TransCoop

ESPRIT Basic Research Action 8012

TRANSCOOP DELIVERABLE II.1

DATE OF ISSUE	January 17, 1995
EDITOR	Peter Verkoulen, Thomas Tesch
ORIGIN	Universiteit Twente
STATUS	final
CONFIDENTIALITY	USE WITHIN TRANSCOOP
RELATED ITEMS	TRANSCOOP Technical Annex, TC/REP/GMD/D2-2/207
FILING CODE	TC/REP/UT/D2-1/014
ABSTRACT	This document describes the requirements for the TRANSCOOP specification language as they have been encountered in the selected applications (Cooperative Document Authoring, Design for Manufacturing, Workflow).
KEYWORDS	Specification Language Requirements, Cooperative Document Authoring, Design for Manufacturing, Workflow
APPROVED BY	TMG
DISTRIBUTION	GMD,UT,VTT

DOCUMENT HISTORY

version	date	reason

Contents

Contributions to this Deliverable

Besides the editors of this deliverable (Peter Verkoulen & Thomas Tesch) a number of people from all partners have contributed to this document. In the table below we indicate as accurate as possible the contribution of each author

name	*institution*	sections contributed to
Frans Faase	UT	1.4, 4.1, 4.4, 4.5.1, 4.6, 6.1, 6.2
Jari Juopperi	VTT	5.3.4
Aarno Lehtola	VTT	5.3.3
Tor Lillqvist	VTT	5.2
Ville Mikkonen	VTT	5.3.4
Pieter Oude Egberink	UT	4.2, 4.3, 4.5
Olli Pihlajama	VTT	5.2
Juha Puustjärvi	UH	5.1.2
Bart Selders	UT	4.1, 4.2, 4.3, 4.4, 4.5, 4.6
Aija Sladek	VTT	5.1, 5.3.1, 5.3.2, 5.4
Thomas Tesch	GMD	1.5, 3.1, 3.4, 3.5, 7
Jari Veijalainen	VTT	5.3.3, 5.3.4, 7
Peter Verkoulen	UT	1.1, 1.2, 1.3, 2.1, 2.2, 2.3, 4.1, 4.4, 4.6, 6.1, 6.2, 7
Jürgen Wäsch	GMD	3.2, 3.3
Antoni Wolski	VTT	5.3.1, 5.3.2

Acknowledgements

We thank Philips Medical Systems (Best, the Netherlands) and Fokker Aircraft B.V. (Amsterdam, the Netherlands) for making available the necessary information, giving us the possibility to perform our case studies.

We are grateful to the Finnish Telecom for releasing information concerning the 101 service. Special thanks go to Dr. Olli Martikainen, Vice-President (R&D) of the company, who has especially supported this project, and Mr. Panu Hämäläinen and Mr. Atte Nikkanen for interesting discussions.

We also thank the PortNet consortium for mediating information about the system in numerous discussions. We are especially grateful to Ms. Ina Mickelsson and Mrs. Eija Berg for the discussions about the system.

We further thank Mr. Esko Uotila, Marketing Manager of ICL Personal Systems Oy, for giving us all support necessary to analyse and test the ICL TeamFlow workflow management product.

Last but not least, we thank Mr. Håkan West, Vice-President (Systems & Procedures) of Valmet Corporation, and Mr. Tuomo Raitanen from the same company for providing us with an interesting test case to analyse the Valmet Help Desk System. They also put considerable effort in explaining the system and its basic goals to us.

Chapter 1

Introduction

1.1 Introductory Remarks

You are currently reading one of the two deliverables of TRANSCOOP Workpackage 2: *the Analysis of Selected Applications*. This deliverable describes an analysis of several cooperative application scenarios and gives requirements that have been derived for the TRANSCOOP specification language from this analysis. The other deliverable in this workpackage contains similar information for the transaction model . It won't surprise you that there is a lot of overlap between these deliverables. The general chapters, like the introduction and the descriptions of the application areas and the case studies are the same for both deliverables. Only the sections containing the results of the application analysis (like the language requirements that have been found) differ. We have chosen this approach to keep each of the deliverables fully self-contained.

One additional remark has to be made on beforehand. What we have been doing in this workpackage is in fact *requirements engineering*: by analysing some application areas, we have tried to come up with requirements for the TRANSCOOP specification language and transaction model. Like design as presented in the chapter about Design for Manufacturing, requirements engineering is a continuous process. Therefore, one cannot expect this document to contain a complete "design" of the specification language by giving a full list of detailed requirements. In fact, the requirements in this document are still rather abstract and general. During the process of language design, the requirements will also be refined and clarified. So, language design and making the requirements more concrete will go hand in hand. There will be a continuous interaction between the language design and the requirements. In the end, we will have to validate whether the specification language meets the requirements that have been formulated. As part of the work around the TRANSCOOP demonstrator, we will specifically have to show that the specification language is suited for modelling the applications that have been analysed in this workpackage.

1.1.1 Structure of the Deliverable

The structure of this deliverable is as follows.

The chapter that you are currently reading contains the introduction to this workpackage and a very superficial description of the TRANSCOOP project. Also, it represents the most important terminology that has been used in our work. Finally, it contains a brief overview of the application domains that have been studied in this workpackage and their interrelationships.

The second chapter contains a kind of taxonomy that has been used when doing our research. When coming up with requirements for the TRANSCOOP specification language , we wanted to have some common understanding on beforehand in which direction to look for the requirements. Of course, it is difficult to make this chapter very concrete, because that would mean that we would have answers already on the questions we were going to pose.

Chapters 3-5 describe the application areas that have been studied in order to come up with requirements. Chapter 3 deals with the analysis of the Cooperative Document Authoring (CDA) area, which has been performed by GMD. Chapter 4 presents the analysis of the Design for Manufacturing (DfM) application area, which has been performed by UT. Finally, VTT has investigated WorkFlow management (WF), which has been reported in Chapter 5.

Chapter 6 is a summarising chapter, which tries to generalise the requirements that have been found in the three application areas. W describe the similarities and differences between the three application areas. Also, we present a list of generalised requirements. In order to obtain all requirements of all application scenario studies, the interested reader should refer to the concluding sections of the three application chapters (sections 3.5, 4.6 and 5.4).

Chapter 7 concludes this deliverable by superficially repeating the research questions that have been answered in this workpackage.

1.2 The TRANSCOOP project

Today's software tools mostly aim at supporting a single user only. For example, consider CAD-tools, software development environments or text editors. E.g., there are excellent text editors available, but they do not appropriately support multiple authors to work on the same document concurrently. This problem divides the cooperative authoring process into two steps: (1) Authors negotiate without tool support; (2) Authors work on the next update or version of the document on their own. These steps are iterated until the authoring process has finished. Merging the steps can improve speed and quality of the authoring process significantly. Software tools supporting cooperative work in such a way are needed and will play a crucial role in innovative solutions for CSCW applications. Tools available today mainly support cooperative tasks on a very superficial level, e.g., by coordinating multiple users at the level of a window system, e.g., X-Windows, without having any understanding of the data with which multiple cooperating users are working simultaneously.

A major conceptual problem in this framework is to ensure consistency criteria for the data concurrently processed by multiple users. Conventional database technology already provides mechanisms to absolutely guarantee consistency constraints by controlling the concurrent access of different users to shared data. Unfortunately, existing transaction

management concepts are not suitable for supporting and controlling cooperation between users, because they are designed to fully isolate users from each other. But, on the other hand, without providing system-controlled access to shared data, a tool for cooperative work will also be of little value because there is no way to automatically avoid inconsistencies, and ensuring consistency constraints will be up to the users.

Another serious problem is the design of cooperative systems itself, which is a very complex and error-prone task, involving several formalisms and techniques that have up to now hardly been studied in combination. Formal techniques well-known from the field of designing non-distributed information systems as well as some promising approaches accomplished in the field of open, distributed information systems have been accepted by a large user community within each of these fields. But the design of cooperative systems includes the description and formal specification of cooperative activities. That is, formal techniques are needed to capture the notion of cooperation at the design level and to allow for the specification of cooperative tasks. In addition, such specifications have to be mapped to a cooperative transaction model in order to enable appropriate database management support at run time.

Cooperative transactions are thus intrinsically complex operations that are difficult to understand completely. A theory is needed for distinct notions of correctness, for simulation models, for the generation of tests, for compiler building and report generation, for correctness-preserving transformations that allow the designer to map specifications onto implementation platforms in a guided way. The whole process needs to be supported by a coherent set of tools.

1.3 The role of WP2 in the project

According to the Technical Annex, the goal of WP2 is "to identify the requirements for a cooperative transaction model and an accompanying specification language in selected types of cooperative environments and the characteristic differences to existing transaction models." We do this by analysing some cooperative application scenarios. We have chosen to perform this analysis to have a better basis for the language and the transaction model design. Only applying "common sense" would imply the risk of coming up with something that cannot be really used or that does not fulfill the requirements from practise.

The application areas that we have studied are Cooperative Document Authoring (CDA), Design for Manufacturing (DfM) and Workflow (WF). This choice more or less covers the broad spectrum of application areas in the context of computer-supported cooperative work, as will be explained in one of the sections below.

We have decided to assign a specific application area to each partner: GMD has investigated CDA, UT has studied DfM and VTT has looked at WF. This choice was based on the specific expertise of each partner.

We had a taxonomy in mind which would guide all the research in the three application areas into one direction. Because of the different researchers studying very different application

areas in different ways, this has not worked out completely. However, we do present a common framework that has been used in the analysis process below. Although we have not been able to stick to this framework very closely, we still think that it has raised some interesting questions that have been answered in some or all of the application analyses.

This document contains the requirements as we have found them in the three application areas. However, this is a snapshot. Moreover, the requirements are relatively abstract as yet. During the process of designing the TRANSCOOP specification language and transaction model, the designers will have to keep the requirements in mind but also evaluate them. Just like in an ordinary design process, the language and model requirements may be refined during the process.

Finally, when working on the TRANSCOOP demonstrator, we also will have to perform some validation of the specification language and the transaction model. This means that we will have to look again at the cases that have been studied in WP2 and check whether the language and the transaction model meet the requirements that had been set by those case studies. If all goes well, the answer to this question should be positive.

1.4 Important terminology

The purpose of this section is to provide definitions for the most important terminology used in this report. The terminology will be defined using basic notions which are used in their 'conventional' meaning, i.e., not particulary limited to elements in information systems. These basic notions together with their meaning being used here, are:

Cooperate to work or act together toward a common end or purpose.

Event something that happens; something that occurs.

Task an (assigned) piece of work to be done (within a certain time).

Agent one that acts.

Actor the performer of an activity.

User one that uses a certain system or tool.

Some of these basic notions will be defined more specifically as part of our terminology.

The application domains studied in this report are all part of the *Computer Supported Cooperative Work* (CSCW) application area. This area is characterised by a high degree of cooperation between users in achieving a common goal or task, where information systems are used for interactions between users and for sharing information. The applications used in this area are typically referred to as *Groupware*. The following application areas within the CSCW application area are studied in this report: Cooperative Document Authoring (CDA), Design for Manufacturing (DfM) and WorkFlow management (WF). These application areas

are described in more detail in Section 1.5. In this section only the domain independent terminology will be given; domain-specific terminology is given in the introductions to the application domains.

A *CSCW system* is an information system that provides an integrated environment to support the work of a group of people engaged in a common task or goal. Following a common goal or task is an essential characteristic for defining cooperative work. Co-workers should be assisted in communicating, in cooperating and in coordinating their activities.

To develop CSCW applications, there is a need for a *specification language* to specify the tasks to be performed by the application, and a *transaction model* to support the execution of these tasks in an orderly manner. The specification language will be defined in more detail in Section 1.4.7, after all other terminology has been given.

In the context of the system versus the applications, the *system* is that part of the computerised environment which is designed once by the system designers. It is generic and thus supports many applications. *Applications* are those parts of the computerised environment which are specified by the application designers to support the execution of specific tasks which are to be performed by the end-users. The system, used in this sense, is related to the implementation of transaction model of a CSCW system. The specification language shall be used by the application designers.

1.4.1 Agents, users and actors

An agent, being someone (or something) that acts, can be classified according to its abilities. A *social agent* is conscious of its ability of interaction. A *technical agent* is program-driven.

A *user* of a system corresponds (more specifically) to a human being[1] that makes use of the system. A user is a social agent. A *user definition* defines the properties of the users: the abilities (privileges), the group(s) a user belongs to and roles that a user may play. A *user group* is a group that gives certain abilities and rights to the users that belong to it. *(User) roles* are categorisations of users. The categorisations may be, e.g. job titles (a secretary or a sales manager).

An *actor* may be an individual user, a user role, an organisational unit or an application program. Here an organisational unit equals a group.

1.4.2 Activities

An *activity* is a set of events that occur under the responsibility of a single actor. Intuitively we can say an activity is equal to "the work to be done by the actor". Activities, in this

[1] For convenient reading, we will use "he" and "his" whenever needed. This should be read as "he or she" and "his or her" respectively.

sense, include activities done within (or with the help of) an information system, but also those done without (or outside of) an information system. It is possible that a certain activity can be divided into subactivities. Activities can also be grouped according to the purpose for which they are performed. The activities of a task are those activities that have to be performed to complete the task. *Database activities* are operations that are being performed either to change data or to move (retrieve) data.

1.4.3 Scenarios

Scenarios are used to control the execution of certain activities according to some predefined rules (possibly with different parameters). Scenarios have to be restricted to the computerised parts of the activities, and are only useful if the activity can be divided into a number of distinct subactivities. We note that a scenario can have certain transactional properties, but that they usually also cover aspects (e.g., authorisation) which are not relevant for transactions. The scenarios are defined by *scenario definitions* given in a *scenario specification language*. A *scenario definition* is a description of a cooperation pattern, expressed in a specification language. The scenario definition may be hierarchical, i.e. consist of lower-level scenario definitions. Otherwise the definition consists of definitions of activity types together with the definitions of data and control flow (i.e. data and ordering dependencies) between them. The definitions include also any conditions (temporal and other) set for the activity or scenario execution. A *scenario instance* is a uniquely identified instantiation of a defined scenario which is used to execute a specific, concrete activity according to the scenario definition.

In WF applications, the activities of a scenario usually have to be executed in a strict order, whereas in CDA, there are fewer restrictions and the order of execution is mainly determined by the users.

When an activity of a scenario itself is performed according to a scenario, we talk about *nested* (or *hierarchical*) *scenarios*. Such an activity is an activity of a scenario on one level, and (at the same time) could be considered as an instance of a scenario on a lower level.

A *global referee* is an agent of a scenario instance that has certain rights over the other agents to force certain decisions. *Autonomy of agents* is the degree to which the agents can make decisions on their own.

Scenario management is controlling the execution of the (instances of the) scenarios within an application. The *audit trail* of a scenario instance contains the information of the operations performed by the scenario so far. It is the role of a *scenario designer* to create the scenario definitions.

1.4.4 Data

Data in database applications can be divided into volatile and permanent data. *Volatile data* are data that are temporarily held in the workspace of an agent. *Permanent data* are stored in

some (shared) persistent storage. *Shared data* are accessible from (and updatable by) more than one agent. Shared data can either be replicated or stored in a single source. *Replicated* means that the data structures in question are stored in two or more component systems and the different instances are kept identical to each other when updates occur (with a possible delay). *Single source* means the data structures are stored at a central place and queried or updated by the distributed system parts.

1.4.5 Databases

A *database* can be defined as a persistent data storage system in which data is stored according to a database schema and that can be accessed through a well-defined interface. A *database schema* is the definition of the structure of a database consisting of: data definitions, operation definitions and constraint definitions. *Database operations* are operations that can be performed on a database, such as: query operations and update operations. *Database constraints* are constraints that are put on the data in the database. *Database consistency*, first of all, means that the results of database queries are correct with respect to the update methods that have been applied. (This is called update consistency.) This means that data do not get corrupted. Whenever database constraints are defined, database consistency would (normally) require that the database state is consistent with respect to the database constraints.

1.4.6 Transactions

A *transaction* is a collection of database operations (formed during the execution of the application) that should be executed as a unit, such that database consistency is preserved (not necessarily having the traditional properties of a transaction). A transaction can be distributed, cooperative, nested and/or have some (or all) of the ACID-properties. An *ACID-transaction* is a transaction with the following four properties: atomicity, consistency, isolation and durability. The following definitions of these properties are based on [Gra81]. *Atomicity* refers to the fact that all operations of a transaction must be treated as a single unit; hence, either all the effects of the operations are applied to the database state, or none. (This is also called the all-or-nothing property). A transaction is *consistent* if it is a correct transformation of the database state, i.e. the actions taken as a group do not violate any of the integrity constraints associated with the corresponding database state. This requires that the transaction program is correct with respect to all semantical requirements of the universe of discourse. *Isolation* requires that (although transactions might be executed concurrently) it appears to each transaction that results of any other transaction are either applied (completely) to the database before the transaction starts or are only applied to the database after the results of this transaction have been applied (completely). *Durability* requires the results of a committed transaction to be made permanent in the database in spite of failures.

A *distributed transaction* is a transaction that is using more than one database. A *cooperative transaction* is a transaction that involves some cooperating agents, that coordinates these

agents in some (application-specific) way and that guarantees some predefined (application-specific) properties.

Transaction management takes care for the concurrent execution of transactions, such that the required transactional properties are guaranteed.

1.4.7 Specification language

A specification language is meant to describe the relevant aspects of a system, abstracting away from aspects that are related to a particular implementation. A good specification language guards against over-specification (e.g., giving unnecessary restrictions) and under-specification (e.g., allowing ambiguities). Generally, there will not be a unique mapping from a specification to an implementation that realises the specification.

The TRANSCOOP specification language is intended for the specification of the cooperative scenarios that are found in CSCW applications. To be more precise, it is intended to describe the application-dependent aspects, and should abstract from the purely transactional aspects supported by the system. It will thus not specify the transaction model that is used by the system.

This specification language will be based on TM and LOTOS. LOTOS can be used to specify the ordering of interaction events occurring at the interface[2] of a CSCW application with its users. With TM it is possible to specify database schemas. The expressions found in TM can also be used to specify data that are transferred in an event. Because TM incorporates a full strictly-typed functional language, with predicative sets, it is powerful enough to specify any kind of computation.

1.5 Overview of the application domain

This section gives an overview of the investigated application domains which are *Cooperative Document Authoring*, *Design for Manufacturing* and *Workflow*–Applications. By analyzing these different application domains we will to derive requirements that are valid for a wide spectrum of cooperative applications.

Database management systems (DBMS) technology provides data modelling and transaction management facilities suited to model buisiness processes. These facilities are normally not appropriate for cooperative application scenarios. The transaction models of such DBMS are based on isolation of users thus giving each user the impression to use the system alone.

Most of the proposed extended transaction models and their corresponding specification facilities are developed for one concrete application scenario. This has lead to several models for specific application domains that usually provide little flexibility in adapting

[2] This does not include the form and appearance of the user-interface.

the transaction semantics to the needs of other application domains. The objective of the TRANSCOOP approach is to develop a transaction model that overcomes this weakness. The model should support a wide variety of application domains in an adaptive and flexible manner.

All investigated application scenarios belong to the category of *Computer Supported Co-operative Work (CSCW)*. CSCW refers generally to any collaboration between users where computer systems are used for interactions between users and for sharing information. The key areas in the CSCW field are *communication*, *collaboration* and *coordination* [EGR91]. These issues are briefly explained in the following.

Computer technology has led to new forms of *communication* between people like electronic mail systems or bulletin boards. These typically asynchronous communication facilities support information exchange between co-workers using the computer systems and networks as communication medium. New developments in computer technology enable also synchronous communication, such as voice connections and video conferences.

Collaboration is based on the *sharing of information* between co-workers. This sharing of information between people is usually realized by using a common file system or a database management system. Unfortunately, file systems usually provide primitive synchronization and no failure handling mechanisms. On the other hand, database management systems usually isolate users from each other – the users can not interact with each other in a cooperative way.

To reduce communication and collaboration efforts in group activities and for strengthening their effectiveness, *coordination* of activities within a group of co-workers is needed. Without coordination, e.g. a team of co-designers in a DfM environment would often engage in conflicts and repetitive actions.

The objective of CSCW systems [3] is to provide an integrated environment to support the work of a group engaged in a common task or goal. Co-workers should be assisted in communicating, in collaborating and in coordinating their activities.

In contrast to transaction models, CSCW systems do not address concurrency and recovery issues to ensure fault-tolerant processing. The objective of TRANSCOOP is the development of a transaction model that supports the key requirements of the investigated application domains and, in contrast to CSCW systems, provides transactional properties.

In the following the investigated application domains are shortly introduced.

The cooperative authoring scenario, as it is investigated in chapter 3, is characterized by multiple authors working on a common (hyper)document simultaneously. The objective of the investigated authoring process is the creation of the artifact 'hyperdocument'. The authoring process is considered as a design problem solving process [HF86, THS90, HT93]. The decomposition into smaller subproblems and their solution by interacting activities gives the ability to solve the overall design problem. A characteristic of design problem

[3] The term *groupware* is often used almost synonymously with CSCW systems.

solving processes is that the aimed product (artifact) can be described only vaguely on beforehand. The same observation holds for the process of creating a hyperdocument: the sequence of activities that has to be performed is not predictable. Thus, authoring activities require high flexibility for choosing the next actions to end up with the aimed document. Beside the required flexibility, a cooperative authoring environment has to provide facilities that support the design problem solving process itsself, e.g. the ability to externalize the mental representation of knowledge structures of the participating authors. In cooperative document authoring the cooperation and communication properties of CSCW are hardly needed while coordination is only meaningful to the described extent.

The scope of Design for Manufacturing is the engineering of discrete complex industrial artifacts [Ide93, UR93]. The engineering process is usually seperated into upstream processes (product design) and downstream processes (engineering, planning, manufacturing). The essential part of DfM is the early involvement of specialists from downstream processes like production engineering and manufacturing in the upstream design process. Thus, the strengthening of the design process by overlapping design phases requires extensive communication, cooperation and coordination facilities. In comparison to the cooperative authoring process the different phases of the overall process are as well-known as the sequence of processing. Nevertheless DfM environments require the refinement of already performed tasks. The given structure of the DfM process helps to coordinate activities in advance and thus, compared with CDA, reduces the need for communication and cooperation.

Workflows are used to define complicated buisiness processes, e.g., collection of activities organized to facilitate or accomplish the production of goods or services. A workflow consists of a collection of tasks that are partially ordered. Tasks are the basic untis of work that are processed by one responsible actor. The order of tasks defines their execution order and flow of data between them. In the simple case, tasks can be performed one after the other. This is usually not optimal since workflow tasks can be performed concurrently with other tasks, thereby reducing the elapsed time it takes to complete a given workflow.

A workflow specification describes control flow and data flow dependencies between such a collection of activitites. Thus workflows focus on the coordination of activities that are performed in a specific situation. Workflows can be considered as a simple type of CSCW systems that supports asynchronous distributed interactions.

The described scenarios have some common properties but they also have significant differences considering the underlying problem solving processes. All three scenarios can be classified as CSCW applications emphasizing the communication, cooperation and coordination properties of CSCW systems differently. Before further investigation of differences, we have identified the following common properties:

- The exchange of information between different users performing different activities is a basic feature of all three application scenarios.
- Multiple concurrent users involved in common tasks to satisfy a common goal.
- Control flow dependencies between these tasks that are derived from the application domain resp. the concrete problem solving process.

- The need for a consistent execution of concurrent tasks.

The importance of these common properties varies for the three application domains. In workflow systems is e.g. a heavy need for control flow dependencies while this is less important in the cooperative authoring domain. Thus we can assign these properties to each application domain with different weights.

The differences in the described application domains become more clear when considering the structure of problems that are to be solved. We can identify a spectrum from CDA over DfM to WF systems. The problem to be solved becomes more concrete or is more clearly defined respectively. In cooperative authoring systems the problems to be solved are not clearly defined. This is reflected by a problem definition which is marked by gaps that are to be filled. The understanding of those gaps is a significant part of the problem solving process. Thus, the analysis of the problem itsself if often the most important step for the solution.

Creative processes are characterized by the fact that there exists no such thing as an optimal solution. This is one reason for the duration of creative problem solving processes: because it is in most cases impossible to find an optimal solution the designers have to compromise a solution that is "good enough" [Sim81]. But there are no criteria to justify whether a solution is good enough. In other words: there is no predefined stopping rule. The process of e.g. writing a document or creating a new product is often stopped due to deadlines or money issues. The fact that those processes are lacking an optimal problem solution strategy and thus an optimal solution influences the problem solving process.

WF applications are placed on the other side of this spectrum. The problem solving process is clearly definable because the problem and how to derive a corresponding solution for the problem are well-known. In most cases there are stopping rules and a solution can be justified as true or false.

This overview shows that there are some common properties in the investiagted application domains but also significant differences. The objective of TRANSCOOP is to satisfy as far as possible the requirements of all application domains by identifying common cooperation primitives as basic building blocks. The vision is that the cooperation needs of most cooperative scenarios can be modelled by providing those basic cooperation primitives.

Chapter 2

Hints for the Application Analysis

This section contains the most important aspects that have been paid attention to when the three partners were analysing their application scenarios. These aspects seemed to be important on beforehand with respect to the design of the TRANSCOOP specification language and transaction model. It is useful to have some common understanding about this as it helps us to get cohesive results. To assure the quality of the analysis, we needed a general agreement on the main points of investigation. This section is meant as an overall guideline for the aspects that have been investigated in the application analysis. It does not serve as a tight, prohibitive structure that exactly told the partners what to do and what not.

As mentioned in the WP2 workplan (TC/INT/UT/WP2/002), we can distinguish three different kinds of properties when investigating cooperative scenarios/applications. These three areas can be considered as the main dimensions or categories of our analysis:

- User aspects
- Activity aspects
- Data aspects

In the initial taxonomy (TC/INT/UT/WP2/002) we distinguished the definition aspects and management aspects of the above categories. The difference is that the definition describes the allowed runtime behaviour in advance. The management aspect contains mechanisms that govern the execution considering the given definitions.

We suggest to focus on the definition parts because this is the knowledge that is given in a specification. At runtime, there will be some mechanisms that are based on the given information. The investigation of runtime mechanisms should not be the major part of our requirements.

The following sections deal with the separate analysis dimensions, as they were mentioned above.

2.1 Aspects of Users

This category involves all user-related information. The goal is to investigate user roles and their properties. Furthermore it should be reflected how groups are composed and what properties they have.

The following aspects of a single user seem to be relevant:

- Which relationships can be identified between users? This question raises the relationships between users within the organisation (e.g. hierarchy of users) and within groups they may form.
- Which activities may be executed by a single user? Who makes the decision whether a user is allowed to execute a specific activity?
- Which data may be accessed by a user?
- Which activities may be finished by a user? Maybe it is meaningful that a user can start a specific activity but only a user with other permissions is able to finish it.
- Which groups can a user join? There may be a coherence between the permissions of a single user and the groups in which he can participate.
- What further properties does a single user have that are relevant when describing a cooperative scenario?

When describing user groups, one generally has to think about the following questions.

- Are groups permanent (i.e. belonging to the structure of the organisation (divisions, departments)) or are they *ad hoc*?
- Are the groups homogeneous or may a group consist of users with different roles, responsibilities and capabilities (heterogeneous group)?
- What is the event to form a new group? What is the event for a group to disappear? Who has the authority to decide so?
- Which activities can be executed by a group? Which data can be accessed by a group? May this authorisation change during the execution of an activity (e.g. delegation)?
- Which communication facilities are provided between group members? (communication through sharing private group data; communication through sharing of global data; external communication facilities)
- What makes the group a group? There should be a difference in the relationship between two arbitrary users and between two group members. It should be investigated what are the describing elements of groups considering private data, allowed conflicts etc.
- Are there groups that belong to (parts of) an activity?

2.2 Aspects of Activities

This category describes the characteristics of the activities that are being performed in the specific application domain. It had to be found out if they can be described in a hierarchical way and what kinds of execution guarantees are given in specific situations. This covers the aspects of allowed interleaving and duration of activities.

The following questions seem to be relevant for the application analysis:

- Does the activity consist of subactivities etc.? Can the activity be described in terms of subactivities by forming an activity tree hierarchy?
- Which dependencies can be identified between different activities?
- Where do dependencies between activities come from?
- Is there an explicit notion of control flow between different activities that is orthogonal to the data flow?
- What is the semantics of finishing an activity? This question raises the issue of committing an activity. There may be implicit assumptions bound to the execution of activities.
- What guarantees are given/required for the execution of activities (ACID properties, other kinds of correctness criteria)?
- Is the concept of time relevant/important in the application domain at hand?
- Is it possible to undo an activity? Is it possible to re-execute an activity?
- Is communication between different activities required?

2.3 Aspects of Data

This category covers the data aspects of an activity. The observations that can be made are very much dependent on the characteristics of an underlying database system (if there is one) and will be different when investigating a scenario instead of an implementation.

- Are database services being provided?
- Is there a description of activity related data (DB schema)? Is the description of data bound to the physical level by having it hardwired in application programs?
- Is there an external description of activity data available? Are there restrictions by a given data model? Which characteristics of data are described in advance (relationships)?
- Data can be private or shared, persistent or volatile. Which of these "properties" of data are relevant in the application domains at hand?

Chapter 3

Analysis of Application Scenarios: Cooperative Document Authoring

3.1 Introduction

This chapter reports on the results of investigating the cooperative authoring domain and the analysis of the SEPIA cooperative authoring environment. Based on this analysis, requirements for a cooperative specification language are presented. Besides that, it is aimed to derive requirements for a corresponding transaction model (TRANSCOOP Deliverable II.2, TC/REP/GMD/D2-2/207).

First of all, some further CSCW concepts are introduced in section 3.1.1. The cooperative authoring scenario is positioned in the CSCW area and it is pointed out that there is a need for transaction models that cover cooperation issues. Section 3.2 gives a description of the cooperative authoring process in general. Afterwards general system requirements for the cooperative authoring domain are presented (section 3.3). From the observations of the cooperative authoring process and the analysis of SEPIA (section 3.4), specification language requirements are presented in section 3.5.

3.1.1 Motivation

Recent advantages in computer and network technologies have led to increasing interest in providing computer support for cooperative authoring processes. Unlike traditional word processors, which at the best support the cognitive activities of single authors, technologies for supporting cooperative authoring of documents must also provide support for the social needs of writers and their demands on coordination, communication and information sharing.

Figure 3.1 shows a classification of CSCW applications considering the geographical distribution and time dependencies between collaborating users [Joh88]. Classifying CSCW

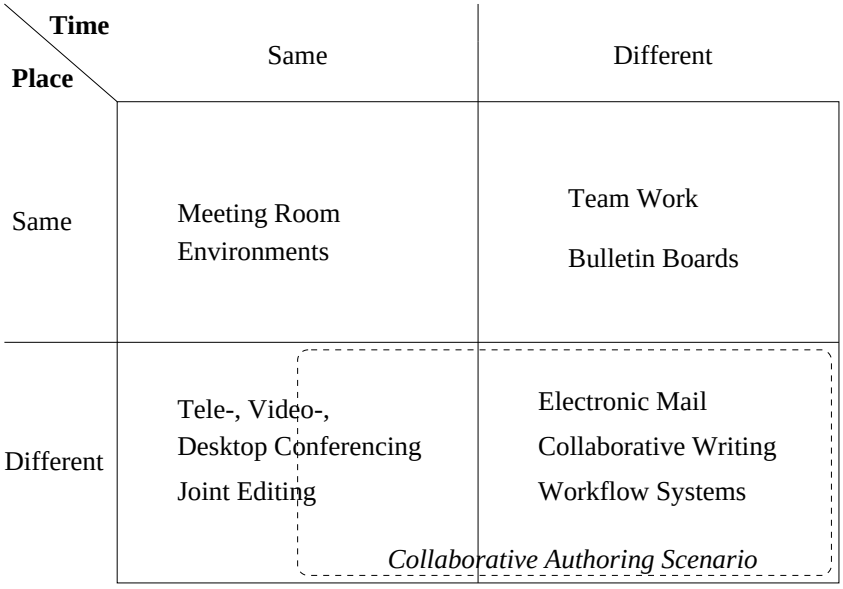


Figure 3.1: The Johnson time/place matrix of CSCW systems

systems by their time and space category is a widely used and accepted approach. The goal of comprehensive CSCW systems is to cover all cells of the time-space-matrix.

The cooperative hypermedia authoring scenario which is investigated in this document is located mainly within the lower right cell of the time-space-matrix. Co-authors decompose the overall authoring task resp. the document to be written into a number of more or less independent parts. After that, the parts of the document are edited by the responsible authors. Later on, they save the modified document parts to a storage system (e.g. a database system) or pass them to their co-workers. In this scenario only one author can modify a part of a hyperdocument at a time. Therefore, a cooperative authoring system can be described as a *distributed asynchronous* CSCW application.

Sometimes situations occur in collaborative authoring scenarios, where it is useful that multiple authors work on a part of a shared (hyper)document *simultaneously*. E.g., this could be the case, if a group of co-authors designs the outline of a document cooperatively. Moreover, they may want to decompose the overall work into subtasks and assign these tasks to specific authors, resulting in a workplan[1]. Furthermore, if an author detects some semantic conflicts within a document part written by another author, it may be the best to solve these conflicts in a tightly coupled cooperation.

The ability to edit e.g. the structure of a shared document simultaneously can save a lot of work and increases the effectiveness of the authoring process. Therefore the cooperative authoring scenario covers also some *distributed synchronous* aspects of computer supported cooperative work.

[1] A workplan itself can be viewed as a shared, commonly accessible document subject to cooperative authoring.

3.1.2 Related work

There are a lot of approaches for coordinating respectively synchronising the work of multiple users on common documents. They mainly differ in the degree of cooperation and in the power and flexibility of their synchronisation mechanism:

Conservative approaches: In CAD environments the concept of a *check-out/modify/check-in cycle* was used to synchronise access to shared data objects [KLMP94, HHMWM88]. A user can check-out specific objects and can perform exclusively operations before checking in the object again. The advantage of this mechanism is the local availability of data without additional communication overhead to the underlying storage system, thus increasing the efficiency of the computer system.

Between a check-out and the corresponding check-in operation the objects usually are not accessible by co-designers [2]. This can result in blocking of designers over possibly long periods of time and in limiting concurrency and cooperation between co-workers.

Floor Control: The notion of a *floor* is introduced to allow multiple users to edit several objects in a cooperative session [GS87, KP90]. Only a single user can be the floor holder and has the permission to perform all kinds of operations while other users can only observe his actions. To enable cooperation between the co-workers, a floor control protocol regulates the passing of the floor between the participants. Usually the applications are using one floor for all objects. This reduces the possible degree of concurrency.

Floor control mechanisms are not suited for applications that need a high degree of parallelism and require high availability of all information. Another disadvantage is the delegation of controlling correctness from a computer-based synchronisation mechanism to the responsibility of the users themselves.

Social Protocols: Another alternative is not to provide any computer protocols for enforcing the correct interleaving of operations of co-workers. The co-workers themselves are fully responsible to synchronise the group's work in a correct way. Social protocols are based on rules and policies that are part of the embedding organisational structure. This approach passes the control of synchronisation up to the users. They have to synchronise themselves by means outside the applications. Of course, the computer system can provide communication channels (e.g. voice/video connections) for the social coordination protocols. Systems using social protocols are described in [EGR91, SFB+87]

Because correctness and failure atomicity guarantees have to be ensured by the user, this approach can result in several conflicting operations that are not detected by the system. For example, useful pieces of work may be undone or are repeated by multiple authors.

[2] Other co-workers may be allowed to *read* checked-out objects. There are also some other extensions to provide more flexibility.

All these approaches do not meet the synchronisation requirements of cooperative authoring environments resp. CSCW applications in general. The goal of TRANSCOOP is the development of transaction management support for cooperative applications that provides appropriate synchronisation mechanisms for those applications.

Transaction models in general are guaranteeing fault tolerance of a given granularity and synchronise concurrent activities of multiple users while ensuring given correctness criteria. In cooperative authoring environments, groups of users are working on a large document, sharing its content. It is required to support those groups by preserving transactional guarantees as far as possible.

Furthermore, cooperative applications like CDA are characterised by operations on complex data objects that are manipulated by several users over a long period. System crashes and deadlocks may affect long-lived activities more often than short ones.

Of course, the classical non-hierarchical ACID transaction model does not satisfy the needs of cooperative systems. It assumes a lot of short transactions consisting of predefined sequences of read/write operations. ACID transactions give users the illusion to be the only user in the system – this contradicts the needs of cooperative application scenarios which require controlled visibility of concurrent users actions. Thus, a cooperative synchronisation mechanism should consider less the competition for data resources like in traditional transaction models. It is rather required to support controlled non-isolated data-access to support a more closely working process and thus give the users the ability to strive for a common goal.

Advanced transaction models support the composition of transaction hierarchies according to the hierarchical structure of the problem that has to be solved. The failure of a subtransaction does not necessarily cause the overall top-level transaction to fail. For example, it is possible to re-execute the subtransaction or to invoke an alternative subtransaction (contingency transaction). If the failed subtransaction is non-vital, the top-level transaction is still able to commit even if the non-vital child eventually fails. The support of different dependencies among (sub)transactions at different abstraction levels provides a flexible framework for transaction execution [CR92a].

Improvements of nested transactions are multilevel transactions [Wei91] and open nested transactions [MRKN92, WS92b]. A subtransaction of an open nested transaction may externalise its results to other concurrent transactions before the overall top-level transaction commits. Because of using semantics by dealing with commutativity between (sub)transactions the ACID guarantees are still preserved in the multilevel and the open nested transaction model [3].

Utilising the semantics of transactions in the open nested transaction model enables us to relax the strong correctness criterion of serialisability that leads to isolation of users. Relaxing the isolation property of transactions can be achieved by using application-dependent conflict specifications instead of commutativity definitions in the strong mathematical sense. This results in application-dependent correctness definitions. Also, some other advanced transaction models have the ability to specify application-dependent correctness criteria instead of using serialisability [NRZ92][HHZ+92] as correctness criterion.

[3] The multilevel resp. open nested transaction model uses multilevel [Wei91] resp. semantic serialisability [MRW+92] as a correctness criterion to ensure serialisability of concurrent executed transactions.

3.2 Cooperative Document Authoring of Hyperdocuments

This chapter gives an overview of both the individual (section 3.2.2) and the cooperative authoring process of hyperdocuments (section 3.2.3) [4]. Based on this overview , in section 3.3 requirements for a cooperative hypermedia authoring environment are derived.

3.2.1 Hyperdocuments

Traditional printed documents are characterised by their presentation medium – paper. This has direct impact on the contents and the underlying structure of the documents. On the one hand, documents can only contain static, non-continuous information, such as text, graphics and pictures. On the other hand, we have restrictions on the organisation of the document. The logical document structure is a hierarchical one. That is, traditional documents consist of several organisational units that are related by a part-of-relationship. For example, an article consists of several sections with subsections and paragraphs. Subsections and paragraphs are parts of a section. This hierarchical (logical) structure is reflected in the table of contents of a document. References within one document, such as cross-references or references from the table of contents to the corresponding document elements as well as references between different documents, such as bibliography references, have to be resolved manually by the reader.

Another restriction subject to traditional documents is the linearity of the final printed document. This restriction has consequences with respect to the presentation of information contained in the document. Moreover, the way a reader can read the final document corresponds to the linear sequence of information that is determined by the author of a (paper) document.

Hyperdocuments form a new class of (electronic) documents, that is based on the concept of *hypertext*. The basic principle of hypertext [Nel81] is the following: a hypertext consists of chunks of information, called *nodes*. These nodes can be related by explicit relationships, called *links*.

The main difference to traditional (paper) documents is that hypertexts are *non-linear*. The concept of links gives us the ability to connect information entities, such as text nodes, in an almost arbitrary way. Therefore, a hypertext has an associative, network-like structure[5].

An important implication of the hypertext concept is the interactive usage of hypertexts by their readers. A reader can freely navigate through the hypertext structure created by an author because links are represented explicitly and therefore references can be interpreted automatically by a computer system. Thus, there is no restriction on the way a user can read a hypertext, e.g. he can decide which link to follow. Of course, there can be some predefined tours through the hypertext created by the author. Such guided tours act as guidelines for the exploration of the hypertext by the reader.

[4] The following sections benefit from the work done at the WiBAS department (Knowledge Based Authoring Systems) of GMD-IPSI.

[5] In the basic model, a hypertext structure can be viewed as a graph in the mathematical sense.

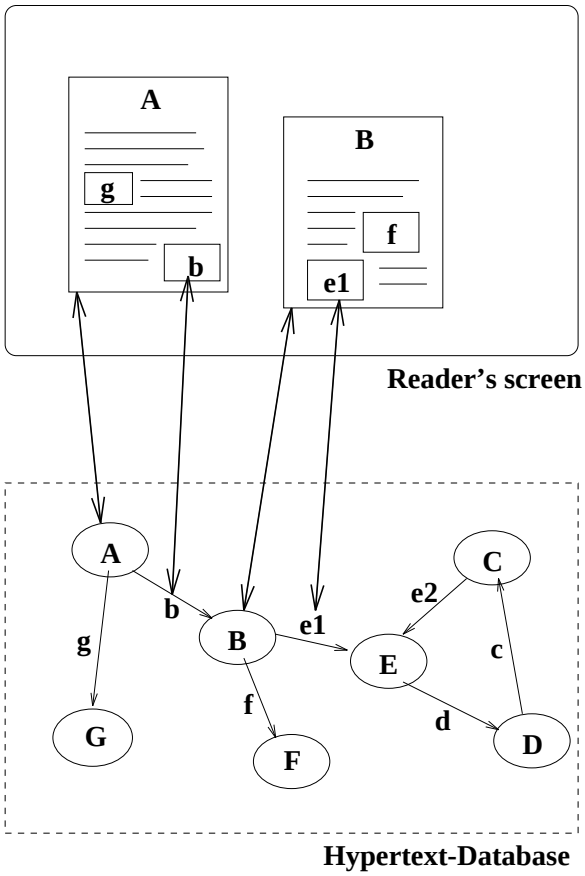


Figure 3.2: The concept of hypertext

Figure 3.2 shows an example of a hypertext structure and the presentation of the hypertext to the user. The hypertext objects (nodes and links) are stored in a hypertext database [6]. A node is presented as a window on the user's screen. Links are displayed as sensitive areas in the window. If a user wants to follow a special link, he just "clicks" on the corresponding sensitive area. For example, if he clicks on "b", a new window, displaying the node "B" pops up [7].

Reader-orientation is a major characteristic of hypertext structures. In the following we will use the term *hyperdocuments* to denote reader-oriented (parts of) hypertext structures that are logically self-contained and can be treated as a single entity. This implies that not every hypertext structure is a hyperdocument.

If the nodes of a hyperdocument do contain (besides text and simple graphics) also multimedia information, such as complex pictures and continuous information (audio, video, animation), the concept of hypertext is generalised to *hypermedia*. The term hypermedia addresses multimedia extensions of hyperdocuments, whereas the term hypertext is more focused on the structural aspects of hyperdocuments.

[6] A hypertext database can be implemented using a file system or a database management system.

[7] Please note, that this is not the only way to visualise hyperdocuments. For example, for the authoring process it may be more appropriate to visualise the hypertext structure as a graph.

There are several extensions of the basic hypertext model, such as composite nodes [Hal88]. Composite nodes introduce the concept of aggregation into hypertext structures. They can contain other links and nodes (another hyper text structure) and can be used to cluster related information entities. Atomic nodes have a set of (multimedia) information as their content, but can not contain further hyperstructures.

3.2.2 The individual authoring process of hyperdocuments

The concepts of hypertext and hypermedia provide new possibilities to structure the contents of a document and introduce new ways of presenting and reading documents. But also some problems inherent to the network-like structure are introduced by this new document type.

For example, a well-known problem for readers of hyperdocuments is "getting lost in hyperspace" [Con87]. Besides this navigational problem, many readers have trouble to comprehend the overall structure of a hyperdocument or the semantics of links [SHH+92].

Thus, hyperdocuments impose new demands on their authors. Authors are responsible for designing the associative structure of hyperdocuments using the appropriate modelling primitives, for editing the multimedia contents of (atomic) nodes and for presenting the hyperdocument to the reader. Having the problems of the readers in mind, this is not an easy task. In contrast to writers of linear documents, authors of hyperdocuments do not have guidelines telling them what their product should look like. Many rethorical decisions are made without widely accepted conventions. Since these decisions entail activities supplementary to the writing of linear documents, such activities are regarded as *cognitive overhead* [Con87].

To raise the quality of hyperdocuments and to make them more readable, authors of hyperdocuments should be supported by special hyperdocument authoring systems that provide appropriate modelling and structuring facilities. Of course, such systems have to be based on sound theoretical foundations.

In the following we present a model of writing hyperdocuments, that was developed at GMD-IPSI [THS90, HT93]. This model of writing is based on an analysis of the cognitive process of writing and the features of the authoring situation.

3.2.2.1 Authoring as a design problem solving process

The objective of the authoring process is the creation of a *"product"*: a (hyper)document. This implies the question how the final reader-oriented hyperdocument should look like. Another aspect is the *process* of creating this product.

Writing hyperdocuments can be seen as a problem solving process [HF86]. Hannemann et. al. [THS90, HT93] refine this model and characterise writing (of hyperdocuments) as a design activity[8].

[8] Problem solving is a general term. The problems to be solved can be assigned to different problem domains, such as design, diagnosis etc.

A characteristic of design processes is that the final state (and the initial state in many cases) of the problem solving process is not known exactly in advance. Moreover, the problem solving process itself usually can not be defined on beforehand.

This holds also for the problem of "designing a hyperdocument". At the beginning we have only a few weak constraints indicating the initial state and the final product. These are, for example, the subject and the objectives of the document that are described informally. Properties of the final hyperdocument, such as the size and the exact specification of the target group of readers may not be unknown at the beginning of the writing process.

During the writing process of an hyperdocument an author has several options how to proceed. He must make decisions without exact guidelines and measures to judge the quality of his work. This shows that creating documents can be regarded as a creative process because authors do not know how to reach a solution and how to satisfy the (implicit given) goals in advance.

Helping an author to travel through this complex design space, an authoring system must rely on the two main features of a design process [9]:

Process aspect: Design is a complex problem solving process, which consists of several subproblems, e.g. acquisition of background material or outlining the document. These subproblems are solved by specific strongly interacting activities, that build up on each others results. During the design process new subproblems may be identified and existing solutions of subproblems may be refined several times. The problem solving process can be seen as iterative refinement and improvement of artifacts.

Product aspect: Design is the construction of a product or *artifact*, which has to fulfill some informal criteria and for which the designer needs some adequate *building blocks* to compose an artifact of high quality.

The interdependencies of extensive planning, production and revision activities are characteristic for the writing process and lead both to an external product – the final reader-oriented hyperdocument and an internal product – the author's knowledge structure. One important issue in authoring environments for hyperdocuments is to help authors to externalise their mental representation of the knowledge structure.

3.2.2.2 The problem of writing hyperdocuments

By regarding the creation of a (hyper)document as a complex design problem solving process three main subproblems for writing of hyperdocuments can be identified [THS90, HT93]:

Content problem: This problem encompasses the acquisition of the contents and background information and the development of a domain model for the hyperdocument.

[9] We will see in the next section that there is another important feature of a design process: design as a social process.

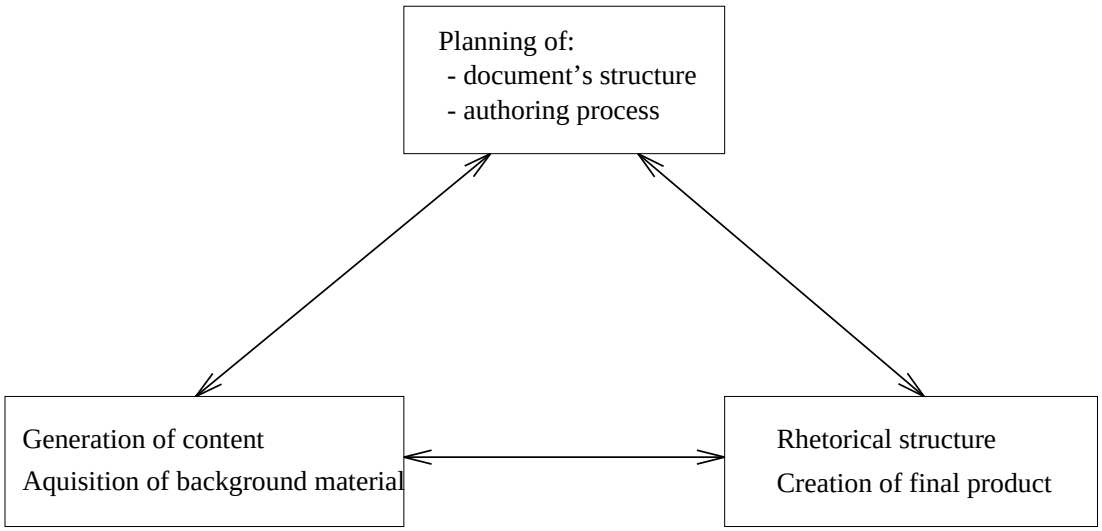


Figure 3.3: The main subproblems of the authoring process and their interdependencies

Rhetoric problem: This problem addresses the rhetorical structure of the hyperdocument. The authors should express their thoughts and ideas in such a way that the reader is able to understand the authors intention.

Planning problem: The third subproblem refers to the planning of the authoring process itself. This includes the basic structuring of the document as well as the development of a workplan for the authoring process.

These three problems depend on each other: for example, the acquisition or generation of content material depends on the actual planning of the document's structure. Figure 3.3 gives an impression of these subproblems of authoring and their interdependencies.

In [SHT89] a fourth subproblem is identified – the argumentation problem. This subproblem addresses the construction of argumentation structures. The document domain knowledge has to be prepared in such a way that the issues and goals identified in the planning problem process are satisfied. The argumentation problem can be seen as a link between the content problem and the planning problem. Solving the argumentation problem is very important for writing argumentative documents.

A hypermedia authoring system should provide appropriate support and modelling facilities for each of these subproblems. Furthermore it should be possible to exchange partial solutions between these problem domains and keep track of the dependencies among the different subproblems. Due to the opportunistic nature of the writing process, the authoring system should not restrict the authors to follow a predefined sequence of actions to solve the above mentioned subproblems.

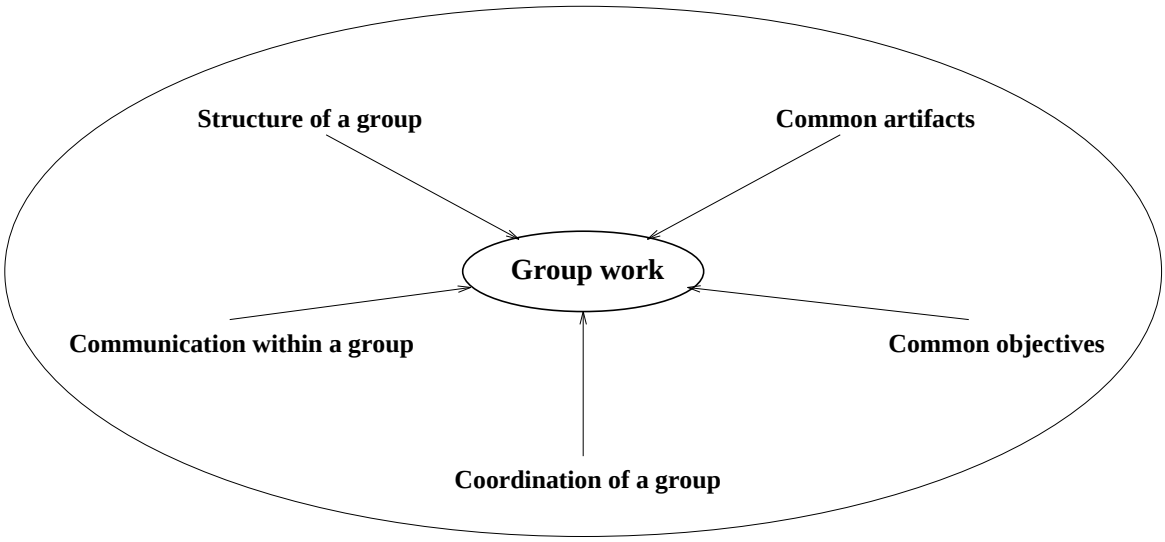


Figure 3.4: Factors that influence the group work

3.2.3 The cooperative authoring process

This section describes the process of cooperative (hypermedia) authoring within a group of collaborating authors.

As stated above, the (individual) authoring process can be described as a design process. Besides the process and the product aspect, there is another main characteristic of a (cooperative) design process. Design usually is a *social process* that involves a group of individuals. For example, the majority of technical documentations is written by groups of collaborating authors.

3.2.3.1 Group work

Group work or cooperative work denotes the social process of several individuals working on a common problem or task. The work within the group is influenced by different factors [LS92], which are shown in figure 3.4:

- A prerequisite for cooperative work is the existence of a group of co-workers. The *structure of a group* covers aspects like size of the group, (hierarchical) organisation of the group and geographical distribution of the group.
- Usually the goal of working together is the creation of *common artifacts* (e.g. hyperdocuments in our scenario). If this is not the goal, at least common artifacts are used as working documents during the group work.

- To enable the goal-oriented creation of common artifacts, *common objectives* have to be maintained. This includes the planning of the cooperative work, the decomposition into workpieces and the assignment of these workpieces to (subgroups of) individual co-workers.
- To maintain these common objectives and to ensure that compatible (sub)artifacts are created, some kind of *coordination mechanism* among the co-workers is needed.
- This coordination of the group's work needs *communication between the members of the groups*. This communication can be asynchronous or synchronous. Synchronous communication (phone, talk, audio/video conference) requires the availability of all communication partners while asynchronous communication (electronic mail, fax) is time-independent.

3.2.3.2 Hypermedia authoring as cooperative work

The collaborative process of authoring a hyperdocument can be basically seen as an extension of the individual authoring process.

In cooperative authoring an additional subproblem (besides the planning, content, argumentation and rethorical problem) can be identified – the coordination problem [Haa94b]. If multiple authors work on a common hyperdocument, there is a need for coordinating the individual authors work to fulfill the common objectives of the authoring process and to produce a single common accepted hyperdocument.

Viewing cooperative hypermedia authoring as group work, the construction of the final reader-oriented hyperdocument is the common goal of the group members. The common objective includes this goal and also constraints such as subject, size, type of the hyperdocument or deadlines, resources, etc.

The cooperative authoring scenario that is investigated in this chapter assumes geographically distributed groups. Communication between the group members can be asynchronous as well as synchronous, depending on the authoring situation [10].

The common artifacts that are created during the cooperative authoring process include the final target hyperdocument, as well as intermediate hyperdocuments, collections of background material and also documents about the cooperative authoring process itself (workplans, task allocations etc.).

From the above it follows that cooperative authoring involves both explicit communication and coordination and implicit coordination through common accessible material, such as workplans and document drafts [Sor87].

[10] Therefore in the time-place matrix of CSCW our cooperative authoring scenario is located in the two lower quadrants. Although we allow some kind of synchronous work in the cooperative authoring scenario, we will not address the field of *joint editing* explicitly. Joint editing denotes synchronous work at the content level (e.g. synchronisation of keystrokes while editing the same document part), that is not investigated in this document.

3.2.3.3 Cooperation on shared documents

In our scenario of cooperative authoring we assume that a large number of co-authors is manipulating a large collection of shared documents, that are stored in an underlying database management system. This shared material are hyperdocuments consisting of a large set of atomic nodes, composite nodes and links. Such hyperdocuments may be partitioned into subdocuments e.g. by using the concept of composite nodes (see section 3.2.1).

Based on [GHMS94] six different modes of cooperation on shared material can be identified. The following enumeration is a slight modification of the described modes.

1. *Separate responsibilities*: The documents are divided into independent, disjoint parts. Each part is edited by one user. Other users may inspect document parts assigned to other users. The cooperation here is quite loose.

2. *Turn taking*: Same as mode 1., but each part may alternate among different users. No more than one user at a time is allowed to edit a given document part. Turn taking requires more support for coordinating users manipulating (one after the other) the same part of a hyperdocument.

3. *Dynamic exchange*: During an authoring session, co-authors may exchange parts of the documents dynamically. One user *A* may want to modify a part of a document that is currently edited [11] by another user *B*. User *A* may *ask* user *B* to transfer the permission to edit this part to user *A*

4. *Alternative versions*: Different users are allowed to create alternative versions of document parts. This allows authors to work simultaneously on the same original part of the document without interfering each others work. Of course, different versions of a document part have to be *merged* later on and this might not be an easy task.

5. *Mutual sessions*: Two or more designers may work at the same time on the same parts of the document synchronously. This requires additional powerful coordination mechanisms to avoid harmful interactions between the concurrently working users

6. *Fully synchronous sessions*: In this mode, different authors working at the same time on the same data are using a shared, global window. All participants of a fully synchronous session share exactly the same view of the shared material (WYSIWIS [12]) and work on the same copy of the document part.

Modes 1 to 3 are typically asynchronous cooperation modes. These modes need support for asynchronous coordination of work. One issue here is the creation of awareness among

[11] In [GHMS94] the term *locked* is used. We avoid this term here, because we view locks as an implementation mechanism for transaction management. Locks in a database management system are not visible to the user of the database management system. Thus, an author can not explicitly lock (parts of) a hyperdocument to ensure that other co-authors cannot access this material.

[12] WYSIWIS is an acronym for **W**hat **Y**ou **S**ee **I**s **W**hat **I** **S**ee.

co-authors about who has done/is doing what in the shared documents. Another issue is to identify changes within the hyperdocuments.

Coordination in these asynchronous modes can take place e.g. by creating annotations for parts of the documents or by using external communication facilities such as e-mail systems or bulletin boards. Of course, also informal synchronous communication channels, such as telephone connections can be used to coordinate the asynchronous work on the shared document.

Mode 4 is also an asynchronous mode of cooperation. Versioning enables authors to work concurrently on shared documents without interfering with each other. The main coordination issue here is version management. Version histories have to be maintained to keep track of the derivation dependencies. Also, differences between alternatives versions have to be identified.

The most difficult task in mode 4 is the merge of alternative versions into one common version. The most authoring systems do not provide (semi-)automatic mechanisms to merge alternative versions. Therefore, large parts of the merge process have to be performed manually by one or more authors.

Mode 5 represents a synchronous cooperation mode on shared documents. There is a variety of variants of this cooperation mode depending on the degree of isolation between the users. For example, if traditional transaction management is used, co-authors are fully isolated. Advanced transaction models, such as the open nested transaction model [Elm92, WS92b] can be applied in order to relax the isolation properties. Another approach is to use one short transaction for each operation reading or modifying the hyperdocument. This ensures visibility of all modifications, but ensures consistency only on the storage layer.

In most systems the concept of notifications is applied to create awareness among co-authors. Notifications inform co-authors about changes in their shared parts of the hyperdocument. An extension of this notification mechanism is to broadcast changes of one author to the synchronously collaborating co-authors on this part.

The *loosely coupled cooperation mode* of the SEPIA- system[13] (see section 3.4.4.1) refers to such a variant of mode 5. In SEPIA mutually sessions are implemented using short transactions and a broadcasting mechanisms to update the user interfaces of co-authors participating in the same session if changes to the hyperdocument occur.

Mode 6 is also a synchronous cooperation mode, that corresponds to the *tightly coupled mode* of the SEPIA authoring environment. The main difference between mode 5 and mode 6 is whether a shared view is maintained at the user interface level or not. In fully synchronous sessions, all collaborating co-authors have the same view of the document part being edited (WYSIWIS). Events at the user interface level, such as scrolling or moving of a shared window are synchronised.

Usually additional coordination facilities such as telepointers [14] and audio/video communications are used in situations similar to mode 6 to coordinate the synchronous work of multiple co-authors .

[13] SEPIA is an acronym for *Structured Elicitation and Processing of Ideas for Authoring*

[14] A telepointer is a cursor that is attached to one user and occurs in all windows of the shared view.

3.3 Requirements for Cooperative Document Authoring

In this section we will present requirements for the design of a cooperative hypermedia authoring system based on the descriptions above and on other related literature. We will concentrate on those issues that are relevant for the TRANSCOOP project. For example, video communication surely is beyond the scope of transaction management support for cooperative applications and thus we will not describe such issues in detail.

The description of requirements tries to follow the categories that are introduced in subsection 3.2.3.1 and are shown in figure 3.4. These categories are: common artifacts, coordination of a group, communication within a group, structure of a group and common objectives. Obviously, it is not always possible to assign the different requirements to exactly one of these categories because they are not disjoint.

3.3.1 Common artifacts

As mentioned in the last section, common artifacts, like outlines or drafts of hyperdocuments, intermediate working documents and the final hyperdocument are a prerequisite for cooperative authoring processes. Therefore, the hypermedia authoring system must provide mechanisms for sharing information among collaborating authors. Usually the sharing of documents is realized by using a multi-user database management systems or a common file system[15].

3.3.1.1 Requirements on the underlying storage system

To enable sharing of information and concurrent access to common artifacts, the underlying storage system must provide an appropriate storage granularity. For example, if a document is represented as a single file, an entire new copy must be written, even if there is a minor modification in the hyperdocument. For hypertext networks the best storage granularity seems to be the hypertext objects themselves (links and nodes) or the attribute level of the hypertext objects.

The storage system must provide data access at this level of granularity. Therefore, common file systems seem to be inappropriate for sharing information in a cooperative authoring system. Relational database management systems are able to provide the required granularity. Unfortunately, their data modelling languages and data manipulating facilities usually are not powerful enough to support the complex data structures needed in hypermedia systems. Therefore, using object oriented database management systems (OODBMS), such as VODAK [16] [KAN93, WA94] seems to be a good choice for storing common artifacts in a cooperative hypermedia environment [17].

[15] We will see in the following, that using common file systems may not be the best choice.

[16] VODAK is an acronym for **V**erteiltes **O**bjektorientiertes **Da**tenban**k**system, that was developed in the VODAK department of GMD-IPSI.

[17] There are other advantages in using OODBMS, e.g. associative access to objects via an object oriented query language, automatic index-maintenance, extensibility by type inheritance, possibility to define views and active capabilities.

3.3.1.2 Consistency of shared artifacts and concurrent access

There are basically two possibilities to work on a shared hyperdocument [Haa94b]:

- *Asynchronous work:* Only one author at a time is allowed to edit a part of a hyperdocument.
- *Synchronous work:* Several co-authors can work on the same document part simultaneously. This cooperation mode is also called synchronous session.

A cooperative authoring system should support *both cooperation modes* [Haa94b]. There may be other predefined, application-dependent cooperation modes between these two extremes. For example, it can be allowed to read parts of documents that are currently edited by another user. Which mode of work is appropriate depends on the author's situation within the cooperative authoring process. Beside different modes of cooperation the cooperative authoring system should support also smooth transitions between the available modes [Haa94b].

Maintaining consistency of shared artifacts is one of the main issues in cooperative authoring systems. Ensuring consistency during asynchronous work is much more easy than to ensure consistency in synchronous cooperation modes. If multiple users are allowed to edit the same part of a document at the same time, conflicts between operations of the cooperating users can occur. These conflicts should be solved (semi-)automatically by the cooperative authoring system.

To ensure consistency during asynchronous work on shared documents, the system must guarantee that only one author at a time can edit a specific part of a hyperdocument. There are several approaches to ensure this property, like floor control mechanisms [KP90, GS87] (see also section 3.2.3.3).

To support coordination/communication in an asynchronous work mode, the hypertext model must allow to enrich data objects with *coordination related attributes* [GS87]. Coordination related attributes are for example co-author annotations/comments [NKCM90, HN93], contextual attributes like creator, modifier of an document part [GS87], application dependent document states (outline, draft etc.) or information about the user currently editing the object (see also section 3.3.2).

In synchronous cooperation modes, *synchronisation of concurrent operations* issued by co-authors participating in the same synchronous session is the main issue. Appropriate mechanisms and protocols are needed to resolve conflicts resulting from concurrent execution of operations. Approaches used in traditional transaction management, like serialisability of transactions, are inappropriate in a cooperative authoring environment and hinder tightly coupled teamwork [EG89]. This is because they try to isolate the users in a way that they do not see effects of other users concurrently working on the same data.

Flexible concurrency control schemes are needed to synchronise operations of cooperating users in synchronous sessions [Hal88]. Many synchronisation mechanism were developed

in the past. Usually the only thing they have in common is that they synchronise concurrent operations of co-authors in an ad-hoc way.

For example, many systems try to solve this problem providing "user locks" and notifications, leaving correctness control completely to the collaborating authors. Similar examples are described in section 3.2.3.3, mode 5 and 6. Other proposals for concurrency control in synchronous work on a shared document can be found e.g. in [EG89, GS87, WL93].

Sometimes also shared views of the data at the user interface level are used for synchronous sessions. We can distinguish relaxed shared views, strict shared views (WYSIWIS) and joint editing [HN93, LSW92]. A concurrency control algorithm for real-time groupware systems can be found in [EG89].

3.3.1.3 Version management

Version management is an important issue in authoring processes because maintaining different versions of hyperdocuments allows users to access past states of the documents. The concept of versioning enables authors to explore alternatives and to "undo" modifications by selecting an older version of the document.

This is also true for single-user systems but version management is more important for co-operative (authoring) environments because the danger of accidental modification is higher. Moreover, a user may want to see a stable state of a hyperdocument even though another user may be modifying the hyperdocument at that very moment [GS87].

If alternative versions are allowed by the version model of the cooperative authoring system, the system itself should support the merging process of alternative versions into a consistent new version. Unfortunately, in most systems the merging of versions is completely left to the authors.

Although a variety of version control models have been developed [Kat90, Haa92a, Haa94a], they all rely on the ability to define and maintain dependency relationships or links among objects. These basic mechanisms should be provided by the underlying database system. Another issue in version management is to minimise the storage cost of multiple versions corresponding to a single object.

In cooperative systems also contextual information, such as who created an object version should be recorded in addition to the object version's content. This information allows to select object versions on the basis of properties of group interactions [Haa92b].

3.3.1.4 Access control

Another important issue in sharing information is access control and authorisation. Effective access control is important for groupware systems which tend to focus on activities and to increase the likelihood of user-to-user interference [EG89].

The right of an author to perform an operation on an object may depend on the user himself, the user's social role within the authoring process, the object to be accessed, the operation to be performed and on constraints about the data objects [GS87]. Access permissions are not static, they may be granted, revoked or delegated during the authoring process.

Therefore, the underlying storage system must provide basic authorisation mechanisms for building appropriate access control models [RBKW91]. The hypermedia authoring system should also provide facilities for negotiation about access rights (see also 3.3.4) and has to handle emergency situations [GS87], e.g. if the only authorised co-author is on vacation.

3.3.1.5 Private and public workspaces

Related to access control issues is the concept of private and public workspaces [18]. Private workspaces enable authors to work on a hyperdocument on their own. The authors can explore alternatives, work on private solutions and create private working documents, that are not visible to the co-authors in a cooperative writing scenario. If an author wants to make some documents public, he can explicitly "publish" them by transferring the hyperdocuments to the public workspace [Haa94b]. The concept of private and shared workspaces can be extended by shared workspaces, that are used by a dedicated group of authors.

3.3.1.6 Support for long lived activities

Hypermedia systems clearly need support for long-lived activities. For example, if the hypertext network contains large documents as node contents, a user may require anywhere from a few minutes to several days to make a single update to a node [Hal91, Hal88].

Therefore, the system should try to minimise the amount of work lost in case of a system failure or network partitioning [19]. The most systems use checkpoint or savepoint mechanisms to do so. The authors have the ability to "save" their work and to "restore" to a checkpoint after a system failure has occurred.

3.3.2 Coordination of a group

A coordination protocol is a mutually agreed upon way of interacting. Coordination may be built into the hardware and software (technological protocols), or left to the control of the participants of a cooperative hypermedia authoring process (social protocols) [EG89]. Coordination of co-authors' activities can be classified in two categories [Haa94b]:

[18] This concept is similar to checkin-checkout mechanism in some CAD systems [KLMP94, KBGW91]. The local availability of the data is used to increase the efficiency of design operations.

[19] In the case that transaction management mechanisms are used to implement these long-lived activities, this means a loss of work because of non-serialisable execution (deadlocks)

- Preplanned coordination
- Spontaneous coordination

Preplanned coordination usually is used in structured, asynchronous authoring processes, whereas spontaneous coordination is more often needed in unstructured resp. synchronous work situations. Ellis et. al. [EG89] claims for an integrated coordination support for structured/unstructured and asynchronous/synchronous activities.

3.3.2.1 Activity management facilities

For some activities in an authoring process, it seems to be appropriate to execute them asynchronously. The overall work is divided into some more or less independent subtasks. These are for example, the authoring of a special part corresponding to a specific subject that is addressed in the hyperdocument, the creation of an image or the preliminary collection of bibliographic references [Thi94].

These subtasks can be assigned to an individual author, to a group of authors or to a special role of an author or group within the authoring process (see also section 3.3.4). Each co-author can then separately deal with the tasks under his responsibility.

Usually the different subtasks are not fully independent of each other. For example, there may be some ordering constraints and data dependencies between asynchronous subtasks or temporal constraints on the execution of the tasks, e.g. deadlines. To enforce these constraints and to ensure that a coherent hyperdocument is produced, there is a need for system supported coordination of the different tasks occurring during an authoring process, e.g. informing users of required tasks [EG89]. Therefore, activity management seems to be very similar to workflow management [Lom93].

The specification of such an activity structure corresponding to a special authoring process can be made beforehand. This requires that a careful analysis of the different authoring processes has been done before the specification. But in most cases such predefined specification is too inflexible, because the activity definition itself evolves during the authoring process [NKCM90]. Thus, an authoring system should provide mechanisms to modify the task's specification, e.g. reassignments of tasks to authors or dynamic creation of additional subtasks.

Another possibility is to include the activity specification process in the authoring process itself. In this case the co-authors develop cooperatively a "workplan" for their specialised authoring activities [Haa94b]. This workplan may be a formal description of the authoring process, so that the system itself can coordinate the authors' tasks. If the workplan is more informal, it is used in the writing process as a additional resource in deciding what to do [NKCM90]. In this case, the authors still have to coordinate their activities by themselves.

3.3.2.2 Coordination of asynchronous work through annotations

To coordinate loosely coupled asynchronous work in cooperative authoring, some kinds of annotations are used [NKCM90, HN93] in most systems. Annotations in hypermedia documents are usually of the granularity of an hypermedia object. They should also contain additional information about the context of the annotation, like creator and creation time.

Annotations can be used in a variety of directions. Authors can use them to describe the modifications they have been made or to describe which work another author should do in the future. Moreover, annotations can be used for commenting on hyperdocuments, which is important in an authoring process [NKCM90]. To support this wide range of annotations, some systems use different types of annotations, like private annotation, public annotation and comment annotation.

3.3.2.3 Group awareness in cooperative authoring

Awareness of the co-authors' activities is another important issue in cooperative authoring environments. Authors should be informed about each others work. They must know when other users make changes to the hyperdocument that affect their work [EG89]. Awareness usually is implemented through notifications, that inform users that something has changed in the document [GHMS94, HW92a, EG89].

Awareness is needed to monitor the progress of the authoring project and to detect semantic conflicts in the work of different authors. If such conflicts occur there is a need for spontaneous coordination to resolve the conflicts. For this type of coordination, in the most cases synchronous communication facilities (see section 3.3.3) are used, like shared views to a hyperdocuments together with telepointers and meta-communication channels, such as audio/video connections [EG89, HW92b, LSW92].

3.3.2.4 Other types of coordination

Activity logging: Someway related to the concept of awareness and version management is activity logging [LFK88, HN93]. To coordinate work on a shared document authors should be able to keep track of changes to the documents. For example the authoring system should answer such questions like "what has been changed in the introduction since my last edit session?".

Coordination through social roles: This type of coordination is closely related to authorisation issues (see 3.3.1), activity management and the structure of the group (see 3.3.4).

Meta-communication channels (see 3.3.3): Usually these facilities are used to satisfy spontaneous coordination needs of co-authors.

3.3.3 Communication within a group

Communication between co-authors can be divided into asynchronous and synchronous communication mechanism [Haa94b]. Some of the listed communication mechanism are already discussed above [20] or they are outside the scope of TRANSCOOP, so they are not described further in this section.

Asynchronous communication mechanisms

- Communications through shared artifacts resp. draft passing
- Communication through annotations
- Electronic mail systems
- Bulletin boards

Synchronous communication mechanisms

- Audio/video connections (meta-communication channels)
- Shared views (WYSIWIS) and telepointers

The above listed communication facilities should be tightly integrated within a cooperative hypermedia authoring environment [HN93].

3.3.4 Structure of the group

This section focusses on the coordination of the cooperative authoring process through the structure of the group of participants. The topic of defining the structure of a group is closely related to the issues of authorisation, activity management and shared artifacts in general.

A response to the coordination problem is to support the definition of *social roles* within a group. Defining social roles reduces the coordination problem by specifying "proper functions", like responsibilities, access rights, privileges and patterns of interaction [NKCM90, EG89].

Which roles are appropriate depends on the organisational structure of the embedding organisation and the specific type of authoring process. In [LFK88] for example, three types of generic social roles are used: co-author, commenter and reader. In other situations, there may be also social roles like editor and reviewer.

Roles can be organised in a hierarchical way. For example, in [LFK88] a commentor is defined as a specialisation of a reader and a co-author as a specialisation of a commentor. In general, social roles within a group can be defined in terms of [NKCM90]:

[20] This shows that communication and coordination are closely related. Communication can be viewed as a mechanism to enable coordination.

- Types/Sets of hypermedia objects
- Types/Sets of operations
- Cooperation modes

In addition to defining social roles for an authoring process, a set of *user definitions* is needed. A user definition usually contains the username or user-id and the roles that each named user is allowed to assume [GS87].

To simplify the mapping of users to roles an additional concept of a *group* can be introduced. Users are associated with a group and this group is assigned special roles. Also groups can be combined to larger groups, resulting in a group hierarchy orthogonal to the role hierarchy.

3.3.5 Common objectives

Maintaining common objectives within a group of co-workers is important to generate coherent hyperdocuments. In a single-user system authors usually do not articulate their goals and objectives resp. changes of objectives and constraints during the writing process. In cooperative authoring environment the externalisation of goals and objectives is much more important [NKCM90].

Common objectives are, for example, the subject of the hyperdocument, the type of readers and constraints about the final hyperdocument, like size of the hypertext network and style guidelines. Some of these objectives can be ensured automatically by the system – others have to be maintained by the co-authors themselves because they are informal. Such informal objectives could be stored in the system as special working documents.

Of course, goals and constraints can change during the authoring process. A cooperative authoring system should provide mechanisms to ensure that the co-authors are aware of the actual objectives and constraints within the group of collaborative authors.

Also authors often need to communicate about the constraints in order to refine their views of the goals that co-authors have generated and increase the likelihood that they will generate compatible products [NKCM90]. Therefore, a cooperative authoring system has to provide appropriate negotiation mechanisms to agree about the actual objectives and constraints imposed on the authoring process.

3.4 Analysis of SEPIA

3.4.1 Introduction

This section reports about the design and implementation of the SEPIA cooperative authoring environment. The idea of SEPIA (*Structured Elicitation and Processing of Ideas for*

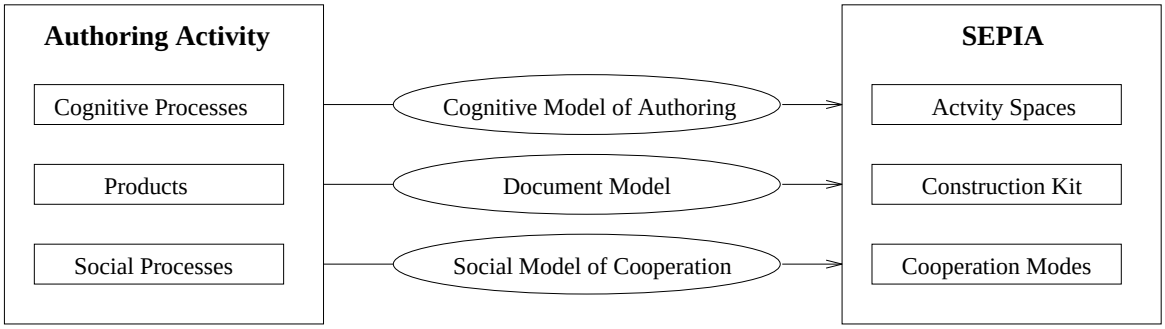


Figure 3.5: Research and development strategy for SEPIA

Authoring) and its basic design principles as well as implementation issues will be presented. Finally it is discussed whether the support of cooperation by the current system is satisfactory for authoring processes and how much knowledge about the authoring process itself is hardwired in the system. By identifying these things, we aim to end up with a list of requirements for the TRANSCOOP specification language and the transaction model. The following presentation is based on the results of the SEPIA research which are presented in [SHH+92, HW92b, SH93].

SEPIA is a cooperative hypermedia authoring environment. Its purpose is to support groups of authors who collaboratively create hyperdocuments. In SEPIA, the largest unit of work is a *project*. Such a project roughly corresponds to a book together with all background material in traditional publishing. Each project consists of four *activity spaces*. These are dedicated workspaces which fulfill different purposes.

Associated with each activity space is a dedicated browser which offers a certain set of functions on the objects it presents. These objects are hypertext objects, i.e. nodes, links and composite objects. In SEPIA, nodes are typed, and each activity space provides specific types for nodes. Similarly, links are first-class objects, and each activity space offers a set of link types which are relevant in its context. Composite objects are used in all activity spaces to cluster information. One can also open a browser on a composite object to display its subobjects. For all browsers, a construction kit offers several basic options on how information can be arranged.

During the design of SEPIA the three main aspects of authoring activities have been considered: the cognitive process, the final product itself and the social aspect of authoring. Figure 3.5 shows the relationship of the authoring activity, the theoretical background, and the resulting components of SEPIA. Paying attention to the process aspect requires to develop and refine a model of the cognitive writing processes and to transform these results into a corresponding functionality as e.g. the activity space concept. Looking at hyperdocuments as a product with new rhetoric properties leads to requirements for a corresponding functionality, as e.g. the construction kit in the rhetorical authoring space. Considering that most large documents are prepared by a team, social cooperation models have to be defined, and SEPIA has to provide corresponding cooperation modes.

Thus, detailed knowledge about the processes, the product and the social situation played equally important roles in the development of the SEPIA system.

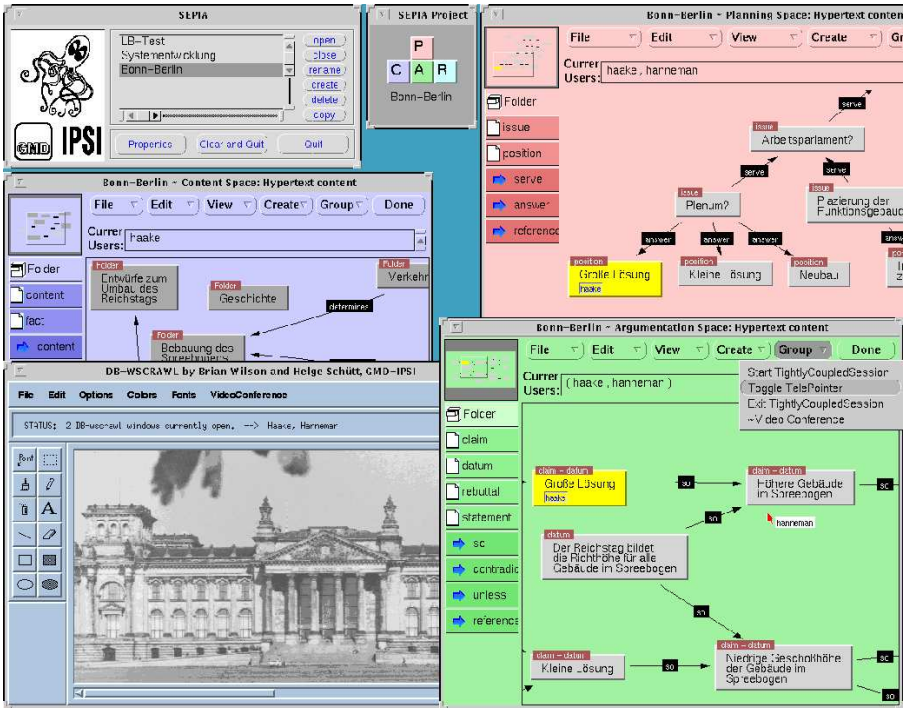


Figure 3.6: User-Interface of SEPIA

3.4.2 Cognitive Model of Authoring: Activity Spaces

3.4.2.1 Introduction

The process of writing of a hyperdocument is regarded as a complex problem solving process that consists of at least three subproblems. The mental representation of these three problems can be described in terms of interacting problem spaces. A problem space is formed by different constraints, design objects and operations in which different knowledge sources are modelled. Design objects can be *atomic nodes*, *links* and *composite nodes*.

The decomposition of the design space into subspaces is used as a basis for identifying the components of the authoring environment. These cognitive problem spaces are mapped in the SEPIA system to corresponding activity spaces. Each activity space provides specific design objects and operations appropriate to facilitate the author's activities when working on a specific subproblem. More precisely, this means that space specific functionality is offered by typed links and nodes according to the semantics of the space.

These cognitive problem spaces are mapped in the SEPIA system into the following activity spaces:

- Planning space:

  In the planning space an author has the opportunity to structure his intentions. The space serves as a platform to externalise writing plans respectively document goals,

to construct issues to be concerned with the document, and to establish an agenda for the authoring activity. Consequently, this space can be regarded as a metaspace for the coordination of different authoring activities. Furthermore it can be used by the authors as a facility for controlling the progress of the design solving process.

The planning space does not provide special support for controlling the work progress or informing authors about deadlines etc. But the authors can communicate through contents in this space that have the agreed functionality (coordination of activities, status of work).

- Content space:

  Within this space SEPIA gives the authors support for idea dumping and the grouping of background material in topic related clusters by composite nodes and the connection via typed links. The objective of this space is to support the development of a document domain model. A design object in the content space can represent background information from internal (other SEPIA documents) or external sources (databases).

  By collecting all information that belongs to the document domain the authors get an impression of the facts they can use for their argumentation structure. By structuring the material the authors get an idea of how to compose the document because the classification of facts helps to get a better view on the problem.

- Argumentation space:

  In the argumentation space the user can create an argumentation structure based on an extension of the argumentation schema developed by Toulmin (1959). For this, appropriate design objects are provided. Authors can elaborate argumentation by generating objections on different levels, by formulating contradictions and by constructing argumentative chains.

  An argumentation structure is build up by linking several hypertext nodes (datums, claims) together. These nodes can be connected with specific link types (e.g. contradict) for modelling an argumentation structure.

- Rhetorical space:

  The idea of the rhetorical space is to create the reader-oriented final hyperdocument. The objective is the assistance of the production of different document styles by providing alternative reading paths resp. different arrangements of the hyperdocument.

  Reading facilities for hyperdocuments are provided. The readers can browse in each space by activating nodes and links and navigate in that manner through the hypertext network.

  Hypertext document types constitute a scale ranging from strictly linear to strictly non-linear documents. Notice that hyperdocuments can vary in the degree of their linearity between these two extremes. Nevertheless, they all should satisfy one major requirement: in order to support comprehension and navigation on behalf of the readers, they must appear as coherent entities. Therefore, the rhetoric space provides a special 'construction kit' based on the concepts of coherence consisting of design objects that are explicitly tailored to the requirements of designing artifacts.

  Please note that the 'construction kit' is an external part of the current SEPIA implementation.

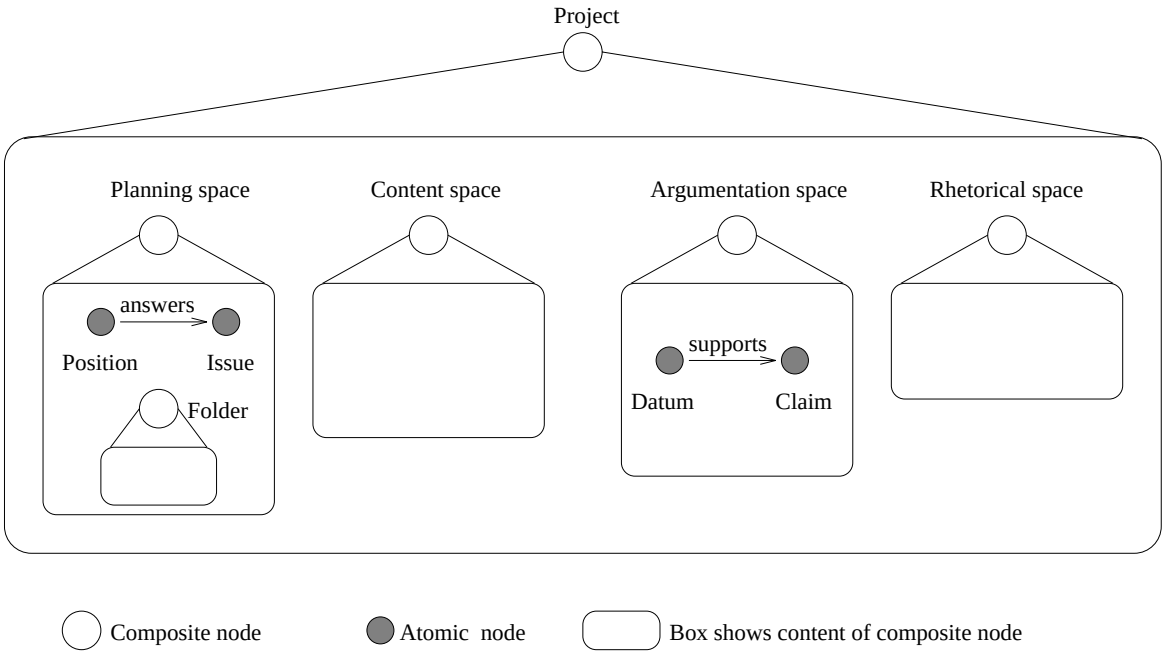


Figure 3.7: Relationship of composite and atomic nodes

3.4.2.2 Data model

A SEPIA project consists on the highest level of the four different authoring spaces as described in section 3.4.2.1. Each authoring space contains specific types of nodes and links that are dedicated to support the activities of the authoring process.

Nodes are hypermedia objects that can carry specific types of contents in accordance to their type definition. This can be plain text, graphics, audio, or a substructure formed by nodes and links (called composite content). If a node has a composite content it is referred as a *composite node*, in the other case it is referred as an *atomic node*.

The implemented inheritance hierarchy realizes links as specialisations of nodes. Thus, they have the same functionality as nodes. All classes that provide objects for carrying specific node contents (text, graphic etc.) are specialisations of a generic content class, that implements the basic functionality. The specific content classes are refined according to the specific needs of the media types.

By using composite nodes it is possible to cluster the hypermedia structures in the different authoring spaces into related chunks of information and to maintain a hierarchical, nested structure. The four authoring spaces of a project as well as the project itself are implemented through composite contents, too (Figure 3.7).

Nodes and links can be shared through different composite nodes. Thus the hypertext structure forms a rather arbitrary graph structure. Sharing is implemented by referencing from different atomic nodes in the activity spaces to the same underlying data object[21].

[21] This is not reflected in figure 3.7

As mentioned above, nodes and links have specific types. Constraints for ensuring specific consistency criteria can be posted to the hypermedia data structure. Examples for these constraints are 'claim nodes occur only in the argumentation space' or 'support links can only exist between datum nodes and claim nodes and cannot have any further substructure'.

With the type concept in SEPIA more semantic of hypermedia objects is provided and thus it is possible to construct meaningful hyperstructures that are able to reflect (sub)problem solutions.

The concept of types in SEPIA differs from the usual type definitions used in (object-oriented) programming languages. Types are more equal to consistency constraints on the hypermedia structure. They do not provide additional functionality or properties of the typed objects, they only state some restrictions on the hyperstructure and give the system the possibility to ensure some properties of the hypermedia document. They also provide the authors with a powerful method to express and structure their intentions and ideas. Thus types in SEPIA are orthogonal to the concepts of types used in object-oriented programming languages or database management systems[22].

In the current implementation, the typing of nodes and links does not have any impact on the cooperation between different authors.

In the following a short description of the different node and link types in SEPIA and their basic constraints are given. Due to the fact that this document investigates in cooperation issues the semantics of the offered node and link types are not further exploited. Afterwards the operations that can be performed by the users on the hypertext structure are presented:

3.4.2.2.1 Planning Space

- Nodes:

  Issue (atomic)

  Position (atomic)

  Folder (composite)

- Links:

  ContributesTo Issue → Issue

  Answers Position → Issue

  Suggests Position → Issue

  Specialises Issue → Issue

  RefersTo {Issue, Position, Folder} → {Folder}

[22] Classes can be used to implement types in SEPIA

3.4.2.2.2 Content Space

- Nodes:

  Content (atomic)
  Document (composite)
  Folder (composite)

- Links:

  RefersTo {Content, Document, Folder} → {Content, Document, Folder}

3.4.2.2.3 Argumentation Space

- Nodes:

  Datum (atomic)
  Claim (atomic)
  Warrant (atomic)
  Folder (composite)

- Links:

  Supports Datum → Claim, Claim → Claim
  Contradicts {Datum, Claim, Warrant} ↔ {Datum, Claim, Warrant}
  Explains Warrant → Supports
  RefersTo {Datum, Claim, Warrant, Folder} → {Folder}

3.4.2.2.4 Rhetorical Space

- Nodes:

  StartNode (atomic)
  ContentNode (atomic)
  DocumentNode (composite) reference to another document
  PathNode (composite): includes a node of type *StartNode*.
  ExplorationNode (composite)

- Links:

  FollowedBy StartNode → {ContentNode, PathNode},
  {ContentNode, PathNode} → {ContentNode, PathNode}
  ExploredBy ContentNode → ExplorationNode
  XRefTo {ContentNode, PathNode} → DocumentNode

3.4.2.3 Available operations on hypermedia objects

In this subsection a short summary of the available operations at the user-interface is given. SEPIA supports a clipboard functionality with the well-known cut-copy-paste operations. These operations can be applied on single nodes and links as well as on selected hypertext structures:

3.4.2.3.1 General operations:

- Cut of nodes and links to clipboard
- Copy of nodes and links
- Paste of nodes and links

The following enumeration shows the available operations on the hypertext structure. Not all operations are available in every authoring space but these differences are not of particular interest here. As mentioned before, they depend on the offered functionality of the specific authoring space. The operations are performed on *HMObject*[23] which is a unified datatype that is used for representing nodes and links. Because of that, the list describes the operations as methods on a fictive HMObject:

- Create a node resp. link of a specific type (text, audio, graphic, composite, . . .))
- Remove an HMObject
- Rename an HMObject
- Move an HMObject
- Set an activity marker on an HMObject
- Release the activity marker of an HMObject
- Create a new content for an HMObject (text, audio, graphics, composite if allowed)
- Show the contents of an HMObject.
- Edit the contents of an HMObject.
- Remove the contents from an HMObject.
- Convert the type of an HMObject to a new (allowed) one.

[23] HyperMedia Object

3.4.3 Document Model: Construction Kit

In the following the SEPIA construction kit is described. As mentioned before it is not part of the current implementation. To get an impression of the SEPIA concepts, its intentions are summarised here. The reader should note that the document model has significant influence on the hypertext as a final product and thus is not so much related to the purposes of this document. However, a deeper understanding of the hypertext structure could help us to learn something about data modelling requirements. This presentation is based on the detailed proposal of the construction kit which can be found in [THH91].

When reading hyperdocuments two difficulties can be identified that are related to the reception of hyperdocuments:

- First, it can be difficult to find the way in a complex hyperdocument. Readers get often 'lost in hyperspace', i.e. they do not know where they are in the overall structure of the document, or where to go next or how to reach a desired destination.
- Second, it can be difficult to understand a hyperdocument. Readers have often problems to understand the context of all the text distributed over various nodes. In that case, they fail to develop an understanding of the document as a coherent entity of closely related facts. Instead, they tend to see it as an aggregation of loosely related chunks of information.

The basic design options of a hyperdocument are nodes and links. In order to increase the coherence of a hyperdocument, nodes and links can be enriched with properties which facilitates the recognition of coherence relations. The resulting set of design objects consists of specific node and link types which can be used as building blocks for the creation of hyperdocuments. They allow a large variety of structures ranging from linear hyperdocuments to non-linear, interconnected documents. In any case hyperdocuments are modelled as a graph of typed nodes and typed links.

An authoring system has to provide design objects with specialised semantics so that authors are able express their ideas decisions in order to perform their tasks. Therefore, three main components are proposed that are suited to represent final as well as intermittent solutions:

1. Content parts: For the generation and selection of relevant information there is a content part. Content nodes consist of different kinds of information, e.g. text, graphic etc. Each content node carries a name and may represent information, e.g. facts, opinions, claims etc. Atomic content nodes may contain a variety of data, and form an entity which can not be subject to further separation. Atomic nodes with external sources manage the access of information stored outside the SEPIA storage system. In contrast to atomic nodes, composite content nodes carry substructures consisting of atomic content nodes resp. composite content nodes.

   Content links establish substantial connections between these nodes. The semantic relationship between source and destination node can be seen as destination *serves* source resp. source *is explained by* destination.

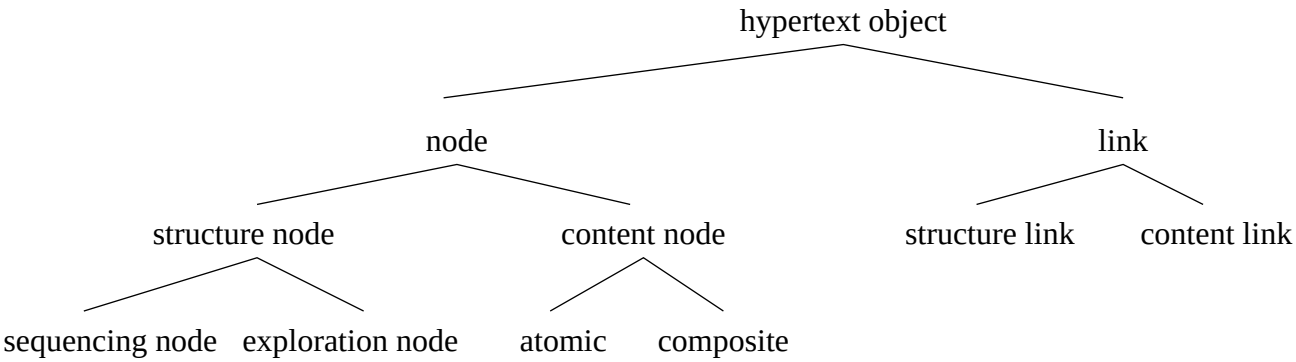


Figure 3.8: Design object hierarchy

2. Organisational parts: For the structuring of information an organisational part is needed that serves the purpose of structuring and organising information under a reader-oriented perspective. Structure nodes can be used to organise the contents in a specific manner while structure links are regarded as coherent relations for the connection of document parts. The objective is to adapt the document to the needs of different readers.

   Structure nodes comprise a subset of nodes and links of the content part and are implemented as composite nodes. Each structure node carries a name and has a starting node. Two kinds of structure nodes are distinguished which assign different degrees of freedom to the reader's navigation.

   (a) *Sequencing nodes* represent author-defined reading sequences through specific parts of the content net. Thus a sequencing node can be regarded as a predefined path providing navigation through the content net.

   (b) *Exploration nodes* represent a subset of the content part. In contrast to sequencing nodes they do not provide additional ordering of the content nodes, thus the reading sequence is not determined in advance.

3. Presentation parts: The presentation part covers the aspects of different presentation styles corresponding to the design objects. Each node contains another specific type of information that requires a specialised presentation.

The above observations are centrally based on the notion of coherence. The construction kit is a tool kit that present several design objects and rules for the creation of hyperdocuments. In [THH91] the design object hierarchy has been presented in figure 3.8

SEPIA is, based on the analysis of the cognitive process of authoring, founded on the activity space concept. Each activity space is related to support specific activities of the authors, like planning or gathering information. The three components of a hyperdocument that are proposed here (content, organisation and presentation part) are planned to be the output of SEPIA's rhetorical space. This space should provide assistance for the creation of document structures that consider the described types of nodes and links.

3.4.4 Social Model of Cooperation: Cooperation Modes

As described in section 3.2.3 the objective of environments like SEPIA is the support of communication and coordination facilities for groups of authors. The competing requirements are to provide concurrent work without isolating the authors from each other and thereby to preserve the consistency of the hyperdocument. Following the argumentation of section 3.1.1, blocking of users contradicts the requirements of the CSCW application domain and should thus be reduced to a minimum.

A group that processes a task collaboratively, needs the ability to access a shared information base that contains the working product. In case of cooperative authoring an information base is a document base for co-authoring. During the group work different situations have to be distinguished. In the first place cooperative workers can work on different parts of the document base at the same or at different times. This working situation is called *individual work*. The second situation is the concurrent access to the same parts of the document base, called *loosely coupled work*. In this situation awareness of co-workers presence and activities is required to fulfill the coordination needs. The third case is the cooperative[24] access to common parts of the document base. This scenario is considered as *tightly coupled work*. When co-workers are going to solve a problem together they establish tightly coupled work. For this SEPIA provides additional communication channels and a shared environment including a common view on the document base (WYSIWIS).

The three described collaboration situations are distinguished in the authoring situation as it is realized in SEPIA [SH93]. The cooperation process proceeds by shifting between the above situations. The transitions are driven by the actual need for cooperation which arises from the state of the work and the next goals that are to be achieved. Therefore, smooth transitions between these modes have to be supported.

CSCW applications like cooperative authoring scenarios require a shared document base for supporting the described cooperation model. The concurrent access of common document parts by multiple authors needs to be synchronised with adequate synchronisation strategies to guarantee a consistent document base and to provide the authors with a consistent view on the document. Furthermore strategies for ensuring awareness of co-authors work have to be supported. In SEPIA, activity markers are used to signal the current status of an object and update notifications are needed to inform other authors of latest changes for ensuring a consistent view on the document base. In loosely and tightly coupled working mode SEPIA applies the consistent view beside the document base to the user interface level, too (position of objects on the screen and display style information; window sizes in tightly coupled mode).

To satisfy the need for smooth transitions between the different cooperation modes, additional information which describes the status of the actual collaboration has to be recorded. This mainly includes the users who are currently accessing an object. This additional information is needed to establish state transitions among cooperating users and hence depends on their current interactions.

[24] In SEPIA terminology this kind of access is called 'synchronous access'. Because this terminology is sometimes a confusing we will avoid it here.

In the following, some remarks to the system architecture have to be made to give an understanding of the implementation aspects of cooperation. SEPIA is based on a client/server concept in which the whole information base is stored in a server database. The authors are interacting with client interfaces that are requesting information from the server resp. sending updates to the server database. Typical operations that the clients are sending to the server are *remove_node*, *edit_node* etc. These operations are treated on the server as atomic units and thus will be executed in some order. Thus, different client requests are executed as transactions while one transaction covers only one elementary state transition to the hypertext structure[25]. By serialising these operations the consistency of the data structure is guaranteed.

The advantage of this concept is that authors are not isolated from each other because each state transition gets immediately visible to all co-authors.

The disadvantage is that the clustering of a set of such state transitions into a single transaction is not supported — this is not treated as an atomic unit. It follows that one author can not e.g. create five new hypertext nodes and ensure that nobody is able to delete the first one while the fifth is being created. Hence authors are not equipped with the ability to process larger units of work in isolation and to make the results visible when they consider them to be stable. This disadvantage could be resolved by supporting operations *Begin-Transaction* and *End-Transaction* to the authors. The decision to make results visible would be up to the authors. Depending on the concurrency control method this approach would lead in most cases to locks of long duration (2PL).

3.4.4.1 Cooperation Modes

The described collaboration modes need further investigation and are thus described in detail:

3.4.4.1.1 individual mode In this mode an author is working alone in one composite node respectively activity space [26]. This means that any author working on an arbitrary composite node that is not currently visited by other authors works in individual mode although this work is part of a larger cooperative effort.

The case that one author is the only worker on the contents of a composite node cannot cause conflicts. The data is completely available for a single user without any access restrictions due to synchronisation needs. In individual cooperation mode the specific author is able to execute the generic operations that are available on the hypertext structure and described in section 3.4.2.3. The individual mode assumes that an author works on disjoint data that is not involved in the current work of other users.

[25] We distinguish consistency of elementary data structures and consistency of the document structure. The first one is preserved by the system, the latter one has to be preserved by the authors.

[26] Remember: activity spaces as well as the whole project are implemented as composite nodes

3.4.4.1.2 loosely coupled mode The loosely coupled mode supports the work of more than one author in a composite node. When multiple authors are editing the contents of the same composite node, they share this part of the hyperdocument structure. When two authors are accessing the same composite node, SEPIA automatically sets up a loosely coupled session among the concurrent users. A session is at least constituted by two co-authors. This can be dynamically extended or reduced over runtime. If only one author remains in a session the loosely coupled session is finished and continues as an individual session.

To deal with the concurrent access of shared data there have to be concepts for ensuring consistency of the authors work. SEPIA tries to solve this problem with a typical CSCW approach. Authors are able to recognise presence and action of their co-workers in the same composite node (group awareness). This gives the authors the ability to detect conflicts by themselves and coordinate their current work. Nevertheless each author can navigate independently through the document.

Before SEPIA performs operations on objects, they have to be selected. If an author selects a node or link, the corresponding object is marked and is displayed in a different colour on the authors display. This can be treated as a user-lock mechanism. Simultaneously the collaborating activity space browsers are notified to change the colour of the selected object (activity markers). Locking of atomic nodes is used to avoid counter-productive actions like moving a node by two different authors. The name of the lock-holder is always displayed on the corresponding nodes. Furthermore SEPIA shows a status line with the names of all participating users. When a user enters or leaves a session, the co-workers are additionally notified with a doorbell.

This can be seen as a social coordination protocol that notifies authors about the nodes their co-workers have touched and about new as well as leaving participants (group awareness). The notification takes place on the user interface level (visualisation of activity markers, status line information, doorbell). This protocol delegates the responsibility for the correct interleaving of performed actions to the authors.

It has to be mentioned that there are no explicit access permissions on the node level depending on the authors tasks resp. roles in the overall process. This cooperation mechanism is based on the assumption that authors are trusting each other.

In the loosely coupled mode the authors can disappear independently from the common work-context by leaving the composite node or the corresponding activity space or by entering another composite node.

On the user interface level each author has his own view (content and size of the window) and can scroll and resize it independently.

3.4.4.1.3 tightly coupled mode The tightly coupled mode offers the opportunity for a very close cooperation between a bumber of authors.

A prerequisite for using the tightly coupled mode is that the involved authors are editing a common composite node in loosely coupled mode. To change into the tightly coupled mode

one author has to ask his co-authors explicitly for a joint session. If they agree the tightly coupled mode is established.

While the loosely coupled mode synchronises authors that may work on more or less different tasks, the tightly coupled mode is moreover regarded as a tool for solving common problems together. Hence, more support of real cooperation is required instead of awareness facilities for synchronising independent tasks (loosely coupled mode).

In order to perform synchronous cooperation on a shared composite node the following functionality is provided:

- Shared views by enforcing synchronised browsers:
  The editing of a composite node in tightly coupled mode is supported by SEPIA by enforcing shared views for all participants. Therefore, all browser windows in tightly coupled mode have to display the same part of the composites content and must have the same size. Scrolling in one of the browsers must lead to scrolling events in every co-authors browser. This means that all changes to window sizes and scroll bars are immediately broadcasted to all authors. New windows on the user interface level that come up by opening the content of a node appear on all screens simultaneously. This makes sure that all authors view exactly the same parts of the hyperdocument structure and the same node contents.

- Telepointer:
  A telepointer is a cursor controlled by one user that appears on all of the collaborating user screens. A telepointer option for pointing on objects visible in the shared view is provided in tightly coupled cooperation mode. The system accomplishes telepointers by visualising the mouse pointers of all participants. The different telepointers are marked with the names of the authors. It is not possible to perform actions with telepointers.

- Meta-communication facilities for additional information exchange:
  If authors are working in tightly coupled mode it might be necessary to have further communication facilities. Therefore SEPIA provides the usage of digital audio and video channels. When a meta–communication channel is established telepointers help to make make communication more natural.

  On the structural editing level this is of course only meaningful if shared views are enforced by using the tightly coupled mode.

For concurrent editing of atomic nodes, the external graphic editor WSCRAWL is used. In tightly coupled mode WSCRAWL is used in joint editing mode.

3.4.4.2 Mode transitions

The SEPIA system implements automatic transitions from the individual to the loosely coupled mode. The tightly coupled mode can only be reached on user demand.

3.4.4.3 Cooperation on the level of atomic nodes

Beside the different cooperation mechanisms on the hypernetwork structure that are described above, SEPIA provides tools for editing the contents of atomic nodes. The graphic editor WSCRAWL, the AudioTool and the picture editor XV are external applications and thus not part of the SEPIA implementation itself.

Text-editor: In the current implementation of SEPIA, there is no cooperative integrated text-editor. That is why there is up to now no joint editing for text nodes possible. The text-editor in the current version of SEPIA deals with a time-stamp based concurrency protocol. While different users are editing concurrently the content of the same text node, the user who saves first wins. The concurrent work of other users will be lost in this case. There is no notification mechanism implemented that informs other users about the problem when the first user is going to finish.

Graphic Editor: The external graphic editor WSCRAWL supports joint editing. Multiple authors can edit the same graphic together in a special joint editing mode. The edit mode is selected when the application is launched.

If two authors are editing the same node in the single editing mode the changes of the user who saves last are valid. WSCRAWL does not notify its users when they are working in single user mode on the same document.

For atomic nodes with graphical contents the graphic editor WSCRAWL supports telepointers, too (joint editing mode).

3.4.4.4 Comparison of cooperation modes

The following table summarises the differences of the SEPIA cooperation modes. Besides the different cooperation mechanisms on the level of editing the hyperdocument structure it shows also the different behaviours of the editing tools on the atomic node level.

	individual mode	loosely coupled mode	tightly coupled mode
precondition for activation	editing of a composite node by one author	editing of a composite node by multiple authors	editing of a composite node by multiple authors and successful request for tightly cooperation
activity coordination	—	group awareness (activity markers, doorbell, status line of composite browser)	group awareness and telepointers; additional meta-communication channels
concurrency: elementary operations	serialisation of client operations by SEPIA-server client operation executes conceptually as one transaction (move_node, create_node) notification of updates (broadcasting mechanism)		
user interface	independent view and operations	independent view and operations	enforced shared views
text nodes	Text-editor with optimistic time-stamp protocol		
graphic nodes	WSCRAWL (single mode)	WSCRAWL (single mode)	WSCRAWL (joint mode)

3.4.5 Implementation

3.4.5.1 Architecture and Implementation

To get a better understanding of SEPIA's cooperation facilities SEPIA's underlying system architecture needs further investigation.

The current SEPIA implementation is based on the client/server concept.

On the server site resides a single user database acting as a hyperdocument database for storing objects persistently. The single user database executes one request after the other and has no concurrency control component. The different client requests are queued in an additional system component (bottleneck server) that acts as broadcast component (figure 3.9).

The client site consists of the user interface and a local workspace carrying all objects that are needed for the actually performed work. The local client workspace is treated as cache because the clients have transient object copies in their workspaces. These copies are visualised at the user interface and thus are subject to the user interaction.

For supporting the social cooperation protocol the hyperdocument objects carry special activity marker information in additional attributes. Additionally each composite node carries a list of users that have currently opened the node. These additional data are used for providing group awareness as well as establishing communication channels between users.

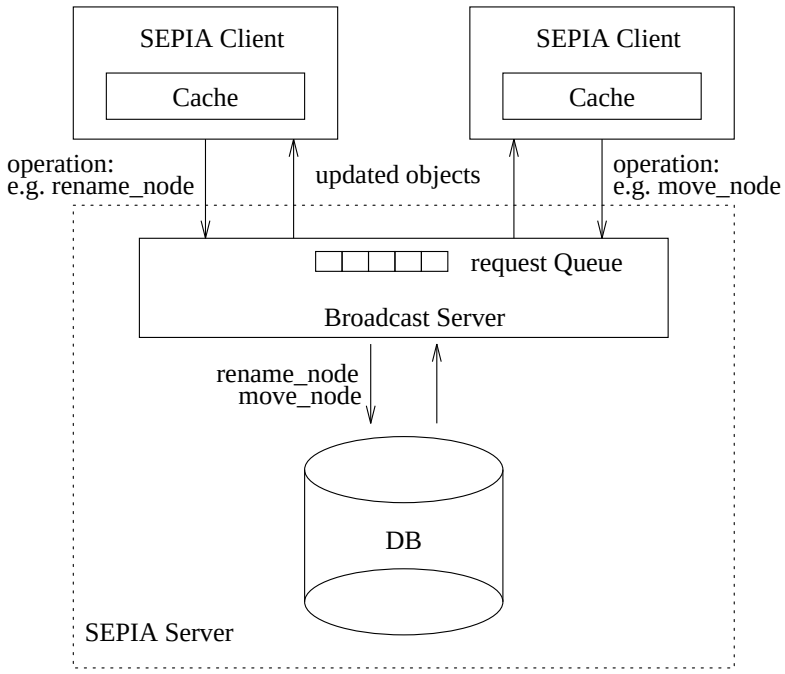


Figure 3.9: SEPIA architecture of current implementation

On top of the single user database, a broadcast component is placed that communicates with the SEPIA clients. The broadcast server receives change notification from any SEPIA instance modifying an object. It broadcasts this information to all other clients. These messages are received and processed by *Broadcast Listener Processes* that are associated with each SEPIA client. This mechanism ensures consist views on the document among cooperating SEPIA clients.

The bottleneck component on top of the database receives all client requests and queues them before they are directed to the database. The queueing of operations guarantees that server requests cannot interleave. This policy preserves the consistency of the basic datastructures representing the hyperdocument. Due to the fact that server requests are always operational units that map the database from one correct state to another correct state anomalies can not occur .

E.g. a loosely coupled session is set up when an author opens a new browser on a composite node which is already in use by one or more co-workers. The new browser retrieves the contents of the node from the database and updates the list of concurrent users (attribute in the composite node object). This update is broadcasted to every SEPIA client. The Broadcast Listener Processes are processing with these incoming notifications by performing server requests that load the new node values into the clients cache. Afterwards the node contents is updated on the screen. Thus the execution of modification actions by one author is broadcasted to all clients and the client application itself retrieves the new values from the database. This mechanism may corrupt actual user requests on the client that are not yet submitted to the server.

The described mechanism enables the dynamic adding and removing of users to running

sessions. This concept is the basis for forming ad-hoc groups as the need of the authoring process arises. The serialisation of client requests by the bottleneck component preserves the consistency of the elementary data structures. It is an important note that the server does not provide any primitives for executing activities of longer duration.

3.5 Cooperative document authoring requirements for the TRANSCOOP specification language

In this section, we examine the requirements resulting from analysing the cooperative authoring scenario to determine how they impact the design of the specification language.

A specification language should not bother scenario designers with all details of one concrete synchronisation mechanism. It is moreover achieved to derive language constructs with clear semantics that are on a high level of abstraction and thus as independent as possible from concrete synchronisation mechanisms. An advantage of such a language would be the opportunity to translate it to various transaction models providing a specific predefined functionality.

Basic concepts for a specification language that satisfies the coordination needs from the cooperative authoring environment are [RS]:

- Definition of activities and their externally visible execution states
- Definition of legal transitions between these states
- Definition of conditions that enable transitions between these states and their properties (rejectable, delayable, forcible)

To describe the cooperative authoring environment in such a specification language, the different authoring activities and their logical dependencies as well as the possible concurrency aspects, have to be specified.

Beside these properties, which can be found in most workflow specification languages, there are some requirements that are characteristic for the cooperative authoring domain, e.g. the need for specification facilities to describe notification events. Events can be database updates as well as user interface events. For close cooperation in the authoring scenario the simultaneous processing of user interface events is required. Furthermore it should be possible to distinguish the different data copies in a client/server environment at the specification language level.

The derived specification language requirements are based on results of the application analysis. The analysis process itself has been performed by following the guidelines from Chapter 2. This description distinguishes three aspects that were taken into consideration during the analysis. The specification language requirements are ordered due to these aspects:

- Activity aspects: this category deals with all questions that belong to the structure of the working process. This means the decomposition of work, dependencies between different tasks, control flow issues etc. The objective is to ensure a correct processing of work according to the application semantics.
- Data aspects: the goal of this category is the investigation of data that is processed by the application. It captures requirements about availability of data and synchronisation needs. Furthermore a possibly required distribution of data is analysed, e.g. properties of distributed data copies were investigated.
- User aspects: this category captures organisational aspects. The relationships of users are reflected as well as access restrictions depending on the users role in the embedding organisation.

Each of the following requirements were assigned to one of the above categories.

3.5.1 Requirements reflecting the structure of activities

1. **Specification of the work process:** The specification language should provide the possibility to specify the activities that occur during the work process. This includes basic activity definitions, the parameters passed to an activity, the constraints to be satisfied before and after the activity executes. Furthermore it covers the transactional guarantees that are required for the execution of a specific activity.

   The description of activities by specifying their external visible execution states is a well known approach in workflow specification languages as well as in advanced transaction models for design environments. State automatons are an abstract model for specifying the behaviour of an activity in terms of state transition diagrams. This includes the specification of *termination states*. When an activity reaches a termination state no further state transitions are allowed. States that are marked as *breakpoints* define consistent activity states.

   This high level requirement will become more concrete by the following requirements.

2. **Hierarchical organisation of activities:** The natural way of working in design processes is splitting large tasks into smaller and more clearly arrangeable parts. According to this procedure the work in the cooperative authoring process is usually decomposed from activities into smaller subactivities.

   The nesting of activities reflects the hierarchical breakdown of work in design environments according to the design problem solving process in cooperative authoring. Therefore the structure of activities reflects both the problem that is to be solved and the organisation of the problem solving process.

   Each (sub)activity that is part of a larger activity has its individual state transition diagram depending on its characteristics (semantics). We can treat these state transition diagrams as the *activity type*. These and other properties have an effect on the concurrency control and recovery mechanism that can be used by the scheduler.

   Thus, the specification language has to provide constructs for structuring the work process in a hierarchical way.

3. **Application specific correctness criteria:** Serialisability is an unsuited correctness criterion for cooperative environments. To support the cooperative working process in an efficient way, new correctness criteria depending on the semantics of the application are needed. Defining correctness means to describe all correct interleavings respectively histories.

   By assuming a need for supporting activity hierarchies, correct histories can be specified along this structure. For each activity in the hierarchy the execution rules of its subactivities should be specified.

   Application semantics do not only determine the allowed interleaving. In case of failures the correct behaviour is as well dependent on the application semantics. Since activities are of long duration, treating an entire activity as an atomic transaction is infeasible. Failures should not cause an entire activity to roll back. Instead, partial rollback with forward recovery must be supported. Since an activity typically consists of several transactions, some of which may already have committed, rolling back may require compensation of subactivities. Forward recovery may involve restarting the activity or executing alternate operations. The language should allow the specification of compensation and alternate operations, and policies for rollback and forward recovery. These policies may include intervention by sophisticated users.

4. **Dependencies between activities:** Dependencies define the linkages between activities and thus form a data- and control-flow definition. Between activities there can be different dependencies expressed:

   (a) Value dependencies: the data parameters that are passed from one activity to another must be expressible.

   (b) Temporal dependencies: temporal dependencies are useful to trigger actions when e.g. a deadline has occurred. This kind of dependency is related to the external time. With interval timer dependencies certain activities can be forced to happen after a specified period of time. Thus we distinguish temporal dependencies:
   - related to an external clock
   - related to an interval timer

   (c) Commit/abort dependencies: [CR92a] introduces certain dependencies of this type. Typical are:
   - Finish dependency ($finish_1 < finish_2$): $activity_1$ finishes before $activity_2$.
   - Commit dependency ($commit_1 \rightarrow commit_2$): $activity_1$ commits only if $activity_2$ commits.
   - Abort dependency: $commit_1 \rightarrow abort_2$: $activity_1$ commits only if $activity_2$ aborts.

   (d) Conditional execution: it is useful to support enabling conditions to provide a dynamic behaviour between activities.

3.5.2 Requirements reflecting data access aspects

1. **Propagation of changes:** When multiple users are working simultaneously on a common artifact they should be able to observe the changes made by others under predefined circumstances. A short response time to reflect the own changes on other user

interfaces and a short notification time, which is the time required for these action to be propagated to everyone's interfaces. This is a strong requirement for cooperative authoring environments because the authors may work on related document parts carrying out content-dependent changes that can not be detected by the system in all cases.

The specification language should provide constructs to define notification events. These events can depend on the status of data as well as on the status of concurrent activities. When the work process requires that concurrent changes become immediately visible, there is a need for corresponding specification support.

Events that causes notification are not necessarily database updates. In the SEPIA system cooperation on the user interface level is provided by simultaneous processing of user interface events (broadcasting of user interface events in SEPIA tightly coupled cooperation mode).

2. **Synchronisation between different kinds of data:** In a cooperative authoring scenario, different levels of data copies have been identified. In the analysis of SEPIA we distinguished three kinds of data copies that can be characterised as follows:

   (a) **Data in the persistent storage system:**
   This data is persistent and usually contains all changes. In SEPIA every operation on the client is sent to the server and executed on the server data. The main properties of the server data are that it is durable and shared.

   (b) **Data in the client workspaces:**
   The client workspaces contain copies of the server objects that are actually accessed. These copies are manipulated directly in the client and sent back to the server. There are two ways of communicating with the server: sending the executed client operations to the server and re-execute them or sending the changed object data of the server. Advantages and disadvantages of both mechanisms are not discussed here. The data in the clients workspace is typically private and volatile. Due to the fact that objects are first manipulated in the clients workspace, these data is more up-to-date than the server data.

   Having in mind that in design environments usually long lasting operations are being processed, it is meaningful to have persistent client data. The advantage would be that recovery operations could be executed in the clients workspace with less server access.

   (c) **Data on the user interface:**
   The users view is based on this third category of data. It usually presents the workspace data considering some presentation issues. This third kind of data has to be taken into account because it is directly manipulated by the user. When working cooperatively on a document, updates to the users view can cause notification events and should thus not be processed deferred in other workspaces (tightly coupled cooperation within SEPIA).

   The specification language has to take these different copies of data into account. It should provide facilities to distinguish them in two different ways. First respect is the relationship between the persistent data, the client workspace data and the

user interface data. Updates to one of these copies may cause changes to the others. Second point is the desired consistency between different client workspaces resp. user interfaces. While the first aspect could be treated as vertical synchronisation, the second reflects horizontal synchronisation issues.

User interface events and the related data are processed by components of a cooperative system architecture while the properties of client and server data are of importance for the transaction model.

3.5.3 Requirements reflecting structural aspects of the organisation

1. **Assignment of resources to activities:** Operations of an activity may be performed by humans or other organisational resources. The same person can dynamically assume different roles for different operations. Access constraints and organisational (e.g. role resolution) policies must be defined to govern the assignment of resources to operations.

2. **Group support:** As described in the cooperative authoring scenario there is a need for supporting pre-planned and ad-hoc formed groups. A group is a set of users that is characterised by some common properties:

   - a common activity that has to be performed
   - rules for the execution of the activity
   - rules that describe events on which group members are notified

   Usually an activity consists of subactivities and database method invocations build the leafs of an activity tree. For an activity performed by a group of authors the execution rules might allow full visibility of concurrent updates. Besides the execution rules that describe how an activity is processed by a group, group members should be notified about each others work. Therefore notification events depending on the activity semantics have to be specified.

   Furthermore a group needs besides mechanisms for information sharing through the database, external communication facilities. They are not considered by a cooperative transaction model but may be of importance for other components of the TRANSCOOP system architecture. Thus it is meaningful to provide corresponding specification facilities. E.g. in certain authoring situations it may be allowed to establish an additional communication channel.

Chapter 4

Analysis of Application Scenarios: Design for Manufacturing

The scope of Design for Manufacturing is the engineering design of complex discrete products. Examples of complex products are aeroplanes and X-ray diagnostic systems for medical purposes. An essential part of DfM is the early involvement of specialists from downstream processes[1] like production engineering and manufacturing in the upstream design process. DfM can be seen as a minimum variant of Concurrent Engineering [Ide93]. DfM deals mainly with manufacturing considerations in the design phase, where Concurrent Engineering takes many more considerations into acount. By studying cases from industry we trace the characteristics of cooperation in DfM.

In section 4.2 we give an introduction to Design for Manufacturing by comparing DfM to the traditional process of product development. In section 4.3 we give an overview of design and manufacturing activities which are relevant for Design for Manufacturing. Section 4.4 gives an overview of the requirements for the information systems to support DfM activities. The three cases that have been studied in more detail are presented in Section 4.5. The three cases are: the Philips Medical Systems case, the cooperative aircraft design case (taken from literature) and the Fokker Aircraft B.V. case. Finally, in Section 4.6 we present the specific requirements for the specification language as found in the cases.

4.1 Motivation and Related Work

Design for Manufacturing integrates as much as possible design, production engineering, and manufacturing, and, hence, corresponds to a cooperative activity of designers, production engineers, and manufacturing experts. This implies that the support of information

[1] A downstream process is a process which execution depends on the outcome of the so-called upstream process. In this section design is an upstream process and production engineering, production planning and manufacturing are downstream processes.

systems has to be integrated too. This integrated information system has to be used cooperatively by designers, production engineers and manufacturers and this imposes certain requirements on the system.

The information systems needed within the field of DfM, are still in the developing phase, due to the complexity of the problems found in DfM. Not only does DfM require the manipulation of complex data structures, but it also requires sophisticated mechanisms of cooperative interactions to support the product development process. These are needed because the product development process found in DfM, tries to combine the creative efforts of experts from different fields. Within DfM, cooperation is found in different levels of the product development project. It is found on the level of project management where people from different departments within a company have to work together, but it is also found between designers that closely work together in the detail design of a certain part.

For these reasons DfM can be considered as an interesting application area for finding requirements for both the transaction model and the specification language that support cooperative transactions.

Research in the field of Concurrent Engineering, of which DfM is a minimal variant, is done in the DARPA Initiative in Concurrent Engineering (DICE) program. The most important participant in this program is the Concurrent Engineering Research Centre (CERC) at West Virginia University. See [CR92b, Cen91] for related publications.

DfM uses the principles of Computer Integrated Manufacturing (CIM) [Wal92]. Before applying DfM, a good understanding of functional activities in design and manufacturing is needed. This is found in [RY85], which was a result of the ESPRIT pilot project 5.1/34, titled: 'Design Rules for CIM (Computer Integrated Manufacturing)'. See also [UR93].

An overview of DfM can be found in [Ide93].

More information about the 'Philips'-case can be found in [Ble94], together with an extensive overview of literature with respect to (computer support for) engineering design.

4.2 Design for Manufacturing

When engineering new products, different kinds of approaches can be used. The traditional approach towards engineering new products is one in which actions are executed in sequence. Over time, however, market demands change and the traditional sequential approach no longer seemed to be the best one. New approaches like DfM evolved.

To understand the reasons why DfM is applied, first the characteristics and limitations of the traditional process are discussed. After that, the characteristics that distinguish the DfM approach from the traditional approach are presented. In the final part of this section, the consequences of DfM for computer support are presented.

4.2.1 Traditional process

Traditionally the different specialists from design, production engineering, production planning and manufacturing work in a so-called "over the wall" approach. The order in which activities need to be executed and the persons responsible are clearly defined.

When specialists of a certain department finish their job, they "throw" their plan or product over the wall to the next department, which then can do their part of the job based on this plan or product. Usually it appears that a subsequent department finds some shortcomings or impossibilities that result in a throwback of the plan or product to one of the previous departments. Consequently, a new product or plan needs to be produced, increasing the number of product development cycles. In the end, after going through a number of cycles, a completed product is delivered.

In figure 4.1 the process described above is shown.

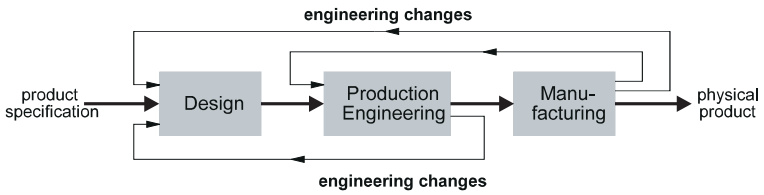


Figure 4.1: The steps in the traditional sequential product development process

The number of cycles and their wideness influence the time-to-market[2] for a great deal. Because of the existing walls in the traditional approach, there is little information exchange and the manufacturing aspects are considered too less in the designers phase. Problems in the design therefore sometimes are detected in a later stage of the product development process.

In the definition and design stages (design) choices are made which commit the final costs of the product development process for a great deal. These costs are actually incurred in the later stages of the product development process. The definition and the design stages have thus a great influence on the cost development, while the actual expenditures take place in the manufacturing phase. This is shown in figure 4.2 [And93].

Bad design choices, made in the design stage, may lead to a lot of effort (and thus costs) in the later stages to achieve the desired quality. A solution to improve the quality of design choices is to increase cooperation by an early involvement of specialist from later stages.

Besides being cost effective and producing good quality, organisations need to be flexible, be innovative and reduce the time-to-market for their products to be competitive. By being flexible, organisations are able to adapt to new situations in the market quickly, while innovativity is important as this creates opportunities to pull customers to an organisation's market.

[2]The time-to-market for a product is the time between the concept start and the commercial release of the product.

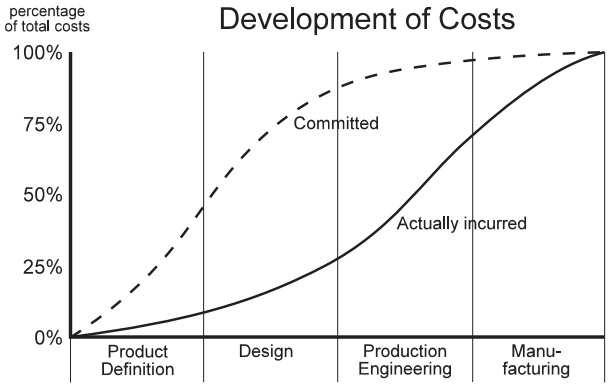


Figure 4.2: Development of costs.

4.2.2 Adapting the traditional process to current demands: applying DfM

To better fulfill market demands like low price, high quality, flexibility and innovativity the traditional sequential process needs to be adapted.

In this paragraph the characteristics that make DfM a better approach, are discussed. In the following section the activities that are part of DfM are presented.

The main characteristic of DfM[3] as opposed to the design phase in the traditional process, is the concept of *early involvement*. By involving production engineers, production planners and manufacturers in the design process, this stage may take more time. Because problems in the design are detected earlier, fewer design cycles are necessary and therefore total development time is decreased, though.

As production engineering, production planning and manufacturing aspects are considered early in the development process, products are more likely to be produced in "the best possible way". To achieve this goal, often, so-called *tiger*-teams consisting of experts of all stages are used during the whole product development process.

Because process planners and manufacturers are provided with small batches of (maybe partial preliminary) information earlier, they can start off with parts of their jobs earlier. In most cases this also leads to a decrease in the time-to-market.

As a result of the early involvement, some of the activities within DfM can, and usually will be executed concurrently. Letting people work concurrently implies that coordination and control are required. This means that frequent exchange of (mostly small batches of) information is necessary, but it also means that its members need to work in a systematic way.

The difference between the traditional sequential and the adapted approach (in which DfM is applied) is illustrated in figure 4.3.

Within DfM the different stages (and substages) of the product development process overlap. Because of this reason it is difficult to point out the exact transitions between the stages.

[3] All the other characteristics to be discussed are related to this main characteristic.

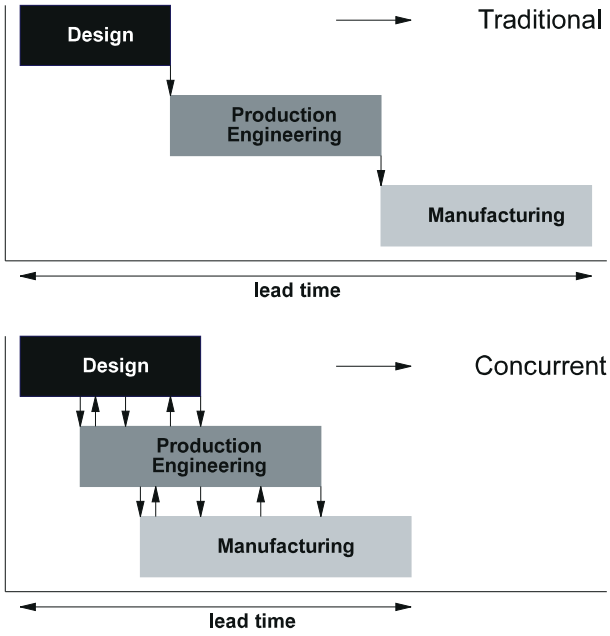


Figure 4.3: traditional process and DfM-applied-process in time

To guarantee the continuation of the product development, formal transition points are introduced. These transition points are known as *milestones* (the 'Philips' & 'Fokker' case), and they are often predefined by stating the required deliverables. They involve a formal evaluation of the status of the product development project.

Although free riding of team members and costs for coordination of the team members are disadvantages of working in teams, cooperating with different disciplines improves flexibility and the level of effort often is increased.

4.3 Design and Manufacturing Activities

The DfM approach stands for a closer cooperation between specialists from the different disciplines like design, production engineering, production planning and manufacturing. To understand this cooperation it is desirable to get an overview of activities of the participating disciplines.

The origin of DfM lies in discrete manufacturing, which is for example applied in the metal industry. Discrete manufacturing can be seen as the manufacturing of products that have a finite three-dimensional geometrical shape. Such a product is often assembled of other discrete items, which must be produced, assembled and held on stock in turn.

We realize that the type of manufacturing control (like small batch or mass manufacturing) has a great influence on the way products are designed and manufactured. Still we will not consider the differences in manufacturing control, because that would make our overview unnecessarily comprehensive.

This section first explains the main functional activities that are applicable for design and manufacturing. More detailed descriptions of the corresponding activities will be given in table form. A discussion about coordination in cooperative design ends this section.

4.3.1 Description of Design and Manufacturing Activities

The main activities in this section are design, production engineering, production planning and manufacturing. They are based on a chosen Computer Integrated Manufacturing (CIM) reference model that provides a functional description of all design and manufacturing activities with a main focus on the metal industry. The latter is principally the case in the presented tables, where is referred to specific *metal* manufacturing operations. However we have tried to abstract from that as much as possible.

The CIM-reference model that we have used, has been developed in the ESPRIT-CIM 5.1/34 pilot project [RY85]. This project was performed by Istel Ltd. (England), Centre for Mathematics and Computer Science (CWI, the Netherlands) and the Computer Science department of the University of Amsterdam (FVI, the Netherlands).

4.3.1.1 Design

The design activity starts with an informal product idea or problem description, that may come from the marketing process or from other members of the organisation. First a rough concept of the design is developed and presented in a design proposal. This proposal is reviewed by different organisational disciplines resulting in a business decision.

When the design proposal concerns a complex product or problem, work breakdown structuring must be applied, introducing coordination problems between the assigned design teams. Decomposition techniques can support work break down structuring. The design teams may consist of specialists from different design disciplines like (among others) mechanical, electrical, pneumatic and systems engineering, and specialists from downstream processes like production engineering and manufacturing. Cooperation between teams and team members is a prerequisite to achieve the final product development goals.

Design analysis, prototyping and reviewing techniques are used to ensure the correctness of the design (decisions) with respect to the previously specified requirements and the extant possible manufacturing processes. Information about the extant manufacturing processes is often present in the form of company handbooks, knowledge bases or just manufacturing people. If a designer specifies that a certain operation must be performed, it must be checked whether it is manufacturable with the present equipment or methods. Manufacturing specialists have the knowledge to evaluate the manufacturability aspects of design choices.

Design modifications must be processed in a coordinated and formal manner, because a modified product could have relationships with other products, that might be changed as well. It is mostly important that the latest version of the design specification is available.

The design teams produce detailed specifications about shapes, materials, part lists and instructions meant for production engineering and manufacturing. These specifications are stored in such a way that they can be accessed by distributed team members or by other

teams. Other processes in the organisation (production engineering, production planning, but for example also marketing) could make use of preliminary information. Management of design versions therefore plays an important role in the product development process.

A complete design specification of all parts of the product evolves in time and is used by other disciplines. This design specification is in the form of a product model.

In table 4.1, a description is given of the most important design activities.

Table 4.1: Design activities

activity	*description*
Concept design	A design proposition is made on the basis of a given product idea or problem description. Solution principles for solving the problem are evaluated and selected. The basic shape together with geometry data will be determined.
Engineering design	Based on the concept design proposition a part list of the design, material and the nominal geometry is determined.
Detailed design	The geometric modelling of all parts is worked out and a detailed part list is generated.
Design analysis	With the use of finite element analysis methods[4]and/or other modelling techniques it can be predicted by simulation if the design will satisfy given specifications.
CAD administration	This activity is responsible for the planning & control of the design process and data management.
Design modification & engineering changes	For controlling design versions it is necessary to formalise all requests for modifications and manage the consequences of engineering change propositions.
Engineering prototype build	This deals with making a prototype of the design.
Engineering test	With the use of a prototype an empirical evaluation in practise can determine the satisfaction to the given specifications.

4.3.1.2 Production Engineering

Production engineering is the set of activities that determines how the parts of a design should be manufactured[5]. This includes an evaluation of the different proposed manufacturing technologies and results in the selection of one of these technologies. For this, cooperation with design and manufacturing specialists is necessary.

In case new manufacturing equipment needs to be developed, production engineering is responsible for its design and manufacturing. This may include plant layout, material

[4]Finite Element Analysis (FEA) method is a mathematical technique to solve differential equations. FEA is used to determine stress and vibrational characteristics of a product and of many other calculations in such fields as heat transfer, thermodynamics, fluid dynamics, metal fatigue, etc.

[5]Any place in this document where manufacturing is considered, assembly could also be considered.

handling equipment and assembling equipment. When existing manufacturing equipment is used, only additional manufacturing tools must be designed and manufactured.

Production engineering takes care of the generation of process plans and manufacturing instructions. A process plan defines the manufacturing operations, the sequence of operations and the (type of) machines that must be used.

In table 4.2, a description is given of the most important production engineering activities.

Table 4.2: Production Engineering activities

activity	*description*
Process planning	The set-up of an operation plan that describes which types of operations and which machines can be used and in which sequence of operations the product can be made.
Plant layout	Determination of which machines and what formation are needed on the shop floor to reach product volume demands.
Part programming	The generation of an instruction set required to machine a component in the determined manner.
Production tool and fixture design	The design, specification (and manufacturing) of the required tooling and fixturing equipment for the different machines.
Material handling	The design, manufacturing and control of equipment required to store, transport, load and unload materials and products in the manufacturing processes.

4.3.1.3 Production Planning

Production planning is the set of planning activities that covers the range from short-term scheduling of production orders to long-term forecasting of product demands. Here it is determined how many products at what time will be made and which manufacturing equipment will be used. All this is recorded in production plans. Attempted is to achieve a balance between meeting customer order due dates and the efficient use of available manufacturing resources, like manufacturing equipment and operators.

(Preliminary) information about the parts and the proposed materials come from the product model. Generated information like production plans are stored.

In table 4.3, a description is given of the most important production planning activities.

4.3.1.4 Manufacturing

Manufacturing is the making of a product out of materials on the right equipment and according to the right manner specified in process and production plans. Depending on the product in view, assembly can be considered as an important part of manufacturing.

Table 4.3: Production Planning activities

activity	*description*
Long-term planning	The determination of the required manufacturing facilities and capacities on the basis of sales prognoses and historical (sales) information.
Production planning	The interactive process of levelling the order demands over the production planning horizon. The goal is to achieve a balance between meeting customer order due dates and efficient use of available manufacturing resources.
Production scheduling	The shop floor control activities which contain the scheduling of machines for example on a daily basis.

Manufacturing activities include in general the storage and transportation of materials, tools and (half) products before, during and after manufacturing operations.

In the manufacturing of complex products one will in general use computer controlled manufacturing systems. Instructions that may come from production and process plans (the manufacturing model) must be provided to computer systems and -operators in order to start and control manufacturing activities. Information generated during the manufacturing of products is stored and used for control and feedback to production engineering and production planning activities.

In table 4.4, a description is given of the most important manufacturing activities.

Table 4.4: Manufacturing activities

activity	*description*
Cutting tool & fixture management	The supply of the machines with the required tools and fixtures like they are specified for a certain operation and the refurbishment of used equipment.
Material management	The steady supply of the machines with the correct rough material and components/subassemblies and the knowledge about the condition and location of such material.
Part program management	The administration and control of the NC-programs and the conversion of programs in a form that can be read by (different) machines.
Production management	The coordination of all previously described activities such that all production scheduling plans can be carried out and the control of the progress of the manufacturing activities.

4.3.2 Coordination of Cooperative Design Activities

In the way we look at DfM, cooperative design is important. The aim of cooperative design is to produce a design which is agreed upon by each team member.

Team members in cooperative design can have different roles. Some of the most important roles are listed below:

- *team leader role* : The team leader is responsible for the progress and coordination of activities of the team members.
- *designing role* : Designers make and analyse design choices. When making design choices, constraints *on the design* are introduced, deleted and modified.

  Designers may come from different design disciplines like mechanical engineering, electronical engineering, software engineering, etc. Which disciplines are involved, depends on the kind of design in view.
- *reviewing role* : Reviewers evaluate and criticise design choices and propose modifications. They also can put constraints on the design, which must be satisfied. They act as checkers with the ability of vetoing design choices.

  Reviewers are specialists who could come from the design disciplines (possibly another member of the design team) as well as from downstream processes, like production engineering and manufacturing. The latter is of special meaning in the scope of DfM, as described in the previous section.

Design proceeds by successive refinement of design models. New specifications evolve in time because new insights are required or new design choices are made, which may lead to new (detail) requirements.

Successive design activities are often organised with the use of commitment steps. A commitment step is defined as a set of design choices that must be formally committed to by a given time. Formally, this means that these steps have to be recorded for safety, liability and quality reasons. Commitment steps are often embedded in formal organisational procedures.

The commitment steps can take place by design review processes. In figure 4.4 an example of such a design review process is presented. The design review process here consist of three basic steps: evaluate, comment and vote. When voted, design choices are committed by all specialists and these choices will further be used during the next step of the design process. Design reviews are often planned ahead by project management.

Conflicting constraints (requirements) on the design, introduced by different specialists, must be negotiated. This may result in relaxing certain design constraints. The negotiation on one side could take place during design reviews resulting in a formal voting procedure. On the other side (a group of) specialists can informally get in touch with each other to exchange and negotiate design choices.

From the aspects mentioned above, it is clear that coordination is important, when cooperative design is applied.

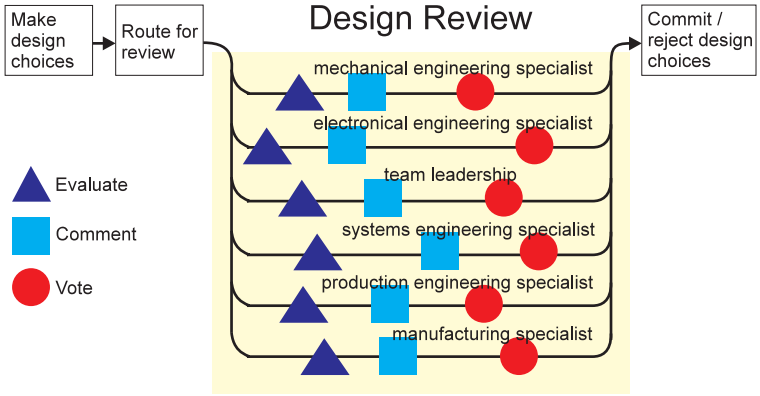


Figure 4.4: The design review process

4.4 General requirements

In the following sections we will sketch the main characteristics of the information systems that are (to be) used for supporting DfM. Section 4.4.1 deals with aspects concerned with the organisation of information and Section 4.4.2 deals with the operations in the information model. Before we look at the operation on the information itself we look at activities within project management that control these operations.

4.4.1 Organisation of the information

The product model is at the centre of the information that is needed for designing an artifact. During the concept design (cf. Section 4.3.1.1), the product model will often be represented as a single set of related data (often called product constraints) which represent the solutions that have to be taken to make a product that meets the initial requirements.

During detailed design (cf. Section 4.3.1.1) and product engineering (cf. Section 4.3.1.2) it is not always possible to maintain a unified product model, and sometimes it is not even desirable. For example in aircraft design we see that the experts from different disciplines use their own models for the representation of a wing. In [BR92] it is stated that the experts of the different disciplines do not even completely comprehend the models used in other disciplines.

When it is not possible to use a unified product model, a complex product model will be used that consists of a skeleton to which all the multiple product views are attached. Because an artifact often consists of many parts that themselves consist of parts again, the skeleton is likely to have a hierarchical structure. Within the field of discrete engineering the skeleton is commonly found in the *Bill of Materials*.

We assume that all information in the complex product model is grouped in logical documents. The concept of logical documents is an extension of the usual concept of a document, such that a certain logical document could also represent a certain view of the product. We assume that the multiple product views are represented in different logical documents. In

the remaining of the report we just use the word 'document' when a 'logical document' is meant.

The documents used during a design project can be divided into documents that contain the information of the product model and into those that contain information about the design process. Information in the latter ones is also referred to as *strategical information*. The first kind of documents we will call the *primary documents* because they are directly related to the product model. The other documents will be called *secondary documents*. These contain information necessary for project management and often refer to primary documents.

In the following sections the product model and the documents, divided into primary and secondary ones, will be described in more detail. Although we have tried to discuss all the different aspects of the informations system in different paragraphs, some overlap is inevitable, due to the complexity of the information system.

4.4.1.1 Product model

We noticed that both unified and complex product models are being used at different stages of the product design. Whether a complex product model will be used depends strongly on the nature of the artifact. In some cases several different (but related) models are used in the different design stages. We assume that the principles that apply to a single product model can also be extended to multiple models.

4.4.1.2 The unified product model

During the concept design (and possibly also during the detail design of a certain component of the product), a unified product model is used to represent the artifact (or part of it). The unified product model is often stored in a single document, for example as a 3-dimensional drawing or as a (product) constraint database. Because of the dynamic way in which the unified product model is manipulated by one or more engineers, there is usually no versioning available. It may happen that several variants, representing different solutions, may exist in parallel as long as no final decision has been made made about the solution that is chosen.

4.4.1.3 Product constraints in the unified product model

There are many ways to categorise product constraints. There are approaches in DfM that consider each datum that defines the product in a certain way, a constraint. We will not follow that nomenclature here. Because we are focussed on describing the information model, we will divide the constraints into *implicit* and *explicit* constraints.

Implicit product constraints are the constraints that are implied by the modelling techniques that are used to represent the product model. If, for example, solid-modelling[6] is used it

[6]Solid modelling is a technique by which the geometrical properties of an artifact are represented by a number of 3-dimensional solids.

means that the product is represented by solids. The data representing a certain solid have to meet certain criteria to be correct. It is usually impossible that two solids overlap in space. This restriction leads to all kinds of explicit constraints. The kind of constraints that can be expressed by implicit constraints are a result of the predefined data-structures used to represent the data in combination with the explicit database constraints that apply to these data. The checking of the implicit constraints is often done automatically by the tools that are used to manipulate the data. An example of the latter is static type checking in a typed specification language.

Explicit product constraints are constraints that have to be expressed explicitly by means of a (logic) specification language. These constraints can be of arbitrary complexity. Explicit constraints are mostly used in the preliminary design stages. In some cases, testing the constraints can be a complicated task, like for example an aerodynamic constraint of a wing. For simpler problems, constraint managers are being used which are able to reason with constraints that are represented in a certain form.

There is another classification of product constraints, orthogonal to the one above, that we would like to mention. The classification is the following. Some constraints simply may never be violated (like "the plane has to be such that it can accommodate at least 70 passengers"). This may seem obvious, but there is another class of constraints: the *deontic* constraints. For example: "the distance between two passenger seats should be at least x meter". We want deontic constraints to be satisfied, but their violation does not cause disasters to happen. You could also call such constraints non-fatal.

It is also possible that some predefined activities have to be carried out upon notice of the violation of a deontic constraint. E.g., when the constraint about the distance between seats is violated, one could decide to diminish the number of seats in the plane or to make it longer. We call this *corrective actions*.

4.4.1.4 The complex product model

Once the amount of information becomes too large, the product model is often represented as a skeleton and a set of related documents. The skeleton can be viewed as a set of objects (components) and relationships between them. Because most artifacts have a hierarchical structure, the skeleton product model will reflect this hierarchical structure. In other words, the skeleton consists of a tree that describes how the artifact can be divided into parts, which themselves can be divided into parts, etc.

In some cases this tree is also used as the production model that states how the parts are assembled together. This is, for example, the case with COPICS (Communication-Oriented Production Information and Control System, cf. Section 4.5.1.4), which is actually a logistic system used during production engineering and manufacturing in the 'Philips'-case. We observed that this system was also being used during the detailed engineering of their products where it serves as a skeleton for the product model.

In other cases the subdivision is standardised by a coding system that gives a unique code to each of the components. These coding systems have a hierarchical structure were a new coding is extended at each level of subdivision.

We note that in some cases, where a line of products is developed, it is possible that the trees of different products have whole branches in common. In this case it is not correct to talk about trees, but is it better to talk about Directed Acyclic Graphs (DAG).

4.4.1.5 Explicit relationships

Besides the top-down relationships between the components and their subcomponents there may also be horizontal relationships between components on the same level. These relationships are often called *explicit relationships*, because they are not implied by the hierarchical structure. The explicit relationships can be between sibling components (e.g. components that are part of the same (super)component) or between components in completely different branches. It is even possible that there are relationships that are not strictly horizontal, but cross the levels in the hierarchy. An example of this is the relationship between the packing material (used only for shipment) and a certain small part of the artifact, that has to be assembled (installed) on site. The packing material is usual on a high level, while the part could be at a very low level.

4.4.1.6 Versioning of the skeleton

If for some reason the skeleton of a complex product model has to be changed, a version mechanism might[7] be needed. The COPICS system, which was used in the 'Philips'-case has a limited versioning system per component in the hierarchical product model.

4.4.1.7 Product constraints in the complex product model

Once the product model is represented by objects and links, the product constraints are likewise divided. There are constraints that apply to a single component (we call these intra-component constraints) and there are constraints between components (we call these inter-component constraints). This division, however, is not obvious, because an intra-component constraint on a certain component can be an inter-component constraint between two of its subcomponents. The inter-component constraints can be divided into those that appear between siblings (components that have a common super-component) and those that are associated between explicit relationships. Often the explicit relations are identified because of existing inter-component constraints.

It might be clear that the representation and the management of constraints in the complex product model is not a trivial problem.

[7] We speak about explicit versioning, which should not be confused with versioning as a well-know technique for enforcing correctness in a concurrent transaction. This remark applies to the whole report.

4.4.1.8 Primary documents

As we saw in the previous section, the complex product model consists of a skeleton and documents containing specific information of the product model. This information includes the product constraints as well. We will use the concept of a document in a much broader sense than only documents that are printed on paper. We would rather define it as a collection of information that is represented and modified as a whole. Examples of documents are: bills of material, solid models, any kind of tables but also sets of explicit product constraints.

Each primary document can be associated with a component (or an explicit relationship when it specifies the interface between two components). Each component can have several documents associated with it, which contain information about different aspects of that component. E.g., a piece of computer hardware could have a document describing its form, a diagram explaining the purpose of its connectors and a formal specification of its functionality. Documents themselves can make references to (sub)components.

4.4.1.9 Versioning of the documents

Especially in the detailed engineering stage of the design process, there is a need for versioning of documents. If more than one version of a document is kept in the information system, versions can be divided into: current, historical and being-updated versions (working copies). Several versioning mechanisms are possible. Most versioning mechanisms allow only a single version to be the current version. In situations where co-operation support is needed, it might be possible that more than one working copy of a document exists. Not all systems keep historical documents on-line available.

Especially during concept design, different alternative solutions are studied in parallel. For this reason parallel versions of a document may be needed.

Special attention should be paid to the versioning of documents where the contents is built (retrieved automatically) from the contents of other documents (cf. Section 4.4.1.11). The version of such documents should depend (by some method) on the versions of their source documents.

4.4.1.10 Constraints between documents

In a certain way the same things that apply to a unified product model, regarding constraints, also apply to documents that represent a certain view of the product or deal with a certain component in the product.

As we saw, the constraints can be divided into intra- and inter-component constraints. Analogously, we can also make a distinction between intra- and inter-document constraints. From the assumption that all information (including constraints) is kept in documents, it means that all inter-document constraints have to be kept in documents as well.

Inter-document constraints lead to inter-document dependencies. This is especially important with respect to the versioning of documents (cf. previous section). Inter-document dependencies are related to the methods in which new versions of documents are made current. It seems to be logical that versions of documents can only be made current if the documents are correct with respect to the constraints. This is important in the context of views: sometimes, (part of) a document will be a view on another document. Consistent versioning is a prerequisite to keep information consistent in such situations. To guarantee this, explicit database constraints could be used to represent the dependencies between the documents.

4.4.1.11 Views

Within the complex product model, different views on the product amy exist. These different views are usually represented in different documents. In some cases these product views are derived from each other. For example, 2-dimensional drawings could be extracted from a 3-dimensional model. This causes a document to be a 'database' view of another document, resulting in an explicit relationship between the documents.

4.4.1.12 Secondary documents

All the documents that contain information that is not directly concerned with the product model or the production process, are considered to be secondary documents.

The secondary documents contain a whole range of documents, from official reports used in project management to private e-mail messages exchanged between two engineers. They can be divided into documents that have to be stored permanently in the information system and documents that are non-persistent. Important aspects of the project management such as workflow, task division and planning will be recorded and be communicated in secondary documents.

In some cases it is difficult to determine whether a document is primary or secondary. For example the results of a complex calculation can be considered as secondary information when it is purely derived information or as primary information if it represents an essential property of the product module.

4.4.2 Operations in the information model

It will be clear from the previous sections that the information systems that should support DfM, are complex. To summarise all possible operations in such information systems will likely be impossible. As these information systems are used by teams of engineers, a high degree of concurrency in the operations performed on the information system, is very likely. Clearly, there is a need for supporting co-operation between the users of the information system.

The operations can be divided into operations related to project management (including all communication between team members) and operations on the product model. But before we describe these operations, we first discuss scenarios on different scales of the project development process.

4.4.2.1 Scenarios

Within a product development process many tasks can be identified that have to be performed to achieve the desired result.

Not all the tasks are of the same level of management (scale of interest). It is possible to identify several levels of management (or scales of operation) with in a development process.

How a certain task has to be performed can be described by a scenario. This means that scenarios are found on different levels of management within the product development process.

In this sense, a whole product development project or a stage of the project, can be considered as a single task. Nevertheless, it is not very useful to look at scenarios at this scale.

At the scale of documents, scenarios to create and modify documents are often implemented through a versioning mechanism. The process of making one or more documents actual, is considered as the execution of a single scenario instance. Because documents depend on each other and on the product model skeleton, these scenarios can be complicated and can have a clear co-operative element when negotiation is required.

On an even lower scale we find scenario instances within documents. These are needed when documents are modified by different users. A document, for example, may consist of a traditional database that contains specific information or it may consist of a complex drawing with multiple views that are manipulated independently. Traditional database transactions can be considered as a limited form of scenario instances. It often happens that a drawing is used as a reference drawing for another drawing. It may happen that two users who work on different aspects of the same component, want to see each others drawings concurrently. In all these cases (co-operative) transaction mechanisms are required to support the scenarios at the different levels of management.

4.4.2.2 Project management

Project management is important for the success of the project. Because the operations on the product model are all directly or indirectly controlled by project management, we will describe the different activities of project management first. We remark that certain project management activities depend strongly on the information stored in the product model.

A primary function of project management is the allocation and effective application of resources. Human resources form the most important resource for the product development process. Project management will focus on the formation of project teams and the

organisation of the activities in the project team. For large projects it is possible that several independent (sub)teams are used, which all take care of their own project management.

In the following paragraphs, we will focus on a number of specific functions and activities that are part of project management.

4.4.2.2.1 Project lead function The project lead function is responsible for the progress (and quality) of the product development process. For this reason it needs information about the status of the process; it should take the appropriate measures if needed. Important points in the product development process are the transition points between the different stages. The process lead function is also responsible for settling the matter in case of unresolved conflicts.

4.4.2.2.2 Reviewing and Sign-off activities The purpose of a review is a quality check of the proposed plans/designs. Therefore an evaluation of design decisions takes place by experts and other responsible people.

A review may be formal or informal. A formal review is planned by project control at formal stage transitions of the product development process. The state of the plans/designs is frozen while being examined. The outcome is prescriptive for the further progress of the product development process. The intention is not to backtrack to design decisions made before the formal review.

Informal reviews are meant to increase the quality and to decrease the uncertainty of design decisions during design activities. Here plans/designs are signed off by the participating reviewers. For every different design, a different group of reviewers may exist or even be formed dynamically. After a sign-off, a design or plan can be released.

4.4.2.2.3 Task identification At the beginning of each stage of the product development process, tasks have to be identified that have to be performed to ensure a successful completion of the stage. Also in the middle of a stage it can be necessary to rearrange tasks or to subdivide tasks into subtasks. Task identification relies on information stored in the product model.

4.4.2.2.4 Task assignment Once tasks have been identified they have to be assigned to team members. In some cases new team members have to be involved in the project. The identification of tasks will often be recorded in secondary documents. Formal tool support may be necessary. Most tasks will involve the creation of (new versions of) documents.

4.4.2.2.5 Planning Closely related to task identification and assignment are planning activities. The complexity of planning activities is influenced by the complexity of the product model, the number of activities and the dependencies between the activities.

4.4.2.2.6 Workflow management Especially when a product development project has a large team, workflow management becomes necessary, especially when the team members are distributed geographically or make use of distributed systems for the storage of the documents. Workflow management involves all the activities that are needed for the distribution of (primary) documents between team members and the notification of team members (often through secondary documents, like schedules and review reports). Workflow management will be dealt with extensively in Chapter 5.

4.4.2.2.7 Authorisation Authorisation is closely related to task assignment and planning. Whether authorisation is needed depends on the size of the project. Authorisation includes all activities that determine who can see and/or change what information. This should also include the abilities of managing tasks and the abilities of assigning authorisation to other team members. Authorisation can also be used to prevent the loss of information.

4.4.2.3 Negotiation

An activity that deserves special attention is the process of negotiation. Negotiation takes place on different levels of the product development process. Some forms of negotiation are clearly on the level of project management, others, however, can also be seen as part of the product model management. For some forms of negotiation it is not clear on which level they belong, or they shift from one level to another.

Negotiations at the level of project management will usually be performed using formal negotiation procedures (see also Section 4.3.2). Formalised forms of negotiation will be needed when experts from different disciplines have to work together. All team members will contribute to the product development process with their own expertise, concerns and constraints. A common problem is that experts do not have the same mental model of the design and that experts do not speak the same "language".

Negotiating is often a lengthy and iterative process. The process starts with a set of multiple conflicting goals or assertions. Whenever a proposal has been made, all participants in the negotiation process must give feedback to each other about which parts of the proposal they agree or disagree on. In order to arrive at an agreement, suitable modifications must be proposed. It is necessary to predict and to evaluate whether a proposal is narrowing the differences. Justifications and arguments for modifying proposals must be generated, communicated and recorded.

4.4.2.4 Recording design history

Because each design process involves a large number of deliberate and unconscious decisions, it is important to record design decisions. In our division between primary and secondary documents, the design history is part of the secondary documents. For practical reasons it will often be recorded as part of the version management of the primary documents. Design history is needed for back-tracking in case errors are found. Recording design decisions is also useful to prevent reconsideration of alternative solutions that already have been rejected in an earlier stage.

4.4.2.5 Product model management tasks

As we saw in Section 4.4.1 the organisation of the product information is complicated. This and the fact that this information is modified concurrently by the team members (preferably in a co-operative manner), requires an effective transaction mechanism.

In the following paragraphs we describe the different categories of operations that can be applied to the product model.

4.4.2.5.1 Constraint management Especially in the early product development stages the properties of the product are specified by the use of explicit product constraints, which are derived from the product specification. In a later stage, these explicit constraints are 'implemented' in documents (like drawings and solid-models).

When constraints are expressed explicitly, they are usually stored in a constraint management system. Of course, the contents of this system can be considered as a document (or as a number of related documents), but this is not an adequate approach, because the constraints are often modified and there is a high degree of co-operation. For these reasons, constraint management can be considered as a specific application area. It is an application area that receives a lot of interest in current research.

When constraints are implemented in a document, it means that the explicit constraints are expressed indirectly by the contents of the document and its interpretation. Because the constraints are expressed in an implicit way, tools are needed that check whether they are in accordance with the explicit constraints. (That this can be difficult is proven by the fact that sometimes it is done by making prototypes of the product.)

4.4.2.5.2 Modifying the product skeleton Modifications in the product skeleton, once it is being used, can have great consequences, especially when the components involved are higher up in the hierarchy. Modifications are complicated even more, because each component can have a document attached to it and can have explicit relationships with other components. When there are active transactions going on on the documents of the components involved, these might have to be aborted.

But apart from the technical problem of applying changes, they can also be expensive, especially in later stages of the product development process.

Because modification of the product skeleton is so complicated and can have far reaching consequences, it is possible that this can only be done as part of formal procedures that involve the project lead function.

4.4.2.5.3 Version management of documents The primary documents that are attached to the components in the product skeleton, usually make use of versioning. Even if there is

no explicit versioning, but a system that can make whole documents part of transactions, an implicit versioning mechanism is available. This includes the situation where a document is made in a local workspace (cf. Section 4.4.2.5.6) and, after being accepted, transferred to a global workspace.

Typical operations on versions of documents are the creation of new working copies and releasing working versions (by changing the 'being-updated' status of a version into 'current'). It is clear that all the current versions of documents should be consistent, both internally and with respect to all related current versions. Changing the state of a number of versions that belong to one or more documents is a complex operation that involves consistency checking (of implicit and explicit constraints).

We noted that in some systems releasing versions of documents is regulated by project management procedures which includes a formal sign-off by authorised team members.

Because of these reasons, advanced transaction mechanisms are required to execute these operations.

If names of team members are attached to working copies, it is also possible to find out who is working on what. Versioning could also be used to prevent that two team members create working copies of the same document (or of closely related documents) without knowing it.

4.4.2.5.4 Checking procedures As we noticed before, checking whether a document is in accordance with the explicitly stated product constraints can be complicated. In aircraft development we see that this is done by creating a specialised model (a finite element model for example) on which a certain computation is applied to calculate some property of the design. In this particular example only the results of the calculation will be made available to other team members, because the data required for the computation are not interesting. Whenever working versions of documents are released, some kind of consistency checking will be performed.

We conclude from these examples that checking procedures are used to guarantee that no mistakes are made during the product development process. The checking procedures are also used to detect conflicts that have to be resolved by means of negotiation. It will be clear that for a large development project, formalisation of these procedures will be necessary.

4.4.2.5.5 Negotiation in co-operative tasks In the previous section we already discussed negotiation as part of the project management activity. Besides the forms of negotiation that take place on the level of project management (through a negotiation procedure), negotiation is also found within co-operative design tasks. This kind of negotiation is informal and loosely structured. To support this kind of negotiation, 'blackboard' discussion systems might be used. The designers should also be able to investigate alternative solutions and be able to discuss these with other team members.

4.4.2.5.6 Global and local workspaces The idea behind global and local workspaces is that team members first create some documents in their local workspaces and if they are ready with their work, transfer them to the global workspace, so that they are visible for everybody. Transferring documents to the global workspace can be compared with releasing work copies of a document. In some cases however, team members (or subteams) do not want to make their documents available to the whole team, because they are either too large or contain specialised information that cannot be fully understood by the other disciplines. Also, secrecy policies may be involved here. There could also be technical reasons why some documents cannot be made available to every team member. The mechanism of local and global workspaces can be realized by a combination of version management (that allows work copies) and a proper authorisation mechanism.

4.5 Specific DfM cases: Fokker, Philips and Cooperative Aircraft Design

By studying cases from industry we will try to trace the characteristics of cooperation in DfM. This section describes selected cases from industry and from literature. First, we describe the case Philips Medical Systems. Afterwards, a case from literature about cooperative aircraft design is presented. We conclude with the case of Fokker Aircraft B.V.

4.5.1 Case Description Philips Medical Systems

One company that deals with cooperative activities in engineering design is Philips Medical Systems [Ble94]. In this section first an introduction to the business activities of Philips Medical Systems is given. After that, the structuring of the development process and the project team approach are presented. The information system COPICS is shortly discussed. In the final part of this section cooperation aspects are presented.

4.5.1.1 Introduction to Philips Medical Systems

Philips Medical Systems is an international oriented product division of N.V. Philips' Gloeilampenfabrieken. The headquarters of Philips Medical Systems are located in Best, the Netherlands.

Philips Medical Systems produces high quality image handling diagnostic systems and X-ray equipment. It develops large complex non-consumer systems in small batches of 50-200 per annum. Development projects take about 2-3 years, while the life cycle of products –without development time– is usually longer than 5 years.

Besides these systems, customers are also supplied with world wide support and educational services. Research, development, manufacturing and supply are situated in several international centra. Philips Medical Systems in general only develops and assembles products. Manufacturing is mainly subcontracted to other divisions and to (external) companies.

In one of the Business Groups of Philips Medical Systems, professional X-ray systems are being developed for cardio, vascular and surgery disciplines. Such complete X-ray systems consist of several products developed by different Product Groups. The following products can be distinguished: the image detection system, image display system, table, control desk, X-ray tube with generator and stand.

4.5.1.2 Structured Development Steps: Systems Management

Within Philips Medical Systems the development process is structured and controlled with the help of an in-house developed management method named *Systems Management*. This method is comparable with the systems engineering approach.

Systems Management divides the development process into seven stages in which certain activities are carried out by project teams. These stages are:

1. Policy Study
2. Feasibility Study
3. Overall Design (or Concept Design)
4. Detail Design
5. Engineering, Integration, Testing & Market Preparation
6. Preproduction, Market Introduction
7. Production, Sales, Installation & Maintenance

Each stage is formally closed with a review, based on a report that contains all results and the (consequences of) taken design decisions. All relevant departments who participate in the development team make contributions to each report. One of the results of the review is a go/no-go decision, taken by the management.

Systems Management aims at involving all departments in each stage of the development process and to enhance communication and commitment.

We will limit ourselves to look at stage 3, 4 and 5 which are represented in figure 4.5. The steps belonging to those stages will be described in more detail, among others with the use of an example.

Tilt table example *As mentioned earlier an X-ray system consists of an image detection system, an image display system, a stand, a control desk, an X-ray tube with generator and a table. Here we take the development of a new kind of table as an example.*

Within an X-ray system the function of a table is to position a patient on such manner that physicians easily can access the patients body, and that the X-rays can reach every part of the patients body. Therefore

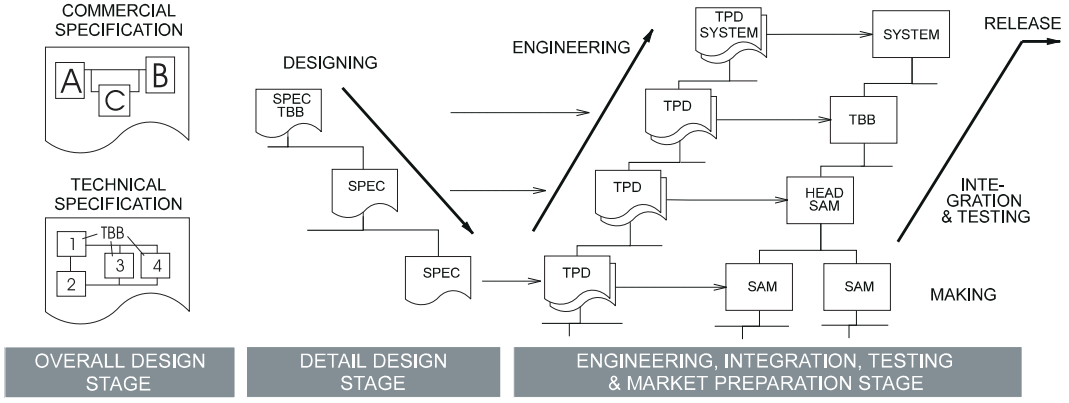


Figure 4.5: Stage 3,4 and 5 of Philips' Systems Management

a big C-arc is developed which contains the X-ray tube and the image detection system in the two outer ends of the arc. This C-arc, which has the ability to rotate in many directions, can be moved around the table, both manually and motorically. Because of that, it is possible to make several X-ray intersections of the patients body at different places with different angles.

For several medical purposes the patient should be canted while being examined by an X-ray system. Therefore a new kind of table, called a tilt table, has been developed. With this tilt table the patient can be motorically tilted in different directions while at the same time the C-arc is moving around the table. Because a predefined relationship exists between the C-arc movements and the tilting of the table, control mechanisms are being used to adjust the trajectory of the C-arc on basis of the table's position. The control systems of the table and the C-arc are thus coupled and form part of the interface between the table and the rest of the system.

This new kind of table is a redesign and contains new and old parts. The tilt table can be used with several types of X-ray systems, which all behave as modular building blocks. Of course additional features have been developed for the tilt table, which are not described here.

4.5.1.2.1 Overall Design The Overall Design stage starts with input from the Feasibility Study: the commercial requirements by means of the application functions, the commercial product modularity, the price and the introduction point in time. In the overall design stage the technical specification is developed and added to the commercial specification.

The following activities are performed in order:

1. A Requirements Specification on system level is drafted.

   In this specification first the purpose, the application area and the main characteristics of the system are described. Furthermore technical requirements on the functional, physical, performance, safety, environmental, manufacturing and logistics properties of the system are stated. These technical requirements are added to the commercial ones which are provided from an earlier stage.

   Tilt table example: from a safety point of view a requirement could be that a mechanism is needed to prevent a collision of the tilt table (with a patient on it) with the rest (the C-arc) of the X-ray system. This requirement and others must be identified in the requirements specification.

2. This Requirements Specification is discussed and approved (validation) after multiple walkthroughs[8] by the team members.

3. A Design Specification on system level is drafted.

   The desired system is partitioned into Technical Functions (TF). TFs are assigned to Technical Building Blocks (TBB) which accomplish the required technical functions. Interface definitions, which describe the relationships between the TBBs, are made.

   Tilt table example: the tilt table is one of the TBBs which form together the whole X-ray system. On a high level, a technical function of the tilt table is to let the patient lay down easily, while technical functions of the C-arc could be holding the X-ray tube and image detection system in line and positioning the C-arc. Another technical function, the complex control of all subsystems, deals mainly with the design of software. The geometry interface definition between the technical building blocks is a good example of an interface: the table and the C-arc must geometrically fit and not collide.

4. This Design Specification is approved (validation) after multiple walkthroughs by the team members.

5. A Verification Specification is drafted to prepare the testability.

   Here it is described how the the system will be tested to prove that the system satisfies the requirements from the Requirements Specification. Three separate verifications are distinguished, namely physical tests (dimensions, materials, subitems, etc.), functional tests (aspects stated in the Requirements Specification) and the beta-test (related to service and customers aspects).

   Tilt table example: a method is defined to test if the mechanism that prevents collisions really works. This means that a test suite of the X-ray system with tilt table is configured and e.g. determined at which angles of the table other parameters should be monitored during testing.

6. For each TBB a Requirements Specification is drafted (TBB level).

   What has been done at system level, is repeated at TBB level. In other words a refinement of earlier decomposed parts takes place. Besides earlier defined requirements and technical functions at a higher level also the introduced relationships between TBBs must be inspected and reviewed.

   Tilt table example: The required tilting function in the table leads to new requirements on a lower level, such as the need to have a motor drive for tilting, the need to control this motor and the need to supply this motor with electricity. Another (more detailed) requirement at TBB level could be that the table must be positioned at a height of at least 1.2 meters above the ground.

7. For each TBB the Requirements Specification is approved (validation) after multiple walkthroughs by the team members.

As output of the Overall Design stage the following can be recognised:

–> Plans which describe the tackling of the project, the quality assurance and the configuration management.

[8] A walkthrough is an informal meeting for discussing a document. One of the authors initiates the walkthrough and collects the remarks, which are further processed.

–> Stage reports which describe among others the commercial and technical product families, the commercial product description, the system specifications and all TBB specifications. In fact all design results with argumentation are collected.

–> Management Summaries.

After the Overall Design stage a formal review is organised. All reports and plans are evaluated. The goal from the team's point of view is to pass this evaluation without any changes.

4.5.1.2.2 Detail Design The Detail Design stage starts with input from the Overall Design stage. The following activities are performed in order:

1. TBBs will be further decomposed and detailed in products: for each of these products Requirements, Design and Verification Specifications are drafted and validated in the same order. Drawings are made. All this may happen in parallel with the necessary agreements and control.
2. If relevant these steps will be repeated on lower levels.
3. Detailed experiments, testing, simulation activities may be performed and prototype models of parts are made. This is done to decrease risks and failure problems with respect to the designed parts.
4. Management of Technical Product Descriptions (TPD) take place, including the management of interface documentation.

Tilt table example: As described before the tilt table is build up from new and from already existing parts. The drawings of existing parts are modified if necessary. Drawings of the new parts are generated. An example is the specification of the (new) motor drive: on basis of this specification the most satisfying motor could be selected, which will most likely be purchased.

As output of the Detail Design stage the following can be recognised:

–> Plans which describe the tackling of the project, the quality assurance, the market introduction and the configuration management;

–> Stage reports which describe among others the detail specifications, the test specifications, capacity and activity plans, and the spare parts lists. All stage results are collected in these reports.

–> Management Summaries.

After the Detail Design stage a formal review is organised. All delivered reports and plans are evaluated by a team of independent technical experts and management representatives. Again a go/no-go decision is taken.

4.5.1.2.3 Engineering, Integration, Testing & Market Preparation The Engineering, Integration, Testing & Market Preparation stage starts with the input from the Detail Design stage. The following activities are performed:

1. The Technical Product Description (TPD) will be completed;
2. One or more (prototype) models are manufactured and integrated for testing and validating purposes.
3. After testing and evaluation eventually changes are made to improve the design.
4. Finally, after everything has been integrated and tested, the release for ordering takes place.

As output of the Engineering, Integration, Testing & Market Preparation stage the following can be recognised:

–> Plans which describe the tackling of the project, the quality assurance and the configuration management;

–> The Release for Ordering instruction

–> The Bill of Material and other parameters needed for the logistic system, named COPICS.

–> Stage reports which describe among others the test models, the testing specifications and results, the full Technical Product Description and the service and marketing approach.

–> Management Summaries.

Tilt table example: With the use of prototype parts, it is tested whether these parts operate together, and whether the specifications are met when parts are put together. Testing a prototype tilt table proves for example that the chosen electricity wire dimensions in practise are correct so that no fire can start. Also attention is being paid to the selection of manufacturers & suppliers of tilt table parts. Initial agreements are set up with suppliers about how and when the delivery of parts will take place. Through the close relationship of developers with their customers, configured (prototype) tilt tables are sometimes lent out to selected hospitals. Here the last tests are being performed, just before production starts.

After the Engineering, Integration, Testing & Market Preparation stage a formal review is organised. All delivered reports and plans are evaluated by a team of independent technical experts and management representatives. Again a go/no-go decision is taken.

4.5.1.3 Cooperation through Project Teams

At Philips Medical Systems development projects are executed with the use of project teams. In such a project team all relevant disciplines involved in the development of a product, are represented. Philips Medical Systems does not use computer support tools for group meetings yet.

In general three types of teams can be recognised: the system team, the project team and technical development teams. Several other teams can be established for short periods to work on specific issues. Formal and informal teamwork occurred frequently throughout the process. Team members report to the project leader on operational matters and to functional heads on technical aspects (matrix organisation).

Special attention is given to specifications, planning, overall progress and synchronisation of the progress of team members. Important technical choices and decisions are discussed in a meeting (walkthrough) and decided on within the project team using a blackboard.

Membership is determined by the project leader. Permanent members are the project leader, the project manager, the product manager, (the) technical development leader(s), the service product team leader, the application specialist and the quality assurance engineer. Furthermore, specialists from manufacturing, purchasing, etc., are involved.

In every project, attention is being payed to administration of technical and commercial product documents. This administration is called *configuration management* and deals with the inspection, acceptation and authorisation of documents before they are filed and released. Also distribution and identification of documents are important issues.

Different versions of documents are being kept and released by the person who has generated them. Documents change from draft to released and authorised as a project review or internal walkthrough is held successfully. The latest version is in paper form stored in a central non-digital project file and accessible to everyone. When a document is distributed to team members and stored in the project file, it becomes an official document. Each formal (i.e. prescribed in the Systems Management handbook) document is manually provided with a uniquely identifiable number that has been released centrally. Depending on the working method within a group or team, each new version of a document gets a new number or keeps its existing number.

For each component the existence of a set of 2D-drawings is recorded in a computer system. From that time modifications in these drawings must be handled formally. These concerned drawings have usually been evolved in a definitive form, before they are recorded. This computer system, which is mainly used for the management of part lists and that will be discussed in the next part, is thus responsible for the versioning of drawings.

4.5.1.4 The COPICS System

The COPICS information system is used as a logistical MRP-system[9]. It keeps track of all components needed for the assembly of products. Each component (which could be an

[9] MRP stands for *Material Requirements Planning* and is used among others for planning of the availability of material and the production activities

assembly of other components), released by the constructor, is registered in this system and has a 12-digit number as a unique component code. The last digit of the number represents the version number of the component.

The components are part of assembly trees, which explain how components have to be assembled into complete products, including packing and documentation. These final assembled products are sent to customers.

A component can appear in more than one assembly tree if it is a component that is used in more than one product.

In an assembly tree, components occur that are bought from suppliers or that are assembled by Philips Medical Systems itself. The first ones are labelled with a code '4', the latter ones are labelled with a code '1' or '9', depending on whether the component must be delivered from stock or not. Besides end products, some subassemblies can also be produced at stock. This especially happens if these components are used in more than one end product or when it is cheaper to produce them in batches.

With each component comes a set of drawings, possibly of different types. From each kind of drawing several versions can exist, but there is only one current version (overlapping versions do not exist). The version of a drawing is determined by its release date. The history of the drawings is stored in the COPICS-system. Each drawing has a single author. If another author wants to update an existing drawing, the version digit of the component number must be increased.

Whenever a component is modified, in general all related components must be revised. Each component also has a list attached that describes the people that have to be notified whenever any change occurs. These people include the buyer and the planner for a code '4' component, and for a code '1' or '9' also the person responsible for the assembly.

When a component is changed, there are three possibilities:

1. A change that does not effect the functionality of the component;
   For example: a change in supplier. In this case the component number stays the same.
2. An upgrade of the component, meaning that functionality has been added;
   In this case the last digit of the component number is increased by one. This means that this COPICS system can deal with nine formal changes (1 digit) at most.
3. Any other change of (functionality of) the component.
   This means that a new component number must be created and that the new component has to replace all appearances of the old component in existing assembly trees. This usually implies that the component numbers on which this new component relies, have to be incremented or changed also. Whenever a component is changed significantly, this also implies that related components have to be changed in parallel.

Once a component has been registered in the COPICS-system, each change of it is done only after a request for change has been evaluated by the responsible engineer.

Anybody, even the assembly personnel, can suggest changes of a component. After a written suggestion has been sent to the "chief" engineer it is judged and evaluated. In case the suggested change involves a major change in the functionality of the component, the consequences are reviewed by other project members. The suggestion for a change is either rejected or accepted.

Because COPICS is a logistic system, it lacks the ability to store some of the relationships that are needed during the detailed engineering design stage of the project. The link with the drawings is only administrative and currently only manually implemented. There is no support from the COPICS-system regarding correctness checks between the drawings of related components.

The COPICS system is good in detecting vertical component relationships, e.g. the relationships that can tell which component is used in which assembled component. However, the horizontal component relationships are not stored at all. This often leads to problems where the functionality of a component is changed and the effects of this change on other parallel components is forgotten.

4.5.2 Case: Cooperation in Aircraft Design

Not only from industry, but also from literature interesting cases can be selected. In this section the case "Cooperation in Aircraft Design" [BR92] is described.

Parts of the case are modelled in a kind of SADT, a technique which offers a clear and simple graphical language for describing information and material flows. In SADT each process is modelled as a box, that has an input (ingoing arrows from the left), an output (outgoing arrow to the right), that is controlled (ingoing arrow from the top) and uses a certain mechanism (ingoing arrow from the bottom). For more detailed information, the reader is referred to [DM88].

In the next part of this section a short introduction to the case is given. This introduction is followed by a rather detailed description of the case. The detailed description consists of two main parts. First, the global development stages are considered. They are expressed using SADT. In the second part, the communication among the specialists is described.

4.5.2.1 Introduction to the case

Alan Bond and Richard Ricci have described how aircraft are being designed in a large organisation. In their paper they describe

1. how the cooperative product development process is initiated by the specification from the customer;

2. the work that is done by the different specialists who are involved in the development process, the models they use, and their interactions;
3. a typical complete scheme of design goals and stages in the development of an aircraft to prototype stage;
4. the kind of higher level negotiation that is taking place in aircraft design.

In this section the information from their paper will be used and transformed into the framework of this report.

4.5.2.2 Global product development stages

We can see design as an activity to which some input is provided that, under control, is transformed into some output by some mechanism. The *input* is provided by the customer who will usually supply the designers with a specification of the aircraft he wants to have. This specification typically consists of information like volume, storage capacity, performance characteristics and target cost. The *output* of the design activity is a full production design that serves as an input for the manufacturers. Designers and other specialists from areas like aerodynamics, structures, aeromechanics, weights/loads and costs but also from downstream processes like manufacturing and their (computer)tools are the *mechanism* that performs the actual design activity. The *control* consists of a set of models used to determine the quality of the design.

The development process of transforming the customer specification into a production development can be divided into three main stages:

1. *conceptual design*: global design of the aircraft
2. *preliminary design*: detailed design of the aircraft
3. *production design*: redesign in which new technological advances are taken into account

The relationships between them are shown in figure 4.6.

In this figure we see that the customer controls all three development stages. In all three development stages the customer has to give an approval that finalises the (conceptual, preliminary or production) design. Once these approvals have been given, the next stage can be started.

As a production contract is often awarded a few years after the preliminary design stage has been finished, redesigning is done in the production design stage while taking technological advances into account.

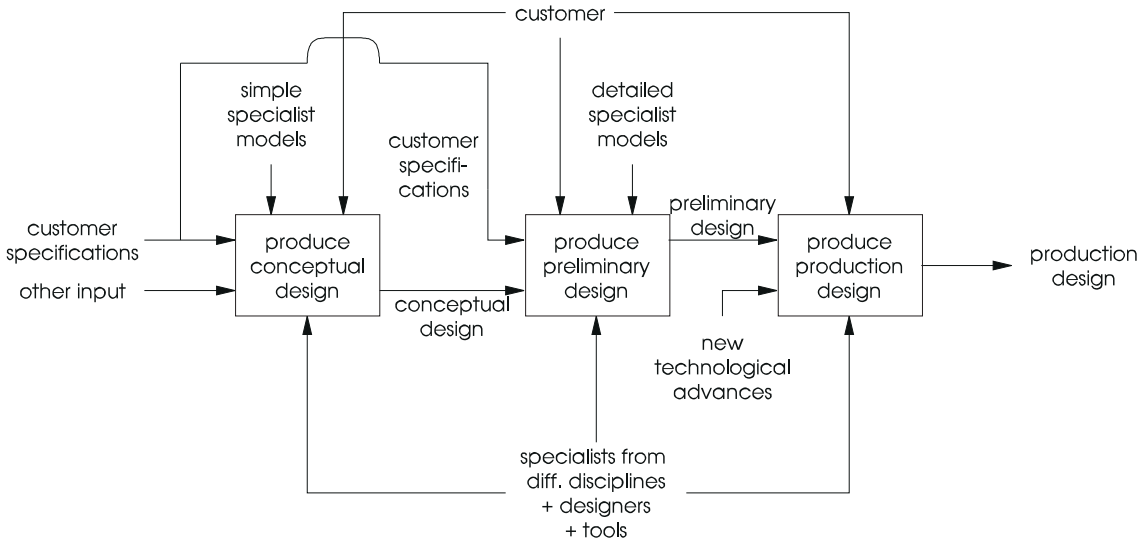


Figure 4.6: The three main product development stages

The production of the conceptual and the preliminary design is shown in figures 4.7 and 4.8 in more detail. Note the similarities of the main activities in these figures. Although the main activities are the same, the stages conceptual design and preliminary design are separated because, as said before, the customer has to give his approval for the final conceptual or preliminary design. As a result, the data in the two stages differs in level. In figure 4.7 data on a conceptual level is produced, while in figure 4.8 data on a more detailed level is produced. Also, to produce this more detailed data, more detailed models are used.

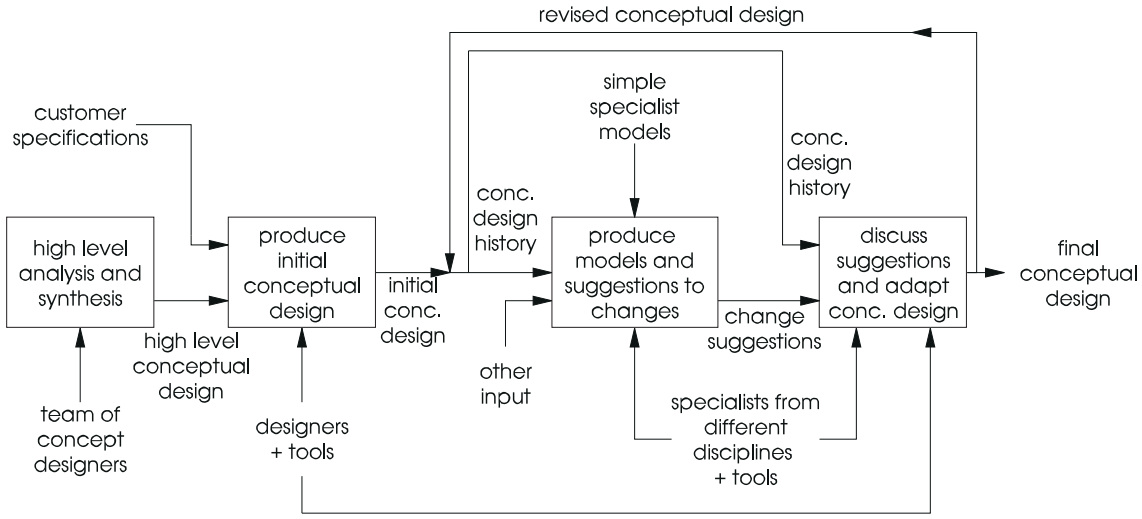


Figure 4.7: The conceptual design stage

The step "produce models and suggestions" in these figures is executed by specialists from different disciplines. The specialist's discipline determines the contribution that he is making to the design.

Some specialists (from aerodynamics, structures, aeromechanics, weights/loads and costs) use information from the common product model to construct their own specific test models. From their test models, specialists can determine whether the common product model is okay

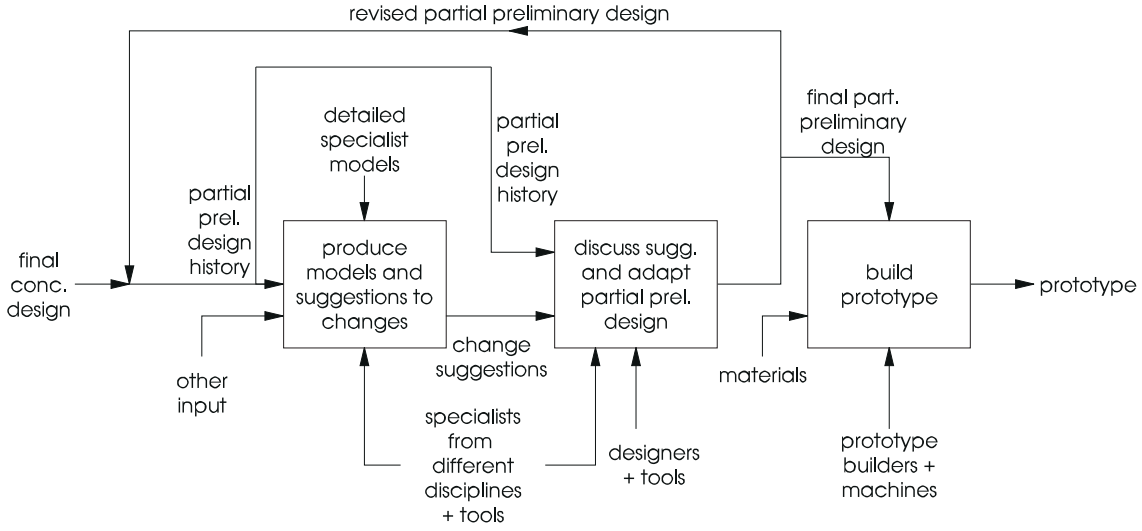


Figure 4.8: The preliminary design stage

as far as their specific knowledge is concerned or that some changes to the product model are required. Cost specialists, however, will usually not actively think of change suggestions as they do not actually help construct the aircraft. In the conceptual design stage the common product model is described on a conceptual level and therefore only conceptual information can be used as an input for the specialist's model. In the preliminary design stage the common product model contains more detailed information which can be used to produce more detailed model output.

The models used by the specialists and the output generated are shown in table 4.5.

Table 4.5: The models and their output per specialism

specialism	*model*	*model output*
aerodynamics	3 D quadrilateral grid of points	lift, drag and pressure distribution of vehicle for given set of flight conditions
structures	3 D lumped finite element model	internal loads and configuration deflection
aeromechanics	simple lumped model representing basic structures	system vibration and stability characteristics for certain flight conditions
weights/loads	lumped model representing $1g$ loading configuration, centre of gravity etc.	gravitational loading distributions and dynamic loading cases
costs	analytical cost model	cost estimates for given aspects of the design

Also specialists from manufacturing, quality assurance, reliability, maintainability and supportability will think of changes that have to be made to the product model based on their specific knowledge. The difference with other specialists is that they will not use models to compute data from which they can determine whether changes to the product model are necessary. Most of these specialists mainly contribute in the preliminary design stage.

The kind of communication that takes place in the steps shown in figures 4.7 and 4.8 is discussed in the next paragraph.

4.5.2.3 Communication among specialists

In order to let specialists work cooperatively, they have to communicate over certain data. The communication that takes place depends on the information that needs to be exchanged between specialists.

4.5.2.3.1 Interaction among specialists All specialists need to have product model data from which they can calculate their specific models. Apart from this, some specialists need additional information that is produced by other specialists when they calculate their models[10].

Structure specialists need external pressure distribution data from the aerodynamic and weight/load specialists. The weight specialists need empirical data that is based on existing aircraft and vendor data. The load specialists need pressure distribution data from the aerodynamic specialists. The aeromechanics specialists need $1g$ weight data from the weight department. This kind of data exchange takes place between the different specialists involved: not all specialists take part in this data exchange. The format in which the data are represented has to be agreed upon.

Sometimes we see that exchange of certain data between more than two groups of specialists takes place (e.g. when product model change suggestions have to be discussed). If this is the case it happens that selected specialists (representing a group of specialists) sit down together to discuss variations of the proposed changes which can alleviate problem areas. Compromises will be suggested and chosen. The designer can change (part of) the common product model according to these choices. The compromises will then be taken back to the separate departments to evaluate them. Models are adapted according to the chosen compromises and new output will be generated. From this, other problem areas may arise which then will have to be discussed. During these discussions a shared vocabulary is used (which could consist of a drawing with some additional textual information on the topics to be discussed).

The process of iteration, described above, continues until all parties will be satisfied. Besides this, the number of iterations is limited by the available development time and money.

During development of the design, the composition of the group of interacting specialists may change because of the specific knowledge of certain specialists. As stated above, some specialists (e.g. from reliability, maintainability, supportability and quality assurance) are mainly involved in the preliminary design stage. Also reasons like illness, retirement or resign may cause a change of the group of interacting specialists.

[10] In figures 4.7 and 4.8 this input is represented by "other input".

4.5.2.3.2 Negotiation and decision in design Designing an aircraft proceeds step by step by refining the specialist's models. Therefore, most design decisions taken are refinement decisions.

In each design step a commitment step is made. A commitment step is a set of joint commitments negotiated by all the design agents at the end of a design step. These commitments are based on best estimates for decision choices and they will be used by all agents during the next step. During long lasting cooperation the aim of negotiation is to find a compromise with which everyone involved is comfortable.

The step by step refinement of the design implies a top-down way of working. First, the different specialists perform a high-level negotiation process, from which lower level constraints evolve. Examples of higher level decisions from which lower level constraints evolve are customer specification and safety and other legal and government-dictated requirements. Lower level design choices depend on the lower level constraints. Usually higher level decisions are negotiable later, but because renegotiation is costly and difficult, it is discouraged.

Besides these (engineering) design decisions also decisions concerning the organisation of the project have to be made. These decisions concern topics like the deliverables that have to be produced by the different members cooperating in the project, the information every member needs to have to perform his task, the way communication is going to take place between the different members and the tools that are going to be used (including the way the data is going to be transformed into the appropriate format and its security aspects).

4.5.2.4 Summary

From the information presented above, it can be concluded that aircraft design is an iterative process of refining a common product model. The product model consists of large amounts of (complex) data. Specialist models are derived from (parts of) this product model. A new version of the product model results in new derived models. The refinement of the product model can be found again in the way decisions are made. First higher level decisions (considering both engineering decisions and project management decisions) are made. These result in lower level constraints. Only in few selected cases, higher level decisions will be renegotiated.

From the customer specifications a high-level conceptual design is made. Then, in every iteration a group of specialists from different specialisms compute appropriate detailed models from the product model available at the beginning of the iteration.

Depending on the specialism, the output of the models or simply the design serve as an input for discussions on improvement of the design. During these discussions a shared vocabulary is used. The kind of specialism of a specialist determines the contribution the specialist is making to the discussion. This signals the different kinds of roles that may appear.

By negotiating on the suggested changes, a new design, edited by the designers, will evolve. During a new iteration other change suggestions may arise which again will be discussed.

It seems to be the case that the designers are the only ones allowed to change the product model. Other specialists only are allowed to read data from the product model.

Besides these engineering decisions, another kind of decisions on which negotiation takes place is the class of project management decisions. These project management decisions consider topics that are related to the coordination of the cooperation. Tasks per cooperating member and the tools that are going to be used are two of these topics. Related to the second topic is the format in which data is exchanged between different specialists.

4.5.3 Case Description Fokker Aircraft B.V.

After the previous section, in which we described a case about cooperative aircraft design, obtained from literature, we describe in this section the cooperative development process of Fokker Aircraft B.V. This case description is based on short visits and on several reports from this company.

First an introduction to Fokker is given. After that, the structuring of the product development process is described. The management of design information and used tools are presented.

4.5.3.1 Introduction to Fokker Aircraft B.V.

Fokker Aircraft B.V. is one of four the subcompanies of Fokker Holding B.V., which has been raised in 1993 on account of cooperation between Fokker and Deutsche Aerospace AG (DASA).

Fokker Aircraft B.V. designs, builds and sells civil aeroplanes and has a leading position in the market segment of aeroplanes with 65-130 seats. The most important products are the Fokker 50 and the Fokker 100 aeroplanes. Besides this, Fokker Aircraft B.V is involved in several civil and military programs and acts as a subcontractor for other aircraft developing companies.

Fokker Aircraft B.V. is divided into operational units which are responsible for the production of electronic systems, composites/metal laminates parts and sheet metal components, and for the subassembly and assembly of components into aeroplanes.

The central engineering department is located at Schiphol Airport, Amsterdam, while the production departments are distributed over several locations. Fokker has subcontracted many of its manufacturing activities to suppliers. Besides that, Fokker is considering to cast off some of its operational production units.

4.5.3.2 Structured product development process

Like Philips Medical Systems, the development of new products at Fokker is based on the Systems Engineering concept. A division into stages is used to structure the development

process. At the end of each stage – at a stage transition – a baseline is presented and a review of this baseline takes place. Fokker distinguishes the stages and baselines as is shown in figure 4.9.

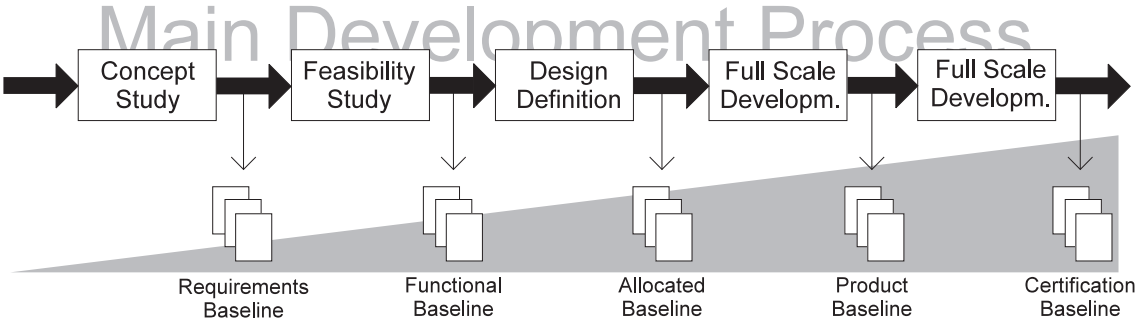


Figure 4.9: Systems Engineering at Fokker Aircraft B.V.

Each baseline consists of a set of documents. These sets of documents contain 'technical' design information as well as 'project management' related design information. Design information must be managed in the time. This means that at defined times (milestones) a structured amount of information (collected in baselines) must be available, which describes the status of the evolving design.

Although some manufacturing engineering activities already are performed parallel to engineering design activities, Fokker is experimenting with the use of design build teams. Through these design build teams, formed of specialists from different disciplines, the contribution of manufacturing experts is increased in the beginning of a development project. According to Fokker this results in a higher quality of the product and a shorter development lead time.

Because the case of Philips Medical Systems described the systems engineering project approach in more detail, we will not look at it anymore in this section. Instead of this, we will take a better look at the used information systems and information management.

4.5.3.3 Information management in the product development process at Fokker

4.5.3.3.1 Generating design information During the development stages of an aeroplane, a lot of design information is generated. Unlike project control information, the technical design information does not depend on the project or organisation form.

Fokker has recognised that in its iterative stages of the product development process there exists a lack of a satisfactory structuring of design information. Therefore Fokker actually is introducing new concepts and working methods for a better management of their technical design information.

The technical design information is generated by answering the following three questions iteratively:

- *What kind of function(s) must be performed by the product?*
  Ideas for a new product arise in various ways. With this information, the primary functions must be defined that the future product will perform. Besides that, requirements from customers or imposed by other design choices, must be stated. Functions and related subfunctions are organised in a hierarchical function structure.

- *How is each function realized? Which principle solutions (systems) are being used?*
  For each function a principle solution or system must be selected from a set of new or available technical solutions. This must be done in such a way that the solution satisfies the requirements imposed on the function. Like functions, the principle solutions are organised in a hierarchical system structure. Relationships between the functional and system structure indicate which function is fulfilled by which systems (and which system performs which functions).

  Alternative solutions (e.g. applied in the past) may be collected and made accessible in an engineering knowledge base. Sometimes a principle solution can fulfill more than one function or a function may require more than one principle solution.

  After a solution has been selected for a certain function, new subfunctions could be defined. Subfunctions thus depend on the previous 'higher' choices of principle solutions while forming a lower level of the function structure. This causes among others the iterative nature of engineering design.

- *With which (composition of) physical item(s) is each principle solution realized?*
  The split up and detailing of the system tree continues until a physical product component can be selected that realizes the concerned part of the system. Physical product components may be manufactured within the company or purchased from suppliers. If a product component does not exist already, a new one must be defined. Which product components are actually selected for a certain product, is indicated in a hierarchical product structure.

  Relationships between the system structure and the product structure indicate which products are chosen for which system.

The answers to the previous questions form different hierarchical structures (see figure 4.10). These structures reflect the iterative decomposition of functions, principle solutions and products. Because all three hierarchical structures finally represent the same product, e.g. an aeroplane, Fokker speaks about three different views on the same product.

Besides these three named views, two other views are also proposed at Fokker: the production structure and the product-support structure. During the product development process, related information is generated about the manufacturing and support process. In most cases, these two views are not complete when the product definition is released; in fact they are even supplemented and detailed when the product is being manufactured and serviced.

Generating design information means ultimately introducing, establishing, breaking and modifying the many relationships that may exists. The three views describe the product development stages that have been taken in the past (design history), so it is possible to retrieve why and how certain steps have been taken. Whenever necessary, it is also possible

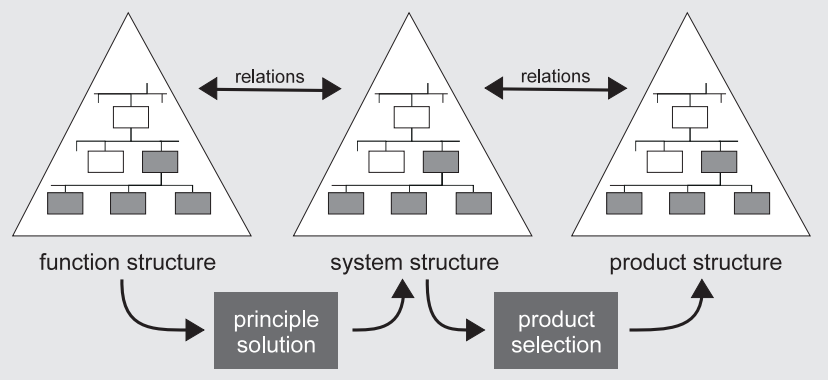


Figure 4.10: Three different structures belong to one integrated product model.

to backtrack to a previous structure and restart the (re)design process from there on. An good example is a situation where after a detail specification a chosen principle solution appears not to be realizable. Furthermore, it is possible to trace all other functions, systems or products that are touched by a redesign modification.

4.5.3.3.2 Geometry aspects of design information Besides the technical administrative information, including the hierarchical structures described before, a very important part of the design information is formed by geometry. That is because geometry defines the physical dimensions of a product which is closely related to the functions a product can accomplish.

Geometry can be modelled and visualised two-dimensionally (2D) or three-dimensionally (3D). 3D-modelling has advantages in comparison with 2D-modelling.

When a 3D-model is used, 2D-drawings can be extracted from this model, which can be seen as views on the same 3D-model. The other way around is not possible (although there are techniques to generate a 3D-approximation from a number of 2D-intersections).

During the definition of functions and requirements, it is necessary to determine the 3D-geometry (required space) and the topology (the location within the aeroplane). Therefore a space-allocation structure is build: available spaces are split up (following requirements) while at the same time chosen (sub)systems (according to system structure) are allocated to these spaces. In other words, spaces within the aeroplane are *reserved* for chosen systems in an early stage of the product development process. When an article is selected (according to the product structure) the real geometry of the product is known. At that moment it is possible to replace the space reservation with the real geometry.

4.5.3.3.3 Integrated product model The integrated product model links all structures, geometrical models and other relevant design information to each other. Fokker is planning to use a generic integrated product model which is empty at the beginning of the product development process, and which is being filled during the product development process.

In this integrated product model all alfa-numerical and geometrical design information is accessible for all users from the different disciplines.

4.5.3.4 Information systems in the product development process at Fokker

Fokker today uses many computer supported systems for their product development. These systems are shortly discussed below.

CADAM, which was introduced at Fokker in 1981, is used to produce 2D-drawings. However the use of CADAM is decreasing more and more. Because the lifecycle of aeroplanes is in general about 45 years, Fokker must still use and keep many 2D-drawings.

Fokker has been using CATIA software for 3D-geometrical modelling since 1984. The use of 3D-design is now stimulated by Fokker to a larger extent, which means that CATIA will be the CAD-platform for the next decennium.

For the management of technical information the TBE-system is being used, which holds all vertical relationships between assemblies. All relevant alfa-numerical technical information, like the parts list of components, is stored in this hierarchical TBE system. From this system the bill of materials is extracted and further used in a MRP-system. In fact this system represents the product structure, as described before.

The system EDC/1 (EDC stands for Engineering Data Control) is used by Fokker to manage formal changes of drawings. It uses the method of WORKING/VERIFICATION/RELEASE to control the release of versions of drawings. Besides that EDC manages information about documents like document identification, the author, the issue number, the CAD-file, etc.

Fokker uses a MRP-system (Material Resource Planning) to schedule the start and finishing dates at the different Manufacturing Operations units. Each of these units use their own logistical planning system, which cooperate with the main MRP-system.

4.6 DfM requirements for the TRANSCOOP Specification Language

In this section, we describe the requirements that came up during the study of the various sources as described in this chapter: the 'Philips'-case, the aircraft design literature study and the design study at Fokker. The 'Philips'-case provided especially global information on a high level of abstraction. The 'Fokker'-case offered the most concrete input. The cooperative aircraft design case was somewhere in between.

We have split this section into four subsections. This has been motivated by the structure of section 2, which distinguishes aspects of activities, users and data; we have also included a subsection with some general requirements. Sometimes, a requirement could be classified in more than one subsection. However, we have not done this; all requirements have only been mentioned within one category.

4.6.1 Requirements: activity aspects

4.6.1.1 Different levels of management

Scenarios occur on several levels of management within the product design process (cf. section 4.4.2.1). The highest level is related to project management (including workflow); the

lowest level is related to updating records in a database document (cf. database transactions). The scenarios on the different levels of management are usually not clearly linked, but the design and execution of scenarios on a lower level can be controlled by those on a higher level. Because this control spans different levels of management it is not clear whether this layering can be described sufficiently by open nested transactions [MRKN92]. On all levels, forms of co-operative transactions are needed.

Requirement:
The specification language should be capable of specifying scenarios on multiple levels of management and the way in which they influence each other.

4.6.1.2 Overview of ongoing scenarios

Because of the aforementioned multiple levels of management, the transaction model should have the ability to overview ongoing scenarios. The scenarios on one level could be considered as objects on a higher level. In current practise, the high-level scenarios are often implicitly implemented in the operations on objects.

Requirement:
The specification language should be able to express the ability of scenarios on a higher level of management to manipulate scenarios on a lower level as objects.

4.6.1.3 Hierarchical transactions

The 'Fokker'-case supports the need for hierarchical transactions: one states the need for a design methodology that is based on a structured way of decomposing [Tim94]. A complex design process often has a hierarchical structure. This requirement is related to that from the previous section. However, this form of hierarchy is much more concrete and in fact, it is provided already by modern high-level transaction models [Elm92]

Requirement:
The specification language has to allow for specifying hierarchical transactions, both in a top-down and in a bottom-up way.

4.6.1.4 History of scenarios

In a design environment, there is a need for knowledge about all design solutions that have been invented ever in the past. This way, the designers won't repeat work that has been done already or look for solutions in directions that have already proven to be useless. Many scenarios in an information system that is used to support design activities, are related to design history.

Requirement:
Specification and manipulation of historic information should be supported by the specification language. Within a scenario, it should be possible to incorporate transactions that have been executed in the past.

4.6.1.5 Milestones and deadlines

In all design projects, milestones and deadlines play a certain role. When deadlines are not met, this may mean that corrective actions have to be executed.

Requirement:
The specification language has to support milestones and deadlines. Among other things, this means that it has to be possible to specify that corrective actions are pursued when a deadline is not met. More generally, one could say that description and enforcement of project phasing have to be supported.

4.6.1.6 Timing and scheduling

The requirements from the previous section also imply the need for some mechanism of *timing*: one has to be able to specify deadlines and actions that have to be performed upon reaching a milestone. An example of the latter is making data available to other employees, because they have been approved now. The concepts of timing and scheduling are required for other purposes too. An example in the area of authorisation is that one may want to be able to deal with a situation where a certain designer may have exclusive access to certain data during a certain period of time, after which he has to share it again with his colleagues. Another example is the situation in which some action can only be performed after some other actions have been terminated successfully.

Requirement:
This all means that the specification language should incorporate the concept of time.

4.6.2 Requirements: data aspects

4.6.2.1 Authorisation

Within DfM the aspect of authorisation plays an important role. Not in the first place to control team members, but especially to prevent loss of information and to guide workflow.

Certain data can only be made available after a formal approval (cf. section 4.6.1.5). Note that this restricts the concurrent way of working in DfM: e.g., product support can only start after the engineering data have been made available. It depends on the concrete scenario at hand to what extent the concurrent process is limited by such constraints.

Requirement:
There will be rules stating who may access or modify which data (possibly for some specified time period(s)). It should be possible to state these rules, using the specification language.

4.6.2.2 Version management

In order to support the development of design documents, the transaction model should be able to support versioning techniques that are used. Versioning is applied to both the product skeleton (cf. section 4.4.1.6) and the documents (cf. section 4.4.1.9). Among other things, this also means that information can have a certain status (draft, approved, secret, public, etc.). This is related to the aspect of authorisation (section 4.6.2.1).

In section 4.4.2.5.3 we discussed aspects of version management for the documents. An important aspect of version management is the release of a document. This constitutes of making a work-copy the actual version through some kind of procedure. In the Philips case, a document is released by its creator. This generally happens after a project review or an internal walkthrough. The latest version of a document is stored in a general accessible central archive. Of course, one may also imagine a completely different scenario, where some central daemon releases the documents and is responsible for version management.

Releasing documents is strongly related to consistency matters, and thus directly related to the database constraints of the information system, and indirectly to the product constraints. We can only say that many different scenarios will occur in practise.

Requirement:
The TRANSCOOP *framework should be as flexible as possible, in order to be able to deal with as many different versioning mechanisms and version management scenarios as possible. It should at least support versioning for complex data structures where there are complex relationships between the documents to which versioning is applied (cf. section 4.4.1.9).*

4.6.2.3 Complex data structures and relationships

Complexity of the data structures is both found in the structure of the information systems and in the data stored in these systems. The structure of the information system consists of a skeleton, documents (primary and secondary) and the relationships between these. On top of this, versioning is applied to almost all information. Altogether this results in a complex data structure and complex operations performed on this data structure.

But not only the information system used to manage the data, but also the data themselves consist of complex data structures. In the 'Fokker'-case, it has been observed that 3D information plays an essential role in the design process: the design process is intrinsically a three-dimensional one. This means that complex data structures with associated primitive operations are needed for storing and manipulating this information.

Another important observation is the following statement that has been made in the Fokker document [Tim94]: "The *kind* of data is not influenced by the organisation or the project structure [...]; in other words: when the same plane is developed within a total different organisation or project structure, still the same kind of design data will be relevant." Fokker proposes to use this stable factor to develop generic design support tools.

Requirement:
From the above it follows that generally we need to be able to describe dedicated complex data structures and sets of associated primitive operations. Besides that the language should be able to express this, some standard facilities (e.g. a library) would be helpful, because it would prevent the cooperative scenario designer from starting the development on the most primitive level. Exchange of data between different data structures has to be supported. This is necessary to be able to deal with views, which is also a requirement (cf. section 4.6.2.5).

4.6.2.4 Shared data structures

Often, information is distributed physically over an organisation. Especially for large DfM projects, geographical distribution is inevitable. By coupling all available data directly (all data available to everyone) or indirectly (only title and global contents available to everyone) via a network, a database is accomplished that can be accessed from different locations and by different disciplines as one logical engineering database.

Requirement:
The specification language has to be suited for describing all different kinds of distributing and making the data available. This requirement seems to be general for all situations where there is common information to be managed.

4.6.2.5 Relationships between data

As is stated in the document [Tim94] from Fokker: "[. . .] in many places and in different stages, data are recorded that are related to each other via reuse and iteration. These relationships "cross" all design stages." An example is the generation of maintenance and instruction manuals from the product information. Another example is the (automatic) generation of NC-machine control data from the design information. These NC-machines can be used to make products, but also to generate production tools.

Moreover, it has to be possible to combine different data into one engineering database and extract certain data from this engineering database again. In all cases, also the need of the designers emerged to use each others information and to be able to cope with relationships between data of several disciplines.

It has been mentioned in the previous section that the information about the design is recorded in several kinds of documents, giving different views on the artifact that is being designed.

Finally, in the context of quality insurance, facilities are needed to "connect" measured values about really produced goods to the original design, in order to determine the need for changes, improvements, etc.

Requirement:
A system supporting DfM has to support this, i.e. specify different views and the relationships

between them, and provide mechanisms to compute and manipulate these views and their relationships.

The specification language should allow to define interfaces (between subsystems, but also between geometrical constructs like to connected compartments) in a straightforward way.

4.6.2.6 Constraints

Design information is often recorded in terms of constraints.

Properties of the product can be represented by the product constraints as we saw in section 4.4.1.3. The product constraints can either be explicit (stored as data of a constraint management tool) or implicit (implied by explicit database constraints of the data model). In the complex product model the constraints are divided over the components (cf. section 4.4.1.7), leading to inter- and intra-component constraints. Once these constraints are stored in different document, dependencies between the documents are created (cf. section 4.4.1.10), leading to document constraints.

We can conclude that constraint management (cf. section 4.4.2.5.1) is a complex task, that involves both product as database constraints. Also, for the checking procedures (cf. section 4.4.2.5.4), a mechanism for constraint validation is needed: does the current state of the system satisfy the constraints? Finally, a mechanism is needed to recognise inconsistencies and redundancies. Although we know that decidability problems may occur here, it is clear that the more help a system offers for consistency and redundancy checking, the more useful that system is.

Requirement:
Within the TRANSCOOP *framework we should be able to specify the database constraints, and possibly also the product constraints (when they are related to the database constraints). These could be part of scenarios used in formal sign-off procedures.*

We should support both the distinction between fatal and deontic constraints and the specification and handling of corrective actions.

4.6.3 Requirements: user aspects

4.6.3.1 Single user aspects

In a realistic DfM situation, there will be a number of different people involved. These people will be organised in several project groups. Some of these groups may have an *ad hoc* nature, whereas others may be formally established ones. E.g., in the Philips situation, group membership is determined by the project leader. A group has permanent members, like the project leader and the quality assurance engineer. There are also temporary members,

like specialists from manufacturing or purchasing. A similar remark can be made for the relationships between the users: some relationships will be *ad hoc* and temporary, whereas others may be of a more permanent nature. It is hardly possible to make general remarks about this: each organisation will organise its design process in a (more or less) different way.

Requirement:
The best advise we can give to the language designers is to try to be as flexible as possible. Try to support as many kinds of user scenarios as feasible!

4.6.3.2 User groups

Generally, there are groups that belong to the structure of the organisation, but not all of them do. Some groups are homogeneous, others may be heterogeneous. There are formal, (in most cases permanent) groups, but also informal ones, that generally exist only for a limited period, to solve a certain problem. The temporary groups disappear either when they have solved their problem, or when a supervisory group or person declares that the work will be stopped. In the cooperative aircraft design case, we observed that the composition of the group of interacting specialists may change because of the specific knowledge of certain specialists.

In general, the task a group has to perform justifies the existence of the group. Generally, this implies certain authorisations and duties for the group members.

Generally, design teams are heterogeneous. Group members may have different roles. In the specification language, it should be possible to distinguish different roles. However, this will mainly boil down to different authorisation for different people.

Requirement:
The same remark that has been made in the previous subsection, also holds here: all aspects of user groups that have been mentioned in the taxonomy seem to have some relevance for the project. However, the DfM application area is not specific enough to be able to make deterministic statements about these aspects. Within one particular organisation, some aspects will be very important, whereas in another organisation, the situation may be completely different. We will have to be as flexible as possible to this respect.

4.6.4 General requirements

4.6.4.1 Flexibility

A system supporting DfM has to be flexible enough to support changing requirements during the project trajectory. One aspect of this is the aforementioned consistency checking of constraints (as the system has to control the consistency of an evolving set of constraints), but there is more to it.

This aspect is one of the most important characteristics of DfM. For DfM scenarios generally take a long period of time and involve a lot of people. It often occurs during a DfM project that changes have to be pursued. This may vary from other people being involved in the same activities to a complete redefinition of (some of) the most important starting points in the design process. This may for example result in the need for different kinds of `commit` statements in the transaction model, where some committed action is only "really" final when some form of approval has been issued. A related issue is that of information evolution, where the committed data have to remain useful even after a schema evolution.

Requirement:
The specification language has to be designed with this urge for flexibility in mind, although we can't assess at this moment what exactly is needed.

4.6.4.2 Heterogeneous systems

In DfM, heterogeneous systems occur: it may be the case that different designers use different systems under different operating systems. This is not a special characteristic of a DfM scenario. It may occur in any situation where many people from different departments or organisations have to cooperate.

Another remark is that the environment of a design situation often imposes requirements on the information that is produced by the organisation or constrains the information that is provided to the organisation. E.g., Fokker has to provide certain documents in a certain form to governmental safety commissions. Another example is the fixed format in which subcontractors deliver their designs.

Requirement:
The specification language should allow the description of heterogeneous systems.

Also, a facility would be useful that enables one to describe "black-box" situations. Often, we only want to say that something is happening, without already specifying what exactly this "something" is.

It is important to be able to specify interfaces between (parts of) the organisation and external parties.

Chapter 5

Analysis of application scenarios: Workflows

5.1 Introduction to workflow systems

Workflow management became an issue in the beginning of the 80's. At this time workflow management was a part of *imaging systems*. Their main task was to *control the flow of digitised documents* through series of processing steps. In the early 90's it was discovered that automating single business tasks did not increase the productivity considerably but automating of a whole *business process* would be needed instead. Workflow management became a key issue because it was expected to enable the management of any kind of business processes. Workflow management was also seen as the solution to interoperability problems between "islands of automation" in larger corporations.

There has been no commercial breakthrough for the workflow management yet, probably because of the immaturity of the workflow products. However, a general belief that workflow management has at least the potential of being a major step forward in the use of information technology, still does exist.

5.1.1 Motivation

Workflows belong to the field of Computer Supported Cooperative Work. Traditionally they are the most predictable and well-defined form of the cooperation there is (the convey-belt approach): the rules of performing the work and the routing of work from worker to worker are well-known in advance. However, in real environments the workflows can have unpredictable, undefined parts as well. This is when the business process being automated is ad-hoc by its nature. Therefore, workflow management has a lot in common with the Cooperative Authoring and Design for Manufacturing fields but, with its well-defined cooperation patterns, gives a valuable new angle to the TRANSCOOP project.

The literature of workflow is dominated by tool vendors and therefore there is no established, common set of workflow concepts. The concepts vary immensely from product to product.

In the TRANSCOOP project, we have used a set of basic concepts to serve the analysis work. Some of these concepts are used in a slightly different way, to reflect the definitions that exist in the workflow field.

5.1.2 Related work

A workflow is a collection of activities organised to accomplish some business process. Specification of a workflow involves describing the relationships among activities and their execution requirements. An activity defines some work to be done. It may be defined in a number of ways. For example, the processing of a form, procedure, email or a transaction may be a task. As a result, there is a number of concepts which can be regarded as a model of workflow. To clarify the diversity of workflow models, we classify the work on workflow models into five categories: *non-transactional workflows, extended transaction models, cooperative transaction models, transactional workflows*, and *transaction metamodels*.

The category of *non-transactional workflows* typically includes early workflow models (e.g. [CC82, DAZ81, WL86]). Those models did not address data sharing, persistence and failure recovery. The models, however, include a notion of an activity, also called a task, a step or a procedure. The control flow between activities was specified by triggers or Petri nets. Later, it has been argued that the approaches based on classical Petri nets are not currently succinct and manageable enough to be useful in specifying workflows [LA94]. To alleviate these shortcomings high level Petri nets [Jen91, Ver93] have been intensively studied during the last couple of years.

A number of *extended transaction models* has been defined in the last years. These models focus on selective relaxation of atomicity or isolation in order to better match the requirements of various database applications. Typically in these models some form of compensation is used and global serialisability is not required as a global correctness criterion.

Extended transactions have constituent (sub)transactions corresponding to workflow activities. An extended transaction model defines the relationships among activities (subtransactions) and their execution requirements, i.e., it can be treated as the specification of a workflow. Such a workflow obeys the same correctness criterion as the used extended transaction model.

For example, a *saga* [GMS87] is an extended transaction. It consists of a set of ACID subtransactions. The application programmer defines a linear execution sequence of the subtransactions as well as as the specification of the compensating transactions. If the execution of a subtransactions fails, the system automatically invokes required compensating transactions. A *flex transaction* [ELLR90, RELL90, BEK93, KPE92] is more general than a saga. It consists of a set of activities, and with each activity a set of functionally equivalent subtransactions can be defined. If a subtransaction fails, a functionally equivalent subtransaction can be executed. The designer of a flex transaction specifies it's acceptable termination states.

Nested transactions [Mos85] extend the notion that transactions are flat entities by allowing a transaction to invoke atomic transactions as well as atomic operations. An advantage of nested transactions is that they allow the potential internal parallelism to be exploited. Variations of nested transactions are e.g. *multilevel transactions* [Wei91, WS91] and *open nested transactions* [WS92a, WDSS93, GR93]. The relaxed correctness criteria related to open nested transactions are semantic serialisability [WDSS93] and semantic atomicity [WDSS93]. These criteria allow more concurrency than the criteria used traditionally. Also the notions of *long running activities* [DHL90, DHL91] and *nested sagas* [GMGK+91] allow nesting within a subtransaction.

Asynchronous transactions have been developed for loosely-coupled multi-databases which may be in an inconsistent state for a limited amount of time. Asynchronous transactions require that interdatabase consistency requirements are specified by a collection of descriptors. These descriptors are used to convert a transaction updating a data item in a single database into transactions that spawn or trigger various activities needed to maintain interdatabase consistency. The concepts as *polytransactions* [SRK92], *active databases* [MD89] and *production rules* [CW90, CW92b, CW92a] have been developed for such an environment.

The third category, *cooperative transaction models*, are related to groupware systems [EGR91, Ell92]). *Groupware* supports a group of users engaged in a common activity. It aims to assist groups in communication, in collaborating and in coordinating their activities. Workflow systems constitute a subclass of groupware systems. Cooperative software engineering, engineering design and text editing are typical applications of groupware systems. In such cooperative environments transactions cooperate by having intersecting views, and by allowing the effects of other's operations to be visible without producing conflicts. Cooperative transactions do not provide dynamic workflow nor do they provide compensation. The concepts as *split transactions* [PH88, Kai90, KP92], *the cooperative transaction model* [NRZ92], *coordination consistency* [HOS90] and *lazy consistency criterion* [NG92] have been developed for transaction management in cooperative environments.

The category *transactional workflows* comprises the models which include a workflow specification language and which explicitly address the issue of concurrent workflows. In general, if the integrity constraints do span multiple systems, concurrent workflows may interfere each others and thereby there is a need to control concurrent workflows. To manage concurrent workflows the *Contract Model* [Wäc91, WR92] introduces a control mechanism over ACID transactions by invariant predicates which can be specified to hold across tasks, and which prevent possible interference of concurrent workflows. In the approach presented in [BDS+93] a workflow specification model [ASSR93, SR93] and an open nested transaction model [WS92a] is combined. In the proposed architecture the workflow management component enforces termination dependencies and the control and data flow dependencies within a workflow. The semantic transaction management component of the architecture manages the compensation dependencies, and the dependencies between incompatible tasks of different workflows to ensure multi-system consistency.

Finally, *transaction metamodels* provide a common framework within which one can specify and reason about the nature of interactions between transactions in a particular model. Using a metamodel, existing transaction models can be described and compared. Klein [Kle91] has shown that most known transaction dependencies can be represented already by

Create-, Finish-, Commit-, Abort-, Compensation- and Weak-Commit-dependencies. ACTA [CR90b, CR90a, CR92a] is an other metamodel. It allows one to specify transaction dependencies as well as the conflict relations of operations. The generality of the conflict relations allows ACTA to capture different kinds of type-specific (e.g. abstract data types) concurrency control. Moreover the notions of delegation [CR93] and notifying [CR92a] make the ACTA framework applicable also in cooperative environments. There is also a comprehensive framework for enforceable specification of extended transaction models and transactional workflows [GH].

5.2 Product study

The product study in this section discusses functionality, provided features and the major shortcomings of some current workflow products. Together with the case studies (later in this chapter) this study gives valuable information about the requirements still to be fulfilled in the workflow management and thus to be considered in the TRANSCOOP project.

In this study we first give a general overview of the current workflow management software and then look briefly to the future of workflow management. After that, we have included in-depth analyses of three selected workflow products. All of these three products can be considered as developed in Europe. Of these products Staffware represents the pioneers in the workflow scene being also one of the most widely used product in the world. IBM FlowMark is a new product exploiting object-oriented technology and scientific research in it. Thus it represents the future direction of the workflow products. ICL's TeamFlow is an extension to the TeamOFFICE office automation suite that is perhaps the most widely known and used commercial software product developed in Finland.

5.2.1 Classification of current workflow products

In the past two or three years the workflow technology has become a remarkable issue in the software market. Currently, well over 100 vendors offer some kind of workflow automation. Their products range from intelligent e-mail systems to workflow management systems for enterprise wide complex business processes. Workflow products can be categorised according to the tasks the products automate [McC92]):

Ad-hoc tasks are performed in unique or occasional cases like developing strategic plan or a corporate brochure, or performing some other project-oriented task. Ad-hoc workflows exist only for weeks and therefore their automation should be quick and easy. Integration of the everyday office tools and document handling is essential.

Administrative tasks are normally repetitive, simple and somewhat structured sequences of activities. Travel request processing is an example of this kind of tasks.

Production tasks consist of repetitive structured business processes that are always performed following well-defined, but often very complex, rules and policies. Production tasks tend to be business critical since often customer service and financial aspects are involved. Therefore, monitoring with audit-trail functionality is needed. For the same reason recoverability, data management and security issues are important. In many cases, a versatile mechanism for integration with other systems, both office and mainframe systems, including legacy systems, is needed. Insurance claim or mortgage loan processing are often given as examples of production type tasks.

A good example of a product suitable for ad-hoc tasks is Lotus Notes[1] which is actually a groupware product having limited (ad-hoc type) workflow capabilities.

Administrative workflow products are usually form-based and can be very well based on extended electronic mail capable of transferring different kinds of documents and data. Defining and implementing administrative workflows usually can be done without extensive training or consulting.

Production workflow products provide rich but variable sets of features and usually demand lots of training. Larger production workflow automation projects involve usually business process re-engineering and they often have to be carried through in cooperation with the consults or representatives of the product.

Recently workflow products have grown more and more versatile, thus blurring the workflow market segmentation between the aforementioned three groups.

In our product analysis we concentrate on the products suitable for production workflows. However, we keep in mind that flexibility and ad-hoc features can be valuable supplement for a production workflow software when a conveyor-belt approach is not adequate.

5.2.2 General features of current workflow products

Workflow systems for automating production workflows typically consist of the following parts:

Workflow management engine coordinates and synchronises the execution of the scenario instances according to the rules from the scenario definition. It manages the internal database(s) in which scenario definitions, user information and data structures as well as audit trail information are stored.

Workflow definition tools allows the definition of scenarios, users and data structures.

Worklist functionality offers tools for the end-users for managing, selecting and performing the work assigned to them.

[1] From Lotus Development Corp.

Administration and monitoring tools allows the definition of the security and privilege levels of users, supervision, management and monitoring of scenario instances and actor performance.

Integration mechanisms includes the APIs and other ways to integrate external applications and documents with the workflow system.

Workflow systems usually do not offer tools for business process analysis (BPA) and re-engineering (BPR). These tools have their own separate market These tools produce typically *descriptive models* of the business processes which cannot be transformed to the *executable models* used by workflow products [Mie94].

In the following paragraphs we summarise the workflow-related definitions in commercial workflow products.

5.2.2.1 Scenario definition with current production workflow products

5.2.2.1.1 Control flow A scenario is typically seen as a series of activities that are assigned to users. Between the activities there is control flow that forms a *route* for the activities. Almost all products offer more than only sequential routing. Conditional routing, parallel routing with synchronising re-joining control flows and looping capabilities are common features. Production workflow products tend to lack features that would empower end-users in their work thus leading to a kind of conveyor-belt approach. All the possible exceptions must be defined in advance. Of course, in this way many products enable scenarios and activities to be predefined so that routing is dependent of e.g. value of some variable (that can be updated in activity). Some recent products, however, make exceptions and aim more to user empowerment. For example ECHO[2] has capabilities to redo and skip activities and re-route tasks to other users. Many production workflow systems have added also ad-hoc capabilities of some degree to handle e.g. situations not considered in the original scenario definitions (e.g. ViewStar[3]).

5.2.2.1.2 Dependencies The ability to describe more detailed dependencies (see Section 5.4.2 for PortNet requirements) than sequentiality between two activities is usually absent with few exceptions (e.g. ECHO, FlowPATH[4], and ProcessWise Integrator[5] [Mie94]).

5.2.2.1.3 External events In most products it is possible to define or at least implement external events which start scenario instances or alter already running instances.

[2] From Digital Equipment Company.
[3] From ViewStar Corp.
[4] From Bull Information Systems.
[5] From ICL.

5.2.2.1.4 User definition Almost every product enables defining roles, groups or (rarely) even the whole organisation structure. However, making user or role hierarchies or defining any other relationships between users, roles and organisations are rare features. Often there is a limited set of predefined user properties and customising this set of properties (e.g. adding more named properties to users) might be impossible.

5.2.2.1.5 Activity assignment Usually activities can be assigned to users or entities consisting of users (like roles). Some products offer also activity assignment at runtime, based on some data related to the scenario instance, (e.g. field value given by actor). When a task is assigned e.g. to a role consisting of many users, then depending on the product, the work request is sent either to a shared workqueue related to the role or to all personal workqueues of the users in the role. In both cases the first user selecting the work item locks it from the other users. Many products do not have a possibility to assign a task to an ad-hoc group as this is not a predefined entity.

Some products support work balancing that enables activity assignment to change dynamically. None of the products we have seen include shared calendar or scheduling systems for users that could be used for deciding (automatically or manually) to whom a certain activity assignment suits the best.

5.2.2.1.6 Setting deadlines Most products have some kind of deadline features for activities while it is less common that they exist also for the whole scenarios. Deadline definitions can be relative (e.g. to the release of the preceeding activity) or absolute time expressions. Various actions can be taken when a deadline expires. Most commonly a notification is sent to the responsible user, the system administrator or some other supervising user. Task priority can be raised (if priorities are supported by the product). The task can be also redirected to another user or be withdrawn, maybe causing scenario instance to terminate. However, the most flexible way to handle deadlines is to let the scenario specifier set alternative actions that will be taken if deadline is ignored (e.g. Staffware[6]). In Staffware deadlines can be also conditional.

5.2.2.1.7 Data definition In addition to related documents, there is usually a need for storing and handling various kinds of data related to the scenario instance or its activity instances. Therefore products enable definition of variables of different base types and complex data structures can be built from the base types. These variables and data structures are used for storing all the relevant data for the users working with the activities as well as for exchanging data with external applications. Furthermore the same data might be a basis for conditional routing and execution of activities.

In some products all the data is stored in the data vectors or data repositories specific to instances of scenarios and common to all activity instances in the scenario instance. Usually data dependencies between the activities are not explicitly defined. An exception to this is FlowMark[7], in which data flows have to be defined explicitly (cf. [IBM94]).

[6] From Staffware Plc.
[7] From IBM Corp.

5.2.2.1.8 Definition tools Graphical scenario definition tools have emerged in products lately. They are usually based on some proprietary flowchart technique with different kinds of icons representing various activity types and arrows between the activities representing the control flow. In some products hierarchical modelling and reusing stored scenarios or sequences of activities as an activity are possible (e.g FlowMark).

In a graphical presentation, one usually cannot express complex rules, dependencies or a detailed definition of an activity. Therefore products enable refinement of definitions by form- and dialog-entries (like in CASE tools).

Few products (e.g. TASC-Flow[8], FlowMark) also offer animation (using graphical scenario definition) or simulation of the defined workflow model for testing purposes. Some products (e.g. Staffware) have a separate environment for testing scenarios before taking them in production use.

5.2.2.1.9 Definition language Most products provide some ASCII representation of the scenario, data and user definition. That representation is usually used for exporting and importing definitions e.g. for backup or for transferring definitions between different platforms. Exported ASCII type definitions are also meant for versioning, since versioning workflow definitions usually are not included in workflow products.

These representations can be considered as definition languages but most of them are not intended to be any kind of programming languages. However some products have programmable high-level languages specifically to support business procedures. Such languages are for example PML (Process Modelling Language) in ProcessWide Integrator, CTDL (Case Type Definition Language) in ECHO [Mie94] and FDL (FlowMark Definition Language) in FlowMark[IBM94].

5.2.2.1.10 Transactional properties Production workflow systems typically use some relational (or recently also object-oriented) databases for storing and accessing scenario definitions and the data involved. They are also often referred to as *transaction-based* workflow systems. This term can be misleading since current commercial workflow management systems themselves don't offer much in the area of transactional workflows. Workflow products usually offer ACID properties to an elementary activity *inside the workflow system, but these properties are usually not applied to its operations outside the system* (e.g. updates into external databases, external program executions). However, some products provide some level of concurrency control for document file handling. They still don't allow to define e.g. requirements for isolation or failure atomicity of an activity, group of activities or a whole scenario. Backward recovery relates closely to these issues and we haven't found any workflow product offering backward recovery. However, IBM has plans to include backward recovery of some degree in future versions of FlowMark.

[8] From TASC

5.2.2.2 Workflow management functionality

5.2.2.2.1 For end-users Worklist functionality offers the primary interface for the end-users for selecting activity instances assigned to them, retrieving details about the work to be done and perform the work. Traditional worklist functionality and outlook is similar to email systems (actually some of the workflow systems are bundled with mailing system). In the era of GUIs worklists have become more sophisticated with colours, icons and buttons but the base idea is the same:

- When a new *work request* related to some activity is sent to a user (by the workflow management engine) it appears into his worklist as a *work item*.
- User can select it, thus locking it (locking work-items is important when work requests are sent to a group of users with the intention that only one group member does the job).
- Some products empower users by enabling them to reject or delegate the task by sending the work request to someone else.
- In some cases selected work items can be put back to the worklist (if work requests are sent to a group of users).
- When the work is done the work item disappears from the user's worklist and the activity is *released*.

In addition to this base functionality products often offer filtering and ordering worklists as well as some degree of customisation. In addition to this there might be some additional facilities e.g. for sending messages.

Some products (like FlowMark) give users a view of the whole scenario and status of the workflow by showing e.g. a graphical representation of the scenario which includes the history. This is one way to motivate users in their work enabling them to see (and understand) the whole business process and their importance in it.

5.2.2.2.2 For system managers In addition to technical matters the workflow system manager is usually responsible for authorisation, manual exception handling and monitoring. Furthermore he is responsible for setting up and maintaining roles, organisation definitions and possibly other workflow specific system structures. This might lead to tight co-operation with scenario definers and implementors. In some workflow systems manager's tasks can be shared with some other privileged persons.

Getting the status of running scenario instances is one of the most important features of the workflow systems and actually one good reason to use workflow management systems. This is especially true in customer-oriented business processes since customers should be able to be informed about the state of their case at any time. Closely related to this is the

capability to search scenario instances or data related to them by different kinds of criteria. Almost every product offers at least some kind of monitoring tool, but they do not all offer extensive searching capabilities.

In addition to the status of single scenario instances it might be important to get a larger view of the system status. This way bottlenecks can be found, e.g. when some user gets too much workload. The system manager (or some other privileged person) can then reassign tasks to other users.

It would also be important to be able to evaluate whether changes in business processes automation have been successful by comparing the overall statistics retrieved from both old and new implementations. For this purpose there should be some metrics for measuring the productivity. This kind of advanced monitoring features are not very common yet in current products (e.g. ProcessIT[9] is an exception).

5.2.3 Standardisation and future of the workflow management

It is not wise and it is not even possible to include all the functionality into one workflow product since users can have different and very specialised needs. By developing open workflow products with standard interfaces it is possible to replace some of the workflow functionality with third party products that suit the user better. For example:

- The gap between BPR tools and workflow systems could be bridged by offering some standard definition language understood by both products.
- Worklists could be replaced with "workflow enabled" products which offer all workflow functionality needed by the end-user. This way, the end-user would not need his own workflow product in addition to the everyday software he uses for performing the work.
- Enterprises having many workflow systems from different vendors would like to have one consistent interface for administering and monitoring all their workflows.

Standardisation of the workflow systems is currently in progress by the Workflow Management Coalition (WMC). Their standardisation work includes [Swe94]:

- Common APIs to workflow services and functions.
- Interoperation between different workflow products.
- Exchange of business process definitions between different workflow models.

In WMC's model there is a central logical centre, *Workflow Enactment Engine*, that has Workflow Application Interfaces (WAPI) supporting five kinds of applications. These applications form the functionality needed in workflow management system (cf. figure 5.1):

[9] From AT&T Global Information Systems.

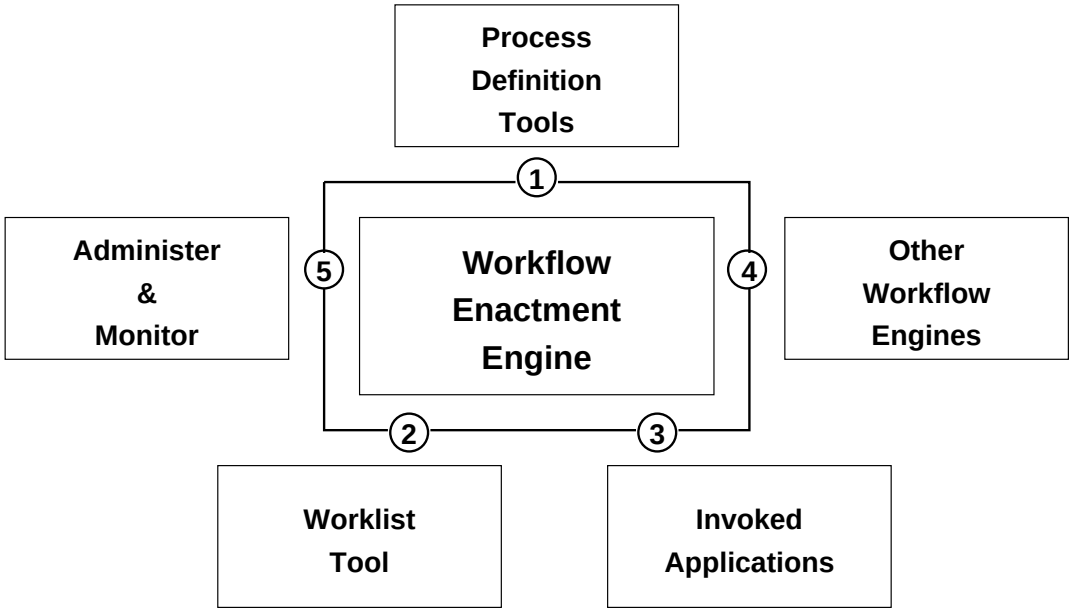


Figure 5.1: Reference model for the workflow system by Workflow Management Coalition. [Swe94]

Interface 1 for BPA, BPR, definition, modelling and simulation tools.

Interface 2 for end-user interface and "workflow enabled" applications.

Interface 3 for integrating external tools and services (interactive or automatic). Also for legacy system integration.

Interface 4 for interaction between workflow management engines (of different vendors).

Interface 5 for administering all the workflow management engines with the same tool. In the same way, monitoring and analysing all the workflows (regardless of the workflow product) would be enabled with one and the same product.

Current workflow products are considered as "umbrella" applications over the services used in business processes. The next and actually already ongoing step is to provide standard interfaces that allow other products to work together with workflow products and share workflow objects. While Lotus Notes is already a good example of this kind of development with numerous add-on products, in the future e.g. Workflow Management Coalition standards will have an important role in the development of next generation workflow products and integrating them with other products.

Object orientation is also fast becoming a key technology in workflow management systems. OO technology especially suits well to workflow objects that combine documents and data with processing rules in a reusable way. In addition to this, there are now great efforts by the Object Management Group (OMG) to develop a standard for object oriented middleware that aims to interoperability among the applications in a heterogeneous network environment.

Since the OMG consortium consists of nearly 350 companies including all major software and hardware companies, their work will have significance also in developing workflow systems.

It is also believed that workflow features will be added to the applications making them "workflow enabled" and finally workflow management would be incorporated into the operating systems. This way workflow management would come eventually ubiquitous - just another functionality in the applications. Distributed object management (DOM) standards might enable this vision also in a heterogeneous network environment. Here OMG's CORBA (Common Object Request Broker Architecture) specification might be the concept to be aware of.

5.2.4 In-depth studies

5.2.4.1 Staffware

5.2.4.1.1 Overview The first version of Staffware[10] was released in 1988 and it is currently one of the most widely used workflow products in the world.

Staffware is a client-server based product where the server always runs in a UNIX-environment, while the client (offering the end-user interface) can run under UNIX using character terminals or under Windows[11] over a local network. Staffware stores data in its proprietary flat file database but has also special versions that can be installed on top of Oracle[12] or Informix Online[13] relational databases.

5.2.4.1.2 Scenario definition In Staffware a scenario definition is a *procedure* and its instantiations are *cases*. It consists of activities called *steps* that have an *addressee* (name of the actor assigned to the step). Staffware usually has a *form* attached to each step.

Staffware procedures are defined by using character based UNIX development tools (see figure 5.2) or the Windows based Graphical Workflow Definer (GWD)[14]. The GWD modelling technique is adapted from the flowchart technique. An example of a graphical procedure definition can be seen in figure 5.3.

A Staffware graphical procedure definition consists of icons that are connected to each other with the control flow. Icons represent different kinds of steps and control structures. A *question mark* icon symbolises conditional routing and an *hour glass* icon symbolises a wait statement (Here activities to be waited for are connected by a dashed connector). *Document* icons are activities with forms attached. If a document icon has a *clock* sign a deadline is attached to it. An external event is defined with a *lightning document* icon.

[10] In this analysis we refer to the following Staffware product versions: *Staffware, version 5.1* (for Solaris), *Staffware for Windows 2.0, Staffware for Windows Graphical Workflow Definer, version 1.0*

[11] Windows is a trademark of Microsoft Corporation

[12] Oracle is trademark of Oracle Corp.

[13] Informix Online is trademark of Informix.

[14] GWD lacks some important functionality (e.g. forms design tool) and part of the development must be done still using character based UNIX-client.

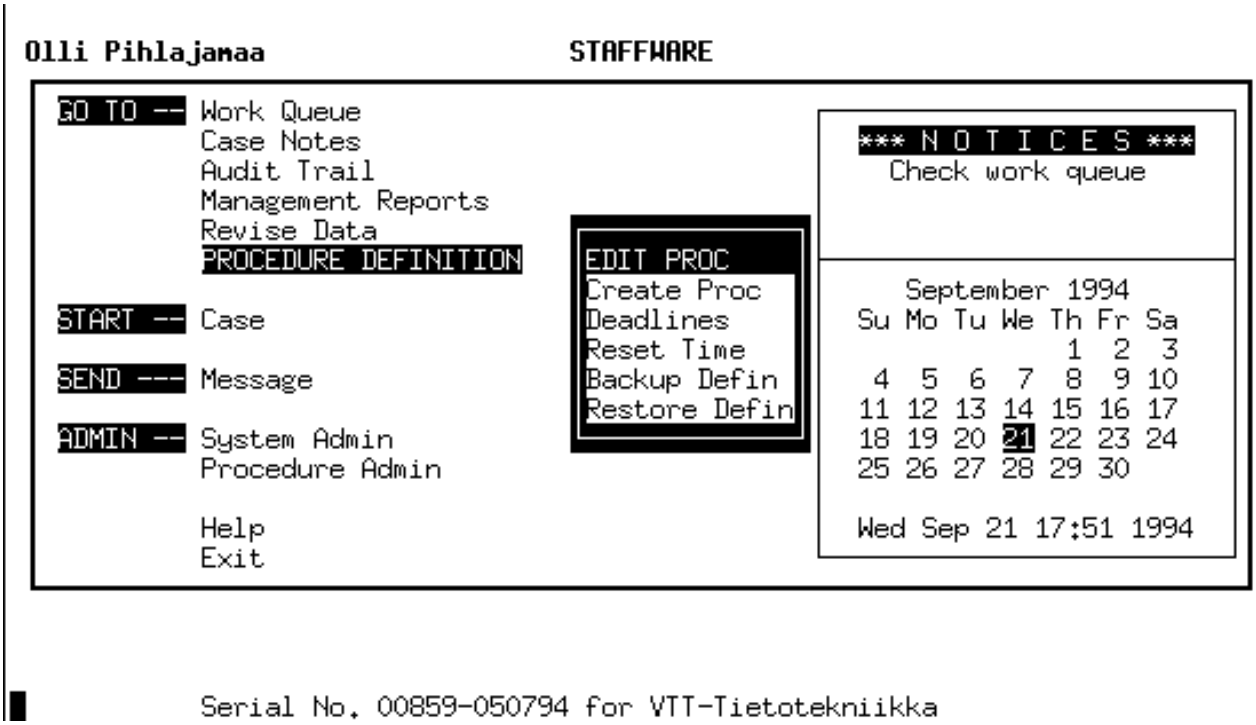


Figure 5.2: Staffware character based client software screen (system administrator menu).

Staffware procedures can be exported to "xfr-files" as well as imported from those files. This ASCII text file can be used for e.g. transferring a definition to a different platform. Staffware, however, does not have any programmable high-level language for defining procedures.

Staffware procedures are not hierarchical which means that steps cannot have substeps and on the other hand Staffware procedures cannot be reused in new procedures as steps. There are different three kinds of steps in Staffware:

- A *(normal) step* is the most common step type to which always a form is attached.
- *Events* are activities that can be used for controlling an already running case or altering the data related to it. By using events it is also possible for one case to start other cases.
- An *automatic step* executes an external program without user interaction and after execution transfers the control to the next step.

Each step consists of four possible parts: an *addressee*, a *form* that the addressee receives (in case of a normal step), an *action* that takes place after the release of the step (e.g. addressee completes the form), and, optionally, a *deadline* by which the addressee should have completed the form. In the following paragraphs we go into more detail about what can be defined in Staffware procedures. Forms are discussed later in "Data definition".

User assignment is done according to the addressee given in the step definition. An addressee can be a user, group or role. In case of the latter two, the individual users are determined at runtime by looking at the user and role definitions. The addressee also can be determined

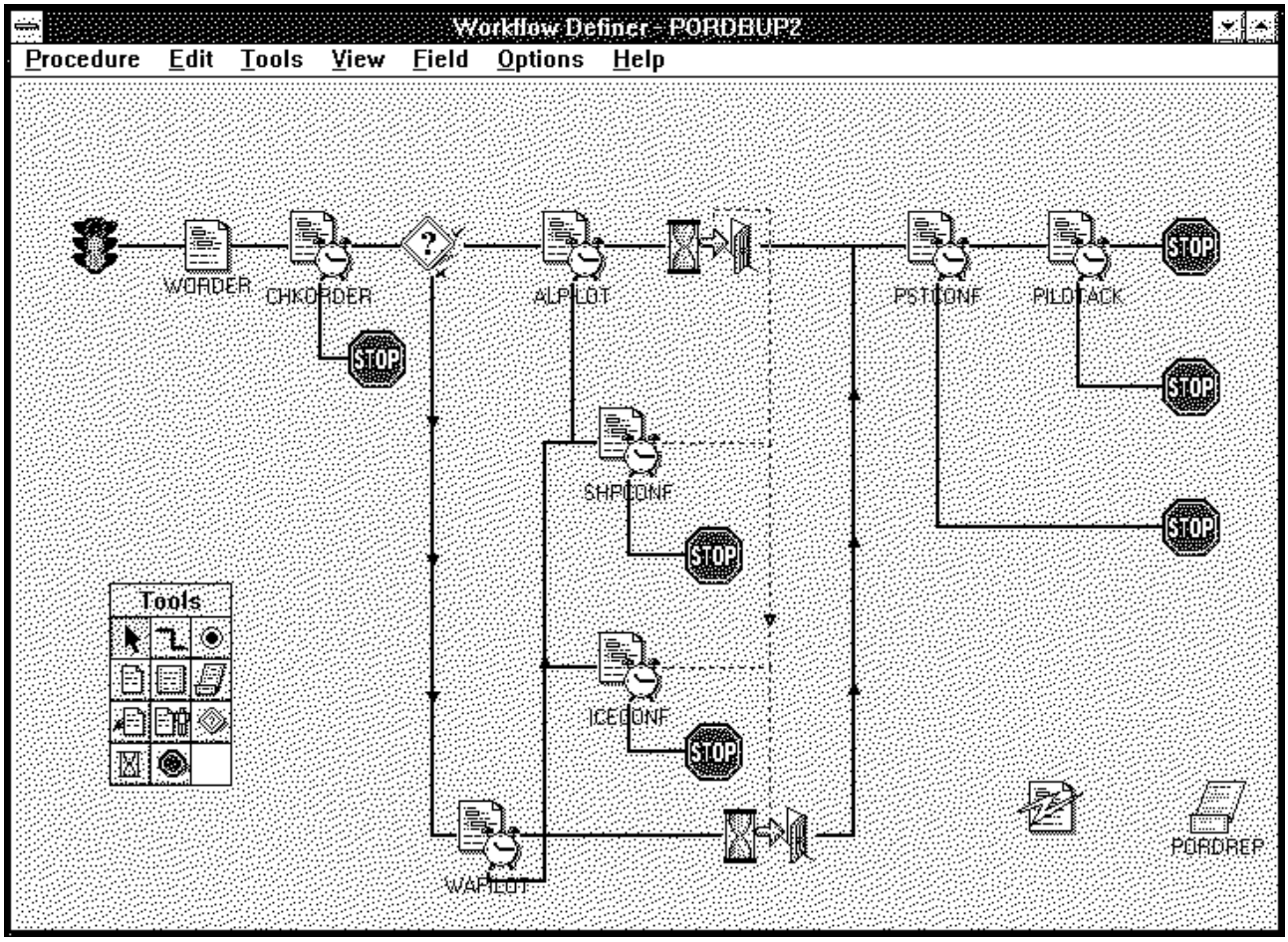


Figure 5.3: Defining a procedure with Staffware Graphical Workflow Definer.

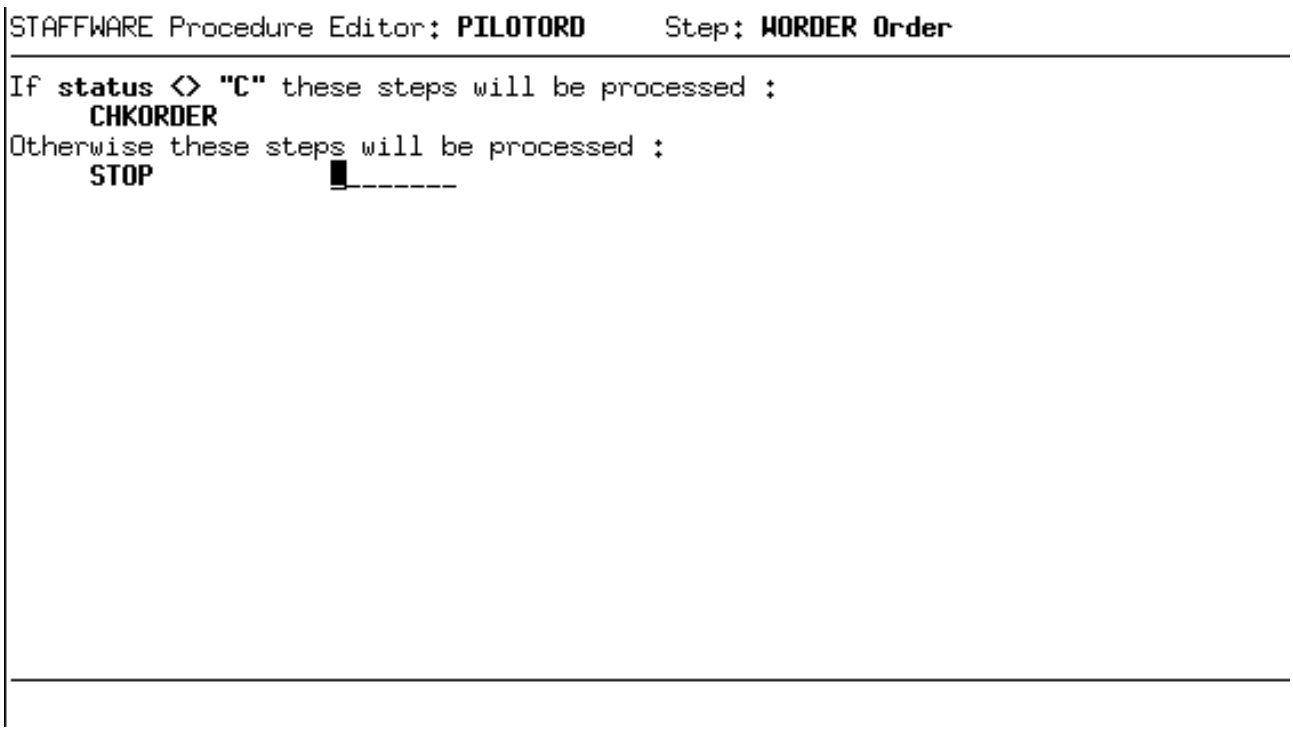


Figure 5.4: Defining the next step by entering actions.

from the value of any case variable. (This allows e.g. the addressee of the current step to decide who will be assigned to the next step if he can update the variable instance.)

Routing from one step to another is specified by actions defined in the steps (see figure 5.4). An action is simply the name of the the next step to be processed after the current one. Here also the predefined *stop action* can be given. The stop action is a special step that terminates the execution of the case. Loops can be defined by specifying the "next" actions of a step appropriately.

Forking the control flow is done by defining many actions in one step. However it is important to notice that parallel steps should not be allowed to update the same variable instances since there is no concurrency control for the data management in Staffware. This is also true when a step has been assigned individually to many addressees when they are allowed to select the same form from their work queues for user interaction (see "Data definition") and therefore have access to the common data of the case instance through the fields defined in the form. Splitting workflows to parallel steps or assigning the same step individually to many addressees makes sense in Staffware only when the step is a document without update possibility (e.g. memo) or when updating is made conditional so that each branch or addressee has access to a different portion of the data.

Conditional (rule-based) routing can be accomplished by defining conditional actions (see figure 5.4). Conditions can be based on the data of the case. In addition to this, with the *wait* statement it is possible to define the release of one step to be dependent of the release of some other step (`Release` $\mathcal{A}$ $\rightarrow$ `Release` $\mathcal{B}$). This way it is also possible to synchronise re-joining control flow branches.

```
STAFFWARE Procedure Editor: PORDBUP     Step: WORDER Order
Deadline Date Expression :

Deadline Time Expression :

The deadline expires after :  0__  minutes
                                   hours
                                   days
                                   weeks
                                   months
                                   years

If the deadline is ignored, these actions will take place :
     STOP
Withdraw the form when deadline expires?  : N

A deadline is set under any of these conditions :
```

Figure 5.5: Defining a deadline for the step.

The optional *deadline* section in the step definition provides a mechanism for entering a date or time by which a step must be released (see figure 5.5). It can also be a relative expression or a result of a calculation. Alternatively, deadline can be defined as the period of time from sending the form to the workqueue to the release of the step. The actions to be processed if the deadline is ignored are also defined in the deadline section. Deadlines can be conditional.

The rules in the procedure that are too complex to be solved by defining conditional actions may be programmed using Staffware *scripts* which can be called using *field* and *form commands* which are triggered by certain field and form actions (see "Data definition" below). Scripts are also used for running series of external program calls (e.g. DDE interaction with Windows applications).

Procedures can be *changed* even during their execution but these changes only have an effect on the remaining steps of the procedure instances since Staffware does not have different entities for procedure definition and execution. This feature causes problems when running cases get into a situation where a subsequent step to the current step does not exist anymore in the updated procedure.

A single case is strictly bound to follow the rules defined in the procedure. Such dynamic features as bypassing or redoing a step or totally rerouting the case are lacking.

5.2.4.1.3 Scenario management Scenario management is mostly controlled by the Staffware server. A case is started by using client software or an external utility program. After the server has received the start request it reads the authorisation information and finds the

first step from the procedure definition. Then the form belonging to the step is sent to the addressee or automatic step is executed without user interaction (depending on the step type).

In case of an interactive step the addressee has access to the form in question through his workqueue that is a list of messages, work-items, indicating pending tasks. Single users have their own *private workqueues* while user groups have *shared workqueues* to which every member has access. When one group member selects a work item from the shared workqueue the server locks it to him and other members cannot access it anymore. The step is locked until the addressee has finished and released it. A step may also be allowed to be delegated to some other addressee by *forwarding* the work item to another user, group or role.

After releasing the step it is deleted from the workqueue and the control flow goes to the step that was defined as an action of the released step. A form related to the next step is sent to corresponding addressee by the server. The flow proceeds this way until the terminating stop-action is reached.

With the audit trail function it is possible to track how a certain case has progressed giving a date, time and the users involved when:

- the case has been started,
- the steps have been processed to the assigned addressee,
- the steps have been released,
- the possible deadlines have been ignored,
- some other actions have been directed to the steps or the whole case and
- when the case has been finished.

Another predefined report is *status report* which shows the current status of each step in a given case.

Staffware also provides functionality for producing customisable *management reports* which enable to produce different kinds of summaries and lists of selected cases.

Like most of the workflow products also Staffware lacks built-in functionality for getting a general view of the work status or distribution e.g at the corporate level. Similarly work balancing and forecasting the duration of a case are not supported.

If the Staffware server cannot assign a step to any addressee the corresponding work item is sent to system administrator's work queue (e.g. because of the change in the procedure). The system administrator can also force the case to terminate by executing the *close case* command.

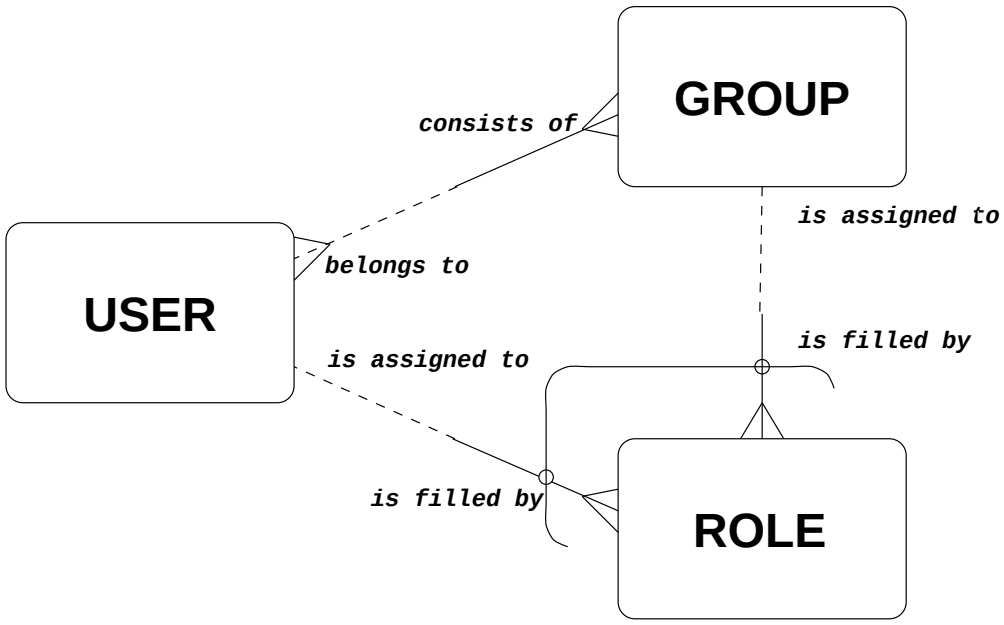


Figure 5.6: Relationships between users, user groups and roles in Staffware.

5.2.4.1.4 User definition and management In Staffware a user can be a single user or a user group. It is not possible to build any kind of user or group hierarchies. Both users and groups are defined within a *user directory* that is a simple list of user and group names.

For each user there is a *user profile* that contains a common set of *attributes* used for all Staffware users. In addition to predefined attributes (e.g. NAME and PASSWORD) any other custom attributes can be defined (such as employee number or supervisor's name). These attributes can later on be used in the implementation of activities, authorisation.

Users have four privilege levels: *Ordinary user, Manager, Procedure Definer* and *System Administrator*. These privileges are assigned to users by defining what kind of menu they use. While a system manager has every functionality available, an ordinary user has only few functions (e.g. managing his workqueues and starting a case).

There is also a separate *role directory* where Staffware users (single users and groups) can be assigned to roles. Role names can be used for routing in much the same way as user and group names. Role directory is common for all procedures. This causes problems when the same role needs to be assigned to different users in different procedures. Relationships between users, groups and roles are shown in figure 5.6.

Creating users, groups and roles as well as maintaining them is assigned to the system administrator.

Staffware does not support organisation structure definition. It is possible to partially overcome this shortcoming by defining some clever custom attributes for users.

5.2.4.1.5 Data definition In Staffware all data are defined by creating different types of *(case) variables* and forms that are stored in a database managed by the Staffware server. In Staffware terminology these variables are confusingly called "fields", without any distinction from the input and output fields defined in the forms. In this text we use the term *field* for the input or output space in the form.

The variable types available are:

- *text*,
- *number* that can be both integer or decimal,
- *date*,
- *time*,
- *memo* that are used for storing large text entries and
- *attachment* that allows attaching a file of any type to a form, by storing the full path for the file in this field.

Staffware enables to define *tables* for commonly used information in the cases (e.g. customer details). In addition to this, it is possible to build simple internally or externally stored data *lists* that can be used for *list of values* functions in the form fields. The values in the lists and tables can be used for field validation purposes in Staffware forms. The tables and values for them as well as lists can be defined only by the system administrator.

Filling out a form is the way for an addressee to perform the work associated to a certain step. The fields defined in the forms can be considered as views to the case variables having the same name as the field in question (see figure 5.8). Field types are derived from corresponding variable types. When the fields in the form are defined to be updatable the changes of these field values imply corresponding changes in the values of the case variables.

A form can be defined by means of the Staffware form editor (cf. figure 5.7) and portions of it (including display fields) can be read from an external text file at runtime. A whole form definition with e.g. updatable fields, however, cannot be read from an external file at runtime. Forms may also have *form conditions* by which it is possible to make some of the fields or data visible only in certain circumstances.

Staffware forms offer ways to integrate external documents and programs to its fields using *field command* option. By this option it is possible to call external programs with parameters or Staffware scripts as well as make assignment operations (e.g. to other fields). Staffware and external programs are exchanged via external ASCII files. If a field command option is used in an attachment field, the file name attached is used as a parameter for the field command.

With *form commands* forms also enable calling external programs or scripts when a user releases the step (from the form - not in his workqueue) and when he exits from the form (without releasing it).

In addition to this, the form itself can be implemented with some third party software and then called by *initial form command* instead of Staffware form[15]. Staffware offers a limited

[15] Staffware "Open Forms" concept.

```
STAFFWARE Procedure Editor: PILOTORD     Step: WORDER Order
──────────────────────────────────────────────────────────────────────────────
                  ORDER OF A PILOT (AGENT FORM)
──────────────────────────────────────────────────────────────────────────────
SHIP

Name:  SHIPNAME                          Call Sign: SHIPCALLSIGN
Depth: SHIPDEPTH

Agent: AGENTNAME                                          Tel: AGENTTEL

──────────────────────────────────────────────────────────────────────────────
PILOT INTO THE SHIP:

From: PILOTFROM                          Date: PILOTFROMDATE Time: PILOTFROMTIME

──────────────────────────────────────────────────────────────────────────────
PILOT OUT OF THE SHIP:

To:   PILOTTO                            Date: PILOTTODATE   Time: PILOTTOTIME


──────────────────────────────────────────────────────────────────────────────
Left margin:    1   Right margin:   78  Row:   1 Col:   1   Insert ON
```

Figure 5.7: Defining a form for a step with the Staffware form editor.

```
STAFFWARE Procedure Editor: PILOTORD     Step: WORDER Order
──────────────────────────────────────────────────────────────────────────────
            Definition of Field :  SHIPNAME

            Type                :  Text
            Length              :  25
            Origin              :  Required
            Command             :  _________________________

Valid responses or calculations
Values                                    Fields          Lists

Help for this Field :
```

Figure 5.8: Defining a field characteristics in Staffware.

Staffware Application Layer (SAL) API that can be used only in conjunction with some other mechanism to pass data across. Staffware also has the special *Form Helper* for integration with Visual Basic[16].

5.2.4.1.6 Data management Accessing case variables through the fields defined in the forms does not provide any concurrency control. This means that many users can read and update case variables simultaneously from different forms if the procedure definition gives the possibility for that (e.g. parallel activities having common input fields). The philosophy of Staffware seems to be that the procedure specifier (rather than the system itself) makes sure that common data stored in case variables may only be accessed by only one user at time. In this way it is enough to lock only the form when an addressee selects it from his workqueue. Variables themselves are not locked — only copies of the values are read by the client when the form is selected from the work queue and all the values are written back when the step is released (finished) allowing forward recovery from this last saved state if some system failure occurs during the next step.

When the case is terminated, all the data related to it remain in the database if the *auto purge* option has not been switched on. These data can be queried using Staffware management functions. It is also possible to retrieve these data of already finished (or currently running) cases by executing an external Staffware utility program. However, there is no way to see what data values were on the fields at the specific point of the workflow (after certain step). Staffware also does not have any mechanisms for backward recovery.

5.2.4.1.7 Conclusions Staffware is intended and has traditions for automating administrative and production tasks with well-defined rules. It allows graphical procedure (scenario) definition using Windows-based tools. Hierarchical definitions and reusability are not supported. This is the reason that defining very complex production type scenarios is difficult. It also does not have a programmable high-level process definition language.

Staffware offers typical routing features (sequentiality, forks, joins, loops and conditional routing) and external events. Its wait feature enables to define `Release` $\mathcal{A} \rightarrow$ `Release` $\mathcal{B}$ type dependencies in addition to normal `Release` $\mathcal{A} \rightarrow$ `Begin` $\mathcal{B}$ type dependencies. There is also some amount of flexibility offered, e.g. a step can be delegated to another user and user assignment can be based on the value in the updatable case variable (allowing ad hoc assignment).

Monitoring functionality has audit trail and status reports. In addition to these typical functions Staffware offers customisable management reports about the data related to cases (scenario instances).

Staffware does not support ad hoc workflows. Nor does it have a possibility to change the rules for a single case (e.g. by-passing or redoing steps) thus empowering the end-user.

Like all workflow products also Staffware lacks backward recovery as well as the possibility to define transactional properties for activities. It also does not have concurrency control for the case variables that cause problems if parallel steps are allowed to update same variables.

[16] Visual Basic is trademark of Microsoft Corporation

5.2.4.2 IBM FlowMark

5.2.4.2.1 Overview IBM introduced the first version of their own workflow management system *FlowMark* to OS/2 platform in 1994[17]. It is fully object oriented and the first commercial product using an object oriented database for storing workflow data[18]. FlowMark is a distributed workflow management system that operates on local area network, offering cross-platform support for OS/2, AIX and Windows. It has three different types of components (see figure 5.9):

Servers that consist of:

- FlowMark server which is the workflow management engine.
- FlowMark ODI ObjectStore object oriented database (OODB) server that is used by the FlowMark server for managing workflow data objects.

Servers are provided for OS/2 and AIX platforms.

Buildtime client for modelling scenarios on OS/2 workstations and for administering the system on OS/2, AIX and Windows workstations. A buildtime client has its own OODB client that can connect directly to the OODB server.

Runtime client for starting processes and using work lists on OS/2, AIX and Windows workstations.

5.2.4.2.2 Scenario definition A *process (model)* (scenario definition) is described in FlowMark by using a graphical diagramming tool and entering more specific data into so-called *notebooks* and other dialogs that can be opened from the diagram. The modelling technique is based on the *metamodel* concept and its mathematical formulation is presented in [LA94]. According to the metamodel, a process is presented as a weighted, coloured, directed graph of *activities* where both control and data flows occur as different kinds of edges. (cf. figure 5.10)

The whole *workflow model* including scenario definition, data definition, and user definition can be represented using the *FlowMark Definition Language (FDL)*. FlowMark enables to export and import workflow models or parts of them using FDL. However, FDL does not offer more than what can be defined by using graphical diagramming and data entry tools provided by the buildtime client.

Activities represent a piece of work that the assigned actor can complete. FlowMark offers three activity types:

[17] In this analysis we are referring to FlowMark version 1.1
[18] Other object oriented workflow products store their objects in a relational database.

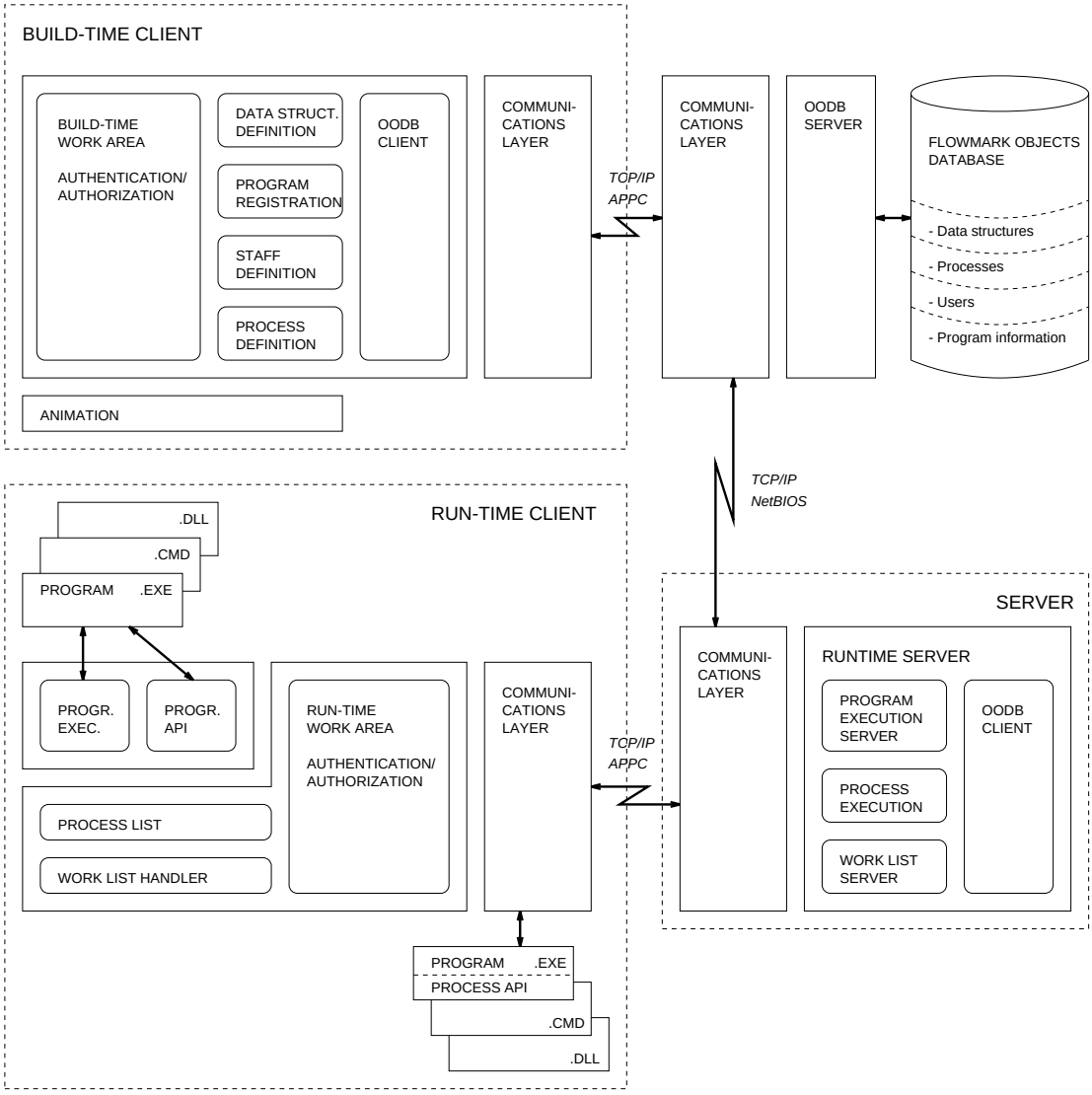


Figure 5.9: FlowMark client/server architecture.

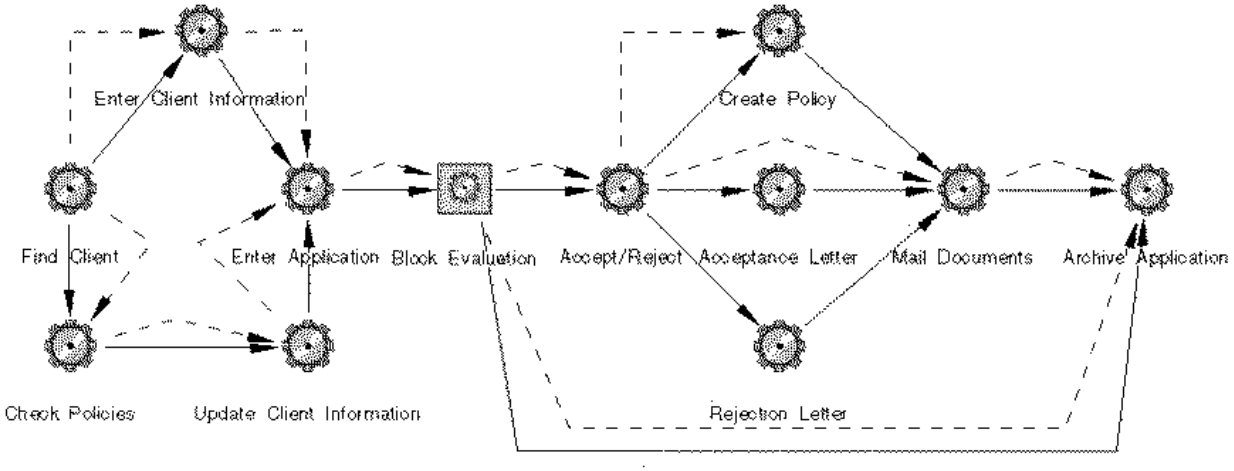


Figure 5.10: Example of FlowMark process model. (source: *IBM, FlowMark for OS/2, Insurance Model*)

1. A *program activity* has an executable program assigned to it. This program performs the work, either alone in the background or in cooperation with the user via a dialogue. Since FlowMark itself does not have any tools for implementing the program activities (e.g. for building forms) they must be implemented as external programs and called via command line calls or provided APIs. The information needed for the external application integration with FlowMark is stored in a *program registration* database.
2. A special kind of activities, *blocks*, is used for grouping a certain sequence of activities into one entity. Grouping can be used for reducing the complexity of the process diagram. It also allows to model loops in which the activities in the block are executed repeatedly until the exit condition of the block evaluates to true. Third purpose for block type activities is to implement *bundles* which allow single activities to be replicated to multiple (parallel) activity instances. The number of instantiations is determined at runtime.
3. There is a possibility to model processes by reusing other processes as activities. Such activities are called *process activities*.

Activities are linked by *connectors* forming potential sequences of activities and describing transmission of data between the activities. In the FlowMark process model there can be two kinds of connectors:

1. A *control connector* which can have a *transition condition* associated to it allowing the control flow through it only when the condition evaluates to true. A *default connector* is an "otherwise" connector through which the control should flow if the transition condition of no other control connector leaving the activity evaluates to true. By defining multiple connectors starting from one activity (and having transition conditions evaluated to true[19]) it is possible to define parallel control flows.

[19] A control connector without transition condition is evaluated as a control connector having a transition condition which *always* evaluates to true (i.e. control always flows through that kind of control connector).

2. A *data connector* specifies the data flow between the *data containers* of the activities. There can also be multiple data connectors leaving from or incoming to one activity. If the data structures in connected data containers do not match exactly, it is possible for the scenario specifier to map the individual members of the output data structure onto members of the input data structure.

   The data are written into and read from the data containers by the programs associated to the activities via APIs. In addition there are some system provided data such as process instance name, activity name, return code, etc. The evaluation of the transition conditions as well as the exit conditions (see below) uses the values in the data containers. So the programs influence the control flow within the process indirectly.

In addition to the conditional routing accomplished by transition conditions it is possible to set *start conditions* and *exit conditions* for the activities to determine when the activity may start or end. The start condition enables to synchronise the re-joining control flows since it is possible to choose the start condition to evaluate to true either if at least one of the incoming connectors evaluates to true or only if all incoming connectors evaluate to true.

Assigning *staff* (actors) to activities can be done either as a *dynamic assignment* or as a *specific assignment*. In a dynamic assignment the actors for the activities are determined at runtime by resolving different kinds of criteria. The criteria can be related to people's levels, organisations, roles or some combination of these (see "User definition"). In the specific assignment an activity is bound directly to one or many users.

A deadline definition in FlowMark is implemented allowing only to define durations for processes and activities in the granularity of hours or days. Moreover it is limited only for notifying the specified person when the process or activity is not completed in the time allowed. It is not possible to define a replacement activity that has to take place if a deadline is ignored.

Another shortcoming in the scenario definition is that it does not include external events.

FlowMark has an animation facility that lets a scenario designer move through the created process model using the process diagram for checking that it is correct and works as expected. This is not a tool that would enable to construct a model for simulation of the real world but just a tool for manual debugging of the logic flow of the process model. During the animation programs associated to the activities are not called. The data are stored into the containers and regarded by the testing person.

5.2.4.2.3 Scenario management After the completion of the process definition it is translated into a *process template* that is the master for all the executable processes and is an entity totally separated from the process model. Once a new process instance has to execute, a copy of the related process template is made and used for steering it.

The runtime client offers a graphical interface for the end-users to start those *process instances* (scenario instances) and to handle *work lists* (see figure 5.11) to which announcements of

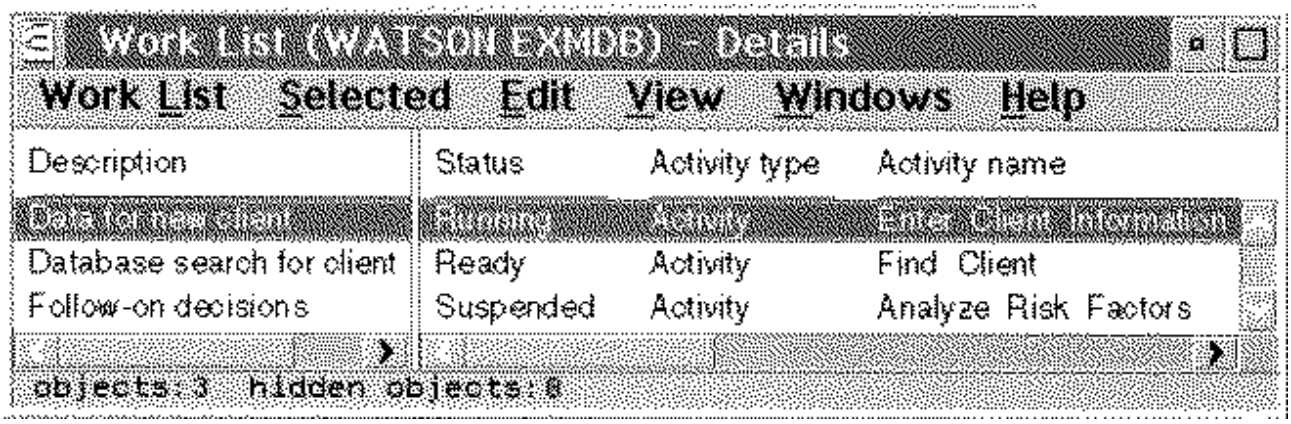


Figure 5.11: Example of a worklist in FlowMark *(source: IBM, FlowMark for OS/2, Insurance Model)*.

the new *work requests* are sent. In the runtime client the *process list* gives a user access to all of the process templates that he is authorised to instantiate (start) as well as to all process instances he is allowed to see. The user interface also allows the filtering and sorting of the process instances in execution.

A *work list handler* provides an end-user interface for managing the work requests FlowMark has generated:

- work requests can be transferred to another actor,
- new processes can be started,
- the list of requests can be sorted or filtered,
- a graphical view of the current status of the process can be obtained.

The server part of FlowMark is a coordinator and synchroniser for the work requests and data related to the activities. The server is also responsible for executing the process instances obeying the defined rules translated into the process template.

When a process instance is started via the graphical interface or via the API the *process execution* component of the FlowMark server executes the process template in the following way:

- First, it finds the start activities and distributes them to the appropriate actors.
- The evaluation of transition conditions determines how the control flows. Connectors that have a transition condition evaluated to true are selected.
- For the non-start activities the start control is checked after one or all (depending how re-joining control flows have synchronised) transition conditions of the incoming connectors have been evaluated to true or marked to come from a *dead path*. Dead paths are subgraphs having activities that can no longer become executable because the transition conditions of all the connectors leading to them are evaluated to false. Activities should *not* rely on data coming from activities on the dead path.

- When no further control connectors with a transition condition evaluated to true are found, the process is finished.

Distributing work requests using either specific or dynamic assignment as well as keeping track of the requests is done by the *work list server*. When there are many persons (e.g. when the actor is a role) receiving the same work request in their work queues, the first person choosing it from his workqueue becomes the owner of the task and at the same time it becomes unavailable in other person's workqueues[20]. It also interacts with the work list handler in the runtime client to handle manual (interactive) program invocations.

With program activities, the *program execution server* queries the program registration database in order to find the appropriate target host and other information for executing a program bound to an activity. The program can be executed at any host that has the FlowMark program execution process running.

FlowMark provides only the monitoring of process instances with a graphical process diagram from which every user can look how the process instance has progressed. There is no other kind of tools provided for e.g. audit-trail reporting. However, FlowMark has an audit-trail server that logs all events into the ASCII files which can be looked at by authorised persons (or can be loaded into the database.)

5.2.4.2.4 User definition and management In its *staff* definition the FlowMark buildtime client lets an authorised person define not only users and their roles but also the whole organisation structure. A limited set of user information management is also possible using the runtime client (e.g. changing the password and marking a person absent).

For a *person* (user) it is possible to define:

- General details (user ID, password, names, telephone numbers etc.).
- Role assignments. A person can have multiple roles (and many persons can be assigned to the same role).
- The organisational unit where the person belongs to. A person may belong to only one organisational unit.
- A substitute person to whom all the work requests are forwarded if a person is marked absent.
- Authorisation to some or all of the following tasks:
  - Define staff
  - Model processes

[20] FlowMark has only personal workqueues unlike Staffware where user groups had their own shared work queues.

- Manage process instances in specific categories
- Access other people's activities

FlowMark has a simple *level* feature for classifying persons. Levels can be named freely and thus used for appropriate purposes in corporate (e.g. as a security or skill level).

A role definition consists of name, description, role's optional coordinator and related persons. FlowMark has three predefined roles: *administrator*, *manager* and *coordinator*. The user who has the administration role does for example, all the authorisations in FlowMark.

Organisational units are defined by giving it a name, description, manager's user ID, related persons and all the parent and child relationships in the organisation hierarchy.

One shortcoming in staff definition is that the data structures are not customisable. For example in many cases it would be useful to put some more information into a person database than is possible in FlowMark.

The level feature in FlowMark might also be too restricted since it enables to define only one set of classification criteria features. E.g. in some corporation, levels can be used for defining skill levels but what if security levels for the staff have to be covered in addition to this?

5.2.4.2.5 Data definition Each activity and process has a private *input container* and *output container* for the parameter storage and exchange between the activities. Each data container is defined by a data structure that consists of data structure *members*. The type of data structure members can be one of the basic types provided (string, long or floating point) or can refer to another data structure thus forming a *nested* data structure. Data structures can also form arrays. All the defined data structures form a pool of reusable data structure definitions.

There are no data types or structures for date and time. Nor is there any kind of conversion functions for date and time handling. This shortcoming has its implications to the deadline definition.

5.2.4.2.6 Data management Since all the data in the scenario instance are stored in private data containers there is no concurrent access to common data. Furthermore, since the contents of the data containers as well as synchronisation and coordination data are kept in the database in case of system failure, forward recovery is always possible from the last saved situation.

Currently, FlowMark does not provide backward recovery. However, IBM has done research in order to create support for backward recovery of scenario instances. In [Ley] a notion of *spheres of joint compensation* is presented introducing a compensation based partial backward recovery of the control flow. In this paper it is also presented how this backward recovery capability can be added to the metamodel [LA94] that is the basis of the process definition in FlowMark. Currently, this concept is being enhanced towards inter-scenario dependencies and a prototype of a "business transaction manager" is under development. It is not clear how exactly and when these concepts will be included into FlowMark.

5.2.4.2.7 Conclusion FlowMark is a robust workflow management system intended to automate production workflows. It is the only product we have seen separating control and data flow. It allows hierarchical modelling and re-usability. The modelling tool provides animation for correcting errors but it lacks advanced simulation tools.

Scenario definition:
Sequentiality, splits, joins and conditions are supported for control flows. In addition activities may have start- and end-conditions. Also loops are enabled by using block activities. Although a process model does not directly support replicating activity instances in a process instance, this kind of bundle activities can be implemented by using block activities and external programs.

External events can not be defined. There is also no possibility to define advanced dependencies between the activities (see Section 5.4.2).

Transaction model:
There is no need for concurrency control in the management of the data belonging to the process instances since shared data do not exist. Data are always stored in the private containers of running activities. If data have to be shared between processes or parallel activity instances they must be stored in shared data base. Common to all workflow products, also FlowMark lacks the possibility to define transactional properties for the activities as well as backward recovery.

Shortcomings:
Deadline features are too limited for many cases and datastructures and conversion functions for handling dates and times are missing.

Although FlowMark enables to define roles and organisation structures it lacks customisability in user definition. FlowMark has only one classification for the users, namely levels, which might be inadequate.

FlowMark does not have any audit-trail reporting tools or sophisticated monitoring aids besides the graphical representation of the status and history of single process instance. Other more detailed information must be retrieved from the audit-trail files externally.

5.2.4.3 TeamFlow

5.2.4.3.1 TeamFlow TeamFlow is a workflow extension to the TeamOFFICE package under development at the Finnish ICL company ICL Personal Systems Oy. TeamOFFICE includes TeamMail (electronic mail, with gateways to X.400, Memo and fax), TeamCalendar (a distributed office calendar), TeamForum (a bulletin board system) and TeamLibrary (a distributed document and file storage system)[21]

[21] This product suite was originally developed by the Finnish company Nokia Data Systems. ICL acquired it when they bought Nokia Data Systems in 1991. ICL supplies also another office automation suite for production workflow management, *OfficePower* that has Staffware-based *PowerFlow* as an add-on product. In addition to this, ICL has a new set of tools, known *ProcessWise* for business process re-engineering.

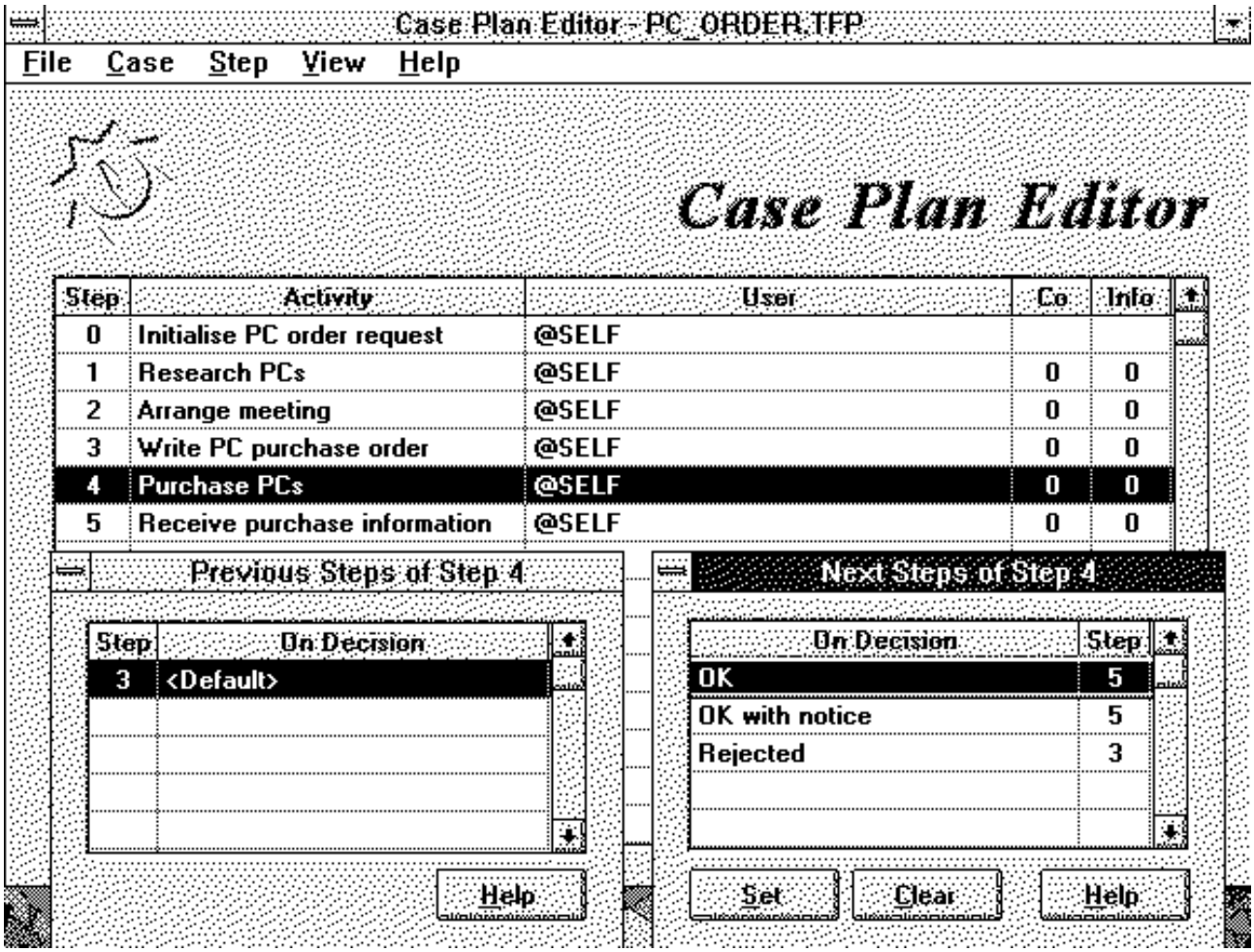


Figure 5.12: Defining case circulation order with TeamFlow Case Plan Editor.

TeamOFFICE is based on a client-server architecture with the server running under OS/2, UNIX SVR4, Novell NetWare or Windows NT systems, and clients being PCs running under Windows $3.x$. TeamFlow servers, however, run only on OS/2 and Windows NT platforms. Systems with multiple servers are supported.

TeamFlow stores the data of the scenario instances in TeamMail messages and associated attachment files that are routed by the TeamFlow server. The TeamFlow server keeps an audit trail log in a Microsoft SQL Server 4.2 database on the Windows NT server or in Gupta's SQLBase on OS/2 server. TeamMail and the other TeamOFFICE products use their own proprietary database.

5.2.4.3.1.1 Scenario definition Scenarios, called *case plans* in TeamFlow, are defined using the Windows based *Case Plan Editor* (see figure 5.12) where definition information is entered by filling different kinds of forms and tables. TeamFlow has no kind of graphical diagramming tool (found in many other products), nor does it have a programmable scenario definition language. An existing scenario can be used as a template when defining a new scenario, thus enabling reusability. Hierarchical scenario definitions are not supported.

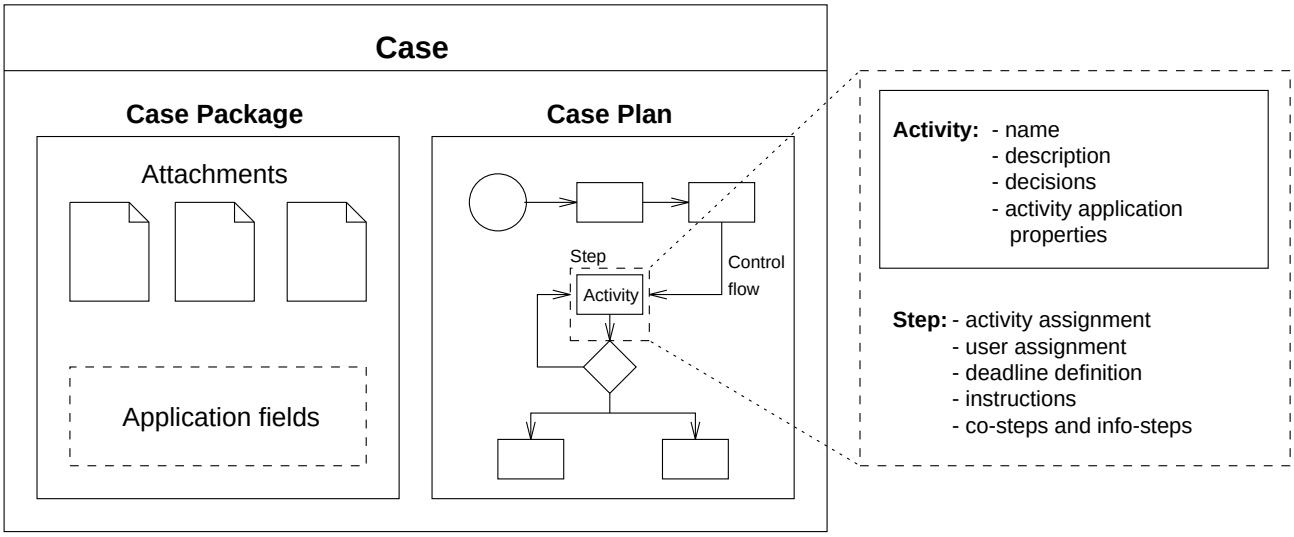


Figure 5.13: TeamFlow case (scenario instance).

Rather than being a full featured workflow definition tool TeamFlow provides a good set of tools and services for *implementing* custom workflow applications. For that it has extensive C/C++, Visual Basic and SQLWindows APIs. In addition to this TeamOFFICE includes an *Integration Kit* which enables easy integration of the most popular office software.

In TeamFlow a scenario instance is called a *case* (see figure 5.13) that consists of:

- *Case package* that may include:
  - *Attachment files* such as documents, spreadsheets or pictures.
  - *Application fields* that may be used for storing and transferring data specific to the case.
- *Case plan* that defines the routing of the case package from one user to another. A case plan is composed of successive series of *steps* that include:
  - assigning TeamFlow activities to TeamFlow steps,
  - assigning the case user (see "User definition" below) to TeamFlow steps,
  - deadline definition,
  - instructions,
  - optional co-steps and info-steps (see below).

In the TeamFlow terminology an *activity* is a definition of a certain piece of work. It always has a TeamFlow-aware activity application associated to it, enabling the performance of the actual work. TeamFlow provides a *Folder Application* as a default activity application that enables a TeamFlow user to view a case, perform the tasks (associated to the attachments) and forward the case to the next user with optional comments. TeamFlow activities include also a list of *decisions* that are the possible outcomes of a completed activity. (cf. figure 5.14)

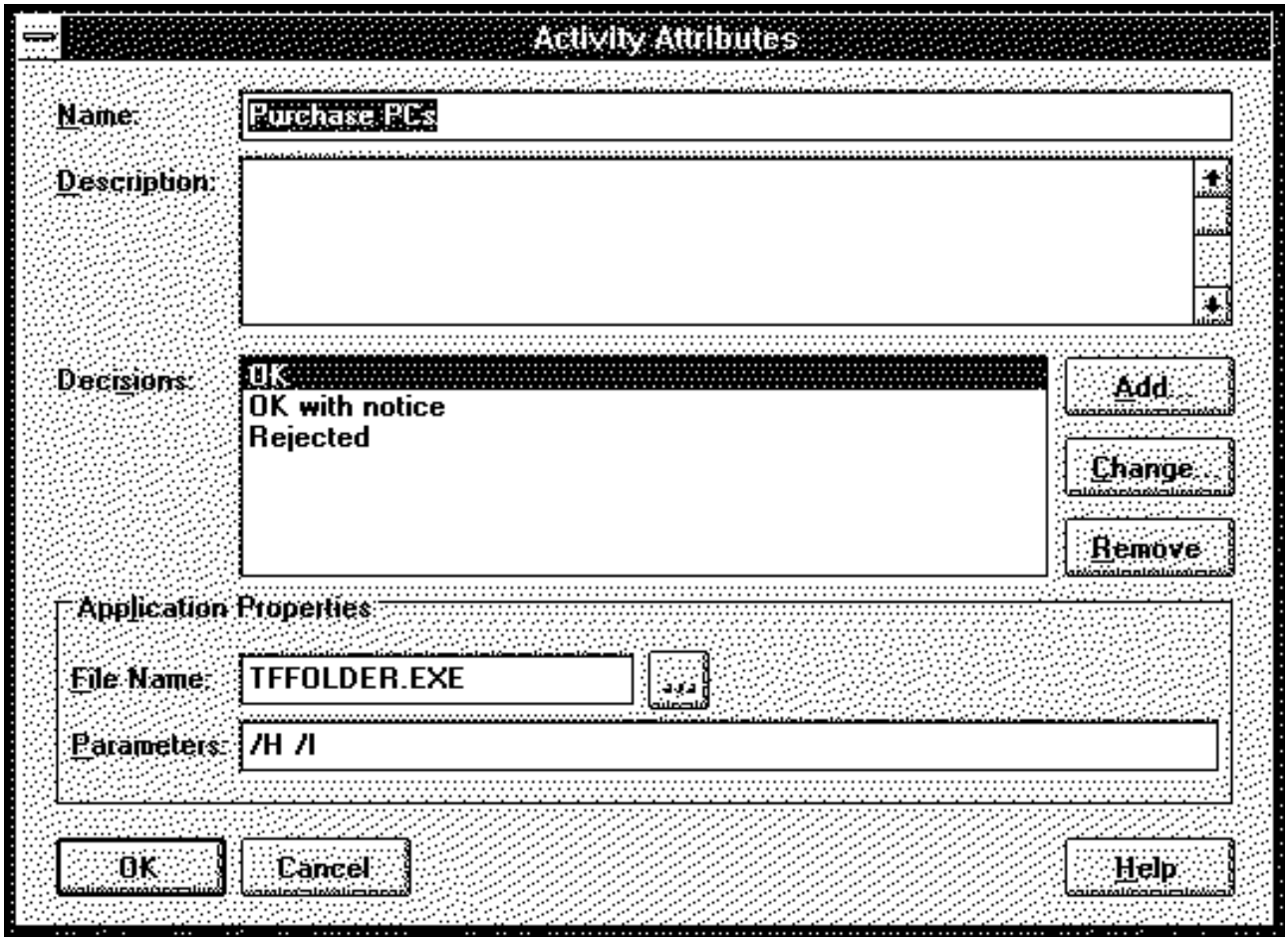


Figure 5.14: TeamFlow activity definition window.

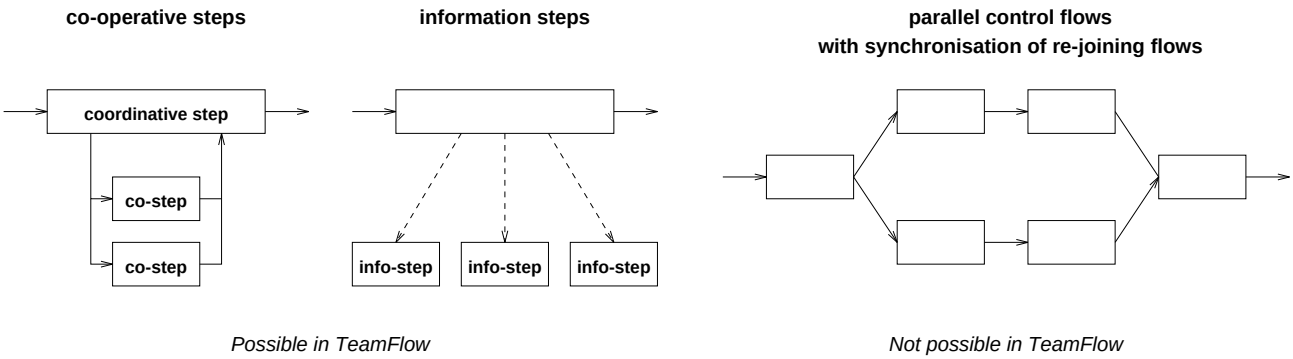


Figure 5.15: Parallelism in TeamFlow case plans (scenarios).

The *control flow* between the steps, the *case circulation order* (see figure 5.12), is established associating the subsequent steps to the decisions of the activity in the preceeding step. Since the choice of the subsequent step is dependent on the decision it is possible to establish conditional, *response-based routing*. The control flow can also have loops. Since the control flow describes the routing of the case package the dataflow is always included in the control flow.

Parallelism is supported only partially. By associating *co-operative steps* with a step it is possible to start execution of many parallel steps. The step having co-operative steps coordinates the co-operative steps and when they are finished the data associated to them are sent back to the user of the coordinative step. This user is then responsible for handling the work received from co-step users (but the user need not wait for the finish of all co-step executions). TeamFlow does not allow parallel branches (having multiple steps) with synchronisation of re-joining control flows (cf. figure 5.15).

In addition to the co-steps TeamFlow has *information steps* that can be used for sending a case package to some user only for informative purposes. This user can not participate in the execution of the case.

For each case plan a set of TeamOFFICE users is defined as *case users* that can be assigned to the steps. These users can also include user groups that are actually distribution lists for TeamMail. If this kind of user group is assigned to the next step in a case the actor of the current step must explicitly choose a single user name from the group before the case can be sent forward. Thus one step cannot be assigned to many users. Nor does TeamFlow provide any shared workqueue mechanism.

Deadlines for a step can be defined either using an absolute date and time expression or a relative expression for the duration of the step. It is also possible to define whether the step user is to be warned before the deadline.

Every case has a *supervisor* that can be defined explicitly as a case attribute. If the supervisor is not explicitly defined the case initiator is assigned to this job. A supervisor has rights for administrating and monitoring all of his cases. (cf. "Scenario management" below.) A supervisor is also the one who is informed about his cases with *notifications*. Notifications to the supervisor can be defined to be sent each time after:

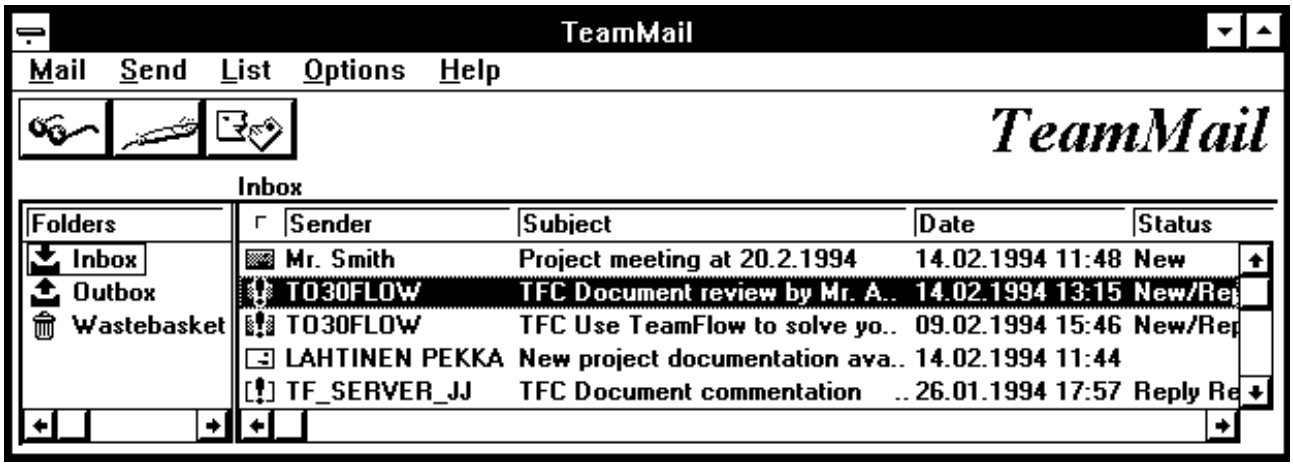


Figure 5.16: TeamMail mailbox containing ordinary messages and TeamFlow cases. (Source: *TeamFlow 3.0-2, Getting Started: Participating in TeamFlow cases, ICL Ltd. 1994*)

- a case is started
- a case is completed, and/or
- every step completion.

5.2.4.3.1.2 Scenario management When a new case is started a message containing a case plan and a case package (with attachment files and application fields) is sent to the virtual user mailbox in the TeamFlow server (called a *TeamFlow Agent*) by the TeamFlow client. After the server has received the message it determines the first user of the case from the case plan and forwards the message to the user assigned to the first step.

The assigned user gets the message to his mailbox with the subject field of the message preceded by the string "TFC" (so that he can distinguish TeamFlow cases from the other mail messages at a glance) (cf. figure 5.16). Thus the mailbox is also used as a work queue. TeamFlow cases are selected for viewing and handling like ordinary mail messages into the *TeamMail Received Mail* window that reveals more information about the case.

In the TeamMail Received Mail window the user can start a TeamFlow activity application for performing the work. This program might be the default Folder Application or any TeamFlow-aware application (which means that the application is programmed to work together with TeamFlow by using TeamFlow APIs). After finishing the work associated to the step the user selects the decision[22] and *forwards* the case to the next user associated to the next step. Instead of sending the case directly to the next user it is always first sent to the TeamFlow server which does the audit trail logging and checks if the case plan has been modified before sending it to the next user.

For activity applications TeamFlow APIs provide services for modifying cases in execution. Another way to alter cases is to use the TeamFlow Case Plan Editor as an activity application.

[22]The decision can also be solved automatically by the activity application.

In both cases it is possible to modify case plans and case packages of single cases in execution in an arbitrary way enabling flexibility, ad-hoc features and user empowerment.

TeamFlow also provides quite extensive monitoring and administering with its *Supervisor Tool*. The supervisor tool lets a supervisor or *TeamFlow administrator*:

- monitor the current status of cases
- view the history of the cases
- see the contents of any case he supervises
- get different kinds of reports about cases
- interrupt an active case (so that it can be modified)
- modify an active case (e.g. changing its route)
- abort an active case
- remove a completed case from the TeamFlow system databases

While a supervisor has administering and monitoring rights over the cases he supervises, the TeamFlow administrator has these rights over all cases (handled by a certain server).

5.2.4.3.1.3 Data definition and management TeamFlow data are stored in a case package which is routed to the appropriate users obeying the rules defined in the case plan. Two kinds of data can be defined:

- *Attachment files* that can be files of any type. For each attachment file it is possible to define *access rights* (Read, Modify, Remove) as well as an external program that is used to handle the file. It is notable that access rights cannot be defined for single steps but only for the whole cases. If external program association is omitted TeamFlow uses the default association (based on file name extensions) defined in the user's Windows settings. (cf. figure 5.17)
- *Application fields* (see figure 5.18) that can only be string variables or lists of strings for transferring case specific data between the steps. Application fields can also be defined to be stored in the audit trail database. Hence they can be used as a reporting criteria.

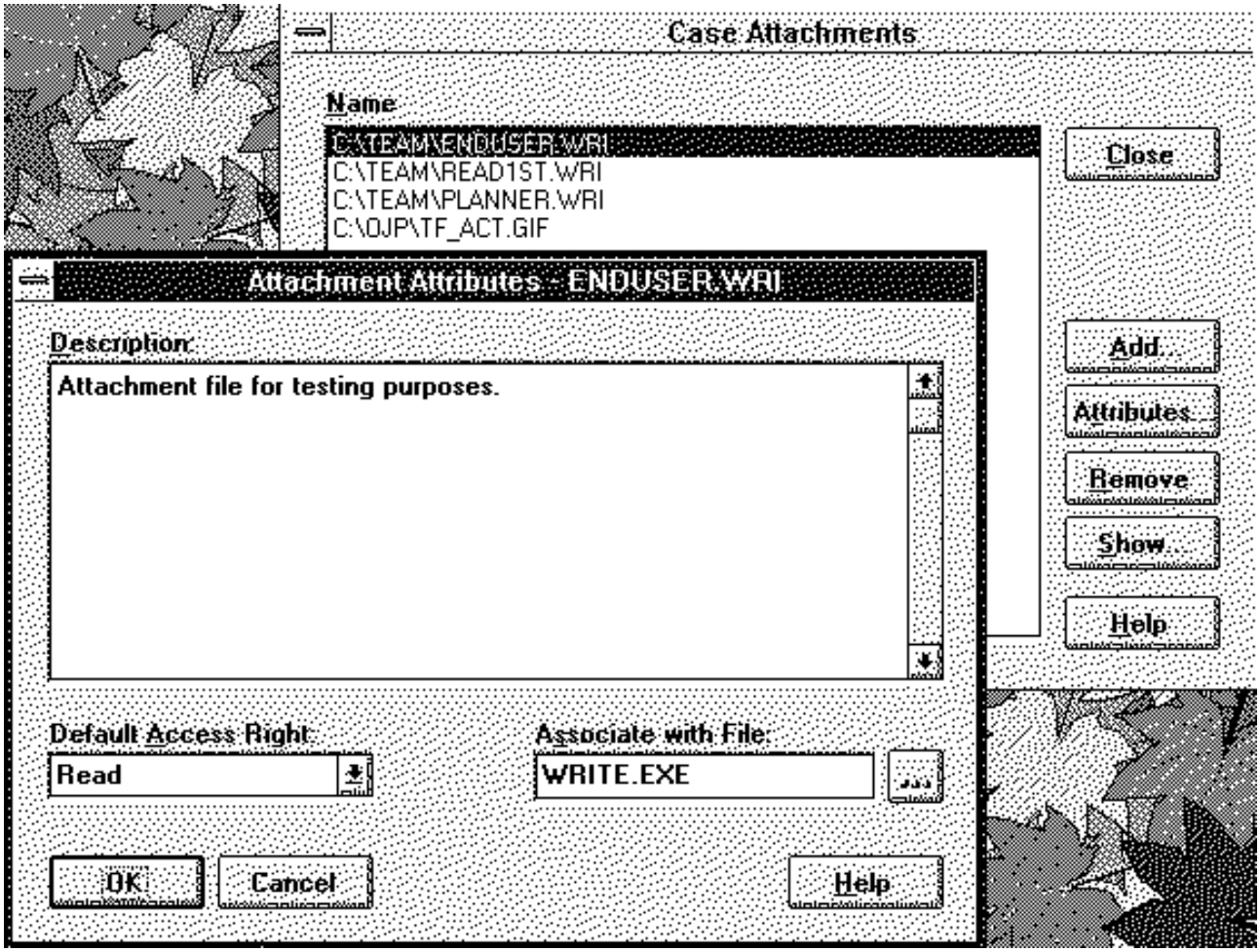


Figure 5.17: Defining TeamFlow attachment files.

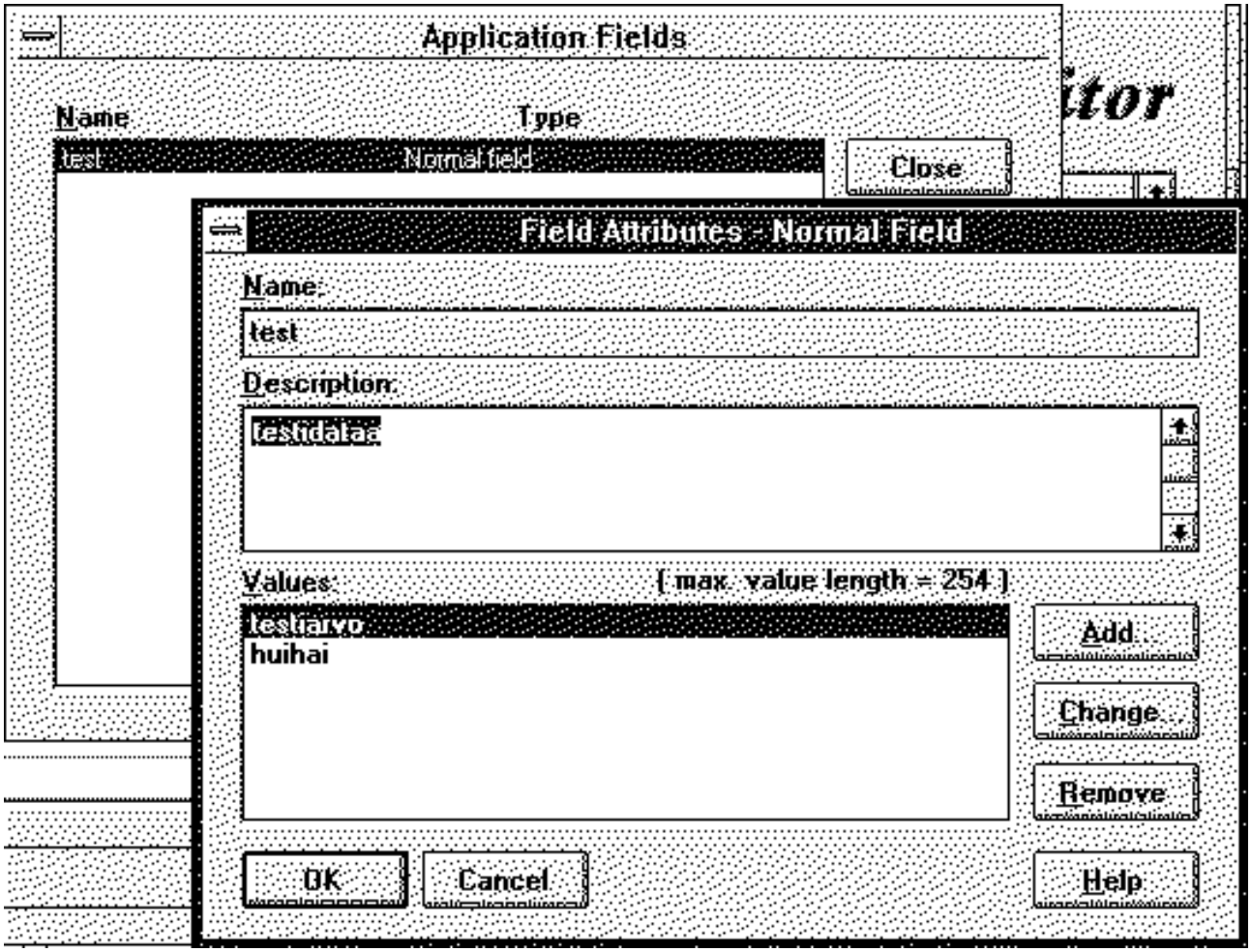


Figure 5.18: TeamFlow application field definition.

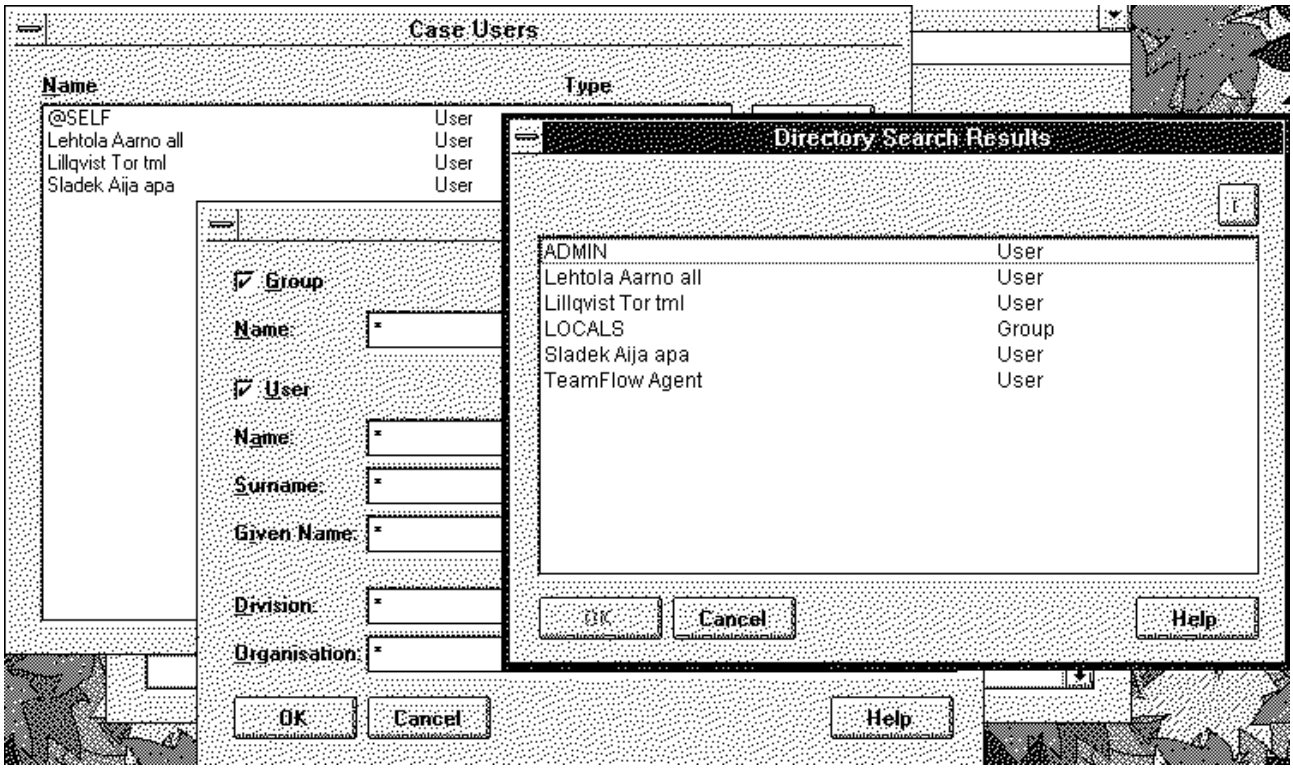


Figure 5.19: TeamFlow case user definition.

5.2.4.3.1.4 User definition and management TeamFlow uses the TeamOFFICE user database. TeamOFFICE lets a system administrator define a comprehensive set of personal details as well as mail details for a single user. Above the personal details TeamOFFICE allows to grant access rights to TeamOFFICE services.

TeamOFFICE also allows to define user groups that are used as distribution lists by TeamMail. However, in TeamOFFICE roles or organisation structures cannot be defined such that they could be used in TeamFlow. Especially the lack of role definition makes user assignment inadequate (see "Scenario Definition" above).

A set of *case users* must be defined for each TeamFlow case plan. Case users may consist of TeamOFFICE users or groups. User assignments to the steps of the case plan must be associated to these predefined case users. (cf. figure 5.19).

5.2.4.3.1.5 Conclusions TeamFlow is an extension of TeamOFFICE, intended for automating administrative and ad-hoc tasks. It is a hybrid solution including features from mail-based and client/server based architectures and thus providing interesting features like support for ad-hoc workflows and modification of scenario executions but still having extensive monitoring capabilities.

TeamFlow is, however, too limited for defining complex production workflows. For example support for parallelism is limited and the rules based routing is lacking. Exceeding a deadline causes only a warning to the actor and a notification to the supervisor who must handle the situation manually.

TeamFlow lacks a graphical workflow diagramming tool, hierarchical modelling and simulation of the pre-production workflows. Moreover, TeamFlow does not provide a programmable workflow definition language. Simple scenarios, however, are easy to define with table and form entries in the Case Plan Editor.

For application programmers TeamFlow provides C/C++, Visual Basic and SQLWindows APIs for developing custom workflow applications. In addition to this, TeamOFFICE includes an integration kit that enables easy integration of the most common Windows office applications.

The TeamOFFICE user definition lacks a definition of roles as well as the definition of the organisation structure. Thus steps cannot be assigned to roles (or to user groups) but only to individual users. This shortcoming is not compensated by the possibility to assign a step to a distribution list from which the user must select to whom the next step is assigned.

TeamFlow could make better use of other TeamOFFICE features. For example TeamCalendar services could be integrated with TeamFlow for intelligent resource allocation and assignment of the case steps.

5.2.5 Summary of analysed workflow products

	Staffware	**FlowMark**	**TeamFlow**
Architecture	Client/Server	Client/Server	Client/Server, Mail-Based
Server platform	UNIX	OS/2, AIX	OS/2, Windows NT
Client platform	UNIX, Windows	OS/2, AIX, Windows	Windows
DBMS	Proprietary, Informix, Oracle	ObjectStore	SQLServer, SQLBase

Scenario definition functionality

Graphical diagramming	Yes	Yes	No
Reusability	Poor reusability in definition	Every object can be reused	Existing definition can be used as an template for new definition
Script language	Yes	No	No
Programmable high level scenario definition language	No	FDL	No
Animation/ Simulation	No	Animation	No
Testing scenarios before production use	In test environment separate from production environment	By animation facility	No separate testing environment. Testing is done by assigning the testing user temporarily to all tasks
Import/Export scenario definitions	Yes	Yes. Uses FDL that enables import/export also user and data definitions	Internal representation is ASCII type file
Misc.	Staffware includes a character based form editor for defining and implementing forms for the activities.		

Activity definition features

Activity types	Interactive, automatic	Interactive, automatic	Interactive, automatic
External events	Yes	No	No
Support manual activities	Yes	Yes	Yes
Hierarchical scenario definition	No	Yes	No
Reuse existing scenario as an activity	No	Yes	No
Start condition for an activity	No	Yes	No
End condition for an activity	Yes (using form conditions)	Yes	No
Compensative or contingency activities	No	No	No

Activity assignment

To a single user	Yes	Yes	Yes
To many persons (everyone becomes actor)	Yes	No	Yes (with co-steps and info-steps)
To a group of users (only one is chosen the actor)	Yes	Yes	No
To a role	Yes	Yes	No
To an organisational unit	No	Yes	No
Dynamic assignment (solved at runtime)	Yes	Yes	Yes
Ad hoc assignment	Yes (by defining the actor of the next step to be retrieved from an updatable variable)	Yes (by implementing assignment in the programs called by activities)	Yes
Delegation	Yes	Yes	yes
Automatic delegation to a substitute user	No (possible to implement easily)	Yes	No
Work balancing	No	No	No

Control flow definition features

Sequential routing	Yes	Yes	Yes
Rule-based routing	Yes	Yes	No (Must be implemented in the program activated by the activity)
Response-based routing	Yes	Yes	Yes
Parallel routing	Yes	Yes	Limited
Synchronisation of re-joining control flows	Yes	Yes (by start conditions)	No
Looping	Yes	yes	Yes
Bundles	No	Yes	No
Separate data flow definition	No	Yes	No
Ad hoc routing	No	No	Yes
Dependencies between activities	`Release` $A \rightarrow$ `Begin` B, `Release` $A \rightarrow$ `Release` B	`Release` $A \rightarrow$ `Begin` B	`Release` $A \rightarrow$ `Begin` B
Temporal dependencies	No	No	No

Deadline definition features

Duration of a single activity instance	Years, months, weeks, days, hours, minutes	Days, hours	Days, hours
Duration of a scenario instance (or group of activities)	Possible by relative date and time expressions	No	No
Time expression	Absolute, relative (to e.g some variable) or calculated date and time expressions	No	Absolute date and time
Action if deadline exceeded	Launch user definable activities	Notify	Notify
Conditional deadlines	Yes	No	No

Scenario management

Forward activity	Yes	Yes	Yes
Skip an activity in a scenario instance in execution	No	No	No
Modify route of a scenario instance in execution	No	No	Yes
Forward recovery	Yes	Yes	Yes
Backward recovery	No	No	No

Data definition features

Basic data types supported	Text, number, date, time, memo	String, long, float	String
Data structures	No	Yes	No
Arrays	No	Yes	List of strings
Misc.	User definable value lists and data tables		

Data management

Management of the scenario instance data	Global within a scenario instance. Parallel activities can have concurrent access.	Each activity has private data containers of which contents are exchanged as defined in data flow definition. Global data with concurrent access must be stored into external storage.	Data is stored in TeamMail mail items. In case of parallel activities, data items are replicated.
Concurrency control of the scenario instance data	No, concurrent access of data items is possible and may cause problems.	Not needed since all the data are stored in private containers.	Not needed since all the data is in the mail item that is sent to the actor.
Document management	Pointers to physical files using attachment fields	Managed by the programs called by the activities	Managed as TeamMail attachment files
Concurrency control in the document management	Yes (based on UNIX file handling)	No	No (all the documents included in the mail item)

User definition

Single user definition	Predefined set of attributes + user defined attributes	Predefined set of attributes	Predefined set of attributes. (TeamOFFICE user)
User group definition	Yes	No	No
Role definition	Yes	Yes	No
Organisation structure definition	No	Yes	No

Integration and implementation

Integration mechanisms	External program calls from forms. Forms can be replaced by another application. Very limited API support. Visual Basic Form helper utility.	External program (or DLL function) calls, APIs	external program calls, APIs
APIs	Proprietary SAL API, DDE	C/C++, REXX, Visual Basic, DDE, OLE, VIM, HLLAPI	C/C++, Visual Basic
Support for EDI	No	No	No

Administration and monitoring functionality

Audit trail database	Yes	Yes	Yes
Reporting the status of a specific scenario instance	Yes	Yes, graphical representation	Yes
Built-in audit trail reporting	Yes	No	Yes
Customised reports from audit trail data base	Yes	No	Predefined reports with user defined parameters
Suspend/reroute/terminate a scenario instance in execution	Yes/Yes/Yes	Yes/Yes/Yes	Yes/Yes/Yes
Modify data of a scenario instance in execution	Yes	Yes	Yes

Suitability for different workflow types

Workflow type product suits best for	Administrative and production	Production	Administrative and ad hoc

5.2.6 Summary of the major shortcomings in the current workflow management software

Inadequacy for inter-organisational workflows. Although in some cases high-end workflow products (e.g. FlowMark) might be suited for defining workflows that take place in many organisations, none of the products can deal with wide range of heterogeneity and autonomy issues in in this kind of system. (This is also true in enterprise-wide systems.)

Lacking standards. Although workflow products provide many ways to integrate external applications and documents, they lack standard interfaces that would support interoperability among workflow management systems and third party software. The standards to be announced by the Workflow Management Coalition will help overcoming this problem.

Business process analysis and re-engineering. Workflow products usually do not include tools for BPA and BPR for the basis of the scenario, user and data definition. Moreover, although there is a great variety of separate BPA/BPR-tools the definitions can not be exchanged between them and workflow products.

Inadequate support for testing, debugging and analysis of the scenario definitions. Most of the products enable manual experimenting of the scenario definition before putting it into the production use. However, certain correctness aspects (e.g. the correct termination of the scenario in all possible cases) can not be tested automatically. Workflow products also usually lack simulation tools.

Missing advanced monitoring tools. While most of the workflow products provide some kind of audit trail and status reporting, only few of them can produce more advanced reports that would e.g. reveal the efficiency of the system and possible bottlenecks in it, thus supporting continuous enhancing of the system.

Inadequate of totally missing definition of the transactional properties. None of the products we know enables the definition of some or all of the traditional ACID properties an the activity or for a group of activities.

Moreover, none of the products we know contains backward recovery by compensation. Neither do current products have a possibility to define contingency activities for alternative execution after a scenario instance is cancelled or cannot be executed for some other reason.

Managing shared resources also has serious shortcomings. In some products some of these shortcomings are solved by preventing the concurrent access of data items and document files related to the certain scenario instance. Other products leave this problem to the application programmer. Even less is offered by the workflow systems if one needs to control the correctness of the data retrieved from the external data storages that are used concurrently by many scenario instances (or even by several different workflow management systems). This means that none of the programs we know enable to define all the shared data and its visibility. All this leads also to inadequate support for cooperative workflow applications.

Inadequate definition of inter-activity and inter-scenario dependencies. Only few workflow products enable the definition of advanced dependencies above the serial dependency (`Release` $\mathcal{A} \rightarrow$ `Begin` $\mathcal{B}$) between the activities while many other types of dependencies might be needed (e.g. `Cancel` $\mathcal{A} \rightarrow$ `Begin` $\mathcal{B}$ and `Release` $\mathcal{A} \rightarrow$ `Cancel` $\mathcal{B}$.

Dynamic and ad hoc features. Typically ad hoc workflows, modifying the rules or data of the scenario instance in the execution and other features empowering the end users are lacking from the workflow systems aimed to production type tasks. However, there are growing number of products including this kinds of features in their functionality.

5.3 Specific workflow case studies

5.3.1 The graphical description technique used in the analysis

5.3.1.1 Parts of the graphical model

The graphical representation of the scenario definitions is based on concepts known from existing system analysis methodologies. The graphical representation is intuitive and also easily translatable to a formal language. The description is decomposable until the required level of detail is reached.

The diagrams used to define a scenario are discussed below.

The *environment diagram* (ED) has the same purpose as in Structured Analysis (SA) [DeM79] and in object-oriented methodologies [RBP+91]. It depicts the exchange of data and control information between the scenarios to be defined and external entities. The environment diagram has been extended so that the environment is divided into two parts: the inner part contains the actors (i.e. human users, roles, organisational units or computer applications) of the scenario definition, and the outer part depicts external objects that exchange information with the actors rather act as a part of scenario execution. The environment diagram of the PortNet system is shown in figure 5.22.

The *transition diagram* (TD) is a modified life cycle diagram [RBP+91]. In the fashion of Harel's state charts, it depicts the activities (or the lower level scenario definitions) of a scenario definition and all possible transitions among them. Transition diagrams may be hierarchically decomposed. The top-level transition diagram of the PortNet system is shown in figure 5.23. The notation of the transition diagram is described in detail in the following subsection.

Traditional *data flow diagrams* (DFDs) [DeM79] are used here to express flows of data and control information among the activities and scenarios. The top-level DFD is created for the upmost scenario definition whereby, for each activity or scenario definition in the transition diagram, there is exactly one process in the DFD, and vice versa. The data flow diagrams may decomposed in the usual way. The top-level DFDs of the PortNet system with respect to exchange of timetable data are shown in figures 5.24 and 5.25.

5.3.1.2 Notation of transition diagrams

There are two kinds of building blocks in a transition diagram: activity and scenario definitions shown as boxes, and transitions represented by directed edges. From now on, we will call activity and scenario definitions of transition and data flow diagrams *nodes*. A transition is an instantaneous event of moving control from one node to another: a transition drawn from node $\mathcal{A}$ to $\mathcal{B}$ means that when the execution of $\mathcal{B}$ is started, the execution of $\mathcal{A}$ is ceased. A control flow starts at one or more entry points and it ends at one or more exit points. Elements of the graphical notation are shown in figure 5.20.

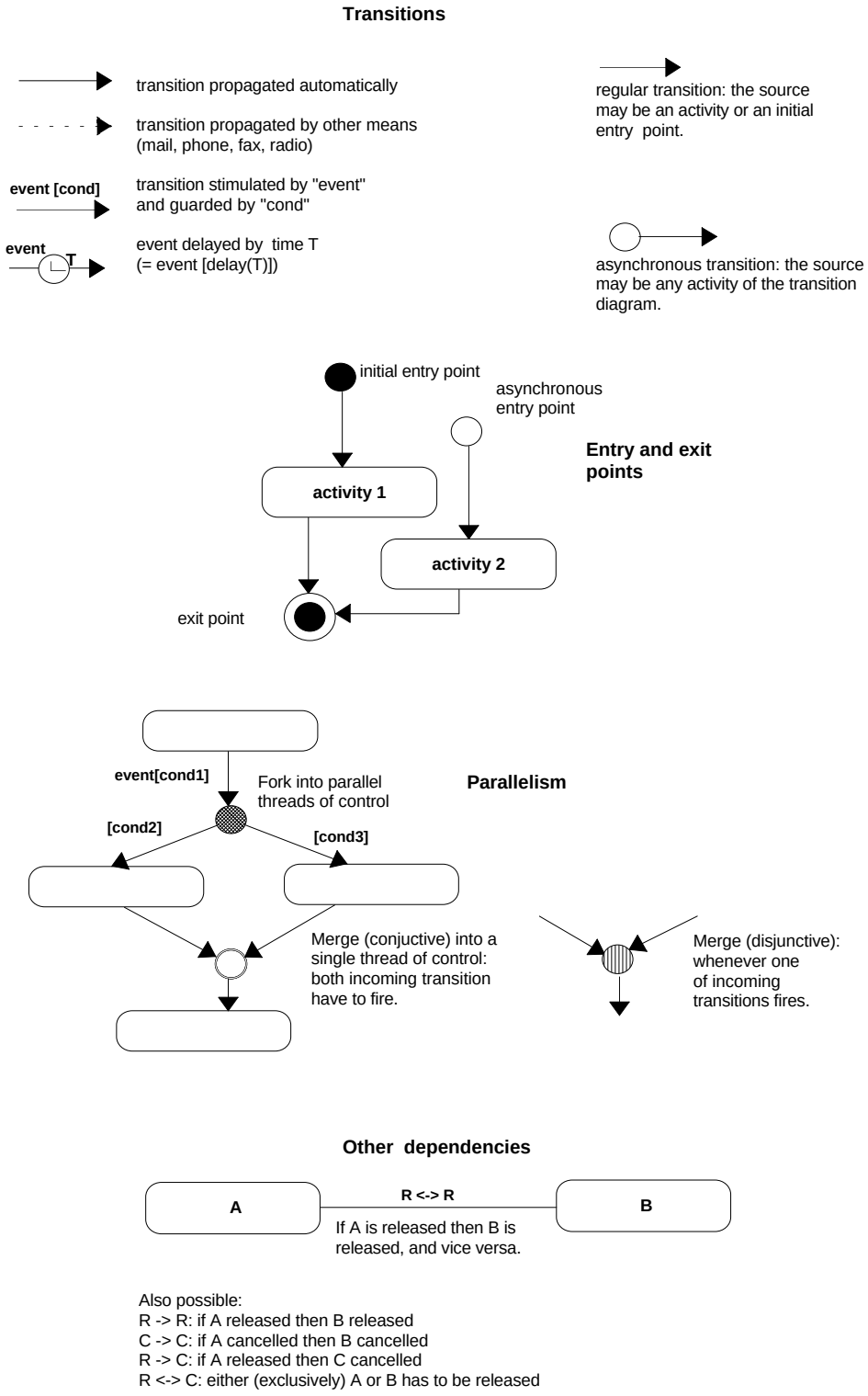


Figure 5.20: Elements of graphical notation for transition diagrams

5.3.1.3 Nodes

There is always an actor associated with a node. If that actor is defined as an *organisational unit*, e.g. a group of people, *cooperation* is present in the node. Nodes may be nested and thus the scenario definition may be decomposed into lower level scenario or activity definitions in a structured fashion. For the purpose of stepwise decomposition of nodes, the actors may be also nested concepts, e.g. a department, a person role, etc.

Non-automatic nodes, i.e. nodes that do not have computerised support for execution of the work being defined, are drawn using dashed lines.

There may be parallelism among nodes at any level of the specification. Possible parallel nodes are depicted explicitly using a special transition notation (see below).

There may be a precondition and a postcondition associated with a node. The conditions are Boolean functions defined over detectable events, and global data (including time). The execution of a node does not start before the precondition evaluates to `true`, and it can not finish (i.e. generate exit events) before the postcondition evaluates to `true`. Preconditions and postconditions are best suitable for nodes which have a simple internal structure (e.g. a single entry state and a single exit point)

Various binary dependencies may be defined among nodes. The most typical ones are shown in figure 5.20.

By default, nodes have *no traditional transactional properties* (the ACID properties). Such properties should be defined at some level of the decomposition if nodes use *global data, persistent data, or data shared by concurrent threads of the scenario definition, or data shared by multiple scenario instances*. The ACID properties as well as any subsets of them (AD, I, etc.) can be declared for a node by attaching the corresponding letters at the bottom of a box representing the node. This declaration implies that the properties will be automatically enforced by the system. If compensation is used for recovery, it should be explicitly expressed in the transition diagram, using nodes to describe the compensating activities and their execution order.

5.3.1.4 Transitions

A transition is characterised by two components: an event stimulating a transition and a guard function (condition) which has to be satisfied in order for a transition to fire (i.e. take place). The event may be any event external to the system, or an event generated in the system, or an event of ending (completing) the execution of the source node. The condition is a Boolean function defined over all global data, data local to the source node and the exit points of the source node.

The notation `e1[c1]` is used to indicate that the transition takes place when event e_1 has happened and the condition c_1 has evaluated to `true`. For example if e_1 happens at time

t_1 and c_1 evaluates to `true` at time t_2, the transition fires at time t_2. If the condition part is omitted, it means that the event fires the transition unconditionally. If the event part is omitted, the default event of ending the source activity (the `released` exit event) is assumed. Therefore, an unlabelled transition between nodes $\mathcal{A}$ and $\mathcal{B}$ means: "the execution of $\mathcal{B}$ is started immediately after the execution of $\mathcal{A}$ has finished".

A special symbol (a "clock") with a time span value `t` stands for the condition "time `t` passed since the event occurred". A transition with a clock symbol thus represents a timer set by the event.

There is a special type of a transition to handle asynchronous (i.e. state-independent) events. An *asynchronous transition* may be fired regardless of the node being executed. In other words, any node may be a source of it. Asynchronous transitions are marked using a special symbol (a circle) as an entry point. Asynchronous transitions are used to model effects of asynchronous events, such as a device failure, a reset command, an expedited abort command, etc.

When an asynchronous transition happens the following takes place:

1. A “Cancel” event is sent to any activity in execution within the decomposition unit (a scenario, an activity or a subactivity at any level) the asynchronous event is sent to.
2. The control is passed to the activity pointed by the asynchronous transition.

Using the above mechanism, a “partial rollback” of a scenario (or an activity) can be easily modelled: An asynchronous transition is drawn pointing to the (sub)activity one wants the processing to be resumed at.

Transitions maintained by non-automated means are represented by dashed lines.

5.3.1.5 Parallelism

There is only one way to denote parallel executions of nodes. A transition may be *forked* to generate parallel threads of control. A special fork symbol is used for this purpose (figure 5.20). Each of the forked transitions may have its own condition which is AND-ed with the source transition condition. Parallel threads of control proceed until a merge symbol is reached. There are two types of the merge symbol. The conjunctive (AND) merge generates a transition when both merging transitions are activated. This is used for synchronisation of parallel threads. The other type is the disjunctive (OR) merge that generates a transition whenever one of the source transitions is activated. In this case, the thread with the inactive transition is stopped by the system.

If the threads do not merge, they terminate independently. The scenario execution does not terminate before all parallel threads have terminated.

Transitions originating in different threads of control can not meet at the same activity without a merge symbol.

5.3.1.6 Entry and exit points

A transition diagram has to have at least one *entry point* and at least one *exit point*. An entry point symbolises the incoming transition, and an exit point symbolises the stopping of processing within the transition diagram and generating a corresponding event. Entry points are unlabelled. An entry transition is labelled with the name of the stimulating event and the condition, and the exit point is labelled with the name of the event being generated.

Although any event may be generated at an exit point, there are two standard exit events: one is `released` meaning a normal termination, and the other is `cancelled` meaning an abnormal termination. The semantics of abnormal termination is specific to the node being depicted.

There is one special exit point used while decomposing a node, which (sometimes) does not generate any event. It is labelled `wait`. When the execution of a node reaches the wait exit point, no event is generated. Rather, an event external to this activity will stimulate a next transition.

The only events the transition diagram shows explicitly are exit events associated with exit points. If an execution of a node generates other events, they are not shown in the diagram. The exchange of events among nodes may be shown on a corresponding DFD diagram.

A *control path* is any distinguishable sequence of transitions between an entry point and an exit point. There is no node outside the set of control paths. A control path may encompass parallel threads if such threads are used.

If a node does not have outgoing transitions, it is assumed it is connected to an exit point labelled `released`.

5.3.1.7 Decomposition rules

The transition diagrams are decomposed much in the same fashion as the DFD diagrams are. An example of the node decomposition is shown in figure 5.21.

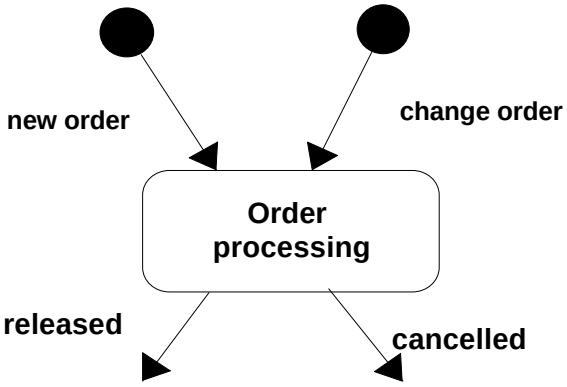
A node shown at level `i` may be decomposed at level `i+1`. Any incoming transition at level `i` is represented as an entry point at level `i+1`. Such a transition is labelled using the name of the source node (omitted if the source is an entry point at level `i`), the event name and the condition. For example, the entry transition labelled:

```
delivery planning: order update [time = 13:00 - 16:00]
```

denotes a transition which originates at node `delivery planning` and is fired if the event `order update` has occurred and the current time is between 13:00 and 16:00.

To ease labelling, the nodes and transitions may be numbered in the DFD fashion.

Any exit point (other than `wait`) at level `i+1` is represented in at least one outgoing transition at level `i`. If there are events in outgoing transitions at level `i`, which are not represented as exit points at level `i+1`, there must exist the exit point `wait` at level `i+1`.



The order processing activity is started whenever a new order or an order change request arrives. The goal of this activity is to store the order, check the correctness of the request, approve the request if acceptable, and generate the necessary feedback to the customer.

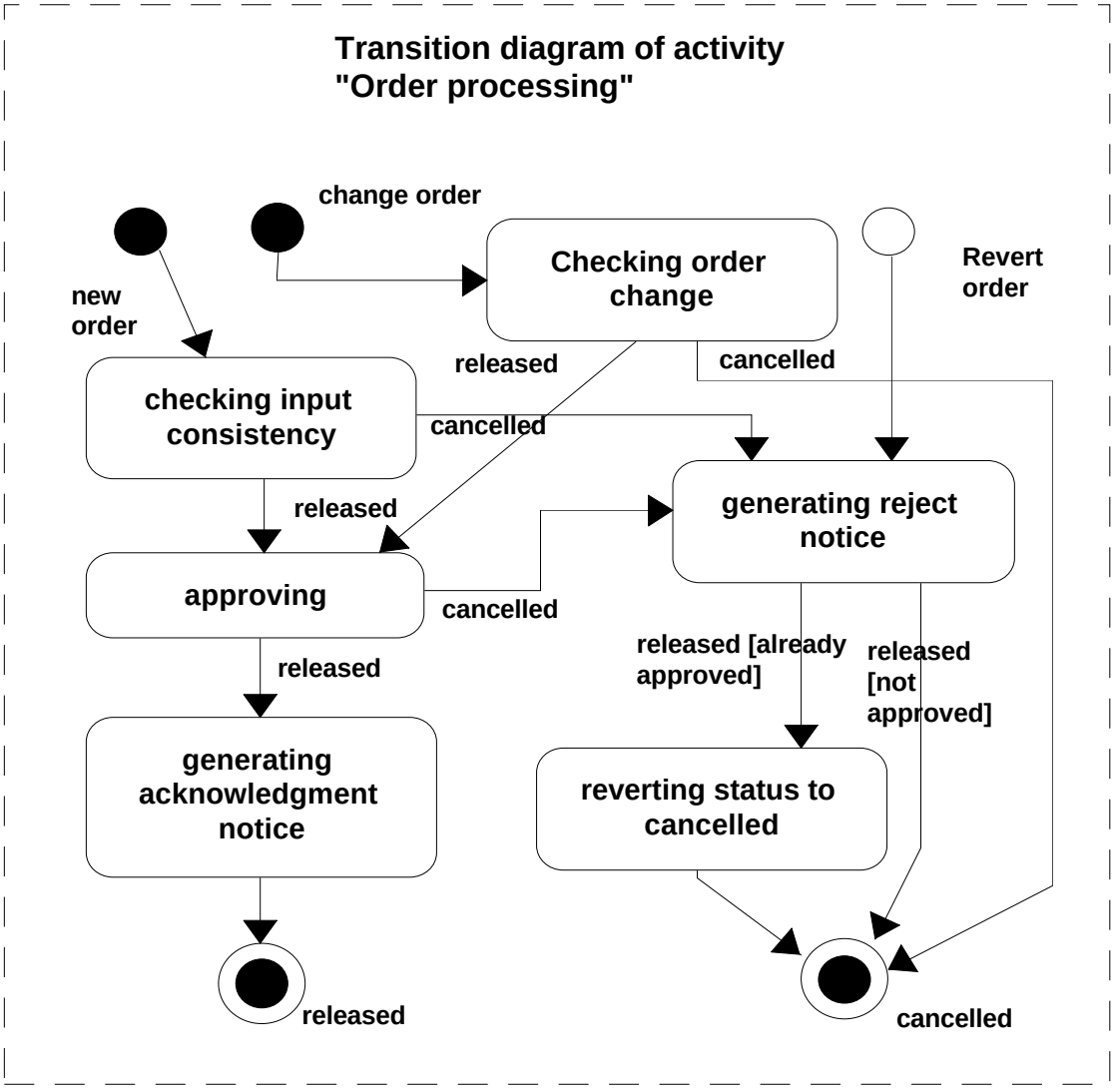


Figure 5.21: Example of a transition diagram decomposition.

5.3.1.8 Rules for incoming transitions

If many transitions point to a node, the first one to fire starts the execution of the node. At this point all other events leading to the node are reset (i.e. their status is "not occurred"). Once the execution of a node has completed, the transitions may fire again.

Looping in the transition diagram is achieved by introducing a transition loop within the same thread of control.

5.3.1.9 Rules for outgoing transitions

Within a single thread of control, only one outgoing transition may fire. Any other situation is considered erroneous.

5.3.2 The PortNet case

5.3.2.1 An Overview of the PortNet System

The PortNet System has been in the making since 1992. It is to handle notices of vessel arrivals and departures. In the near future it will also include the invoicing of vessel visits. The functional analysis of the system was done at 1992 by a Finnish consulting company, EDI Management. During the year 1993 the same company defined the EDIFACT messages used in the system. The implementation work (the PortNet System Project) has taken place during the years 1993–1994 and has mostly been done by a Finnish software house, VTKK Group Ltd.

Ten Finnish ports have participated in the PortNet project, as well as the stevedoring companies in the ports, the National Board of Customs, the National Board of Navigation (MKH), the Organisation of Foreign Traffic Companies, the Finnish Ship Agent Organisation, the Finnish Port Organisation, the Finnish Forwarders' Organisation, the Central Organisation of Finnish Industry, the National Board of Waters and Environment, Alfons Håkans Oy and Neste Oy.

The goals of the PortNet project are to simplify the current business processes of the participants, to create a uniform business process model for all Finnish ports and to gain reduced costs as well as other benefits.

5.3.2.1.1 The Old System In the old, non-automated system, the shipping company agent has to file in several forms, with basically the same data contents, for the different authorities.

Let us look at an arrival of a foreign ship. The arrival data needed is originated by a foreign agent who sends a manifest to the shipping company agent. The shipping company agent:

1. Forwards this manifest to a stevedoring company.
2. Sends the port in question a "Port declaration" form and a "vessel declaration for the Port" form.
3. Sends the local customs office the "vessel declaration for the Port" form, a notice of arrival, a "vessel declaration for the customs" form and later on the vessel's bill of entry.
4. Calls the local pilot station to order a pilot to navigate the ship to the port and
5. Calls the service providing companies for services needed at the port.

Other data exchange involved in the ship arrival are:

- The stevedoring company sends the port information about the container terminals.
- The port sends schedules on arriving (and departing) ships to MKH.
- The local customs office sends notices of arrival to MKH.
- Based on all this information, ports and the MKH produce statistics on the im-Port/exPort cargo flow in the Finnish ports and send these statistics to the interested parties: the port organisation, the press, State Railways, the customs, the city and the forwarders.
- The stevedoring company, the port, the local customs office and MKH also send the shipping company agent bills of the services they have provided for.

5.3.2.1.2 The PortNet System In the PortNet system the agent's work is rationalised so that he sends only the manifest (to the stevedoring company), the vessel declaration for customs and the vessel's bill of entry (to the customs) manually. The rest of the data, the notice of arrival, the service orders and the declaration of dangerous goods the ship may be carrying, are sent to the PortNet system as an EDIFACT message, "the notice of arrival and departure" (IFCSUM,S93.A).

The PortNet message service is implemented and maintained by the VTKK. VTKK translates the incoming EDIFACT messages into so called "in-house" format and inserts the data contents into a VTKK database. The incoming messages are also collected into batches and delivered to the authorities (ports and the MKH) on fixed time intervals (e.g. every half an hour).

The PortNet System provides for terminal connections to the VTKK database. This way interested companies, the National Board of Waters and the Environment, ports and customs can query time schedules, service requests and dangerous goods information of the arriving and departing ships. In the future, this data will be transferred automatically to local systems.

The PortNet PortInvoicing subsystem is currently being implemented. With this subsystem ports can send their invoices in EDIFACT form (INVOICE, 91.1) to agents, stevedoring companies and exPort/imPort companies.

5.3.2.1.3 The Status of the PortNet Implementation Project The PortNet system has been in a test phase since the beginning of 1994. The uncertainty about Finland joining the EU delayed the production use somewhat since a lot of the business processes involved change when Finland joins the EU. The plan is to get major ports and ship agents in production use in the 1st quarter of 1995.

The pioneer users of the system are the ports of Helsinki and Turku, and their related ship agents, MKH and the National Board of Waters and Environment.

Next step of the implementation work is to develop a local Dangerous Goods (IMO) system for ports and to expand the current PortNet IMO database accordingly. Also, there will be a mechanism for the reception and sending of IMO manifests from and to foreign ship agents and forwarders (the PROTECT project). This will be a major improvement to the current situation where the IMO data arrive in the form of a fax message and are typed into the system either by a shipping agent or a port worker manually.

5.3.2.1.4 The PortNet System as a Workflow Implementation A ship visit to a harbour involves a lot of cooperation between different organisations. Upon arrival of a ship, an MKH pilot has to be ready at the right place at the right time, to guide the ship into the harbour. When the ship arrives at a harbour, the service companies and the harbour personnel have to be ready to connect the ship to a dock, to unload and load the cargo and to supply the ship with water and food. When all this is ready, the ship needs a pilot again, to guide the ship back to the sea. A successful visit to a harbour is a question of fine-tuned, well-timed team play. Any delays in getting the requested services cost money. The money loss is counted in thousands of Finnish marks per a wasted hour.

The PortNet System automates sending these service requests as well as sending other information about the properties and current cargo of a ship. The above information is included in a notice sent by an agent. The notice serves as a basis for work planning and triggers manual and automated work at the port in question and at MKH. The agent also gets response on the notice from the authorities, the final response being the bill, the contents of which are based on the original notice plus some additional data gathered during the local processing. Thus the PortNet System is also a typical workflow implementation in which a form gets routed from actor to actor and refined at each step. What makes the PortNet System a special workflow implementation is its inter-organisational nature: currently the system spans three organisational *roles*: the agent, the port and MKH. Several organisations participate in the PortNet System in the roles of agent or port. In the future there will be more roles and more participants.

5.3.2.1.5 The Analysis Work The PortNet System was analysed by interviewing system designers or main users of the pioneering user companies: A shipping agent company Fimag, the Helsinki port and MKH.

The interviewees were asked to name the external actors of the system, and the data gathered from these actors. They were also asked to identify the main (manual and automated)

activities of their local systems and the documents or forms used or produced by these activities. They were questioned about the (temporal and other) business rules affecting the processing. They also named the major disadvantages of the current PortNet system. The results of these interviews are presented in the sequel.

5.3.2.1.6 The Borders of the PortNet System The PortNet System and its communication with the outside world is illustrated in the environment diagram of figure 5.22. Dashed lines indicate manual communication (phone calls, fax messages). The timetable information (ETAs, ATAs, ETDs and ATDs) is shown separately because we analyse that in more depth in the sequel.

5.3.2.2 Transitions of the Inter-Organisational PortNet Scenario, Handle a Ship Visit

The transitions of the Handle a Ship Visit scenario definition are shown in figure 5.23. The processing is initiated by an agent who creates an Advance Arrival Notice (AAN). The execution of the node "Preparing advance arrival notice" can either be cancelled in the middle of the processing by the agent himself or released for further processing. In the former case the whole processing is stopped and the scenario instance is finished. In the latter case the AAN gets sent to the VTKK message service. Note that there is also a recursive transition to the same "Preparing advance arrival notice" node, meaning that in case the agent gets some new information, he can refine the data contents of the message and resubmit it (or, cancel the whole notice). This means that the same node “Preparing advance arrival notice” may have several instances in a scenario instance. The number of instances will be solved at runtime. In the sequel, this kind of multiple instances will be called *bundles*.

When receiving an AAN, VTKK delivers the message to MKH and the port in question. MKH forwards the message to the pilot station closest to the port the ship is going to arrive at. This automated message sending from the agent to the MKH Pilot Station and the port is shown as solid lines in figure 5.23.

All the notices (advance and final) automatically delivered in the PortNet system are of the same EDIFACT message type (IFCSUM,S93.A). The message contains information of:

- The vessel itself (its name, radio call sign, length, width, machine power and so on),
- The arrival to a port or departure from a port in question (the cargo of the ship and the dangerous goods included in the cargo, the depth of the ship with the cargo, the travel plan and the arrival or departure time) and
- The services expected at the port (a pilot, food supplement, etc.).

If the port finds errors in the message it received, for example when an IMO (dangerous goods) code does not make sense, a port worker calls the agent and asks for the correct information. This type of (manual) communication is shown as a dashed line in the figure 5.23.

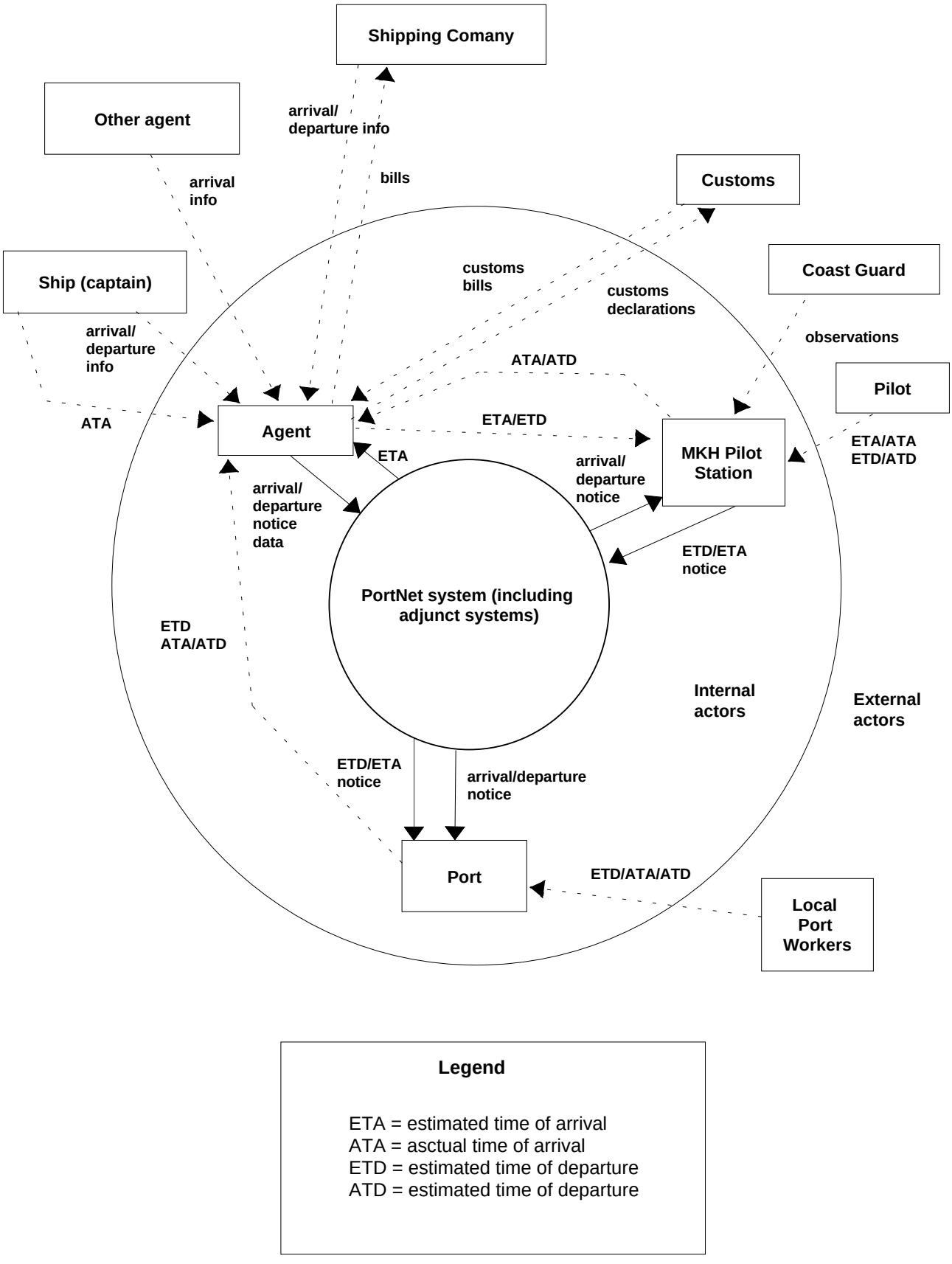


Figure 5.22: Handling a ship visit: environment diagram

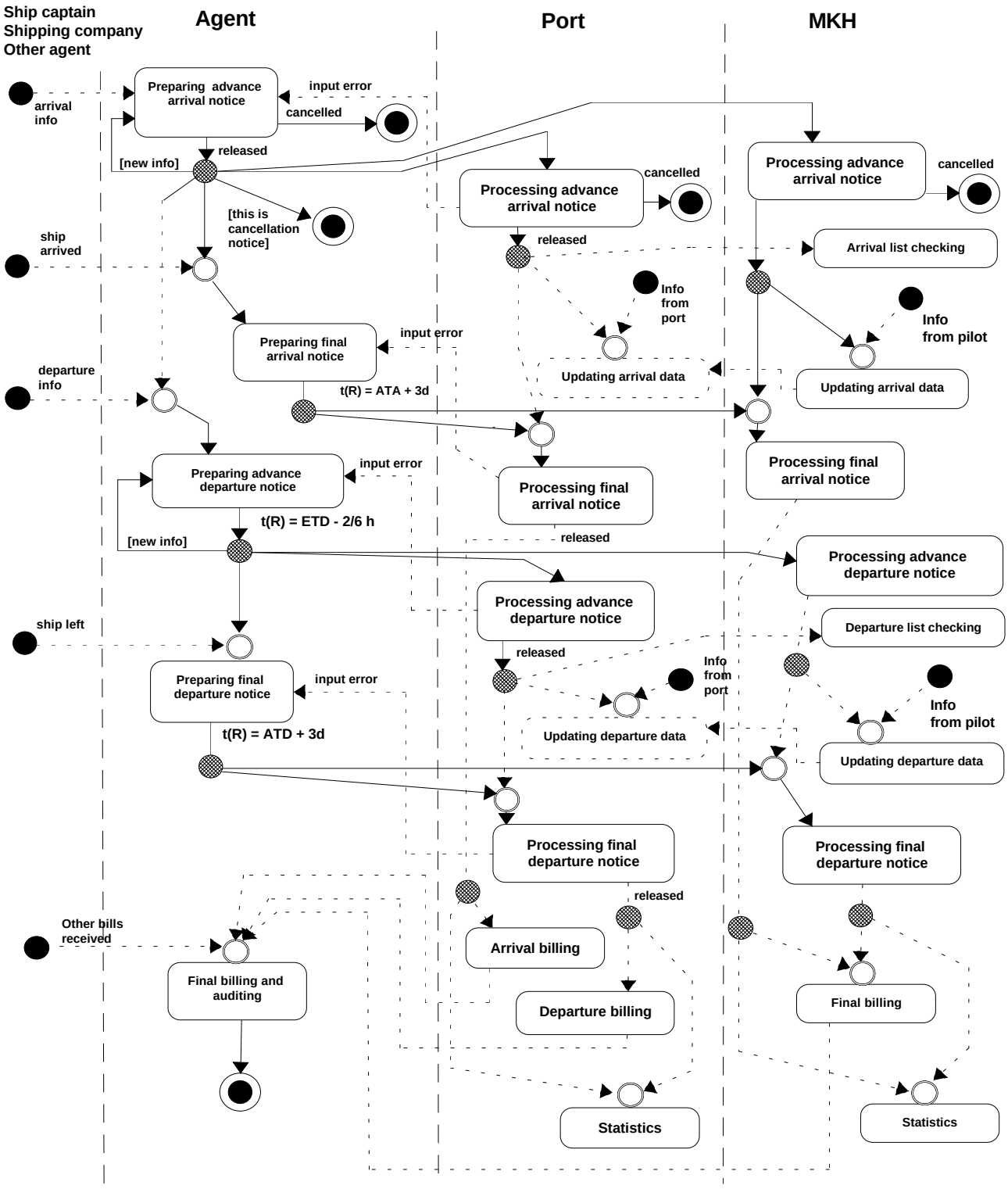


Figure 5.23: Handling a ship visit: scenario transition diagram.

As mentioned before, an agent can also refine the data contents of the AAN (Advance Notice of Arrival). This happens, for example, when the agent gets more accurate information about the ship's estimated arrival time or the dangerous goods or, when the agent finds out there have been errors in the previous AAN. Furthermore, the agent can cancel the whole AAN in case he is informed that the ship will not be arriving after all. After the agent has refined or cancelled the notice, the whole original EDIFACT message gets re-submitted to the MKH and the port, with the new data, a "changed" status code and a reference to the previous AAN. In figure 5.23. the same solid line shows the submission and re-submission of an AAN.

When the port processes the advance arrival notices, an arrival list of vessels is created manually and sent out weekly to the nearest MKH Pilot Station (the dashed line in figure 5.23).

After processing the advance notices, both the port and MKH can find out more about the arrival locally (typically when the agent or the ship captain calls in). This new information triggers the execution of the node "Updating arrival data". This node is manual at Helsinki port (dashed line) and automated at MKH. When a Pilot Station updates the ETA (Estimated Time of Arrival) of the ship, the updated ETA gets sent to the PortNet system automatically as a changed AAN. Currently, only ports receive this message together with other changed AANs and they handle them manually. This is represented by the dashed line between the two "Updating arrival data" nodes.

After the agent has released an AAN and the ship has arrived at the port, the execution of the "Preparing final arrival notice" node can be started. This rule is shown as a bright merge circle in figure 5.23. The execution of the "Preparing final arrival notice" node has to be finished successfully (released) and a Final Arrival Notice (FAN) has to be sent out at the latest 3 days after the Actual Time of Arrival (ATA) of the ship. This deadline (ATA + 3 days) for the node "Preparing final arrival notice" is shown below the node. The FAN notice gets sent to MKH and the port in question in the same way as described above.

The port can start the execution of the "Processing final notice" node only after it has released the execution of the "Processing advance arrival notice" node and received the FAN from the agent (the bright merge circle in figure 5.23). The same rule holds for the MKH Pilot Station.

The port starts the execution of the "Arrival billing" node as soon as the "Processing final advance notice" node has been released whereas MKH waits until both "Processing final arrival notice" and "Processing final departure notice" nodes have been released before it starts the final billing. Both MKH and the Helsinki port have an automated system for invoicing but the data exchange between the PortNet System and these systems is currently still manual.

The agent can start the execution of the "Preparing advance departure notice" node only after the "Preparing advance arrival notice" node has been released and the departure information has been received. Notice that there is no restriction in timing between the "Preparing final arrival notice" and "Preparing advance departure notice" nodes. This is because in practise, the Advance Departure Notice (ADN) can be sent out long before the ship actually arrives.

Otherwise, the processing of the Advance and Final Departure Notices (ADN and FDN) is almost identical to the handling the notices of arrival. However, at the port, the Statistics activity is started only after both ADN and FDN have been received.

After the bills from both MKH and the port have arrived at the agent's and the "Final billing and auditing" node has been released, the Handle a ship visit scenario is finished.

5.3.2.3 Data Flows of the Inter-Organisational PortNet Scenario, Handle a Ship Visit

Figures 5.24 and 5.25 show the flows of timetable data (for the arrival and departure of a ship). *ETA* is the Estimated Time of Arrival, *ATA* the Actual Time of Arrival, *ETD* the Estimated Time of Departure and *ATD* the Actual Time of Departure. We concentrated on the updates of these data items since they have the most evident potential conflict.

Figure 5.24 shows that the ETA is originally created by the agent and sent by the PortNet System to the port and MKH. There, it is inserted into the local data stores either manually (Helsinki Port) or automatically (MKH). The Helsinki port maintains manually an Arrival List which is also sent weekly (on paper) to the nearby Harmaja Pilot Station. Currently this list is found trustworthy and it serves as the basis for planning of the operative work of the station, i.e. the pre-allocation of the pilots. Therefore, the ETAs of the list are updated in the local system.

When getting closer to the estimated arrival time, the MKH local workers often come up with a more accurate ETA which is inserted into the local system as well as sent back to the PortNet System. The port updates (manually) into the Local Vessel Diary both the ETAs defined by local workers and the ones delivered through the PortNet System. Note that the same ETA may be sent by the agent several times, in changed Advance Notices.

At the port, only the ATAs defined by local workers are inserted into the Vessel Diary. At MKH the ATAs are defined by the local workers as well, and updated into the local system. Unlike the ETAs, they are not sent back to the PortNet System, but the agents often call in to ask what the ATA of their ship actually was. This manually obtained ATA also gets written into the Final Arrival Notice of the agent and sent back to MKH and the port via the PortNet system.

Due to this (manual) data exchange, the conflict serialisability graph of the nodes is not acyclic. For example, there is a cycle between the port and MKH:

- Port:Updating Arrival Data -> MKH:Arrival List Checking (MKH Pilot Station updates weekly the ETAs of the Port Arrival List in the PilotNet database).
- MKH: Arrival List Checking -> MKH: Updating Arrival Data (When receiving better information, a MKH Pilot Station worker updates the ETA earlier updated based on the Arrival List).
- MKH: Updating Arrival Data -> Port: Updating Arrival Data (the MKH ETA is sent to the port and updated in the Vessel Diary).

In practise, lost updates and other errors are prevented by human intervention (the port and MKH) or rules hard-coded in the database application programs (MKH). Examples of these manual and automated rules are:

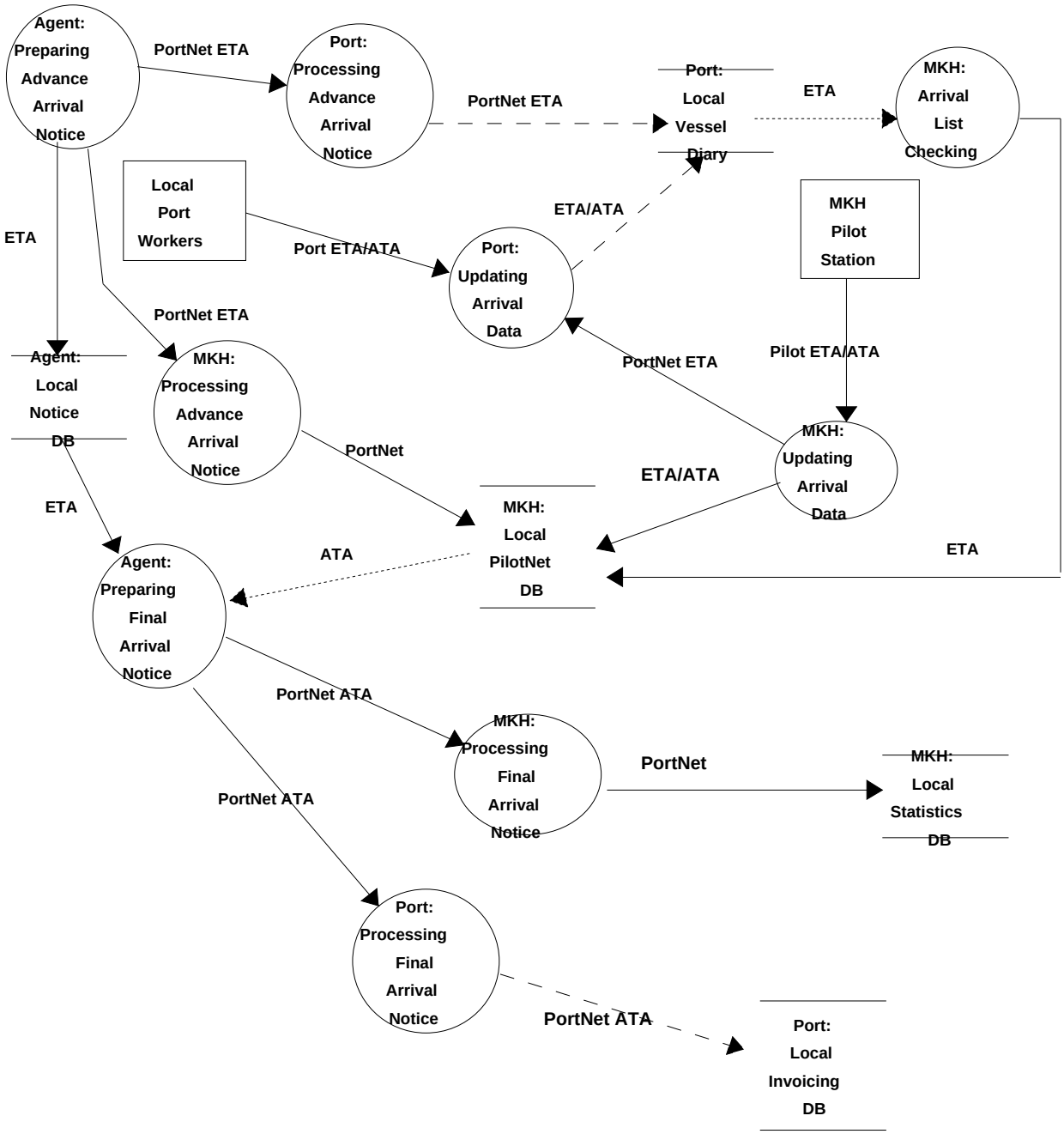


Figure 5.24: Handling a ship visit: scenario data flow diagram: arrival timetable data flow.

- If an ETA is received from the PortNet System after the status of a pilot order has been changed to *confirmed* (the ship captain has called in and confirmed the order), the PortNet ETA will be ignored (automated rule at the Pilot Stations).
- If the Arrival List has ETAs of ships whose arrival has already informed to the Pilot Station some other way, those ETAs will be ignored (manual rule at the Pilot Stations).
- Any ATA received from the PortNet System will be ignored (automated rule at the Pilot Stations).

Clearly such a system is still error-prone and would benefit from support in defining and maintaining these practical rules.

The timetable dataflows in a departure of a ship are almost identical to those in an arrival of a ship.

5.3.2.4 Preparing Advance Arrival Notices

To show how the analysis work would proceed we have refined the node "Preparing Advance Arrival Notice" to two lower-level diagrams: a data flow diagram and a transition diagram. At this lower level there are traditional transactional properties to be supported for the activities: for example the node "Composing the notice" should be isolated and durable.

A cancellation of the notice currently prepared may happen even in the middle of the preparation. This is shown as two asynchronous events: "Cancel preparing of this notice" with the condition that the notice has not yet been sent and "Cancel preparing of this notice" with the condition that the notice has already been sent. In the former case, no additional activities have to be performed and the execution of the "Preparing Advance Arrival Notice" node just finishes unsuccessfully. In the latter case, a sequence of compensating activities has to be executed before finishing. This sequence is: 1. "Composing the Cancelling Notice" and 2. "Checking, approving and sending the notice".

5.3.2.5 Applicability of the Current Products

Despite of its workflow nature, the PortNet System currently does not take advantage of any workflow products, neither globally at the inter-organisational level nor locally at the processing site level.

Using the products globally would be difficult indeed, since there are no workflow products available that could span heterogeneous platforms and at the same time, guarantee any kind of correctness for the overall execution. Currently the inter-organisational communication is based on sending and receiving EDIFACT messages and the products used are data communication software and EDIFACT translators. Locally, activities are performed with existing database applications which are implemented on top of relational database products.

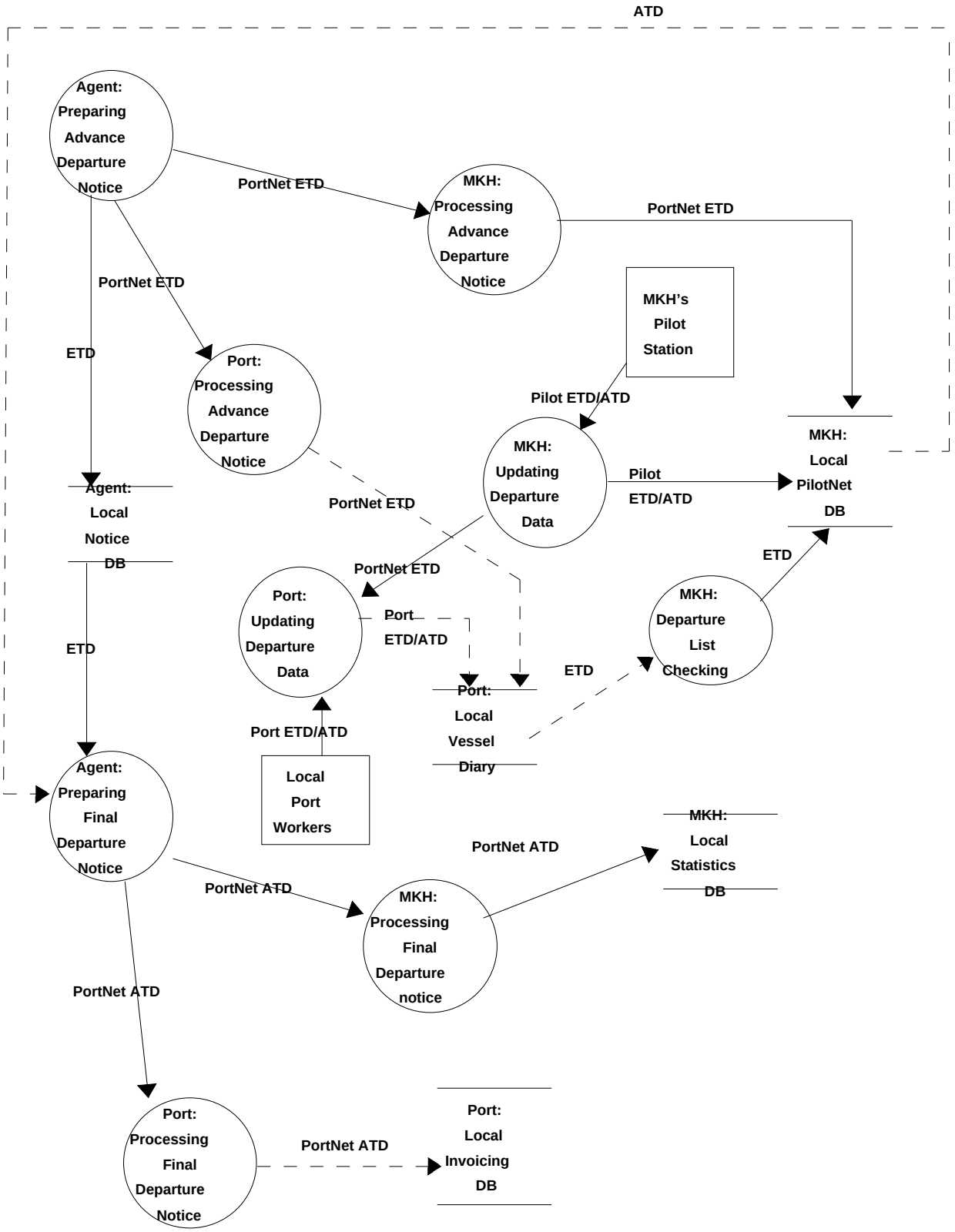


Figure 5.25: Handling a ship visit: scenario data flow diagram: departure timetable data flow.

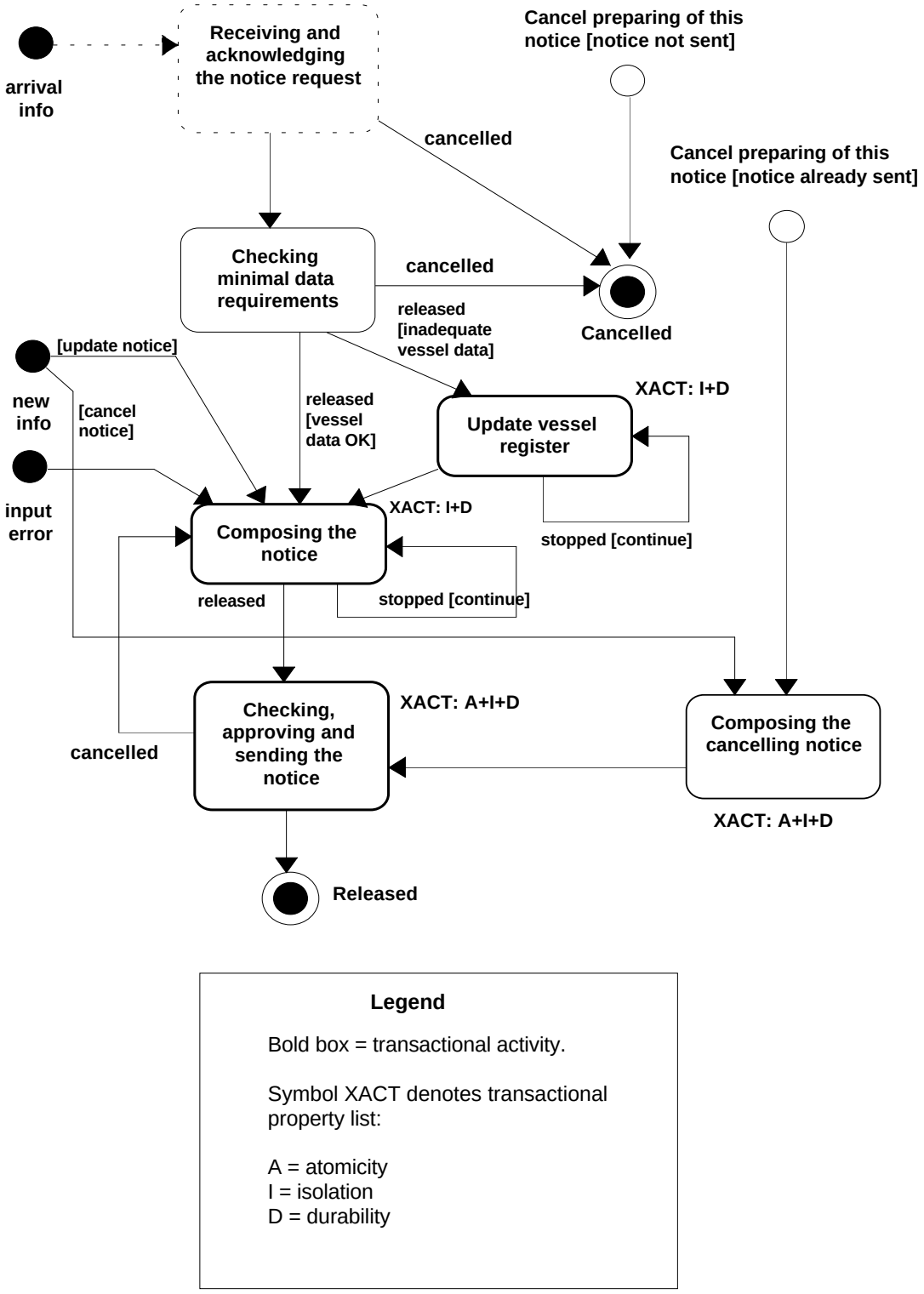


Figure 5.26: Handling a Ship Visit: Preparing Advance Arrival Notice: Transition Diagram.

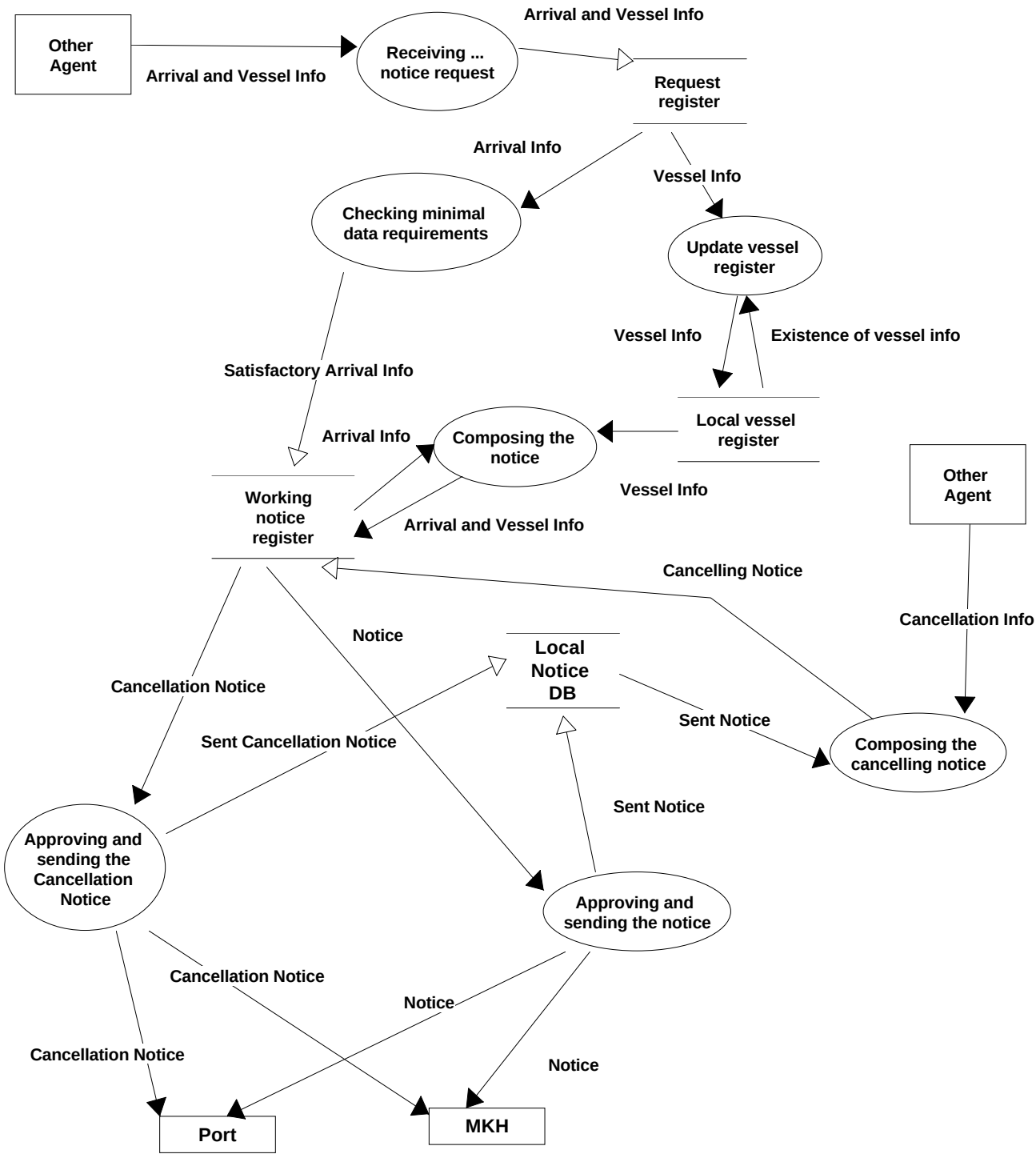


Figure 5.27: Handling a Ship Visit: Preparing Advance Arrival Notice: Data Flow Diagram.

5.3.2.5.1 Specification Tools at the Inter-Organisational Level The state of the art in specifying EDIFACT systems is to define the message syntax. Hardly any emphasis is put into defining the semantics of the messages, the externally visible actions the messages should lead to and the protocols by which the messages should be exchanged. This is the case with the PortNet system, too. Even the rather superficial analysis that was performed on behalf of the TRANSCOOP project convinced the users that a more thorough specification would be worthwhile.

Some existing workflow products, e.g. FlowMark, provide hierarchical modelling tools which would be sufficient for these specification purposes. Some workflow tools also provide support for the verification of the specifications by animation, which is a useful feature in a complex, inter-organisational environment. However, to function correctly, these tools require a specification detailed enough to serve as basis of an implementation since the aim is to generate an executable program.

In the PortNet case a detailed specification is not meaningful because the tools cannot possibly create a program that would work correctly in such a distributed, heterogeneous environment. Also, even a detailed specification is not necessarily favourable at the inter-organisational level since some of the participating organisations may be competitors and they may not want to reveal the details of their local processing. Furthermore, the details of local processing vary from organisation to organisation even though they may act in the same role in the scenario definition. For this reason the detailed specification of the inter-organisational system would be very complex.

Therefore, our conclusion is that the definition of inter-organisational scenarios would require *a black-box approach* at the upmost (inter-organisational) level. The black-box approach means specifying

- The messages exchanged,
- The organisational roles the messages are sent to,
- The exchange protocol and the local business rules that affect the protocol (e.g. reasons to reject a message).

5.3.2.5.2 Implementation of Data Exchange In the PortNet case the automated data flow is one-way: from the agent to the authorities. Any additional information needs are satisfied via phone calls or fax messages. The PortNet system also lacks a return receipt mechanism at the application level. The above design decisions have led to the current situation in which the phone is still heavily used for confirmations. In order to make the system truly automated, the data communication of the system should be enriched with return receipt message types. This is quite possible with the products currently used in the system.

5.3.2.5.3 Local Processing Locally the EDIFACT messages are translated into in-house format and the valid data is either inserted into a local database by a traditional database

application (MKH) or printed and manually typed into the local systems (the Helsinki port). The local data is then processed further by pre-existing local database applications. Therefore, the local systems lack some typical workflow support such as deadline definition, deadline alarming, user, role and organisational unit definition, activity assignment and delegation, audit trails and recovery. This is seen as a drawback by the local users.

Even though the traditional workflow features would be appreciated locally, we are not aware of products that would provide for an easy integration with EDIFACT translators and existing database applications. Applying the products locally would require tailoring which might lead to costly maintenance of the system.

None of the existing products does provide for *backward recovery by compensation*, *support for relaxed ACID properties* of scenarios and activities or *definition of local correctness criteria* which would clearly be useful features for the PortNet system.

5.3.2.5.4 Conclusions Even though there are products that would solve some of the PortNet problems, there is no set of products that could be easily *integrated* with each other and with the existing products and applications of the PortNet case and that would support *the whole life cycle* of an inter-organisational workflow system, from specification to maintenance. Also, there are no workflow products that would *guarantee the correctness of the execution* of an inter-organisational scenario in an *distributed and heterogeneous* environment.

5.3.3 The Valmet case

Valmet Oy is a major metal industry corporation in Finland. It is one of the world's biggest paper mill machinery producers having production units in many countries in Europe and in the USA. The topic of this survey is the problem reporting system that Valmet is implementing for tracking and controlling the maintenance of certain software systems in production units.

The programs which this problem reporting concerns have great significance for the daily functioning of the production units. It is important to get all errors reported quickly to relevant parties. The software maintenance fees depend on the quality of the service. The problem reporting system should also keep track of the service level of each supplier. The system also collects statistical information on the problem frequencies of various software components. The prototype system is in beta-testing, though some of its parts are still under development. It is based on Lotus Notes.

5.3.3.1 The Problem Reporting System Overview

The organisation of software problem reporting at Valmet is depicted in figure 5.28. This system embeds a scenario between many organisations involving production units of Valmet, the corporate level management of Valmet, software suppliers, and a software distributor.

In this report we concentrate on the inter-organisational scenarios. Within each organisation there is a local scenario model that might be rather complicated.

In the production units each software system has a *responsible user* (ruser) who is supposed to know more of the program than the other users. The ruser may locally solve some of the problems other users report to him. Those problems that the ruser cannot solve, are forwarded to the *application manager* (appmanager) of the production unit. The appmanager can publish the new problem report on the corporate level help desk bulletin board. In case of software faults that are covered by maintenance agreements, this publication is followed by a scenario instance where actors include the initiating appmanager, the responsible supplier and the distributor. Other relevant parties may follow how the scenario progresses on the bulletin board.

A new report on the help desk will get the status "reported". If the fault is obvious, the report is marked into the category "software fault", which has two subcategories: "fatal" and "operational limitation". The former subcategory means that the software cannot be used at all while the latter rather poses constraints on the use of the software. Faults are usually supposed to be repaired in 3 to 5 days depending on the software component. The report category "change" is used for software changes that are not clearly covered by the maintenance agreements and that need further negotiation with the supplier. The report category "development inquiry" contains cases that need cooperative planning. This category contains development initiatives and specifications of new versions. The corresponding processing is cooperative planning with concurrent access to the report by several authors. In excess to the use of the bulletin board system the users are supposed to use other communication methods as well. For instance, in case of a fatal error an application manager is more likely to take direct telephone contact to the relevant supplier than wait until the supplier observes the report on the bulletin board.

The *supplier* has partial access to the corporate level help desk and is responsible for getting the relevant reports from it. In case of reports in the "software fault"-category, the supplier is usually supposed to reply within 24 hours to the help desk. After noticing a new report on the help desk the supplier changes its status to "received". The supplier is supposed to give an initial response. It may provide first hand estimates on the correction of the faults and possible workarounds for the time the correction process takes. The supplier is also supposed to provide a correction plan to be accepted by the application manager. After giving the initial response, the status is set to "under work". After the supplier has shipped the corrected version of the software to the distributor, the supplier changes the status to "done". After the distributor has delivered the software to the appropriate production units, the distributor sets the status to "distributed". The unit that initiated the whole process sets the status to "approved" after it has found the result acceptable.

On the help desk bulletin board each report corresponds to a Lotus Notes form, which contains:

- Control and authorisation information, which includes:
  - Author

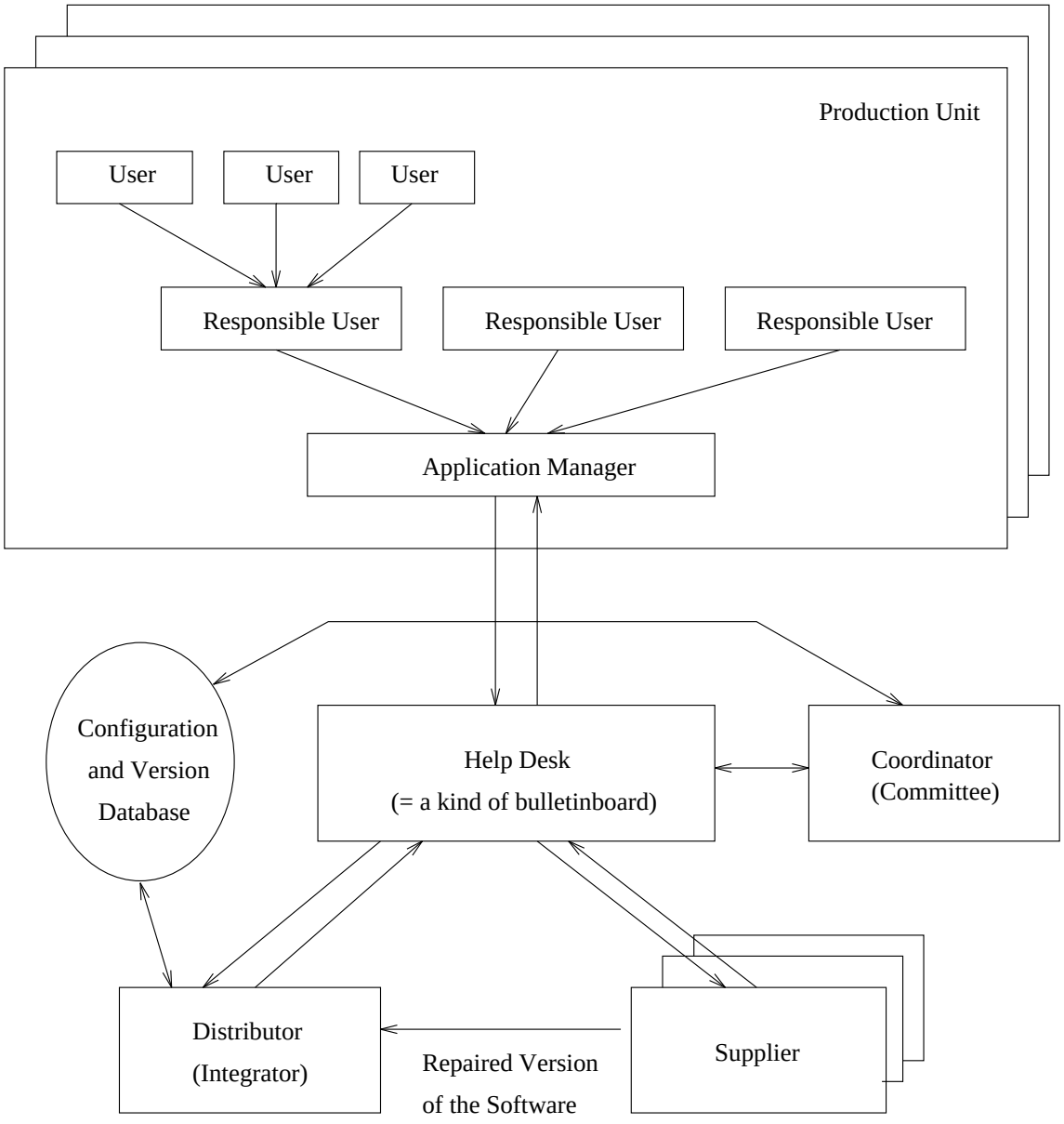


Figure 5.28: Problem Reporting at Valmet

  - Target software identification (also version number)
  - Data access rights for actors
  - Status of processing (reported, received, under work, done, distributed, approved, frozen, cancelled)
  - Report class (software fault, software change, development inquiry, other report)
- Problem description
- Solution description
- Comments
- History of status changes and performance information

The report is divided into four parts that have different access rights:

- reporting area for production units,
- answering area for suppliers,
- common commenting area, and
- measuring area for tracking the quality of service.

All areas are readable to all parties except the measuring area which is visible to Valmet personnel only. The initiating application manager writes in the reporting area and the supplier in the answering area. All parties can write in the common commenting area where each comment is automatically attached with authentication, date and time. The measuring area is meant for presenting two performance indicators which are used for evaluating the supplier. The first one is a function of the time elapsed from initial reporting to the initial response and the second is a function of the time elapsed from initial reporting to the "done" status.

On the corporate level there is a *coordinator* (possibly a group of technical managers and consultants) that makes strategic guidelines and decisions on new software versions and on the overall software architecture that evolves in the production units. The coordinator also gives its opinion in cases where the problems cannot be clearly localised to a single organisational unit. For instance, troubles in interfacing two systems with each other might require solutions at both ends.

The *distributor* gets the fixed version from the supplier and installs and tests it in those production units that want to have it. The distributor maintains and uses the configuration and version database, that is used to track the corporate wide installations. This database is also being used for compatibility considerations.

5.3.3.2 Analysis of the scenario

The underlying bulletin board metaphor provides weak support for scenarios of workflow type. Processing of a scenario is mainly based on the use of the statuses in the corresponding reports. It is up to the users of the system to use the statuses in a meaningful way. The system does not control this.

The control conventions are well-defined in the handling of software fault and change request reports. Development inquiries, and other reports do not have very clear conventions. In those cases the system functions as a discussion forum and workflow modelling is not a natural way for modelling the process. For instance, the cooperative planning process of a new software version needs support for concurrent access of the shared reports by many actors.

Figure 5.29 roughly depicts the scenario for the problem reporting system. In case of software faults and change requests the processing is series of actions. The main actors in this part of the scenario are the reporting application manager, the supplier of the faulty software and the distributor. The changes in the status are marked by italics after each activity. The application manager has rights to set the statuses "reported", "approved", "frozen" and "cancelled". The supplier may set the statuses "received", "under work" and "done". The distributor can set "distributed". Those parties that are not so directly connected to the dialogue can still browse on the bulletin board to find out how the case is developing.

Both browsing and discussion forum activities are presented as sets of parallel activities. For each of the parties an activity thread is launched with an own exit point. The cooperation between parallel activities is handled by data exchange through the report database and the actors may behave asynchronously.

The scenario shows that workflow type modelling suits well to the processing of error reports and change requests but it is not the best means for formulating the planning activities within the problem reporting system.

5.3.3.3 Ideas to further develop the system

The problem reporting system involves many autonomous organisations. The general problem settings behind the system leave room for more ambitious solutions than those in the current beta-version.

The transaction support within the report database could be improved as the current system only maintains one copy of the data and lets the actors do destructive changes to these data.

Status changes should be controlled by the system and there should be time controlling mechanisms, as well. Now users are responsible for such features.

Currently all data is on the bulletin board and the overall response time of the system depends on how active the users are in reading the reports. There is a need to give the system more active role. E.g., it could alarm the responsible actors for delays in processing. This self-monitoring could be combined with forecasting capabilities to estimate durations of scenario instances. In order to give a more active role to the system it might be good to integrate message exchange (e.g. electronic mail) functions with it.

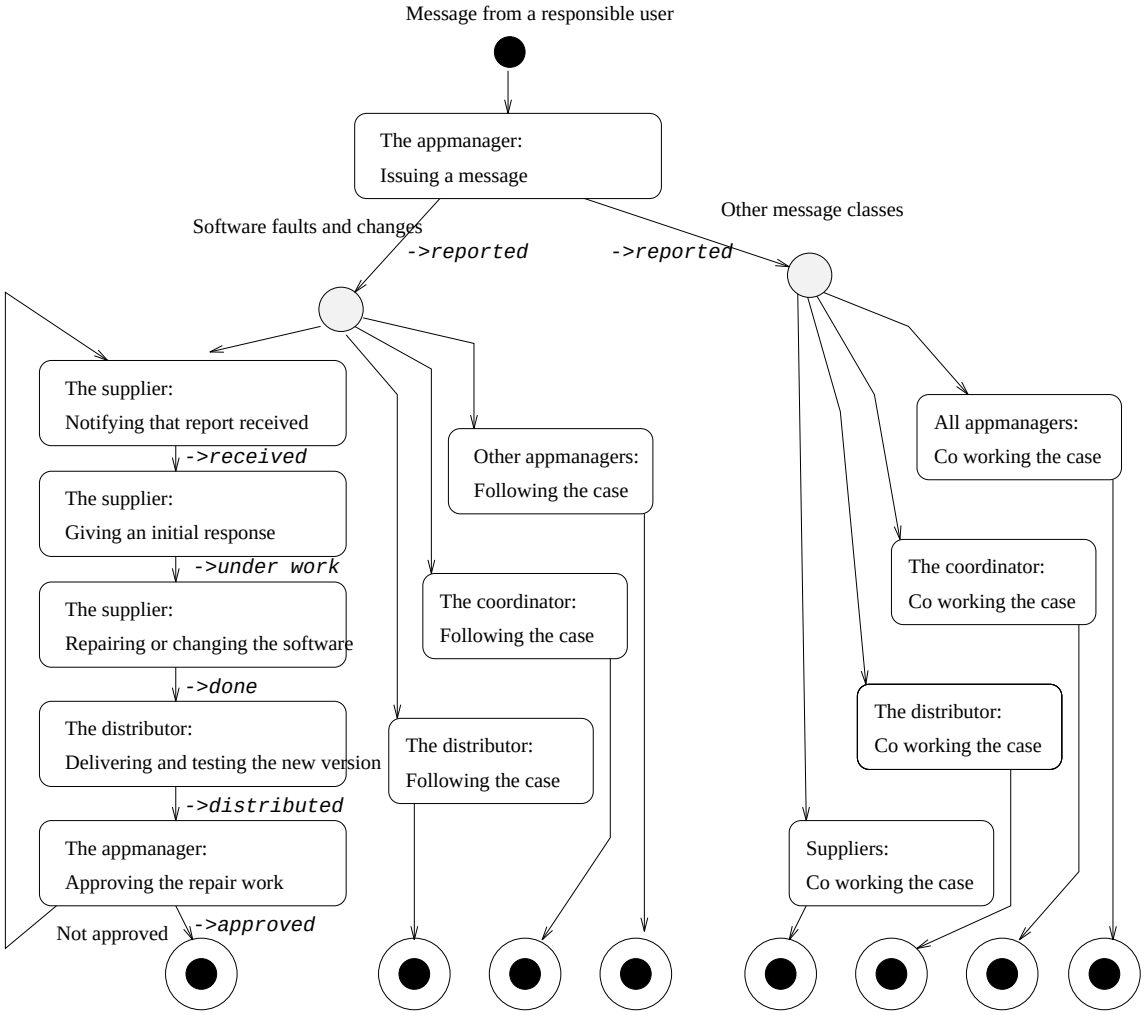


Figure 5.29: Problem reporting scenario

5.3.3.4 Applicability of the current products

The current version of the Valmet problem reporting system is implemented using Lotus Notes[23]. The product suits the current bulletin board based approach well. Properties of Lotus Notes have also had an effect on the design of the current prototype, as the tool was early chosen for the work. However, when more stringent control on the scenario instances is needed, Lotus Notes will not be the most economical solution, as it is merely a document archiving system for shared documents rather than a workflow management product.

There are many commercial workflow management products that could be used for the centralised part of the workflow type of processing in the helpdesk, i.e. the products mentioned in section 5.2 of this deliverable. The lack of product support is more evident when the scenarios span over organisational boarders in a heterogeneous environment. Moreover, in the cooperative design/authoring type of processing there are not suitable products that would enable fine-grained simultaneous cooperation of multiple actors on the same document in tightly coupled manner.

5.3.4 The intelligent networks case

5.3.4.1 Introduction

The term "intelligent networks" (IN) refers to an architectural and functional concept which is intended to ease the introduction of new telecommunication services. Traditionally telecommunication services have been built directly on the hardware of a switch with proprietary tools. This approach is very expensive; it takes several years to specify, develop, test and deploy new services. The crux of the IN approach is that it offers a standardised environment where the software controlling the basic switch functionality is separated from the software which controls the call progression. The aim of the subsequent sections is to give a brief introduction to the concept of intelligent networks, to discuss the relation of intelligent networks and workflow technology, and to offer a vocabulary which helps one to understand the framework of the 101-service described below.

5.3.4.1.1 The evolution of network and switching technology Telecommunication technology has developed rapidly in the past few decades. The advances of computer technology, network technology and switching technology have been the most important technical factors behind this trend of development. The first computer networks suitable for extensive use were introduced 25 years ago. These networks were largely analogous and provided only a poor quality of service, e.g., the probability of bit errors was high. In this respect the packet switched data networks and LANs (Local Area Networks), which were introduced in the late 1970's and in the 1980's respectively, where the turning points of technology.

The switching technology was first manual, then electro-mechanic and in the next stage it was based on analogous circuitry. The control signals were in-band, i.e, the control

[23] Lotus Notes is a trademark of Lotus Corporation

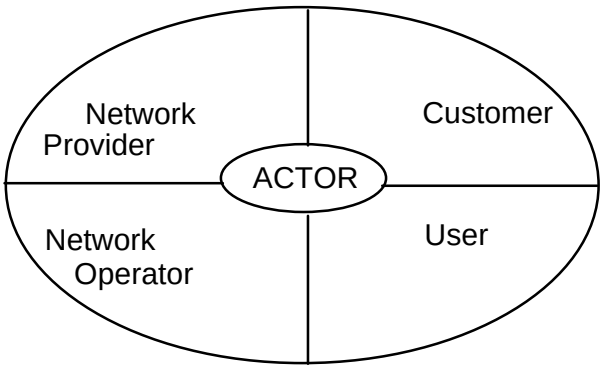


Figure 5.30: Roles occurring in a telecommunication system

signals used the same channels as the signals carrying actual data, and the signals were analogous ones. The advances in circuit technology made it possible to start the design of fully digital switches. During this work it became evident that an essential part of any digital communication network is the set of protocols which control the functioning of the network. For this reason CCITT initiated the work on SS7 protocol stack in 1980. Digital switches employing SS7 and microprocessor technology became commercially available in the 1980's.

The term "intelligent network" was introduced at Bellcore in 1986 to refer to an architecture which supports multiple value-added telecommunications services, which enables the control and management of these services, and which is independent of the network equipment, the services being offered and the type of the underlying telecommunication network. Since the introduction of the IN concept a lot of work has been done to create a widely accepted standard to support the implementation of the IN based telecommunication systems.

CCITT has embarked on a phased standardisation process which aims at the Intelligent Network Architecture (INA). The basic idea of the work on INA is to progress on all areas of IN concepts simultaneously and, despite of this, maintain compatibility between the different versions of the standard. The SS7 protocol stack is an essential part of the new architecture because the network management of INA relies on SS7.

5.3.4.1.2 The actors, roles, needs and objectives related to Intelligent Networks The service model of an IN architecture defines the actors which may occur in a telecommunication system as well as the roles which these actors may have. Figure 5.30 presents a suggested set of roles which an actor may adopt [DO94]. An actor may play several roles concurrently. Consider a situation where a customer orders a telecommunication service from a network operator on behalf of a group of users and also offers technical and administrative support to the members of the group. Because an individual user receives all the technical and administrative support from the customer, the user sees the customer acting like a network operator.

The needs of an actor depend strongly on the role of the actor. A customer wants:

- the possibility to choose the best service at an affordable price in order to use telecommunication as a competitive edge in his own life or business,
- the customisation to meet his own needs,
- a fast response to new service requirements,
- the freedom to use the lowest cost access to network,
- bandwidth on demand.

A user wants:

- the ease of use,
- the same appearance and behaviour for all accesses,
- a high probability for successful communication sessions.

A service provider wants:

- to secure commercial integrity,
- the freedom to customise and personalise service logic,
- the freedom to choose the best network provider.

A network provider wants:

- to secure network integrity,
- the freedom to optimise network performance at competitive costs,
- the reuse of service platforms as much as possible when lower level technologies change.

The principal relationships between the different roles are presented in figure 5.31.

The objective of the IN approach is to offer a new architectural concept that meets the needs of telecommunication service providers. This architectural concept must help the service providers to satisfy the existing and potential market needs rapidly and cost effectively and to improve the quality and reduce the cost of network service operations and management. CCITT has determined the following specific objectives for the IN:

- It should be applicable to all telecommunication networks, e.g., public switched telephone networks (PSTN), including integrated services digital networks (ISDN), both narrow band and broadband, packet- switched public data networks and mobile networks.

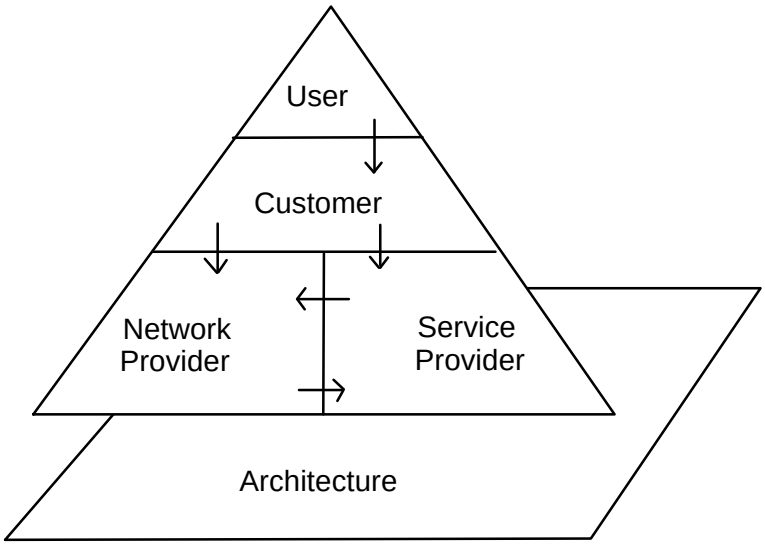


Figure 5.31: The relationships between roles

- It should enable service providers to define their own services, independent of service-specific developments by equipment suppliers.
- It should enable network operators to allocate functionality and resources within their networks and manage their networks efficiently, independent of network-specific developments by equipment suppliers.
- It should be introduced starting from the existing networks and the current CCITT Recommendations.
- It should evolve to reflect implementation experiences, new technological opportunities, and market evolution.

With the traditional telecommunication technology it is not cost-effective to develop short-term services. With IN technology it should be possible to introduce new services rapidly without affecting the available services. CCITT's IN concept defines a large set of standards that describe the interfaces between different network control points. By only specifying the interfaces, IN makes it possible for vendor systems provide with different products and for operators to use any of these products in their network configuration. IN includes also capabilities for other than operators to introduce new services into the telecommunications network.

5.3.4.2 Relationship between IN and workflow

Service creation, service management, service processing and network management are considered to be the most important functional parts of the IN concept. The processes

related to these parts, e.g., the processes related to IN service creation, make workflow concept very useful as a design tool in many phases of IN development. Processes can be described and analysed by means of workflow. Because these processes are supposed to be dynamically changing there is also a need to use dynamic tools for the provision of such flexibility. Furthermore, the workflow techniques require that the processes of the application area are analysed and described explicitly. This helps one to detect problems even before any tools have been used.

One practical example is the flow of documents between different autonomous organisations during the service creation. This process usually contains many unnecessary steps. If we visualise this process there is a high probability that we can notice some of these unnecessary steps even without using workflow tools and thus we can adjust the process more flexible. Making this process more flexible usually means that we achieve improvements in many different ways. One of the main advantages can be time saving. Time is probably the most valuable property associated with these processes and also in any other production process which aims at fulfilling customers needs.

As we have stated before, the IN concept also includes many processes which all are dependent of time and are supposed to be dynamic. We should bear in mind that one of the main purposes of IN is to achieve rapid service provision and subscription. This is the context which is the most obvious connection point between IN and dynamic tools related to workflow. There are many questions to be solved before we can achieve the goal of making telecommunications service subscriber be able to use the service as soon as possible. The IN architecture offers solutions to some of these problems but it does not solve them all.

Telecommunication service providers have also noticed this problem related with service provision time. As a result service provision time is decreasing rapidly. A final goal for the near future has been defined already. This goal is that service provision time should be approximately equal to zero, which means that a subscriber should be able to enjoy the service just ordered by telephone right after hanging up. As we can imagine, this kind of rapid service provision demands enormous dynamic capabilities from the system. Such rapid service provision can not be done with every service. Groups of basic services which can be adapted to a short provision time are usually related to individual subscribers services. On the other hand there are larger services which are usually subscribed to by a company and because of organisational limitations they need not be enabled at once. Individual services usually include changing, adding or removing information from databases which include information about the subscriber and services. These changes are all there is to make a new service available, change old services or subscribers personal information and to cancel existing service. Because of this short period of time to establish a service, it is not possible to make very complicated changes to switches or programs.

As we consider elements that could make it possible to achieve flexibility and in service provision we notice there are still many problems to be solved. These problems are mainly related to IN architectures point of view. Because service provision includes use of databases there must be consistent databases including information about services and subscribers. Because there are many databases there is also a need to build a system which can determine independently what changes are necessary in order to make the service available. This should be an automatic and dynamic process. This means that all the associated information

must be changed in every place where consistency of information is needed and this all have to happen in a short period of time. This process may be complicated depending on whether the system is distributed or not. If the information is located in many autonomous places the refreshing of information has to be done in every place where it is situated. That is why there is a trend to keep different pieces of information in fixed places. This is used in BIN (Broadband Intelligent Network). This information modification process may be carried out by hiding the actual system and communication architecture. There is a need for a kind of "core" which hides these system dependent factors. Here is a natural need for object-oriented programming and instruments. Interfaces are determined in order to make it possible to communicate with the system. These interfaces provide basic services which can be put into practise very easily. Although the interface apparently provides easy tools for changing service and subscriber information there is of course the need for more complex logic behind the scenes.

5.3.4.3 The 101-service of Telecom Finland

Telecom Finland is a Finnish network operator and telecommunications service provider with a turnover of ca. FIM 5 billion/year and 6500 employees. Besides the conventional telephone services (local calls, long distance calls, international calls) and telephone services based on the cellular systems technology (GSM, NMT-900 , NMT-450), it offers more complex services based on the modern IN switches. Freephone, virtual private network and conference calling are some examples of the services based on the modern switching technology.

The purpose of the 101-service is to make it easy for a customer to direct all his long distance telephone calls to the network of Telecom Finland. In principle this is a very simple service but due to the company organisation and actors beyond the control of the company there are many difficult questions related to the process which realizes the service.

When a customer wants to make a long distance telephone call in Finland, he must either implicitly or explicitly select the operator which arranges the connection from the switch of the local operator to the destination. If the customer does not care which company arranges the long distance connection then he only selects the dialling code and the local telephone number as usual. The operator responsible for the long distance connection is then randomly selected among the operators which provide long distance services in this particular geographical area. If, however, the customer wants to govern the selection of the long distance operator, then he must give an operator code before the dialling. To use a certain long distance operator by default, the customer must sign an agreement with the operator. The 101-service is a way to initiate the service agreement process with Telecom Finland.

5.3.4.4 The analysis of the 101-service

Five different actors are involved in the realization of the 101-service. The actors and their mutual principal relations are presented in figure 5.32. *A customer* is a private person who

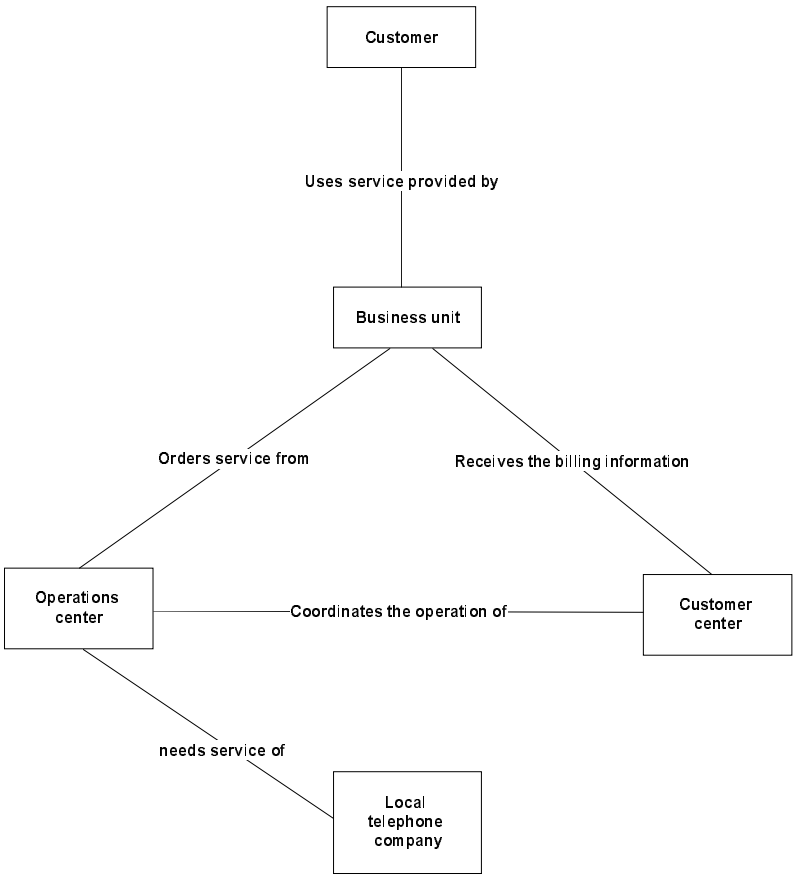


Figure 5.32: The 101-service: environment diagram

wants to direct all his long distance calls to the network of Telecom Finland. *A business unit* is the part of the organisation of Telecom Finland which offers the 101-service to customers. There exists exactly one business unit which offers this kind of service to private persons. *An operations centre* manages all the long distance calls related issues of some geographic area. Telecom Finland has several operations centres around the country. *A customer centre* is responsible for both technical and administrative issues related to local calls in a geographic area. While there are several customer centres around Finland, the customer centres and operations centres do not necessarily reside in same places. The services of *a local telephone company* are needed if the customer lives in an area where Telecom Finland does not offer local telephone services.

The transition diagram and the data flow diagram of the 101-service are given in figures 5.33–5.35. A customer initiates the service process by making a telephone call. An officer of the business unit receives the order and feeds it into an information system. After the check of the customer's credit information, a written agreement is sent to the customer and the operations centre is requested to establish the connection. When the connection is established the customer centre sends a notification to the business unit which in turn informs the customer about the connection. The long distance telephone calls of the customer are then

automatically directed to the network of Telecom Finland.

The main tasks of the regional customer centre is to determine the organisation which is responsible for the local telephone service in the area where the customer lives and to order the physical connection from an appropriate organisation. If Telecom Finland takes care of the local service then the connection order is done via a database. If the local services are provided by another company, a standardised fax message is used to order the new routing of connection.

Several database are used in the production of 101-service. The business unit stores administrative information about the customer in the TG-database. The business unit also initially adds the customer to the billing database. The customer centre stores technical user agreement information in ATLAS-database. Finally, the operation centre stores records of actual calls in a database which is used as a basis for billing. All the mentioned databases are managed by different database management systems.

Several key observations were made of the environment of 101-service:

- The organisation of Telecom Finland is inherently distributed and heterogeneous. Computers and databases are used extensively.
- The organisation of Telecom Finland must satisfy two kinds of needs:
  - the needs emerging from the customer service and
  - the needs emerging from the management of large and very complex technical systems.
- The 101-service contains both manual and computerised steps.
- The quality of the 101-service depends very much on the how the company manages to coordinate and control the various workflows occurring in the service production process.
- There exists actors which can not be steered by Telecom Finland, i.e., the local telephone companies.

5.4 Workflow requirements for the TRANSCOOP specification language

5.4.1 General Requirements

The general requirements of the workflow domain for the specification language are:

1. *Natural constructs* for expressing typical workflow properties. In order to be useful for the scenario specifiers, the specification language has to have a good conceptual correspondence to *workflow thinking*. Our analysis showed that there is an established set of constructs used in workflow modelling. If TRANSCOOP specification language differs too much from this established way of modelling workflows, it may get rejected by its potential users.

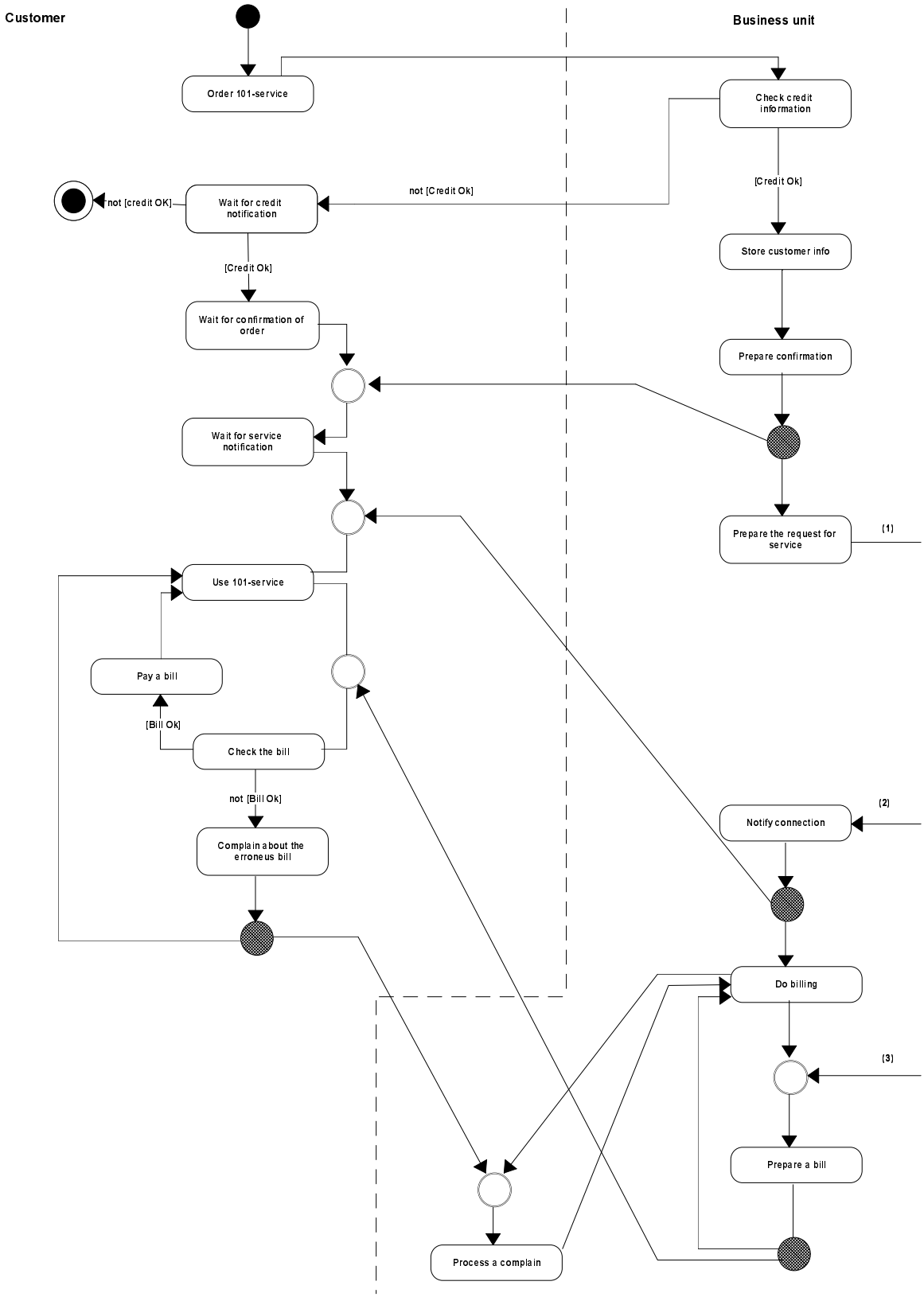


Figure 5.33: The 101-service: transition diagram

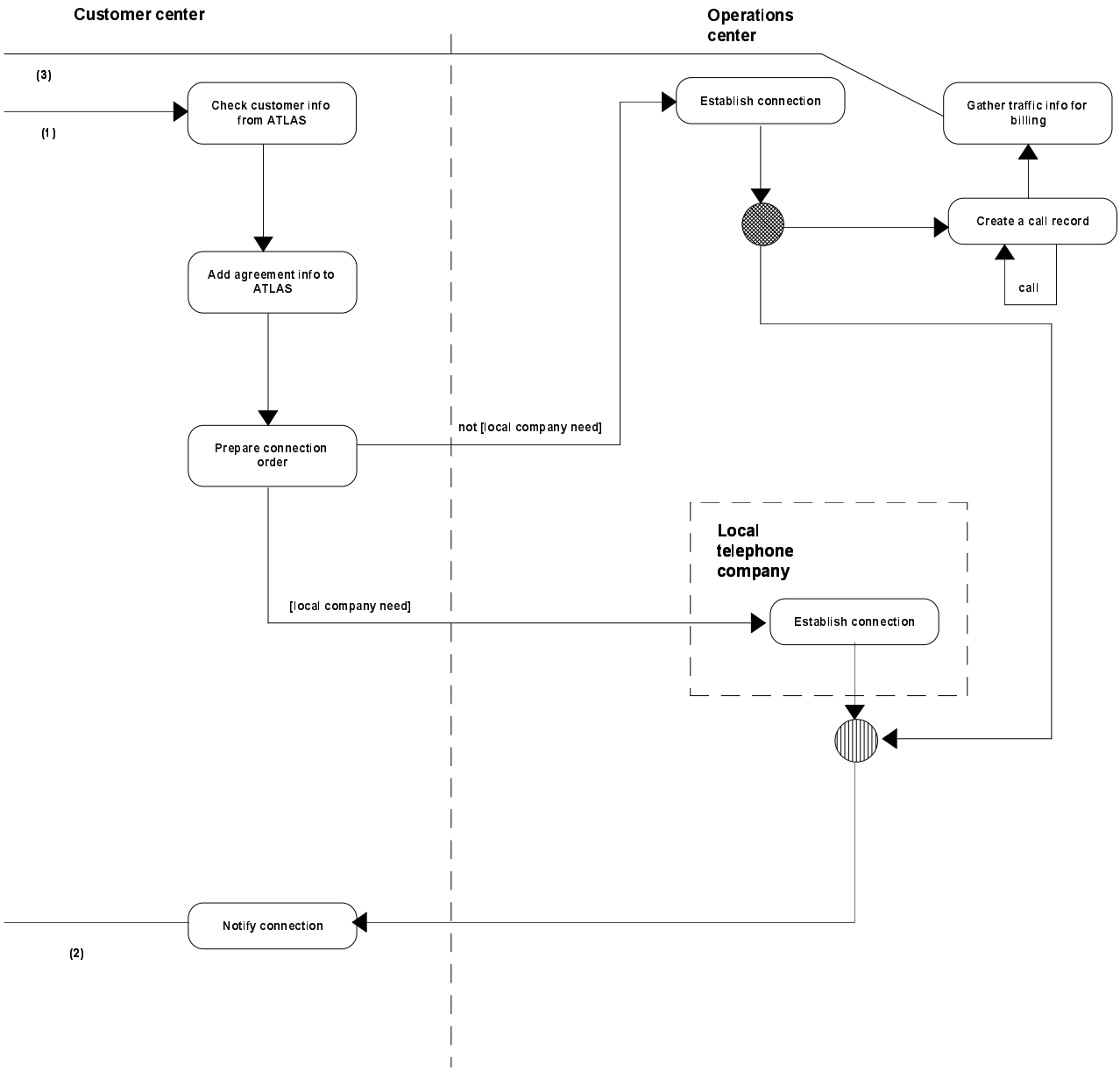


Figure 5.34: The 101-service: transition diagram (continued)

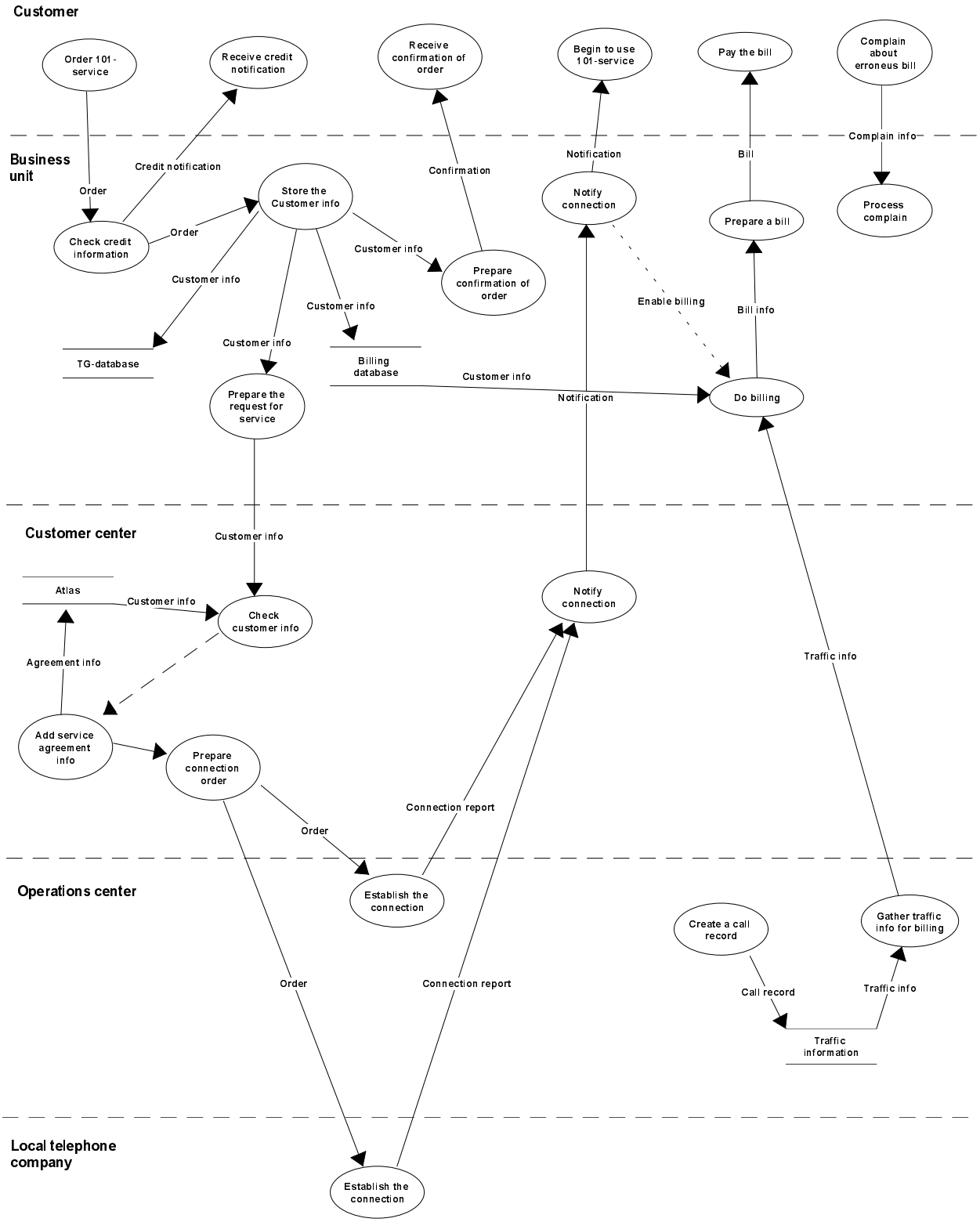


Figure 5.35: The 101-service: data flow diagram

2. *Expressive economy* of workflow specifications. Simple things in workflows, such as sending a message, should yield in simple expressions in the specification language. Also, the maintenance of specifications should be easy.

3. The typical workflow properties the language should be able to express are:

   (a) The hierarchy of specification: scenario definitions consist of lower-level scenario or activity definitions,

   (b) The ordering dependencies of scenarios and activities,

   (c) Conditions for execution of scenarios and activities,

   (d) Multiple entry and exit points of scenarios and activities,

   (e) Parallelism of scenarios and activities,

   (f) Traditional transactional properties (ACID) of scenarios and activities,

   (g) Data dependencies of scenarios and activities.

   (h) Bundles.

4. The specification language should allow a sufficient selection of *data definition and management* features. These features are:

   (a) Definition of complex data structures,

   (b) Definition of relationships between data structures,

   (c) Definition of policies for concurrent access of data and

   (d) Definition of visibility rules for volatile data.

5. The specification language should allow a sufficient selection of *user definition and management* features. These features are:

   (a) Specification of users, roles, organisational units and their properties such as authentication and authorisation.

   (b) Specification of relationships among and between users, roles and organisational units.

   (c) Rules of assignment of the activities to the responsible actors. These actors can be organisational units, roles or users.

6. The specification language should allow definitions of *compensating scenarios and activities* at each level of specification.

7. The specification language should allow definitions of alternative scenarios and activities, at each level of specification.

8. The scenario instances should be *addressable* at the specification language level, i.e. one should be able to refer to other scenario instances from an scenario instance. This is especially necessary when scenario instances start other scenario instances and the instances interact with each other.

9. The specification language should allow definitions of document archiving and version management.

10. The specification language should allow definitions of coordination of a group.

The above requirements will be explained in more detail in the following chapters.

5.4.2 PortNet Requirements for the Specification Language

5.4.2.1 The Role of the Graphical Description Technique Used in Analysis

A graphical description technique was developed and used during the PortNet analysis work. The technique was to capture the minimal set of constructs needed to describe the scenarios and activities in the PortNet application area. During the analysis, we found the technique well suited and expressive enough for its purpose. This purpose, however, is the analysis of a system, not its design. For design purposes, a more thorough methodology and a more detailed specification, will be needed.

Our technique itself is a graphical description language: its transition diagrams specify the control flow, i.e. ordering dependencies between activities and scenarios whereas its data flow diagrams specify the data dependencies between activities and scenarios. The same description technique has been used in all levels of scenario definition from the inter-organisational level ("Handle a Ship Visit") to the elementary local database transactions ("Updating Vessel Register"). Furthermore, the technique can as well be used in specifying ordering and data dependencies *between scenarios*.

The specification language should allow a similar *hierarchical definition of scenarios and activities* and at each level of specification, be able to express at least the same features as the graphical description technique: the ordering dependencies between activities and scenarios, conditions for execution of activities and scenarios, multiple entry and exit points of activities and scenarios parallelism of activities and scenarios, traditional transactional properties of activities and scenarios (the ACID properties), data dependencies between activities and scenarios and definitions of complex data.

5.4.2.2 Stability of Scenario Definitions

After having made the mistake of trying to capture the local PortNet scenarios at the inter-organisational level of specification, we came to the conclusion that at the inter-organisational level, the definition of scenarios should include only:

1. The message types exchanged,
2. The organisation types the messages are sent to and
3. The message exchange protocol, i.e. what message types are expected as a response to a certain message.

This way the actual harbour and the pilot station a message is sent to as well as the local processing details are defined at the time the scenario definition is instantiated. This is necessary because the actual data contents of a message (e.g. the destination harbour) define to which harbour and pilot station the message will be sent. Also, the local processing varies from harbour to harbour and pilot station to pilot station. Therefore, specifying these details at the inter-organisational level is not reasonable. With this approach, the scenarios at the inter-organisational level can and should be pre-defined and they do not change frequently. In the PortNet system, the inter-organisational scenarios can change when participating organisations together decide to develop the system further. This at its most frequent, may happen once a year.

At the local level, scenario definitions describe the actors and the data and ordering dependencies of the local activities. The local organisations may change these autonomously i.e. as often as they find it necessary. The PortNet interviewees did not indicate a great need to change the local scenario definitions, however this may change once the system gets in the production use or it may vary from organisation to organisation.

5.4.2.3 User Definition and Management

The specification language should allow a sufficient selection of user definition and management features. It was not reasonable to include those in the graphical description technique but, the need is clearly present in the PortNet system. These features are:

1. specification of users, roles, organisational units and their properties,
2. specification of relationships among and between users, roles and organisational units and
3. specification of rules of assignment of the activities and scenarios to the responsible actors.

The properties of users, roles and organisational units include e.g. *expertise, authentication and authorisation* aspects. There should also be a possibility for the scenario specifier to define whatever additional properties he finds necessary. For the ease of maintenance, some of these properties should be *inherited* e.g. from an organisational unit to all the users that belong to that organisational unit.

Relationships among and between users, roles and organisational units were typically of the type "belongs to" (between users and organisational units) or "is a" between users and their roles or different organisational units (hierarchical organisations). However, there is no need to pre-define a fixed set of allowed relationships at the language level but the scenario specifier should be able to define whatever additional relationships he finds necessary.

In rules of assignment of the activities to the responsible actors, the actors can be organisational units, roles or users. Typically in PortNet type inter-organisational workflows, at the

higher levels of specification, such as the inter-organisational level, the actors are defined as organisational units and later on in refinement, these definitions are mapped to individual user roles or users.

There should be a possibility to express the rules of assignment based on user-specific status information, e.g. the current workload of individual users (*work balancing*).

5.4.2.4 Alternative activities

The specification language should allow definitions of alternative activities (or scenarios), at each level of specification. In workflow environments, the alternative activities typically take place when a deadline fires or a failure happens. Other conditions for execution of alternative activities may be defined as well.

Examples of possible definitions of alternative activities are:

- "WHENEVER DISKERROR DO recovering_action()" and
- "WHENEVER TIMEOUT DO find_and_assign_a_new_actor()".

Alternative activities may be cascading, i.e. include other alternative activities.

5.4.2.5 Traditional transactional properties

The PortNet analysis showed, that the specification language should allow at least the traditional transactional properties to be specified for the scenarios and activities (to be automatically enforced by the system). These traditional ACID properties are *Atomicity*, *Isolation* and *Durability*. We will talk about *Consistency* later on (see Data Definition).

It seems to be an application-specific issue what subsets of ACID properties are required for individual activities and scenarios. In the PortNet case, the AID-properties were needed for some leaf-level activities, such as "Checking, approving and sending the notice" performed by the Agent. On the other hand, the Agents activity "Checking minimal data requirements" had no transactional properties and it could be interrupted at any time. At the upmost level, the AD-properties were required for the inter-organisational scenario, "Handle a ship visit". Because of the long duration of the upmost scenario and the need to use also partial results, the I-property was not valid at the upmost level.

The compiler or interpreter of the specification language should *guard the correctness of specification of the traditional transactional properties*. For example, the transactional properties specified for a child scenario or activity should not conflict with the properties of the father.

5.4.2.6 Extended Transactional Properties: the Dependencies

With the transitions and events of the graphical description technique, we were able to describe the rich set of *ordering dependencies* that was present in the PortNet case: Let $\mathcal{A}$ and $\mathcal{B}$ be activity definitions at any level i of the scenario specification. $\mathcal{A}$ and $\mathcal{B}$ can have the following ordering dependencies:

1. The execution of $\mathcal{B}$ will be started after the execution of $\mathcal{A}$ has been successfully finished, i.e. released:

   `Release` $\mathcal{A} \rightarrow$ `Begin` $\mathcal{B}$`.`

   This is the most common dependency in the PortNet system. (e.g., the PortNet activities "Preparing advance arrival notice" and "Processing advance arrival notice").

   Similarly, unsuccessfully finishing the execution of $\mathcal{A}$ can cause the execution of $\mathcal{B}$ to start:

   `Cancel` $\mathcal{A} \rightarrow$ `Begin` $\mathcal{B}$`.`

   An example of this dependency would be starting an alternative activity in case the original activity fails.

2. Releasing the activity $\mathcal{A}$ may have its implications in the success of execution of the activity $\mathcal{B}$:

   `Release` $\mathcal{A} \rightarrow$ `Release` $\mathcal{B}$ or,
   `Release` $\mathcal{A} \rightarrow$ `Cancel` $\mathcal{B}$`.`

   An example of the former are the PortNet activities "Preparing final arrival notice" and "Preparing final departure notice", i.e. in case the ship arrival has been successfully informed to all the participating organisations, we must make sure we do not leave the system in a state in which the ship seems to have stayed in the harbour. In other words, the departure of the ship has to be successfully informed to all the participating organisations as well.

   An example of the latter are the timetable data updates originated by the MKH local workers and the Agent. Within a scenario instance, any update released by a local worker cancels any update by the Agent.

3. Cancelling the activity $\mathcal{A}$ may as well have implications to the success of execution of the activity $\mathcal{B}$:

   `Cancel` $\mathcal{A} \rightarrow$ `Release` $\mathcal{B}$ or,
   `Cancel` $\mathcal{A} \rightarrow$ `Cancel` $\mathcal{B}$.

   An example of the former is requiring an alternative activity $\mathcal{B}$ always to finish successfully.

   An example of the latter are the PortNet activities "Preparing advance arrival notice" and "Preparing advance departure notice". In case the arrival is cancelled, the departure must be cancelled as well, in case some departure information has already been given.

Note that $\mathcal{A}$ and $\mathcal{B}$ are at the same level of scenario specification. This is because the PortNet analysis showed no need to specify any inter-level dependencies. Whether this is due to the hierarchical refinement method used or a true property of workflow systems, requires further study.

The semantics of `Cancel` $\mathcal{A}$ is discarding the work done in $\mathcal{A}$ so far and exiting the execution of activity $\mathcal{A}$. `Begin` $\mathcal{A}$ means starting the execution of activity $\mathcal{A}$. `Release` $\mathcal{A}$ means storing $\mathcal{A}$'s results in a stable storage and exiting the execution of activity $\mathcal{A}$. These semantics are the same at all levels of the specification. However, the implementation of `Cancel` $\mathcal{A}$ and `Release` $\mathcal{A}$ may be very different at different levels of the scenario specification. When activity $\mathcal{A}$ is an elementary database transaction executed on top of a relational database system, `Cancel` is implemented as an execution of the SQL command `ROLLBACK` and `Release` is implemented as an execution of the SQL command `COMMIT`. At higher levels of specification, the definition of activity $\mathcal{A}$ may consist of definitions of several elementary database transactions and depending on the transactional properties specified for $\mathcal{A}$, `Release` may imply committing them all at the same time or committing the last one (assuming that the others had been committed earlier). Similarly `Cancel` may imply simply not committing the pending elementary transactions or, compensating the already committed elementary transactions.

These dependencies may have *temporal aspects* as well, such as "activity $\mathcal{B}$ should begin 3 hours after activity $\mathcal{A}$ has finished successfully". However, the temporal condition can be expressed as a pre-condition for activity execution.

In the workflow systems, there might be also *soft* versions of the type 1 dependencies, i.e. even though there is a dependency `Release/Cancel` $\mathcal{A} \rightarrow$ `Begin` $\mathcal{B}$, the actor of $\mathcal{B}$ may start the execution of $\mathcal{B}$ before the execution of $\mathcal{A}$ finishes, with a warning sent to the actors of both $\mathcal{A}$ and $\mathcal{B}$ by the system.

The above ordering dependencies may exist *between scenario instances* as well. For example, cancelling a whole scenario instance typically implies executing a set of compensating activities but it may also require starting a new scenario instance. Further requirements can be made on the success of the execution of the new scenario instance (e.g. `Cancel Scenario1` $\rightarrow$ `Release Scenario2`).

The PortNet analysis also showed *data dependencies* between scenarios and activities. A data dependency between scenarios or activities $\mathcal{A}$ and $\mathcal{B}$ means that $\mathcal{B}$ uses data produced by $\mathcal{A}$.

The specification language should allow specifying the above dependencies for scenarios and activities. These dependencies may exist in all levels of the specification.

5.4.2.7 Conditions

There should be a possibility to define *pre-conditions* and *post-conditions* for activities and scenarios. These conditions are Boolean functions defined over detectable internal or external

events ("until x fires"), global data such as the time ("after 3 hours"), local data possibly queried from a local database ("Arrival.ETA = today") or data of the activity or scenario instance ("$x = y$"). The execution of activity $\mathcal{A}$ does not start before the pre-condition of $\mathcal{A}$ evaluates to true and the execution of $\mathcal{A}$ can not finish before the post-condition of $\mathcal{A}$ evaluates to true.

The specification language should allow definitions of condition evaluation policies as well. These policies can be e.g.:

1. The number of times the condition should be tried in case it always evaluates "false" and
2. The alternative activities to be performed if a condition evaluates false. One alternative activity can be terminating the activity in question or terminating the whole scenario instance.

5.4.2.8 Parallelism

There should be a way to specify parallel executions of activities and scenarios (forks of the control flow). These forks describe loops and recursion, too. Conditions may be attached to forks.

Similarly, there should be a way to specify merges of parallel executions. These can be specified as a pre-condition for execution of the activity (or scenario) in which the forks of control flow join. With pre-conditions we can specify both *conjunctive* (a transition is generated if and only if all merging transitions are activated) and *disjunctive* (a transition is generated whenever one of the source transitions is activated) merges of the graphical description technique, as well as *specifier-defined conditions* for merges (e.g. merge if 2 of the 3 transitions are activated).

5.4.2.9 Bundles

In addition to parallelism specified in the definition level, there has to be a possibility to define bundles, which are multiple instances of different activities (or scenarios) or the same activity (or scenario), to be started concurrently. The actual number of instances is solved at *run time*.

5.4.2.10 Data definition

The data exchanged and stored in inter-organisational workflow applications is typically *well-defined*, *structured*, *complex* and *large* in volume. The state of the art in designing these systems is defining the structure of the messages to be exchanged between organisations.

Therefore, the specification language should provide a comfortable way of defining structured, complex data both exchanged and in local storages. The data structures may also have relationships.

Furthermore, as seen in the PortNet Data Flow Diagrams, the consistency property, i.e. the *correctness* criteria for concurrent handling of the data may be different to the traditional database conflict serialisability. What that criteria may be requires further study.

However, the specification language should at least 1) allow the workflow specifier to choose from a set of different correctness criteria and thus specify scenario-specific or local restrictions for concurrent handling of the data (for example, "TIMETABLE data requires Local Conflict Serialisability"). There should be be both pre-defined, standard correctness criteria, such as conflict serialisability, and specifier-defined correctness criteria. Therefore, the specification language should also allow 2) definition of scenario-specific correctness criteria.

In addition to *persistent data* discussed above, workflow systems include *volatile data*. The volatile data can be private for an activity instance, shared with lower-level activity instances or shared with all activities at any level. The specification language should support 3) defining the *scope* of volatile data.

5.4.2.11 The need to compensate finished activities

In the PortNet case, there clearly is a need to undo the effects of already finished activities (and scenarios), i.e. to execute compensating activities. This is due to the requirement that the overall scenario, "Handle a ship visit" should be atomic.

Because in the PortNet case the results of finished activities may have been used by other activities before the compensation takes place, there might be cascading compensations as well. However, it is not the responsibility of the workflow specifier to define what cascading compensations are needed but it is up to the underlying system to do this, based on the data and ordering dependency specifications (provided by the scenario specifier), and the execution history of activities and scenarios (maintained by the system). For example, if there is a data dependency between activities $\mathcal{A}$ and $\mathcal{B}$, i.e. $\mathcal{B}$ uses data produced by $\mathcal{A}$ and the executions of both A and $\mathcal{B}$ have been successfully finished at the time a need to compensate $\mathcal{A}$ arises, $\mathcal{B}$ has to be compensated as well.

Note that compensating activities may cause *new scenario instances* to take place as well. In the PortNet case, if the Agent finds out that the destination harbour of a ship will change, after he already has sent out an Advance Arrival Notice about that visit, he must 1) re-submit the original notice with the status "Cancelled" and 2) send out another notice with the new destination harbour. The latter step initiates a new scenario instance.

If the Agent already has sent out an Advance Notice of Departure as well, the steps 1) and 2) have to be done for that notice, too. So, in the PortNet case undoing the whole scenario instance, would be "Compensate all the activities done so far". In each participating organisation, this would mean executing the compensating activities as defined in the scenario

specification, for all the finished activities, one by one in a reverse order to the original execution.

Another requirement the need to compensate implies is the definition of *multiple entry points* for activities and scenarios. In workflow environments, the need to compensate, i.e. cancel a scenario instance S_1, may arise in an organisation O_1 at any moment, possibly in the middle of the local processing of the scenario instance S_2 in organisation O_2. The compensating activities should typically *interrupt the normal processing* so that no work would be wasted. In the PortNet case, the cancellation of a ship visit sent to the Port and MKH by an Agent should besides undo also interrupt all the processing concerned with that particular visit. The entry points define the states of activities to which internal or external events lead. For compensation purposes, we may need several of those.

However, in inter-organisational communication the compensation can be less straightforward if the need to compensate is initiated in another organisation and the organisation that receives the compensation message decides to demonstrate its local autonomy. If, for example at MKH, the ship captain has already called in and confirmed the ship visit, any cancellation notices about that visit received from the Agent will be ignored because they are found less trustworthy. In this case, MKH can refuse to compensate whereas at the same time, the Port may decide to execute the compensating activities. This leads to *an inconsistent state of the scenario instance* which is perfectly acceptable in the PortNet case. But, if the captain of the ship has not confirmed the ship visit yet, MKH will take the Agents cancellation seriously and all data concerning that visit will be removed from the MKH local database. At the specification language level, this kind of situations can be handled by including conditions for execution in the compensating activities.

Therefore, the requirements for the specification language are:

1. The possibility to specify the compensating activities which may include existing activities (or scenarios) on different levels of the specification hierarchy,
2. The possibility to specify conditions for execution of compensating activities,
3. The possibility to specify multiple entry points for activities and
4. The possibility to specify data and ordering dependencies between activities.

5.4.3 Valmet Requirements for the Specification Language

The Valmet Problem Reporting system has a dual nature as software fault and change request reports cause workflow processing while the other report classes lead to a cooperative designing/authoring type of processing. This dual nature is reflected also in the requirements implied by the Valmet system. We have also included those requirements that become relevant while enhancing the current prototype system.

5.4.3.1 User definition and management

The specification language should allow specification of users, roles and organisational units, their properties, their relationships, and activities and scenarios assigned to them. The user definition should also cover authorisation and authentication related definitions.

5.4.3.2 Data definition

In the Valmet case, data definition means specifying the structure of a report document. The fields have types (numeric, alphanumeric, date, time, signature etc.) and access right constraints. Depending on the user rights, fields may be readable, writable or not visible.

With regard to concurrent access to the bulletin board documents in planning a new software version, there is a need to specify policies for concurrent access of data. In practise this may mean giving each data construct a concurrent access mode code. Such could be e.g. not shareable, shareable within members of the Valmet software coordinator committee etc. The access mode codes could also be generic and treated as test conditions in the basic data manipulation primitives discussed later in this document.

5.4.3.3 Execution control

Ordering, timing and data dependencies of the scenarios and activities must be expressible. One goal of the Valmet system is to evaluate the services provided by software suppliers. The specification language should provide also tools for defining performance monitoring. Primitives are needed for accessing processing statuses of scenario instances, as well. For instance, an application manager must be able to get information on a scenario instance that has been left hanging due to ignorance of the supplier. Specification of user authentication related processing as electronic signature verification must be supported.

5.4.3.4 Concurrency control in shared documents

Concurrency control in cooperative design/authoring related processing needs facilities for defining the basic data manipulation primitives and their interrelated dependencies (ordering, temporal and data). The idea is that the system designer is given facilities to define the actual primitives and to define execution conditions to their concurrent processing. The complexity of the concurrency control depends on the granularity in which cooperative access is supported. In a document, the access control may be provided on the level of whole documents, document fields or field contents. Even concurrent editing of the same edit field could be supported.

5.4.3.5 Document archiving and version management

In the Valmet system, each document is accessible through many indexes that are maintained with respect to document contents, identity codes, time stamps etc. Also the main actors of a scenario instance retrieve these documents from the document archive via these indexes. Unlike in many other workflow systems, here always the initiative of working with a scenario instance comes from the side of the user. The initiative does not come from the system which is merely a document repository. The language should provide ways to specify the search indexes.

Version management support is important. When specification documents of a new software version are designed as a cooperative process, there is a need for maintaining the history of each document. There is also need for supporting backtracking to earlier design decisions and starting new branches of thought. This may be needed for instance, when the integrity of the document is somehow lost or when the designer in charge wants it. There is a need for a referencing mechanism to the earlier versions and version management related access primitives.

5.4.4 IN requirements for the specification language

The 101-service of Telecom Finland induces the following requirements for the specification language:

1. The support for the verification of workflow scenarios:
   - The designer of a scenario should be able to specify constraints which should hold in every scenario instance. If a scenario instance does not obey a constraint then the used verification tool should be able to indicate the exact location of the violated constraint in the workflow scenario.
   - It must be possible to make a quantitative analysis of the workflow before it is deployed.
2. The description of steps executed by computers.
3. The description of steps executed by human beings.
4. The linkage of steps executed by human beings and computers.
5. Dependencies between steps. (The dependencies are similar to the PortNet case.)
6. The description of actors with multiple roles.
7. The description of heterogeneous data sources.

Chapter 6

Requirements for the TRANSCOOP Specification Language

In this chapter, we try to generalise the results that have been described in the previous three chapters. In section 6.1, we describe the common properties of the areas of Cooperative Document Authoring, Design for Manufacturing and Workflow. Also, we say something about the differences between the implications of the different application areas. In Section 6.2, we summarise the requirements that have been found while analysing the application areas.

6.1 Summary of Analysis

In chapter 2, we have identified the areas of interest for performing the application analysis. We distinguished the categories *activity*, *data* and *user*. Within those categories, we have indicated the kinds of questions that seemed to be relevant on beforehand. After having performed the application analysis, we have been able to answer most of these questions. In some applications, additional requirements have come up. We have classified them in one additional category *general requirements*.

The three application areas span a wide range of cooperative application scenarios. The activities within an organisation could be divided in those related to the work to be done (e.g. designing for CDA and DfM), the organisation of the work (such as planning) and management of the work. We could say that each of the above categories of activities are related to a different level of management, where work management is found on the highest level, and the work to be done at the lowest level. On these different levels of management, different forms of cooperation are found. The forms of cooperation are usually more formal on the more higher levels.

In the following we summarise for each application area, the levels of management it addresses and what is the main focus of the complexity.

- **Cooperative Document Authoring**
  This application aims at supporting the work process of a team of authors working together on writing a (hypermedia) document. CDA aims at supporting activities that are related to the lower levels of management. It mainly supports the design work to be done and some support for the organisation of the work.

  Because document authoring is a creative design process, it will involve quite dynamic forms of cooperation. The complexity of the problems found in CDA is related to supporting dynamic forms of cooperation.

- **Design for Manufacturing**
  Like CDA this application area is also concerned with a design process, but it differs in that DfM requires people from very different disciplines to work together and to communicate with each other. Also during the design process many documents are generated that describe a certain part of the design of the artifact. For these reasons DfM also has a strong focus on work organisation and can also (with large projects) involve quite a lot of work management.

  The complexity of the problems found in DfM is related to the management of complex data structures stored in inter-related documents.

- **Workflow**
  This application area, especially when we look at inter-organisational workflows, is concerned about work management and work planning. It is not concerned about the contents of the work to be done, but about how the different activities are organised. Aspects like authorisation and authentication play an important role.

  The complexity of the problems found in WF is related to reliability, (inter-organisational) consistency and responsibility.

Another remark has to be made about the graphical language that has been introduced in Section 5.3.1. This language illustrates which constructs are necessary to describe workflow scenarios. This language has been designed to present the analysis results in the workflow chapter. It is an *ad hoc* language, that represents in itself requirements for the TRANSCOOP language. It is not a competitor of the TRANSCOOP language, nor is it intended as such.

In the following sections we present a summary of the aspects that were found to be crucial for the application areas. The first section discusses some specification language related requirements. The other sections discuss aspects related to the *activity*, *data* and *user* aspects that the specification language should be able to express. In Section 6.2 a list of detailed requirements is given.

6.1.1 Specification language related requirements

The specification language should not bother scenario designers with all details of one concrete synchronisation mechanism. This can be achieved by deriving language constructs with clear semantics that are on a high level of abstraction and thus as independent as

possible from concrete synchronisation mechanisms. An advantage of such a language would be the opportunity to translate it to various transaction models providing a specific predefined functionality.

The specification language should be such that it can be used by those who want to specify the CSCW applications found in the application areas. This means that the specification language should use concepts that are based on the requirements that are stated in the following sections.[1] The following sections describe the areas that are of special interest for the specification language. Section 6.2 gives a more detailed list of requirements.

6.1.2 Activity aspects

The following important aspects related to activities were found in the application areas:

1. **Different levels of management**
   Different levels of management are most clearly found in the DfM application area, but also in the inter-organisational workflows.
2. **Hierarchical activities**
   In all application areas, decomposition of activities (on the different levels of management) is used to break-down complex activities in more elementary ones.
3. **Activities of uncertain duration**
   The duration of activities in design environments (like CDA and DfM) is usually not predetermined. This unpredictable duration of activities results from their interactive nature and the complexity of the problem to be solved.
4. **Flexible ordering of activities**
   Because of the creative nature of CDA and DfM applications it is important that the ordering of the activities is flexible within a scenario. Within WF flexibility is required to deal with exceptions that can occur at any time during the execution of a scenario.
5. **Transactional properties of activities**
   In all application areas there was found a need to be able to specify the transactional properties of the activities and the dependencies between the activities.
6. **Multi-user cooperation**
   Because of the size and complexity of artifacts and the erratic problem solving process, there is a strong need for dynamic multi-user cooperation in the CDA and DfM application scenarios.

6.1.3 Data aspects

Looking at the aspects related to data, the following were found to be important in the application areas:

[1] We are aware of the vagueness of this requirement. It is the responsibility of the specification language designers to keep this requirement in mind.

1. **Large data volume**
   In all application areas we have to deal with large data volumes. In workflow, large numbers of scenarios being executed independently are found. Design environments are usually characterised by providing joint working facilities for the production of large, shared artifacts.
2. **Complexity of the data**
   Especially in DfM and CDA we find complex data structures. The data is complex with respect to both its contents and its structuring. An example of complexity of the contents of the data is found in DfM, where complex 3D-structures are represented. An example of complexity of the structuring of the data is found in CDA, where complex graph-like structures are being used to represent a hypertext document.
3. **Constraints**
   In all application domains constraints are being used to specify the consistency rules that apply to the data. Especially in DfM, user-defined constraints are being used. In the DfM application area the checking of constraints can be a complex task.
4. **Versioning**
   In the DfM and CDA application areas, versioning of the data plays an important role. Both explicit and implicit versioning mechanisms are being used. Implicit (or hidden) versioning is a technique that can be used to increase data availability but requires (automatic) merging of parallel versions.
5. **Historical data**
   In all application areas there is a certain need to keep records of the activities that have been executed. Especially in DfM it is important to record design decisions, also to prevent designers from considering solutions that have already been evaluated in the past.
6. **Shared and private workspaces**
   In design environments the concept of shared and private workspaces seems to be useful to support design work.[2]

6.1.4 User aspects

As user aspects are mainly important for the levels of management related to work planning and work management, the aspects presented in this section are first of all important for workflow applications. They are less crucial in the other application area.

The following aspects were found:

1. **User definition**
   User definition includes the definition of users, user groups, roles, organisational units and their relationships. In workflow applications this also includes rules for assigning activities to users that will be responsible for their execution.
   Although user definition is important it is not possible to give a fixed set of rules about how it should be done. Thus, flexibility is required.

[2] We do not refer to the notion of physically shared and private workspaces. This notion is orthogonal to the conceptual notion of workspaces.

2. **User management**
   In workflow (and also to a lesser extent in DfM) applications, the assignment of responsible users to activities plays an important role.
3. **Authorisation**
   Especially in (inter-organisational) workflows, authorisation and authentication are important. Authorisation plays some role in all application areas to prevent the loss of data.
4. **Ad-hoc formed groups**
   In design applications there is a need for *ad-hoc* group formation, as in contrast with the above described aspects.

6.2 Specification Language Requirements

In this section, we summarise the requirements that have been found for the TRANSCOOP specification language. What we have been doing in this workpackage is in fact *requirements engineering*: by analysing some application areas, we have tried to come up with requirements for the TRANSCOOP specification language and transaction model. Like design as presented in the chapter about Design for Manufacturing, requirements engineering is a continuing process. Therefore, one cannot expect this document to contain a "design" of the specification language by giving a full list of detailed requirements. In fact, the requirements in this document are still rather abstract and general. During the process of language design, the requirements will also be refined and clarified. So, language design and making the requirements more concrete will go hand in hand. There will be a continuous interaction between the language design and the requirements.

In order to avoid too much duplication, we will shortly describe the most important requirements that have been found in the application analysis and that seem to be used in the application domains. For a more extensive description of each requirement and for requirements with less impact, we refer to the respective sections of the three chapters covering the three application areas.

The rest of this section consists of four subsections, for the activity, the data, the user and the general requirements. Each subsection contains requirements for the specification language, although we did not include again and again a sentence like "The specification language should be suited to express. . .".

6.2.1 Activity requirements

Before listing the requirements for the TRANSCOOP specification language that seem to be general for all studied application domains, we want to generalise a remark about the basic concepts for a specification language that satisfies the coordination needs of a cooperative environment. Such a specification language should support at least:

- the definition of activities and their externally visible execution states

- the definition of legal transitions between these states
- the definition of conditions that enable transitions between these states and their properties (rejectable, delayable, forcible)

With these remarks in mind, we are ready to give the list of general requirements. As mentioned before, this list is very concise. However, it refers to other sections of this deliverable in which more information can be found.

1. **Specification of the work process**
   The language should provide the possibility to specify the activities that occur during the work process. This includes basic activity definitions, the parameters passed to an activity, the constraints to be satisfied before and after the activity has been executed. Furthermore it covers the transactional guarantees that are required for the execution of a specific activity. Of course, the activities in a cooperative work process will often be performed in parallel. (cf. Sections 3.5.1 (item 1) and Section 5.4.2.5.)
2. **Different levels of management**
   The specification language should be able to deal with different levels of management as found in the DfM application domain and as within inter-organisational workflows. More concrete this means that it should be possible to write specifications that can be controlled from the outside (through 'gates' or 'ports') and that can contain black-box parts (which are specified on a lower level of management). (cf. Section 4.6.1.1).
3. **Hierarchical organisation of activities**
   In all application areas, it has been observed that the activities that are performed have a *hierarchical* nature. Performing complex activities is naturally done by splitting large tasks into smaller and more clearly arranged parts. The structure of the activities reflects the problem that is to be solved and the organisation of the problem solving process.
   In CDA and DfM, the activity structure is generally dynamic. Hence, new subactivities may be added at runtime as the need arises. This means that activity hierarchies should be generated dynamically according to the needs of the problem solving process. In workflow, the activities tend to be better structured, although a workflow activity hierarchy may be dynamic too, whenever certain activities are conditional. This also means, that at least workflow applications require a specification language in which things like alternatives and repetitions can be specified in a natural and elegant way.
   The specification language has to provide constructs for structuring the work process in a hierarchical way. In section 5.4.2.1, some examples of such constructs are given, like the ordering dependencies between activities and scenarios, conditions for execution of activities and scenarios, multiple entry and exit points of activities and scenarios parallelism of activities, transactional properties of activities and scenarios and data dependencies between activities and scenarios. (cf. Section 3.5.1 (item 2) and 4.6.1.3.)
4. **Application specific correctness criteria**
   Serialisability is not a suitable correctness criterion for cooperative environments. To support the cooperative working process, new correctness criteria depending on the semantics of the application are needed. The specification language has to be able to express this. This also means that concepts like compensating actions, restarts and histories can be specified by the specification language (Section 4.6.1.2, Section 5.4.2.11 and Section 3.5.1 (item 3)).
5. **Dependencies between activities**
   In all application areas, it is important to be able to describe dependencies between subactivities. This holds for the complete spectrum of application scenarios, from WF to CDA.

The ability to define a lot of different relationships between activities and subactivities, results in a very flexible framework.
Dependencies define the links between activities and thus form a data- and control-flow definition. Between activities there can be different dependencies expressed:

1. Value dependencies: the data parameters that are passed from one activity to another activity must be expressible.
2. Temporal dependencies: temporal dependencies are useful to trigger actions depending on some timing condition, e.g. when a deadline has occurred (cf. also Section 4.6.1.5). Both relative and absolute time temporal dependencies should be possible.
3. Commit/abort dependencies: conditions stating that certain activities may only start after some other activities have reached a certain state, etc. (cf. Section 3.5.1 (item 4) and Section 5.4.2.6.)
4. Conditional execution: it is useful to support alternatives and repetitions to provide a dynamic execution of activities (cf. also Section 5.4.2.4).

6. **Flexible ordering of activities**
It is a typical requirement that designers and actors in a workflow want to have the ability to go back to some earlier stage in the working process or to continue exactly where they were interrupted. It is very difficult to describe such an activity completely in advance, usually it is only possible to specify a range of actions that are allowed to happen. However, the specification language has to provide as much flexibility as possible in this respect.
7. **Interactivity of users**
The previous requirement stated that the activity structure is often dynamic in our application areas. A property that is closely related to this is the interactive nature of these application areas. Cooperative activities are often characterised by user tasks that cannot be exactly predefined. A user may start an activity, execute operations interactively and use a couple of days to perform the activity.
The specification language should be suitable to specify complex interaction sequences between users that can occur during cooperation.
8. **Transactional properties of activities**
Conventional transaction models work with application-independent correctness criteria, like serialisability. The strong correctness criterion of serialisability contradicts the very nature of cooperative applications, because it guarantees the isolation of transactions from one another. In cooperative scenarios, it makes sense to loosen the correctness criteria, e.g. by allowing actors to read uncommitted data. The nature of the desired correctness criteria will depend generally on the application domain at hand.
The implication for the specification language is that it should be able to specify application-dependent correctness criteria.
9. **Compensating actions**
Sometimes, it is necessary to undo a certain transaction by so-called compensating actions. Examples have been seen a.o. in the PortNet analysis, but actually this is relevant in all application scenarios. In fact, compensating actions are "plain" actions again, with some special relationships with other actions (viz. the actions that they should compensate). In this respect, this is a special case of the situation of activities that are dependent on one another, which was mentioned in above with item 5.

10. **Milestones and deadlines**
    In all application areas, milestones and deadlines play a (more or less) important role. Whenever people cooperate, there have to be agreements about which parts of the work will be performed by whom and when it will be ready. This is relevant for each cooperative application scenario, although it will be more important in situations where the activities are prescribed very detailed, instead of described loosely and generally.
    Something related to this is that timing aspects are important in these situations. In terms of the specification language, it will be important to be able to specify deadlines and time-dependent activities.
    More details can be found in Section 4.6.1.5 and Section 5.4.2.6.
11. **Multi-user cooperation**
    In CDA applications a highly dynamic form of cooperation is found where two or more authors synchronously interact with each other, sometimes accompanied with real-time audio and/or video connections. This is described in more detail in Section 3.4.4.

6.2.2 Data requirements

1. **Complex data structures and relationships**
   In all application scenarios, complex data (structures) have been found. The TRANSCOOP specification language should provide facilities to deal with this.
   The complexity of the data is usually reflected in the complexity of the operations needed to manipulate the data. When cooperation is taken in account, the complexity of the operations is likely to increase even more.
2. **Propagation of changes**
   When multiple users are working simultaneously on a common artifact they should be able to observe the changes made by others under predefined circumstances. The specification language should provide constructs to define notification events, i.e. events in which users inform each other about relevant aspects of the cooperation. Version management is an important issue here (cf. Section 3.5.2 (item 1) and Section 4.6.2.1).
3. **Synchronisation between different kinds of data**
   Data can occur in a persistent storage system, in the client workspaces and in user interfaces. The specification language has to take these different kinds of data into account (cf. also Section 3.5.2 (item 2) and Section 4.6.2.4).

6.2.3 User requirements

1. **Single user aspects**
   In a realistic cooperative situation, there will be a number of different people involved. These people will be organised in several project groups. Some of these groups may have an *ad hoc* nature, whereas others may be formally established ones. The best advise we can give to the language designers is to try to be as flexible as possible. Try to support as many kinds of user scenarios as feasible!
2. **User groups**
   There are many different kinds of groups in a cooperative scenario. Groups may be formal or informal, temporal or permanent, heterogeneous or homogeneous, etc. The

same remark that has been made above, also holds here: all aspects of user groups that have been mentioned in the taxonomy seem to have some relevance for the project. However, the investigated application areas are not specific enough to be able to make deterministic statements about these aspects. Within one particular organisation, some aspects will be very important, whereas in another organisation, the situation may be completely different. We will have to be as flexible as possible to this respect.
However, workflow applications require that at least the following can be done:

1. Specification of users, roles, organisational units and their properties such as authentication and authorisation.
2. Specification of relationships among and between users, roles and organisational units.
3. Rules of assignment of the activities to the responsible actors. These actors can be organisational units, roles or users.

3. **Authorisation**
In cooperative scenarios, the aspect of authorisation plays an important role. Control team members is of course important, but its main relevance is to prevent loss of information and to guide workflow in a cooperative scenario.

See for more information Section 3.5.2 and Section 5.4.2.3

6.2.4 General requirements

1. **Flexibility**
A system supporting cooperative design has to be flexible enough to support changing requirements during the project trajectory. This aspect is one of the most important characteristics of cooperative design. For such scenarios generally take a long period of time and involve a lot of people. It often occurs that changes have to be pursued during a project. This may vary from other people being involved in the same activities to a complete redefinition of the most important starting points in the design process.

Chapter 7

Conclusion

Bibliography

[And93] F.A. Andriesse. Topmanagement must break down walls to come to a better innovation process (topmanagement moet muren slechten om te komen tot een beter innovatieproces). *De Ingenieur*, 105, December 1993. in dutch.

[ASSR93] M. Attie, M. Singh, A. Sheth, and M. Rusinkiewicz. Specifying and enforcing intertask dependencies. In *The 19th International Conference on VLDB*, 1993.

[BDS+93] Y. Breitbart, A. Deacon, H.-J. Schek, A. Sheth, and G. Weikum. Merging application-centric and data-centric approaches to support transaction-oriented multi-system workflows. *ACM SIGMOD Record*, 22(3), September 1993.

[BEK93] O. Bukhres, A. Elmagarmid, and E. Kuhn. Implementation of the flex transaction model. *IEEE Data Engineering Bulletin*, 16(2), June 1993.

[Ble94] L.T.M. Blessing. *A Process Based Approach to Computer Supported Engineering Design*. PhD thesis, University of Twente, Enschede, The Netherlands, 1994.

[BR92] A.H. Bond and R.J. Ricci. Cooperation in aircraft design. *Research in Engineering Design*, 4:115–130, 1992.

[CC82] J.-M. Chang and S.-K. Chang. Database alerting technique for office activities management. *IEEE Transactions on Communications*, 30(1), January 1982.

[Cen91] Concurrent Engineering Research Center, editor. *Proceedings of CERC's second Workshop on Product Development Process Capture and Characterization*, 1991.

[Con87] J. Conklin. Hypertext: An introduction and survey. *Computer magazine*, pages 17–40, September 1987.

[CR90a] P. Chrysanthis and K Ramamritham. Acta: A framework for specifying and reasoning about transactions structure and behavior. In *ACM SIGMOD International Conference on Management of Data*, 1990.

[CR90b] P. Chrysanthis and K. Ramamrithan. A framework for specifying and reasoning about transaction structure and behavior. In *ACM SIGMOD International Conference on Management of Data*, 1990.

[CR92a] P. K. Chrysanthis and K. Ramamritham. Acta: The saga continues. In Elmagarmid [Elm92], chapter 10, pages 349–397.

[CR92b] K.J. Cleetus and R. Reddy. Concurrent engineering transactions. In *CE & CALS '92 Washington Conference & Exposition*, June 1992.

[CR93] P. Chrysanthis and K. Ramamrithan. Delegation in acta as a means to control sharing in extended transactions. *IEEE Data Engineering Bulletin*, 16(2), June 1993.

[CW90] S. Ceri and J. Widom. Deriving production rules for constraint maintenance. In *The 16th International Conference on VLDB*, 1990.

[CW92a] S. Ceri and J. Widom. Managing semantic heterogeneity with production rules and persistent queues. Technical Report 92-078, Politecnico di Milano, October 1992.

[CW92b] S. Ceri and J. Widom. Production rules in parallel and distributed database environments. In *The 18th International Conference on VLDB*, 1992.

[DAZ81] V. De Antonellis and B. Zonta. Modeling events in data base applications design. In *The International Conference on VLDB*, 1981.

[DeM79] Tom DeMarco. *Structured Analysis and Systems Specification*. Prentice-Hall, Englewood Cliffs, New Jersey, USA, 1979.

[DHL90] U. Dayl, M. Hsu, and R. Ladin. Organizing long-running activities with triggers and transactions. In *ACM SIGMOD International Conference on Management of Data*, 1990.

[DHL91] U. Dayal, M. Hsu, and R. Ladin. A transaction model for long-running activities. In *The 17th International Conference on VLDB*, 1991.

[DM88] C.L. McGowan D.A. Marca. *Structured Analysis and Design Technique*. McGraw-Hill Inc., United States of America, 1988.

[DO94] R. Davison and P. O'Brien. Service provisioning in a multi-provider environment. In H. Kugler, A. Mullery, and N. Niebert, editors, *Towards a Pan-European Telecommunication Service Infrastructure–IS&N 94*, volume 851 of *Lecture Notes in Computer Science*, pages 258–271, Aachen, Germany, September 1994. Springer-Verlag.

[EG89] C. A. Ellis and S. J. Gibbs. Concurrency control in groupware systems. In *Proc. of the ACM SIGMOD Conference on Management of Data*, pages 399–407. MCC, 1989. Portland, Oregon, May.

[EGR91] C. A. Ellis, S. J. Gibbs, and G. L. Rein. Groupware: Some issues and experiences. *Communications of the ACM*, 34(1):38–58, 1991.

[Ell92] C. Ellis. A model and algorithm for concurrent access within groupware. Technical Report CU-CS-616-92, University of Colorado at Boulder, October 1992.

[ELLR90] A. Elmagarmid, Y. Leu, W. Litwin, and M. Rusinkiewicz. A multibase transaction model for interbase. In *The 16th International Conference on VLDB*, 1990.

[Elm92] A. K. Elmagarmid, editor. *Database Transaction Models for Advanced Applications*. Morgan Kaufmann Publishers, 1992.

[GH] Dimitrios Georgakopoulos and Mark F. Hornick. A framework for enforceable specification of extended transaction models and transactional workflows. To appear in the *International Journal of Intelligent and Cooperative Information Systems*, September 1994.

[GHMS94] K. Gronbaek, J.A. Hem, O.L. Madson, and L Sloth. Cooperative hypermedia systems: A dexter based approach. *Communications of the ACM*, 37(2):65–74, February 1994.

[GMGK+91] H. Garcia-Molina, D. Gawlick, J. Klein, K. Kleissner, and K. Salem. Modeling long-running activities as nested sagas. *IEEE Data Engineering Bulletin*, 14(1), March 1991.

[GMS87] H. Garcia-Molina and K. Salem. Sagas. In *ACM SIGMOD International Conference on Management of Data*, 1987.

[GR93] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann Publishers, 1993.

[Gra81] J. Gray. The transaction concept: Virtues and limitations. In *Proc. of the seventh Int. Conference on Very Large Databases*, pages 144–154, 1981.

[GS87] I. Greif and S. Sarin. Data sharing in group work. *ACM Transactions on Office Information Systems*, 5(2):187–211, April 1987.

[Haa92a] A. Haake. CoVer: A Contextual Version Server for Hypertext Applications. Arbeitspapiere der GMD 665, Gesellschaft für Mathematik und Datenverarbeitung, July 1992.

[Haa92b] A. Haake. Take CoVer: Exploiting Version Support Support in Cooperative Systems. Arbeitspapiere der GMD 686, Gesellschaft für Mathematik und Datenverarbeitung, October 1992.

[Haa94a] A. Haake. Under CoVer: The Implementation of a Contextual Version Server for Hypertext Applications. In *Proceedings of ECHT '94*, September 1994.

[Haa94b] J. Haake. Autorensysteme für die kooperative Erstellung von Hypermedia Dokumenten. PHD Thesis, unpublished draft, 1994.

[Hal88] F.G. Halasz. Reflection on Notecards: Seven issues for the next generation of hypermedia systems. *Communications of the ACM*, 31(7):836–852, July 1988.

[Hal91] F.G. Halasz. "Seven Issues": Revisited. In *3rd ACM Conference on Hypertext, San Antonio, Texas*, December 1991. Final Keynote Talk.

[HF86] J.R. Hayes and L. Flowers. Writing research and the writer. *American Psychologist*, 41(10):1106–1113, 1986.

[HHMWM88] T. Härder, C. Hübel, K. Meyer-Wegener, and B. Mitschang. Processing and transaction concepts for cooperation of engineering workstations and a database server. *Data & Knowledge Engineering*, 3(2):87–107, September 1988. North Holland.

[HHZ+92] S. Heiler, S. Haradhvala, S. Zdonik, B. Blaustein, and A. Rosenthal. A flexible framework for transaction management in engineering environments. In Elmagarmid [Elm92], chapter 4, pages 87–121.

[HN93] J.M. Haake and C. Neuwirth. Collaborative Authoring of Hypermedia Documents. In *Proceedings of "Translating and the Computer, 15", London*, 1993.

[HOS90] W. Harrison, H. Ossher, and P. Sweeney. Coordinating concurrent development. In *Proceedings of the Conference on Computer-Supported Cooperative Work*, 1990.

[HT93] J. Hannemann and M. Thüring. Schreiben als Designproblem: Kognitive Grundlagen einer Hypertext Autorenumgebung. *Kognitionswissenschaft*, Dezember 1993. Springer Verlag.

[HW92a] J. M. Haake and B. Wilson. Supporting Collaborative Writing of Hyperdocuments in SEPIA. Arbeitspapiere der GMD 682, Gesellschaft für Mathematik und Datenverarbeitung, September 1992.

[HW92b] J.M. Haake and B. Wilson. Supporting Collaborative Writing of Hyperdocuments in SEPIA. In *Proc. of the ACM Conference on Computer-Supported Cooperative Work*, pages 138–146, Toronto, Canada, 1992.

[IBM94] IBM Corporation. *IBM FlowMark, Modeling Workflow, Release 1.1*, 2nd edition, September 1994.

[Ide93] A.W. Idema. Computer supported design for manufacturing: A constraint approach. Master's thesis, University of Twente, Enschede, The Netherlands, November 1993.

[Jen91] K. Jensen. *Colored Petri Nets*. Springer, 1991.

[Joh88] R. Johansen. *Groupware: Computer Support For Business Teams*. The Free Press, N.Y., 1988.

[Kai90] G. Kaiser. Flexible transaction model for software engineering. In *Proceedings of Sixth International Conference on Data Engineering*, 1990.

[KAN93] W. Klas, K. Aberer, and E. Neuhold. Object-Oriented Modelling for Hypermedia Systems using the VODAK Modelling Language (VML). In *Object-Oriented Database Management System*, NATO ASU Series. Springer Verlag, Heidelberg, August 1993.

[Kat90] R.H. Katz. Towards a Unified Framework for Version Modelling in Engineering Databases. *ACM Computing Surveys*, 22(4):375–408, 1990.

[KBGW91] W. Kim, N. Ballou, J. F. Garza, and D. Woelk. A distributed object-oriented database system supporting shared and private databases. *ACM Transactions on Information Systems*, 9(1):31–51, January 1991.

[Kle91] J. Klein. Advanced rule driven transaction management. In *Proc. of the 36th IEEE Computer Society International Conference*, pages 562–567, Spring 1991.

[KLMP94] W. Kim, R. Lorie, D. McNabb, and W. Plouffe. A Transaction Mechanism fo Engineering Design Databases. In U. Dayal, G. Schlageter, and L.H. Seng, editors, *Proceedings of VLDB*, pages 355–362, 1994.

[KP90] M.J. Knister and A. Prakask. DistEdit: A distributed toolkit for supporting multiple group editors. In *Proceedings of the 3rd Conference of Computer Supported Cooperative Work*, pages 343–355, 1990.

[KP92] G. Kaiser and C. Pu. Dynamic restructuring of transactions. In A.K. Elmagarmid, editor, *Database Transaction Models for Advanced Applications*, chapter 8. Morgan Kaufmann Publishers, 1992.

[KPE92] E. Kuhn, F. Puntigam, and A. Elmagarmid. Multidatabase transaction and query processing in logic. In A.K. Elmagarmid, editor, *Database Transaction Models for Advanced Applications*, chapter 9. Morgan Kaufmann Publishers, 1992.

[LA94] F. Leymann and W. Altenhuber. Managing business processes as an information resource. *IBM Systems Journal*, 33(2), 1994.

[Ley] F. Leymann. Supporting business transactions via partial recovery in workflow management systems. Copy obtained from author.

[LFK88] M.D.P. Leland, R.S. Fish, and R.E. Kraut. Collaborative Authoring of Hypermedia Documents. In *Proceedings of CSCW '88*, pages 206–215, 1988.

[Lom93] D.B. Lomet, editor. *Data Engineering Bulletin: Special Issue on Workflow and Extended Transaction Systems*, volume 16, June 1993.

[LS92] M. Leadbeater and D. Seeley. Supporting teamwork with asynchronous hypertext. In B.T. Thomas, editor, *Proceedings of HyperOz '92: Workshop on Hypertext Activities in Australia*, February 1992.

[LSW92] A. Lempke, N.A. Streitz, and B. Wilson. WSCRAWL: The Use of a Shared Workspace in a Desktop Conferencing System. Arbeitspapiere der GMD 672, Gesellschaft für Mathematik und Datenverarbeitung, August 1992.

[McC92] Scott McCready. There's more than one kind of work-flow software. *Computerworld*, 26, November 1992.

[MD89] D. McCarthy and U. Dayal. The architecture of an active data base management system. In *ACM SIGMOD International Conference on Management of Data*, 1989.

[Mie94] Derek Miers. *Process Product Watch*, volume Two. ENIX Limited, 1994.

[Mos85] J.E.B. Moss. *Nested Transactions: An Approach to Reliable Distributed Computing*. PhD thesis, Massachusetts Institute of Technology, 1985.

[MRKN92] P. Muth, T. C. Rakow, W. Klas, and E. J. Neuhold. A transaction model for an open publication environment. In Elmagarmid [Elm92], chapter 6, pages 159–218.

[MRW+92] P. Muth, T. C. Rakow, G. Weikum, P. Brössler, and C. Hasse. Semantic concurrency control in object–oriented database systems. Arbeitspapiere der GMD 645, Gesellschaft für Mathematik und Datenverarbeitung, April 1992.

[Nel81] T. Nelson. *Literary Machines*. Mindful Press, CA, 1981.

[NG92] K. Narayanaswamy and K. Goldman. LLazy consistency: A basis for cooperative software development. In *Proceedings of the Conference on Computer-Supported Cooperative Work*, 1992.

[NKCM90] C. M. Neuwirth, D. S. Kaufer, R. Chandhok, and J. H. Morris. Issues in the design of computer support for co-authoring and commenting. In *Proc. of the ACM Conference on Computer-Supported Cooperative Work*, pages 183–195, Los Angeles, CA, 1990.

[NRZ92] M. H. Nodine, S. Ramaswamy, and S. B. Zdonik. A cooperative transaction model for design databases. In Elmagarmid [Elm92], chapter 3, pages 53–85.

[PH88] C. Pu and N. Hutchinson. Split transactions for open ended activities. In *The 14th International Conference on VLDB*, 1988.

[RBKW91] F. Rabitti, E. Bertino, W. Kim, and D. Woelk. A model of authorization for next-generation database systems. *ACM Transactions on Database Systems*, 16(1):88–131, March 1991.

[RBP+91] James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, and William Lorensen. *Object-Oriented Modelling and Design*. Prentice-Hall, Englewood Cliffs, New Jersey, USA, 1991.

[RELL90] M. Rusinkiewicz, A. Elmagarmid, Y. Leu, and W. Litwin. Extending the transaction model to capture more meaning. *ACM SIGMOD Record*, 19(1), 1990.

[RS] M. Rusinkiewicz and A. Sheth. Specification and execution of transactional workflows. chapter 29, pages 592–620.

[RY85] P.J.W. ten Hagen R.W. Yeomans, A. Choudry. *Design rules for a CIM system*. Elsevier Science Publishers B.V., Amsterdam, 1985.

[SFB+87] M. Stefik, G. Foster, D. Bobrow, K.Kahn, and L. Suchman. Beyond the Chalkboard – Computer Support for Collaboration and Problem Solvin in Meetings. *Communications of the ACM*, 30(1):32–47, 1987.

[SH93] H. Schütt and J.M. Haake. Server support for cooperative hypermedia systems. In *Proc. of the Hypermedia Konferenz*, pages 45–56, Zürich, Switzerland, 1993.

[SHH+92] N. Streitz, J. Haake, J. Hannemann, A. Lemke, W. Schuler, H. Schütt, and M. Thüring. Sepia: A cooperative hypermedia authoring environment. In *Proc. of the fourth ACM Conference on Hypertext*, pages 11–22, 1992. Milano, Italy, Nov. 30 – Dec. 4.

[SHT89] N. A. Streitz, J. Hannemann, and M. Thüring. From ideas and arguments to hyperdocuments: Travalling through activity spaces. Arbeitspapiere der GMD 402, Gesellschaft für Mathematik und Datenverarbeitung, July 1989.

[Sim81] H. A. Simon. *The Sciences of the Artificial*. MIT Press, 1981.

[Sor87] P. Sorgaard. A cooperative work perspective on use and development of computer artifacts. In *Tenth Information System Research Seminar in Scandinavia (IRIS), Finland*, August 1987.

[SR93] A. Sheth and M. Rusinkiewicz. On transactional workflows. *IEEE Data Engineering Bulletin*, 16(2), June 1993.

[SRK92] A. Sheth, M. Rusinkiewicz, and G. Karabatis. Using polytransactions to manage interdependent data. In A.K. Elmagarmid, editor, *Database Transaction Models for Advanced Applications*, chapter 14. Morgan Kaufmann Publishers, 1992.

[Swe94] Keith D. Swenson. Interoperability through workflow management coalition standards. In *Workflow '94 Proceedings*, 1994.

[THH91] M. Thüring, J. M. Haake, and J. Hannemann. What's eliza doing in the chinese room? incoherent hyperdocuments and how to avoid them. Arbeitspapiere der GMD 533, Gesellschaft für Mathematik und Datenverarbeitung, May 1991.

[Thi94] H. Thimm. A Multimedia Enhanced CSCW Teleservice for Wide Area Cooperative Authoring of Multimedia Documents (Position Paper). In *ACM CSCW '94 Workshop on Distributed Systems, Multimedia And Infrastructure Support in CSCW*, October 1994.

[THS90] M. Thüring, J. Hannemann, and N. Streitz. Schreiben als Designproblem: Ein integrativer Ansatz. In D. Frey, editor, *Bericht über den 37. Kongress der Deutschen Gesellschaft für Psychologie, Göttingen*, 1990.

[Tim94] H. Timmerman. Idee project baseline 2. Report CC-0417, Fokker Aircraft B.V., May 1994. in dutch.

[UR93] A. Storr U. Rembold, B.O. Nnaij. *Computer Integrated Manufacturing and Engineering*. Addison Wesley Publishers Ltd., 1993.

[Ver93] P.A.C. Verkoulen. *Integrated Information Systems Design: An Approach based on Object-Oriented Concepts and Petri Nets*. PhD thesis, Eindhoven University of Technology, 1993.

[WA94] J. Wäsch and K. Aberer. A tailorable Hyperengine using an Object-Oriented Database Management System. Working Paper, 1994.

[Wäc91] W. Wächter. Contracts: A means for improving reliability in distributed computing. In *IEEE COMPCON*, 1991.

[Wal92] J. Walder. *CIM principles of Computer Integrated Manufacturing*. Wiley and Sons, Ltd, 1992.

[WDSS93] G. Weikum, A. Deacon, W. Schaad, and H. Schek. Open nested transactions in federated database systems. *IEEE Data Engineering Bulletin*, 16(2), June 1993.

[Wei91] G. Weikum. Principles and realization strategies of multilevel transaction management. *ACM Transactions on Database Systems*, 16(1):132–180, 1991.

[WL86] C.C. Woo and F.H. Lochovsky. Supporting distributed office problem solving in organizations. *ACM Transactions on Office Information Systems*, 4(3), July 1986.

[WL93] U. K. Wiil and J. J. Leggett. Concurrency control in collaborative hypertext systems. In *Proc. of the fifth ACM Conference on Hypertext*, pages 14–18, 1993. Seattle, Washington, Nov 14–18.

[WR92] H. Wächter and A. Reuter. The contract model. In A.K. Elmagarmid, editor, *Database Transaction Models for Advanced Applications*, chapter 7. Morgan Kaufmann Publishers, 1992.

[WS91] G. Weikum and H. Schek. Multi-level transactions and open nested transactions. *IEEE Data Engineering Bulletin*, March 1991.

[WS92a] G. Weikum and H. Schek. Concepts and applications of multilevel transactions and open nested transactions. In A.K. Elmagarmid, editor, *Database Transaction Models for Advanced Applications*, chapter 13. Morgan Kaufmann Publishers, 1992.

[WS92b] G. Weikum and H.-J. Schek. Concepts and applications of multilevel transactions and open nested transactions. In Elmagarmid [Elm92], chapter 13, pages 515–554.

