proceedings of the CRIS 88 conference, North-Holland, Amsterdam, 1988.

[Olle 88b] Olle, T.W., J. Hagelstein, I.G. MacDonald, C. Rolland, H.G. Sol, F.J.M. van Assche and A.A. Verrijn Stuart, Information System Methodologies - A Framework for Understanding. Addison-Wesley, 1988.

[Parent 87] Parent, C., The ERC approach: A data model and an entity-relationship algebra (in French: L' approche ERC: un modèle de données et une algèbre de type entité-relation). These de Doctorat Etat, l'Université de Paris 6, July 1987.

[Prabhakaran 88] Prabhakaran, N. and E.D. Falkenberg, Representation of Dynamic Features in a Conceptual Schema. The Australian Computer Journal, vol. 20, nr. 3, pp. 98-104, 1988.

[Reisig 85] W.Reisig, Petri Nets. EATCS Monographs on Theoretical Computer Science, Springer Verlag, 1985.

[Reisig 87] Reisig, W., Petri Nets in Software Engineering. In: Petri Nets: Applications and relationships to other models of concurrency, W.Brauer, W.Reisig and G.Rozenberg (Eds.), Lecture Notes in Computer Sciences, nr. 255, Springer Verlag, pp. 63-96, 1987.

[Richter 82] Richter,G. and R. Durchholz, IML-Inscribed High-Level Petri Nets. In: T.W. Olle, H.G. Sol and A.A. Verrijn Stuart (Eds.), Information Systems Design Methodologies: A Comparative Review. Proceedings of the CRIS 82 conference, North-Holland, Amsterdam, pp.335-368, 1982.

[Rolland 82] Rolland, C. and C. Richard, the REMORA Methodology for Information System Design and Management. In: T.W. Olle, H.G. Sol and A.A. Verrijn Stuart (Eds.), Information Systems Design Methodologies: A Comparative Review. Proceedings of the CRIS 82 conference, North-Holland, Amsterdam, pp. 369-426, 1982.

[Ross 77] Ross, D.T. and K.E. Schoman, jr., Structured Analysis for Requirements Definition. IEEE Transactions on Software Engineering, SE-3, pp. 6-15, January 1977.

[Scheschonk 84] Scheschonk, G., Petri-nets as formal basis for information systems (in German: Petri-Netze als formale Basis für Informationsysteme). Lecture notes, Berlin University of Technology, 1984.

[Schiel 89] Schiel, U. and I. Mistrik, OKAY - Object-Oriented Knowledge Analysis and Design. Paper presented at IFIP WG 8.1 Workshop on Object-oriented approaches in information systems design, Sesimbra, Portugal, June 1989.

[SDW 88] The System Development Workbench, Pandata, March 1988.

[Seligmann 89] Seligmann, P.S., G.M. Wijers and H.G. Sol, Analyzing the Structure of I.S. Methodologies: an Alternative Approach. In: Proceedings of the First Dutch Conference on Information Systems, R. Maes (Ed.), Amersfoort, November 1989.

[Shipman 81] Shipman, D., The Functional Data Model and the data language DAPLEX. ACM Transactions on Database Systems, vol. 6, nr. 1, pp. 140-173, 1981.

[Shoval 88] Shoval, P. and N. Pliskin, Structured prototyping: Integrating Prototyping into Structured System Development. Information and Management, vol. 14, nr. 1, pp. 19-30, 1988.

[Simovic 89] Simovic, D.A. and D.C. Stefanescu, Formal semantics for database schemas. Information Systems, vol. 14, nr. 1, pp. 65-77, 1989.

[Smith 88] Smith, J.B. and S.F. Weiss, Hypertext, Communications of the ACM, Special Issue on Hypertext, vol. 31, nr. 7, pp. 816-819, July 1988.

[Spivey 88] Spivey, J.M., Understanding Z, a specification language and its formal semantics. Cambridge Tracts in Theoretical Computer Science, nr. 3, 1988.

[Teorey 86] Teorey, T.J., D. Yang and J.P. Fry, A logical design methodology for relational databases using the extended entity-relationship model. ACM Computing Surveys, vol. 18, nr. 2, pp. 197-222, 1986.

[Ter Hofstede 89a] Ter Hofstede, A.H.M. and S. Brinkkemper, Conceptual Task Modelling. Technical Report nr. 89-14, Department of Information Systems, University of Nijmegen, September 1989.

[Ter Hofstede 89b] Ter Hofstede, A.H.M., T.F. Verhoef, G.M. Wijers and S. Brinkkemper, Expert-based support of Information Modelling: a Survey. Technical Report RP/soc-89/7, Software Engineering Research Centrum, October 1989. To appear in: Proceedings of the Workshop on The Next Generation of CASE-tools, S. Brinkkemper and G.M. Wijers (Eds.), Noordwijkerhout, NL, 8 - 11 April 1990.

[Tsichritzis 78] Tsichritzis, D.C. and A. Klug (Eds.), The ANSI/X3/SPARC DBMS Framework: Report of the Study Group on Data Base Management Systems. Information Systems, vol. 3, 1978.

[Turner 87] Turner, W.S., R.P. Langerhorst, G.E. Hice, H.B. Eilers and A.A. Uijtenbroek, System Development Methodology (SDM II). North-Holland and Pandata, 1987.

[van Hee 88] van Hee, K.M., G.J. Houben, L.J. Somers and M. Voorhoeve, Executable Specifications for Information Systems, Computing Science Notes, nr. 88/05, Department of Computing Science, Eindhoven University of Technology, March 1988.

[Verheijen 82] Verheijen, G.M.A. and J. van Bekkum, NIAM: an Information Analysis Method. In: T.W. Olle, H.G. Sol and A.A. Verrijn Stuart (Eds.), Information Systems Design Methodologies: A Comparative Review. Proceedings of the CRIS 82 conference, North-Holland, Amsterdam, pp. 537-590, 1982.

[Vermeir 82] Vermeir, D. and G.M. Nijssen, A procedure to define the object type structure of a conceptual schema. Information Systems, vol. 7, nr.4, pp. 329-336, 1982.

[Verrijn Stuart 87] Verrijn Stuart, A.A., Equivalence conditions for information systems representations. Report 87-09A, Department of Computer Science, University of Leiden, 1987.

[Vonk 88] Vonk, R., Analyst workbenches: a reference framework (in Dutch: Analyse Workbenches: een referentiekader). Informatie, vol. 30, nr. 1, pp. 17-32, January 1988.

[Wieringa 88] Wieringa, R. and R. van de Riet, Algebraic Specification of Object Dynamics in Knowledge Base Domains. In: The Role of Artificial Intelligence in Databases and Informations Systems, Joint TC2-TC8 Working Conference, Guangzhou, China, pp. 346-371, 1988.

[Wieringa 89] Wieringa, R., Three roles of Conceptual Models in Information System Design and Use. In: Proceedings of the Conference on Information Systems Concepts, E.D. Falkenberg and P. Lindgreen (Eds.), Namur, October 1989.

[Wijers 89] Wijers, G.M. and H. Heijes, Automated Support of the Modelling Process: a view based on experiments with expert information engineers. Technical Report, Delft University of Technology, December 1989.

[Wileden 86] Wileden, J.C. and M. Dowson (Eds.), Software Processes and Software Environments, Proceedings of an International Workshop, Coto de Caza, SIGSOFT Software Engineering Notes, vol. 11, nr. 4, 1986.

[Wintraecken 85] Wintraecken, J.J.V.R., Information Analysis according to NIAM - Theory and Practice (In Dutch: Informatie-analyse volgens NIAM - in theorie en praktijk)., Academic Service, Den Haag, 1985.

[Wirth 76] Wirth, N., Algorithms + Data Structures = Programs. Prentice Hall, Englewood Cliffs, 1976

[Yourdon 79] Yourdon, E. and L. Constantine, Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design. Prentice Hall, Englewood Cliffs, 1979.

SUMMARY

This research addresses the various types of modelling as they are applied during the development of information systems. The many notational conventions in which modelling can be expressed, are hardly ever accompanied by stepwise guide-lines for constructing a model. The combination of a notation and a procedure in a modelling technique is one of the starting points of this dissertation.

First, we position modelling of information systems in the framework of the systems development cycle. Then the roles of the several persons involved in modelling are discussed. Incorporation in methods and techniques, and the support of modelling by tools are defined. The model triangle offers a philosophical perspective for some types of modelling that occur in the development of information systems.

By superpositioning of the model triangle, we obtain a basis for meta-modelling in a simple way. Meta-modelling is defined as the modelling of a modelling technique. Although mostly not mentioned explicitly, is meta-modelling often applied in various research at the moment. Some examples thereof as well as the techniques for meta-modelling are discussed. The stepwise guide-lines for modelling are called modelling procedures. The requirements of such procedures and their formulation is elaborated in general terms. Underlying is the need for formalisation, i.e. the description in a mathematical systems. We use predicate logic for this purpose because this is a simple and adequate formalism.

In the chapters 3, 4 and 5 the modelling procedures for events, activities, data and tasks are given. We start in chapter 3 with a classification of events in and around an information systems, which gives rise to concrete starting points for modelling in complex organisations. External events are used to model the context diagram, which denotes the interaction of the system with its environment. The processes within this system are modelled as activities, provided their level of abstraction is sufficient. We give a procedure that starts with descriptions of activities in natural language and results in a hierarchy of activity models. Some consequences of the formalisation of activity modelling are presented.

Chapter 4 gives a short formalisation of the Entity-Relationship technique and its modelling procedure. The synthesis of the use of this technique and the NIAM technique in one systems development project is discussed.

The concept of task, a type of process, plays a central role in the presentation of a new technique for task modelling in chapter 5, which we have called Conceptual Task Modelling (CTM). This technique is based on three existing techniques: NIAM, RIDL en Predicate-Transition nets. An extensive formalisation of the technique and of the relationship of task models with activity models and data models is given. The use of the CTM is illustrated with a modelling procedure and a CASE-tool for the technique. This chapter ends with a discussion of further research of this topic.

The support of modelling by means of tools is addressed in chapter 6. In addition to the determination of some terminology, we discuss the incorporation of supported techniques in systems development methods. The support of the transfer of the one technique to the other in a CASE-tool is called modelling transparency. We introduce also a notion for the relation between a method and a tool: method-companionship. Finally, we formulate and formalise the idea of layered modelling in the case of complex techniques.

SAMENVATTING

Dit onderzoek richt zich op de verschillende vormen van modelleren zoals dit wordt toegepast tijdens het ontwikkelen van informatiesystemen. De vele conventies om modellen in uit te drukken worden vrijwel nooit vergezeld van stapsgewijze aanwijzingen om een model te construeren. Het samengaan van een notatie en een procedure in een modelleringstechniek is een van de uitgangspunten van dit proefschrift.

Eerst plaatsen we modellering van informatiesystemen in het kader van de systeemontwikkelingscyclus. Daarna worden de rollen van de verschillende personen betrokken bij modellering besproken. Inpassing in methoden en technieken, en ondersteuning door tools wordt vervolgens gedefiniëerd. De modellendriehoek biedt een filosofisch perspectief voor enkele typen van modellering die van toepassing zijn bij het ontwikkelen van informatiesystemen.

Door superpositionering van de modellendriehoek verkrijgen we op simpele wijze een basis voor meta-modellering, dat wil zeggen het modelleren van een modelleringstechniek. Hoewel meestal niet expliciet vernoemd, wordt meta-modellering tegenwoordig in allerlei onderzoek toegepast. Enige resultaten van zulk onderzoek alsmede de technieken voor meta-modellering worden besproken. De stapsgewijze aanwijzingen voor modellering hebben we modelleringsprocedures genoemd. De eisen aan en de constructie van zulke procedures wordt in algemene termen uitgewerkt. Achterliggend is de noodzaak tot formalisatie, ofwel de beschrijving in wiskundige systemen. We gebruiken hier de predicaten logica, omdat dat een simpel en adequaat formalisme is.

In de hoofdstukken 3, 4 en 5 worden vervolgens de modelleringsprocedures voor gebeurtenissen, activiteiten, gegevens en taken uitgewerkt. We bespreken eerst in hoofdstuk 3 een classificatie van gebeurtenissen in en om een informatiesysteem, die aanleiding geeft tot concrete aangrijpingspunten voor modellering in complexe organisaties. Externe gebeurtenissen worden gebruikt om het context diagram te modelleren, dat de interactie van het systeem onder beschouwing met zijn omgeving weergeeft. De processen binnen dit systeem worden, voorzover zij van een voldoende abstractieniveau zijn, gemodelleerd als activiteiten. We geven de

procedure die uitgaat van beschrijvingen van activiteiten in natuurlijke taal en resulteert een hiërarchie van activiteitenmodellen. Enige consequenties van de formalisatie van activiteitenmodellering worden gepresenteerd.

In hoofdstuk 4 wordt uitgegaan van de concepten van de Entiteit-Relatie-modelleringstechniek. Een beknopte formalisatie van deze techniek en een gegevensmodelleringsprocedure worden gegeven. Er wordt tevens kort ingegaan op het gezamenlijk gebruik van deze techniek met de NIAM-techniek in een systeemontwikkelingsproject.

Het concept van taak, een soort proces, staat centraal in de presentatie van een nieuwe techniek voor taakmodellering in hoofdstuk 5, die we Conceptuele Taakmodellering (CTM) hebben genoemd. Deze techniek is gebaseerd op drie bestaande technieken: NIAM, RIDL en Predicaat-Transitienetwerken. Een uitgebreide formalisatie van deze techniek zelf en die van de relatie van taakmodellen met activiteitenmodellen en datamodellen wordt gegeven. Het gebruik van CTM wordt toegelicht aan de hand van een modelleringsprocedure en een CASE-tool voor de techniek. Dit hoofdstuk eindigt met een bespreking van mogelijk onderzoek op dit gebied.

In hoofdstuk 6 wordt ingegaan op de ondersteuning van modellering door middel van hulpmiddelen. Naast de vaststelling van enige terminologie op dit gebied, bespreken we de inpassing in systeemontwikkelingsmethoden van technieken die ondersteund worden door automatische hulpmiddelen. De ondersteuning van de overgang van de ene techniek naar de andere in een CASE-tool noemen we modeleringstransparantie. We introduceren tevens een begrip voor de relatie tussen een methode en een tool: method-companionship. Tenslotte formuleren en formaliseren we het idee van gelaagd modelleren in het geval van complexe technieken.

CURRICULUM VITAE

The author of this thesis was born on January 18th, 1958 in Monnickendam. In 1976 he obtained the V.W.O. diploma at the Waterlant College in Amsterdam-Noord and started to study Mathematics at the University of Amsterdam. After having obtained the Bachelor's degree, he moved to Nijmegen in 1980. At the University of Nijmegen the Master's degree in Mathematics and Informatics was received in 1984. From then on he was employeded as an assistant professor in the department of Informatics of the University of Nijmegen. He was a member of the Programming Languages and Compilers group (head: prof. C.H.A. Koster) and in 1986 he became member of the newly established Information Systems group (head: prof.dr. E.D. Falkenberg). It was in this group that the research which lead to this thesis was performed.
In 1989 he was appointed as a member of the Working Group WG8.1 'Design and Evaluation of Information Systems' of the IFIP Technical Committee TC8 'Information Systems'. The author is also a member of the research staff of the SOCRATES project of the Software Engineering Research Centre in Utrecht.

During the academic year of 90/91 the address of the author is:

Sjaak Brinkkemper
Department of Management Sciences and Information Systems
College and Graduate School of Business
CBA 5.202
University of Texas at Austin
Austin, TX 78794, USA
Email: sjbr@emx.utexas.edu

Stellingen
horende bij het het proefschrift
Formalisation of Information Systems Modelling
van
Sjaak Brinkkemper

1. De praktijk van informatiesysteemontwikkeling heeft meer baat bij formalisatie van visuele technieken dan bij visualisatie van formele technieken.
   Harel, D., On Visual Formalisms. Communications of the ACM, vol. 31, nr. 5, pp. 514-530, 1988.

2. Modellering is slechts gedeeltelijk te formaliseren.

3. Het is te verwachten dat aan de veelheid van tot dusverre geformuleerde stromingen van technieken voor datamodellering een even zo groot aantal nieuwe stromingen zal worden toegevoegd. Er zal nog vele jaren genoten kunnen worden van de hiermee gepaard gaande fundamentalistische twisten.

4. Voor een persoon die informatie verstrekt aan systeemontwikkelaars is informant een meer geschikte benaming dan de ingeburgerde term gebruiker.

5. Net zoals in het Engels het gebruik van de term 'methodology' in de betekenis van methode vermeden moet worden, dient de term 'methodiek' in het Nederlands vermeden te worden.

6. Assistenten en onderzoekers in opleiding dienen tijdens die opleiding te leren hoe zij een wetenschappelijk artikel in hun vakgebied kunnen schrijven.

7. Het onderscheid tussen Amerikaans-Engels en Brits-Engels is verwarrend bij het publiceren in het Engels.
McNab, S. M., Hoe een wetenschappelijk artikel te publiceren in het Engels. NRC Handelsblad, 16 januari 1990.

8. Er is in Nederland behoefte aan een onafhankelijk expertise-centrum op het gebied van methoden, technieken en hulpmiddelen voor de systeemontwikkeling. Het is dan ten zeerste gewenst, dat vanuit zo'n centrum een grootschalig onderzoek naar het praktisch effect van het gebruik van deze methoden, technieken en hulpmiddelen uitgevoerd wordt.

9. Meteorologische uitspraken op basis van geografische iso-lijnen, zoals: "Boven de lijn Amsterdam - Nijmegen moet rekening gehouden worden met ijzel en natte sneeuw", dienen geïnterpreteerd te worden met kennis van foutschattingen horende bij het genereren van contourplottekeningen.
Brinkkemper, S. en H. Hendriks, A New Algorithm for Contour-plotting. Proceedings of the Eurographics '87 Conference, Ed. G. Maréchal, p. 513-527, Amsterdam, August 1987.

10. Wetenschappelijk onderzoekers worden gemotiveerd door hobbyisme en ijdelheid.