A conceptual model of a business transaction management system

A conceptual model of a Business Transaction Management System

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Technische Universiteit Eindhoven, op gezag van de Rector Magnificus, prof.dr. J. H. van Lint, voor een commissie aangewezen door het College van Dekanen in het openbaar te verdedigen op dinsdag 13 september 1994 om 14.00 uur

door

WATTE JELLE HOFMAN

Geboren te Bozum

Dit proefschrift is goedgekeurd door de promotoren

prof.dr. K.M. van Hee en
prof.dr.ir. J.A.E.E. van Nunen

CIP-DATA KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Hofman, W.J.

A conceptual model of a business transaction management system / W.J. Hofman. – Den Bosch : Tutein Nolthenius. – Ill.
Thesis Eindhoven. - With index, ref.
ISBN 90-72194-39-X
NUGI 855
Subject headings: EDI / workflow management.

UTN Publishers
Willem van Oranjelaan 5
5211 CN 's-Hertogenbosch
The Netherlands

ISBN 90-72194-39-X

However, as the fifth of December is a well known day in the Netherlands for giving and receiving presents, my wife Désirée and I were also expecting a present. We received it in the early morning of December sixth and called it Pieter Jelle. Now this monograph has come to an end, I can hopefully spend more time with both Jelle and Désirée.

Graft, 1994

Preface

Concipiating a monograph is a hard job. Concipiating it as a leisure interest is even harder. Undoubtfully like others I have experienced that writing a monograph requires a discipline in thinking and working. Using formal techniques like timed, coloured, hierarchical Petri nets is certainly a help.

I would like to thank first of all Bakkenist Management Consultants in giving me the opportunity and the time for concipiating this monograph. Without the time it would certainly have taken another year to complete the monograph. I would also like to thank all my colleagues at Bakkenist Management Consultants for their comments to the work. Especially, I would like to thank Andries van Dijk for realizing some of the ideas in the software product EDIT. The realization of those ideas has given me the knowledge either to adjust them or to come with new ideas. In building software, shifts of ideas can lead to frustration with the builders. Fortunately for me, Andries had the patience to work with me. Furthermore, I would like to thank Frits Cramer, who in the end found the spare time to correct my use of the English language.

Secondly, I would like to thank my customers who gave me the opportunity to apply the concepts in practice. Royal Nedlloyd bv gave me the opportunity to specify a first prototype of the software. It was called the Chain Module and has been tested successfully in practice. Stichting Uniform Transport Code which currently applies the result of the ideas with success in external logistics. I would like to thank the Board of Stichting Uniform Transport Code and its secretary that gave me the confidence to work with them. Furthermore, Assurantie Data Network bv. has taken over the concepts and applies them with good results in the insurance industry. Also, a number of customers like Odette Europe for supply in automotive and the Agricultural Telematics Centre in the Netherlands, have taken over some of the concepts and are using the software product EDIT.

Thirdly, a number of students of the Technical University of Eindhoven has been of great help. I would like to thank Bart Kersten, Maarten Elshout, Erik Suijs, Carlo Koop, and Pieter Langereis for their contribution.

The Edispuut has been a forum to reflect my ideas. It has given me the opportunity to see the strong and the weak points of the concepts. Also, Martin has done a hard job in correcting my spelling and wording. I had to trouble him twice with this job.

I will still remember the day when I first started to write this monograph. It was on December fifth 1990, when I was expected to give a presentation for Edispuut.

Contents

1 Introduction

1.1 Background

In their day-to-day operations, commercial companies as well as non-profit organizations provide their products to their clients. These products are goods, services, money, or information. The clients are other companies and organizations, departments of organizations, or private persons. To initiate and to control the product provision process, information is exchanged. The information can be exchanged e.g. by paper documents, telefax, telephone, and electronic messages.
The use of electronic messages, or EDI: Electronic Data Interchange (Hofman, 1989), can offer several improvements to the business processes of organizations. Sokol (1989), for instance, argues that the opportunities of EDI can be found in the improvement of trading relations and the elimination of key-entry errors. According to Sokol, this leads to more accurate and timely shipments from suppliers to their customers. Sadhwani (1987) gives examples of a cost reduction from $50 to $14 for handling an order by using EDI. In the Netherlands, the advantages of the use of EDI are discussed in a series of books (Hofman (1989), Van der Vlist (1992)). Several aspects of EDI have been subject to research. Streng (1993) has tried to develop a tool to support the assessment of the value of the introduction of EDI for decision-makers. Schultz (1994) investigated the social aspects of EDI projects and van Heck (1993) the design management of such projects.
Similar advantages as those mentioned for EDI are often mentioned for the application of so-called Workflow Management Systems. These systems are to control the document flow in organizations by formalizing work procedures (Ellis and Nutt, 1992).
Information is also regarded as a factor that initiates changes in business processes. Hammer (1993), for instance, mentions the costs savings by re-engineering the business processes by making invoices redundant. The importance of information to business processes is also stressed by Creemers (1993) and Davenport (1993). New concepts are introduced for external integration of logistics, based on electronic information exchange (Kreuwels, 1994). Others, like Van der Vlist (1987), stress the relation between organizational and technical networks for the development and implementation of EDI. This combination of networks, also known as interorganizational systems (IOS), is also defined by Barret et al. (1982), Suomi (1989), and Wierda (1991). Both Wierda (1991) and King et al. (1989) emphasize the lack of an accepted theoretical framework for applying the concepts of interorganizational systems. In the area of EDI standardization, several initiatives have been taken to develop the technical aspects of such a framework (UN/ECE WP.4 GE.1, 1992, UN/EDIFACT, 1993, and ISO/IEC JTC 1/WG 3 N255, 1993).

The conclusion is that the electronically exchanged information and it's processing is of major importance to organizations. The majority of the current information systems performs an administrative task in merely *recording* activities carried out by organizations. Data related to business activities can be stored and retrieved in information systems. The functionality of these systems supports a limited number of procedures that have to manage a number of well-known business activities. However, there is a growing demand in offering a flexible response to the changing requirements of the clients. Therefore, an increasing number of automated information systems of different organizations is being interconnected today (Ediforum, 1992). We have entered the paradigm of *flexible communicating information systems* to manage changing business activities.

1.2 Problem definition

To be able to define the problem of this monograph, we introduce the following concepts (figure 1.1):

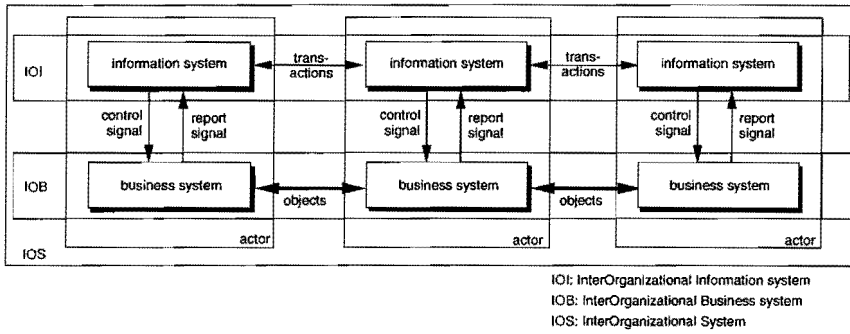


Figure 1.1: Business and information systems

- *actors* are commercial companies, non-profit organizations, private persons, or even information systems;
- an *interorganizational information system* (IOI) is the union of two or more flexible communicating information systems;
- an *interorganizational business system* (IOB) is the system that is to be controlled by the IOI;
- an IOS is either the union of communicating actors, or the union of interorganizational information systems and interorganizational business systems;
- *external communication* is the communication between information systems of two actors;
- *internal communication* is the communication between the information system and the business system of one actor.

Using these concepts, the problem definition of this monograph is the following:
- Is it possible to develop a conceptual model for interorganizational business systems?, and,
- Is it possible to develop a conceptual model for interorganizational information systems that control the execution of interorganizational business systems?

We will demonstrate that a part of an interorganizational information system can be made *generic*. That part can be applied to different situations, e.g. industry, transport, and health care. It also supports different procedures controlling a business process in different situations, e.g. an assemble-to-order production of personal computers and a transport service between Europe and the United States of America. This generic part is a *Business Transaction Management System* (BTMS). In this monograph, we look only at the construction of a conceptual model of a BTMS. Generic software is *parameterized* software. An *instance* of generic software is obtained by substitution of all parameters by possible values. One of the parameters for a BTMS is a procedure.
A BTMS can be compared with generic software components like a DBMS (Database Management System) or a UIMS (User Interface Management System). The use of such generic components will lead to economic improvements, e.g. more rapid application development and lower software cost. As a consequence, even small organizations will be able to automate the management of their business activities.
The process of defining the structure of a specific business system is *business engineering*. Business engineering results in the specification of procedures that are the parameters to the BTMS. Therefore, applying the concepts and the modelling of the concepts as described in this monograph will lead to the reduction of costs in the adaptation of the information system to support a new definition of the business process. Business engineering itself is outside the scope of this monograph, although the introduction of the BTMS might introduce redesign of business processes.

1.3 Research approach

In this monograph, we make a distinction between the reality, the concepts, and the modelling of the concepts. The concepts define the reality in words. Concepts are either domain independent (e.g. messages and actors) or domain dependent (e.g. vessels or patients). Modelling is a mathematical representation of the concepts by assigning a modelling component to a concept. We use timed, coloured, hierarchical Petri nets (Van Hee, 1994) to develop a theoretical basis for business processes. Using timed, coloured, hierarchical Petri nets, we also make a conceptual model of a flexible communicating information system. The parameters of a BTMS that are tokens in a Petri net, are specified by a data model. Since a procedure can also be modelled by a Petri net, this implies that one of the tokens of a Petri net can be a Petri net itself.

We will start by defining our concepts and model those concepts. Based on these concepts we specify the interorganizational information system. We will apply the concepts to external logistics that is an instance of an interorganizational business system. Finally, we will compare our concepts with other approaches and their usefulness in practice. The concepts of interorganizational systems, Workflow Management Systems, business process (re-) engineering, and the concepts developed in EDI standardization are compared with our domain independent concepts. Furthermore, we will assess the value of the domain independent concepts by comparing them with developments in EDI standardization. The value of the concepts is already proven in practice, since we have developed a software product to support message modelling (EDIT, 1993). This product has been used to model messages in for instance external logistics, agriculture, insurance, and supply in automotive.

1.4 Structure of the monograph

The structure of this monograph is as follows:

- chapter 2 presents the conceptual modelling of interorganizational business systems and interorganizational information systems;
- chapter 3 and 4 specify the data and the process structure respectively of the BTMS that is a component of the information system of one actor. The data structures that are discussed in chapter 3, specify the structure of the tokens of the Petri net of the process structure of the BTMS;
- in chapter 5 the application of the conceptual model is given for external logistics;
- in chapter 6 the relation is discussed between the concepts that are presented in this monograph and the concepts that can be found in literature;
- in chapter 7 some conclusions and recommendations are given with respect to the usefulness of the BTMS and the possibility to realize such a system. Areas for further research are indicated.

The structure of this monograph is shown in figure 1.2.
Annex 1 gives an introduction to functional data modelling and timed, coloured, hierarchical Petri nets. These two modelling techniques are used to model the data structures of the tokens of the BTMS and the process structure of the BTMS respectively. As figure 1.2 shows, timed, coloured, hierarchical Petri nets are also applied to model the concepts.

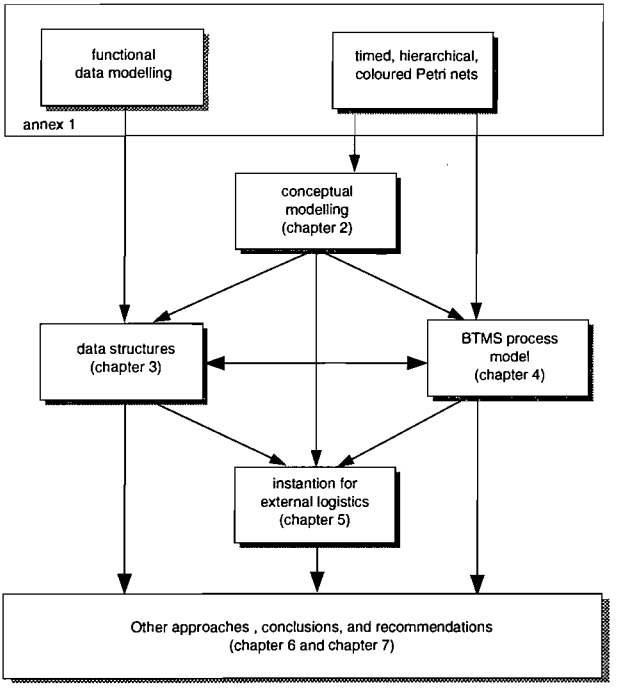


Figure 1.2: Structure of this monograph

2 Conceptual modelling

2.1 Introduction

We have given an informal notion of an interorganizational system (IOS) in the first part of chapter 1. An IOS can be decomposed and specified using the frameworks introduced in 1.2.1 and 1.2.2. There are two ways in which we can consider an IOS. Both approaches can be modelled using timed, coloured, hierarchical Petri nets.
In the first approach, an IOS is a composition of an IOI and an IOB that communicate (figure 2.1). An IOI is a composition of *information systems* (is) that communicate. An IOB can be decomposed in *business systems (bs)* that exchange objects.

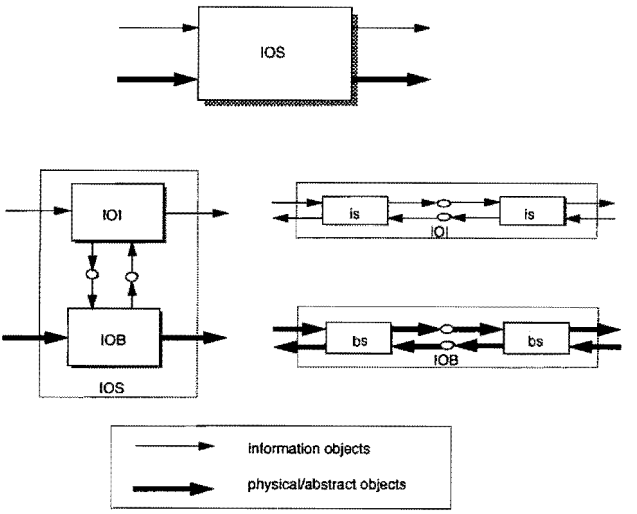


Figure 2.1: Decomposition of an IOS in an IOI and an IOB

In the second approach, an IOS is a composition of two or more actors that exchange physical objects and information objects. An actor is an organization, an organizational unit, or a person operating an information system or a business system, an information system on its own, or an information system that is operating a business system by exchanging signals with that business system.

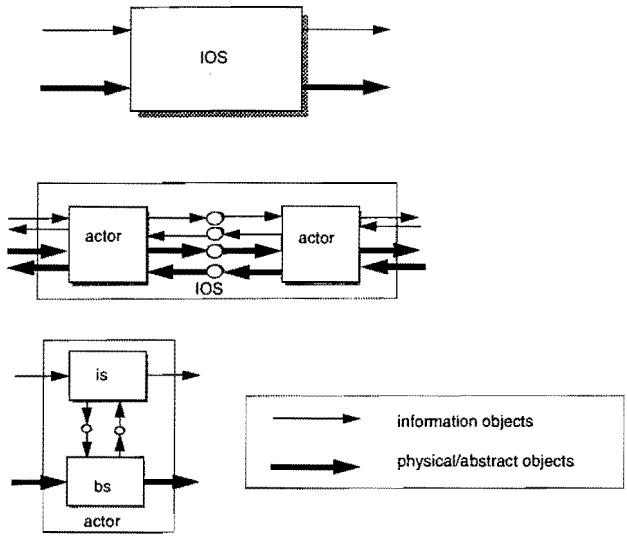


Figure 2.2: An IOS decomposed in actors

In general, actors try to minimize the uncertainty in their behaviour by making agreements with other actors. These agreements relate to the *services* of an actor. A service can be agreed at different co-ordination levels. *Co-ordination levels* specify requirements on the communication between the information systems of actors.
We make a distinction between the *structure* and the *behaviour* of interorganizational systems. The structure of a system is the static part of that system and the behaviour the dynamic part.

If appropriate, each section of this chapter starts with a description of the reality that is to be conceptualized, followed by the concepts (i.e. the concepts and terms that are used to describe interorganizational systems at an abstract level, independent of some specific application domain) and the modelling of the concepts. We use timed, coloured, hierarchical Petri nets to model the concepts (modelling is representing a concept as a model component).
In this chapter, we will present the concepts and the modelling of those concepts for the first approach of an IOS. These concepts also describe the second approach of an IOS. We will start by giving the concepts of business systems (section 2.2). The concepts that describe flexible communicating information systems (section 2.4) depend on the concepts that describe business systems and co-ordination levels of actors (section 2.3). The detailed specification of the information system is given in the chapters 3 and 4.

2.2 Business systems

2.2.1 Examples of business systems

As an example, we consider a business system for the transport of products packaged in containers, boxes, or pallets from a shipper in Europe to their final destination in the United States and Asia. The containers, the boxes, and the pallets are the objects to be transported. The shipper offers packages for transport and determines the location at which they are available for transport, the time at which they will be available, the location to which they have to be transported, and the time at which they are required to be available at their final destination. For example, the shipper offers his products for transport from three production units in Europe. These production units are located in the Netherlands, Germany, and Belgium.

For instance, we envisage a transport service operated by a forwarder on behalf of this particular shipper. The forwarder can arrange the transport for all types of packages offered by the shipper for transport. The transport service consists of two stages: the first stage is the transport by road to a port, and the second stage is the transport between two ports. The transport from the port of discharge to the place of delivery in either the United States or in Asia is arranged by another forwarder. Additionally, the forwarder is requested to arrange the transport from the port of New Orleans in the United States to a destination in the state Louisiana. The ports in Europe in this example could be Antwerp and Rotterdam. Packages from the Netherlands, Germany, and Belgium can be transported to the United States and Asia via both ports. The packages that are going to be transported via Rotterdam have to be stuffed in containers by a container stuffing centre. Those containers that are transported to the United States are transported back to Rotterdam (either full or empty). Containers that are transported to Asia are re-used in Asia.

Figure 2.3 shows the business system of the shipper without using formal techniques. In figure 2.3 the stuffing centre is part of the port of Rotterdam.

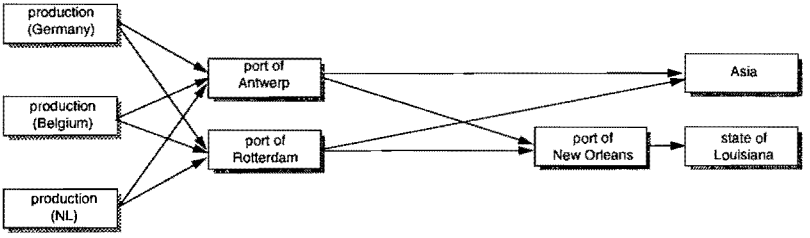


Figure 2.3: Business system of the shipper

Sea transport is arranged by an agent of the carrier (a liner-agent). A liner-agent can arrange the loading of the packages onto and the discharging of the packages from the vessel. The actual loading and the actual discharging is executed by a stevedore. Depending on, amongst others, the sailing schedule of vessels, the forwarder selects a liner-agent.

Similar examples can be given for other business systems. The raising of cows from birth to death can also be considered as an interorganizational business system, where the cows are the objects that are exchanged between the business system of a breeding farm and the business system of a slaughterhouse. Another example of an interorganizational business system is the handling of documents by the tax office. Documents are passed from one desk to another. The desks can be viewed as (internal) business systems, whereas the documents are the objects that are exchanged. The last example we consider is the handling of traffic fines. The police issues fines to persons (e.g. for driving too fast). The fines are passed to, for instance, a central office that controls the payment of the fines. However, if the payment is not received in time, the fine is passed to a legal authority. The actors involved (the police, the central office, and the legal authority) each perform specific activities in the business system for handling fines. The traffic fines are the objects that are passed between the business systems of these actors.

2.2.2 Concepts of business systems

In some business systems, physical objects like containers are exchanged; in other business systems, information that is present on documents, or the documents themselves are exchanged. Furthermore, one business system can be used for many objects, e.g. several containers of different types can be transported from a specific place to another. Moreover, an actor can outsource part of its business system to other actors.

As indicated in the introduction of this chapter, we make a distinction between the structure of a business system and the behaviour of that system. The structure of a business system is specified by the concepts business process, task, service, object type, resource type, and their relations. The behaviour of a business system is specified by the concepts of activity, action, object, and resource. We will define these concepts in this section.

Definition *task, service, business process*
A *task* is an elementary unit of work that is capable of consuming clearly defined input objects and producing clearly defined output objects on the basis of a control signal, possibly using resources and producing a report signal. A unit of work that is elementary cannot be decomposed any further. A *service* is a specific ordering of tasks that has a beginning and an end. A *business process* is the set of services that a business system can provide. A business process can have many services.

An example of a task is the lifting of a container into a vessel by a crane. The definition of a task implies that the control signals that can be consumed by a business process have to be distributed amongst the tasks that can be performed. The report signals of a business process can be produced by one or more tasks.

Input and output objects can be physical objects (e.g. a container) or information objects (e.g. the information that is present on a document or in a control signal). In general, a business process or a task is capable of transforming input objects into output objects. The objects that can be produced by a business process or a task may differ from the objects that can be consumed by the business process. For example, production of car doors is the transformation of steel plates and other material. Special business processes or tasks are:

- those that do not consume input objects (*generation*);
- those that do not produce output objects (*consumption*);
- those that are only able to move object types from one location to another without modifying them (*movement*);
- those that produce two or more output objects on the basis of one input object (*divergent production*);
- those that produce one output object on the basis of two or more input objects (*convergent production*).

A business process can be of physical nature (e.g. the transport of containers from one location to another) or of abstract nature (e.g. the transfer of money from one bank account number to another).

The business process of an actor is, in principle, specified for specific input and output object types and is independent of the input or output objects, although the behaviour may depend on the objects. There are two ways in which an actor can execute his business process on behalf of other actors:

- its business process is a set of services;
- the services that an actor offers depend on the characteristics of the input objects and the required output objects, e.g. products that are engineered to order (Bertrand, 1990).

In both cases, time planning has to be performed. We base the specification of the BTMS on services that are specified by an actor. Therefore, the business processes that we consider in this monograph can also be viewed as the union of all services. If the business process of an actor is of an abstract nature, the services are only specified in the information system of that actor. A service of an actor can have an identification in the information system of that actor. Besides our restriction to services, we do not allow loops of the tasks in a service.

Definition *action, activity*
An *action* is the execution of a task. An *activity* is the execution of a service with uniquely identified input and output objects.

Definition *superior, subordinate*
An actor is called a *superior* of another actor if the first actor can outsource the execution of a task to that second actor. The second actor is called the *subordinate* with respect to the first actor.

Obviously, a subordinate can act as a superior for a third actor if he out sources (part of) the execution of the service that he is responsible for on behalf of the first actor, to that third actor. So, the execution of a task of a superior actor can be the execution of a service of a subordinate actor.

According to our definitions, an activity is an ordered set of actions. The same business process can be executed one or more times, depending on the control signals and the objects that are present. It is not necessary that all tasks of a business process are executed when an activity takes place. An activity is only performed if it can consume a sufficient number of objects and sufficient resources are available. An activity may last some time, e.g. the transport of containers and the handling of documents takes a certain amount of time. That time is called the *duration* of that activity. The duration is, in principle, the same for all activities that refer to the same business process. However, the duration of one activity of a business process has a distribution with a minimum and a maximum. We distinguish between the *expected duration*, the *minimum duration*, and the *maximum duration* of a service and, therefore, of a task. While executing a service, the starting and the completion time as required by a superior are known. To be able to compute the required start and completion time of the execution of each task of a service, the start offset of a task must be known with respect to the expected duration of a service:

- $rs_{\text{action}} = rs_{\text{activity}} + so_{\text{task}}$
- $rc_{\text{action}} = rs_{\text{action}} + ed_{\text{task}}$

where *rs* is the required start, *so* is the start offset, *ed* is the expected duration, and *rc* is the required completion. This mechanism is commonly known as forward control (Bertrand, 1990). It can be applied to, for instance, divergent and movement business processes. A backward control mechanism must be applied to, for instance, convergent business processes. Such a control mechanism requires a delay with respect to the expected completion time of a service. We will call this delay the end-offset (*eo*). The backward control mechanism is as follows:

- $re_{\text{action}} = re_{\text{activity}} - eo_{\text{task}}$
- $rs_{\text{action}} = re_{\text{action}} - ed_{\text{task}}$

Figure 2.4 shows a forward and a backward control mechanism.

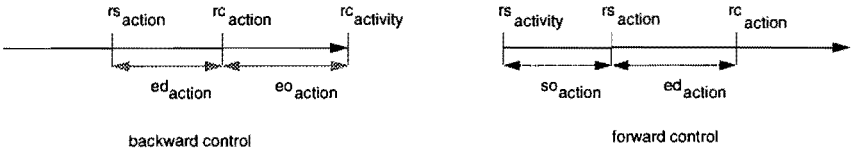


Figure 2.4: A forward and a backward control mechanism

In case of a forward control mechanism, the required completion time of an action has a variation; in case of a backward control mechanism, the required starting time has a variation. We assume that in practice this variation is the maximum duration

minus the minimum duration at the most. If otherwise, the action is to be handled as an exception.
If the required starting and completion times of the activity and the start and the end-offset of a task are known, either a backward or a forward control concept can be applied (we will apply a forward control concept in chapter 4). If neither the required starting nor completion time of the activity are known, three solutions are possible:

- it is not possible to compute the required starting or completion times of an action;
- the required starting time of an activity is assumed to be the time of reception of the information regarding the input objects;
- actors have contractual agreements (section 2.3).

Definition *object, object type, resource, bound resource*
An *object* is a physical thing (e.g. a container), an abstract concept (e.g. a job), or a piece of information (e.g. a message). An *object type* is the set of objects with similar features. Objects can be either input to an action or activity or output of an action or activity. A *resource* is an object that is used to facilitate an activity or an action. A *bound resource* is a resource that is reserved by the information system of an actor for an activity or an action.

Activities and actions consume and produce objects. A specific object that is either produced or consumed by an activity or action is of a certain type, e.g. the twenty-feet container with identification SEAU 1234567 is of the type twenty-feet containers. Another example of an object and its type is: a fine dated July 6th 1993 for passing the speed limit with 20 kilometres per hour, which is of the type we may call speeding traffic fines.
We make a distinction between an input and an output object. An *input object* is an object of a specific type that is input to a business process or a task. An *output object* is an object of a specific type that is output of a business process or a task. For example, certain parts are, at a specific time, input objects to a production action, whereas the products of that production action are the output objects. An activity or an action may consume zero, one, or more input objects at the same time and produce output objects after its duration. Output objects of an activity can be input objects to another activity. More specifically, a container can be an output object of a transport activity and an input object of a transshipment activity.
Resource types are for instance trucks, containers, machines, and persons. Resources can have the following roles:

- if a resource is used by a task it can be used directly by the same or another task after it has become available. For instance, a machine can be used for the production of a specific article and is available to produce other articles after that specific one has been produced;
- if a resource is used by a task, it cannot be used directly by that same task after it has become available (e.g. a milk bottle and a container). These resources can

be used up by one task (filling them with milk or stuffing goods in a container) and can be available after another task has been completed (after consumption of the milk or after taking the goods out of the container);
- a resource can only be consumed and not produced again, e.g. oil burnt by an engine.

The use of resources is limited by parameters such as availability and volume (e.g. a limited number of containers can be loaded on one vessel at the same time).
If resources are reserved for an activity or an action by the information system of an actor, we will call them bound resources. The binding of resources is a planning function of the information process of an actor. It will be discussed in chapter 4.

Definition *control signal, report signal*
A *control signal* is an information object that initiates an activity or an action. It contains information on the input objects, the output objects, and the resources that are to be consumed or produced by an activity or an action. A *report signal* is an information object that represents the result of an activity or an action. It contains information on the input objects and the resources that are consumed by an activity or an action, and the availability of the output objects and the resources that are produced by an activity or an action.

If the same resource or object can be consumed by two or more activities or actions at the same time, it will only be consumed by the activity or action that receives the control signal. A control signal of a task is produced by another task or by an information process, which assigns the possibility to consume objects and resources to activities or actions. Within a specific logistic application, such a control signal is, for instance, an instruction to a person to load a container on a vessel. It can only be given after sufficient capacity of the vessel is assigned to that action by the information process. The person may give a report signal after discharging the container.

2.2.3 Modelling business systems

As mentioned, we use the formalism of timed, coloured, hierarchical Petri nets to model the concepts of business systems. This modelling technique is discussed in more detail in annex 1. We will briefly discuss the basic components here. These components are places, processors, connectors, and tokens. Processors can be composed to nets of processors, they can be elementary, they can be time consuming, and they can fire. Nets consist of places connected with processors. When a processor fires, it consumes at least one token from all its input places and can produce tokens in one or more of its output places. A token is specified by its value, its identity, the place in which it resides, and the time at which it may leave the place.

The concepts that define the structure are modelled as follows:
- a *business process* is modelled as an open net that can be decomposed in elementary processors, modelling the tasks of that business process;
- a *task* is modelled as an elementary processor that can be time consuming;
- a *service* is modelled as a open net;
- an *object type* is modelled by a complex (chapter 3);
- an *actor*, a *superior*, and a *subordinate* are modelled as non-elementary processors.

Because a business process can be decomposed in tasks, the connectors of the processor modelling the business process are connected to one or more tasks. A business process and a task have input and output connectors to consume and produce respectively:
- objects;
- information (control and report);
- resources.

The concepts that define the behaviour of a business system are modelled as follows:
- an *activity* is modelled as the firing sequence of a net;
- an *action* is modelled as the firing of an elementary processor;
- an *object*, a *resource*, a *control signal*, and a *report signal* are modelled by a token. Therefore, they have a unique identification.

A control or a report signal contains the identifications of the objects and the resources that are to be consumed or produced by a service or a task. Each activity is triggered by an initial control signal and the input objects and resources that are represented by that signal. The final report signal of an activity contains the result of that activity. By formally modelling objects, signals, and resources as tokens, they have the characteristics of tokens (value, identity, time stamp, and place). For instance, in external logistics the token representing a container has the value 'container contents', the identity 'container number', and is ready to leave a location at a certain time.

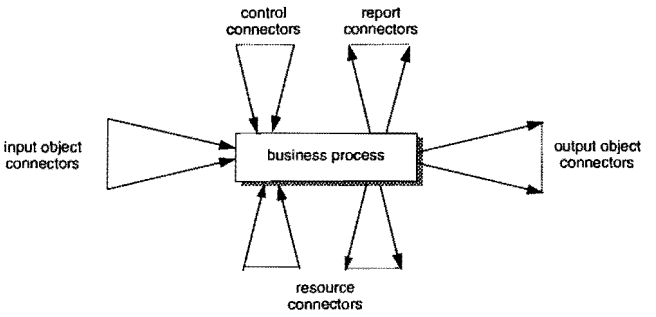


Figure 2.5: Modelling a business process

Figure 2.5 shows a processor modelling a business process. The graphical representation of a task is similar to that of a business process, with the exception that

a task should be represented by an elementary processor. One or more of the output resource connectors of figure 2.5 may be connected to a place to which also an input resource connector is connected.

2.2.4 Modelling an example of a business system and a service

Figure 2.6 shows the business process of the forwarder as presented in section 2.2.1, using the formalism of timed, coloured, hierarchical Petri nets. The figure only shows the places that can contain objects and resources, and the connectors between these places and the processors. The connectors at which the control and report signals can be consumed or produced are omitted. Furthermore, the colour of the objects is not shown and only one of the firing rules of one of the processors is specified as an example. Therefore, the structure of this business process is not completely modelled.

A possible firing rule for the processor for transport from the Netherlands is for instance:

if the containers that are identified by the control signal are available at the time indicated in the control signal (pre-condition),

then the containers are produced after a certain duration in the place connected to the stuffing centre. A report signal is produced containing the identifications of the containers that have been produced in that place and the time at which they have been produced (post condition).

The report signal could for instance be the control signal for the stuffing centre.

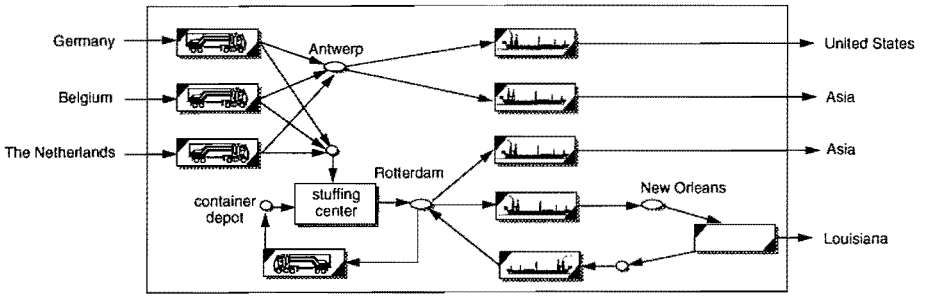


Figure 2.6: An example of the graphical representation of a business process using the modelling technique

The business process of figure 2.6 has the ability to perform many activities in parallel, according to the concepts we have described. One activity is for instance the firing sequence of the transport task from the Netherlands via Antwerp and the transport task from Antwerp to the United States for certain packages. Examples of services are the transport from the Netherlands to the United States via the port

of Rotterdam and the transport from the Netherlands to the United States via the port of Antwerp.

2.3 Co-ordination levels of actors

2.3.1 An example of co-ordination levels

A large multinational may decide to outsource its physical distribution and international transport to a carrier and a forwarder respectively. The multinational requires accurate delivery times to its customers. In the case of physical distribution, the multinational makes agreements with the carrier (who also operates a warehouse) to exchange information concerning the articles that are to be delivered to the customers. The information is exchanged by, for instance, delivery forecasts and delivery orders.
In the case of international transport, the multinational has made agreements with the forwarder. Contractually, the multinational and the forwarder have, for instance, agreed upon the transport of containers specified in a number of Twenty feet Equivalent Units (TEUs) per year from the Netherlands to the United States. In more detail, the multinational gives information on the transport of a number of TEUs from the south of the Netherlands to the northern part of the state of Louisiana. The forwarder decomposes the transport service in three tasks: transport to a transshipment place in the Netherlands (pre-carriage), transport from a transshipment place in the state of Louisiana to the final destination (on-carriage), and transport between the two transshipment places (main transport). Because the main transport is by sea, it is likely that the transshipment places are ports. During the execution, the multinational and the forwarder exchange information on the exact places, times, and containers that are to be transported. Depending on the place and the time, the transport to and from the ports is either by road, by barge, or by train. Sufficient resources have to be available.

Similar examples can be given in other business areas, e.g. a client and an insurance company agree on an insurance contract for insurance against damage.

2.3.2 Concepts of co-ordination levels between actors

These examples show that actors try to optimize their use of resources by making agreements with other actors. Thus they want to minimize their uncertainty in the behaviour of their business process. In general, we distinguish three *co-ordination levels* between two actors:

- *strategic level*: actors make a long-term agreement on the object types to be consumed or produced, and the resource types to be used during the execution of that agreement. The long term agreement results in the specification of one or more services of the subordinate.

- *tactical level*: actors have to agree in more detail on the object types to be consumed or produced by the services specified at the strategic level, thus allowing the subordinate to add more detail to its services. At the tactical level, additional constraints are formulated for the operational level.
- *operational level*: the actual objects are consumed and produced. Co-ordination concerns the processing of actual objects.

The strategic level is a higher level than the other levels, and the tactical level is a higher co-ordination level than the operational level. The services specified at strategic level are for instance an aggregation of the services at the tactical level. In some cases, the tactical level is omitted. Sometimes, the operational level may never be executed (e.g. in case of insurance against damage of object types like a car for which a standard contract is valid).

Definition *contractual relation, incidental relation*
A *contractual relation* is a relation between two actors where they, first of all, make agreements on the structure of an interorganizational business system at strategic and tactical level, and, secondly, co-ordinate the behaviour of that interorganizational business system at operational level. The relation lasts longer than one execution of a task. An *incidental relation* is a relation between two actors for the execution of a standard service of a subordinate. It requires on the one hand the co-ordination between two actors concerning the specification of the standard services of the subordinate, and on the other hand, the delivery by the superior of information concerning the input objects, the output objects, or both. An incidental relation between two actors exists only during the execution of a task.

Prior to a contractual or an incidental relation, an actor may want to exchange information with other actors regarding his services. These services may be stable or may change in time. The difference between a contractual and an incidental relation lies in the number of executions of the same task at operational level by a subordinate: in a contractual relation a task is executed zero, one, or more times, whereas in an incidental relation a task is executed at most once during the relation. The incidental relation is a special type of the contractual relation.
In a contractual relation, the object types that are to be produced or consumed may lead to the reservation of resource types by a subordinate. The business process that is agreed at strategic level is decomposed at tactical level. For example, in a yearly period the object types are expressed in terms of a product family (strategic level), whereas in a given week specific articles are to be produced (tactical level). In such a case, the actors have to agree on, for instance, product families and articles in those families. Another example is a rough estimate of the number of containers to be transported in a year, whereas in a given week the exact number of containers and their identifications can be given.
Besides agreements on the services, other aspects such as the object quality can be agreed upon by contract. Also, the information exchange can be part of the contract.

2.3.3 An example of applying the concepts of co-ordination levels

The example that is given in section 2.3.1 is modelled at all three levels (figure 2.7). The object types and the firing rules of the processors are not specified. The figure shows only the connectors and places of the objects. The connectors and places of the control and report signals are left out. The resources are internal to a process.

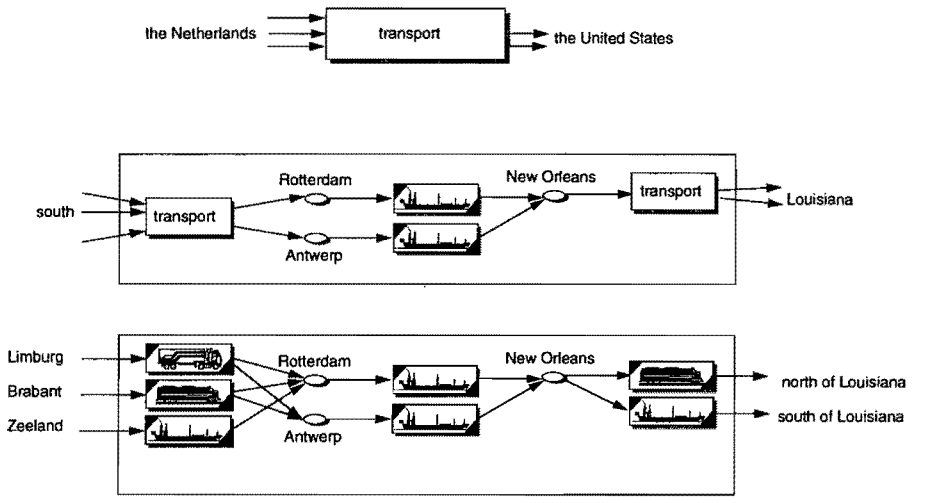


Figure 2.7: Services at different levels

The upper part of the figure shows the structure of the service at strategic level, the middle part shows the tactical level, and the lower part the behaviour at the operational level according to the structure at tactical level. Each level is a decomposition of the higher level. At operational level, the transport service of the tactical level consumes objects from three provinces in the south of the Netherlands and produces those objects in two regions of the state of Louisiana. The transport from the other regions in the Netherlands to the ports of Rotterdam and Antwerp and the other states in the USA can be added.

2.4 Communicating information systems

2.4.1 Examples of communicating information systems

The shipper and the forwarder of the previous example exchange information in order to co-ordinate. For example, the shipper exchanges a transport order with the forwarder, referring to the contract they have. As a result of processing the transport order, the forwarder submits his transport planning to the shipper. By means of the

transport planning, the shipper can carry out his planning for the shipment of the packages. Once the shipment has taken place and the packages are delivered, the forwarder reports on the execution of his service to the shipper (figure 2.8).

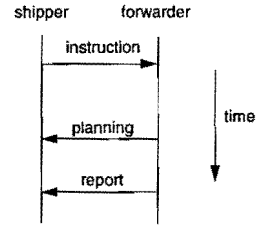


Figure 2.8: Exchange of messages to execute a contractually agreed transport service

The time sequence shown in figure 2.8 can be refined. A shipper can send updates to an earlier exchanged instruction and the forwarder can give updates of the planning and is capable of reporting by several messages.
Sea transport is part of the service agreed by contract between the shipper and the forwarder. If the shipper requires transport of packages, he initiates a *transaction* with the forwarder that contains information to enable the execution of a service. An instruction is the first *message* of the transaction. The forwarder selects the proper service and initiates the execution of the tasks of the service by sending a shipping instruction to a liner-agent for the transport of the cargo with a certain vessel. The planning of the liner-agent tells him that he is able to arrange the transport of the cargo using the particular vessel. He receives a report from the liner-agent after the execution.
A possible time sequence of the messages that are exchanged to support the execution of the service is shown in figure 2.9. Other actors than the ones mentioned thus far are added (e.g. a carrier and a stevedore).

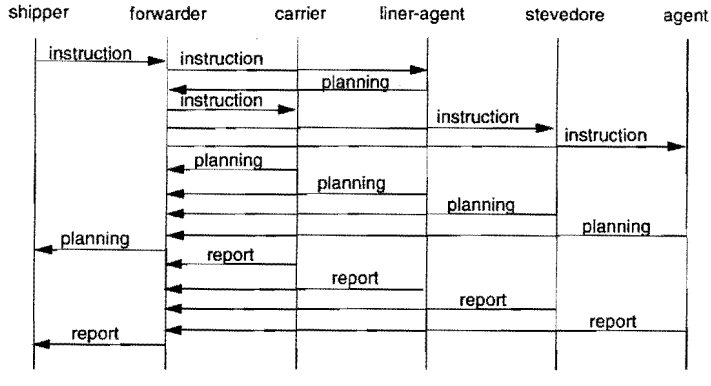


Figure 2.9: A time sequence diagram of the messages of the example

Figure 2.9 shows one possible time sequence of the messages that are exchanged between the actors involved. The sequence depends on the behaviour of the actors involved. Another sequence is for instance the exchange of the report of the carrier before the agent of the forwarder sends his planning. The forwarder can also offer a service to the shipper by which every report received by the forwarder is passed to the shipper. Another service of the forwarder is to exchange the report of the carrier that performs the road transport, which is the report of the first action of the service, and the final report of his agent, which is the report of the final action of the service. Another option is that the forwarder wants to confirm the transactions with the carrier, the stevedore, and the liner-agent, before he has received the report of his agent. These are all options that support tracking and tracing.
Figure 2.9 shows that in time, first of all, the instruction and, secondly, the planning messages are exchanged between the actors involved. Thirdly, all report messages are exchanged. The sequence of the instruction and the planning messages can be called the 'initiation of a transaction'. The exchange of report messages can be called the 'confirmation'.
In this particular example, the forwarder arranges first of all the sea transport. It is also possible that he starts by arranging transport to the premises of a stevedore and waits on the reports of the road carrier and the stevedore before he arranges sea transport. Therefore, the initiation of the execution of tasks can be interleaved with the confirmation of the execution of those tasks.

2.4.2 Concepts of communicating information systems

To be able to execute a service, actors must be capable exchanging messages containing information concerning a service and the input and the output objects of that service, and must agree on rules for the sequence in which the messages can be exchanged. The data structure of the messages is presented in the next chapter. The rules are specified by the concept of a (business) *transaction protocol*. The sequence in which the actions of an activity are controlled is specified by the concept of a *procedure*. To allow the interleaving of the initiation and the confirmation of actions, the rules for a transaction protocol are subdivided in the initiation and the confirmation rules.
As indicated in section 2.2 we make a distinction between the structure and the behaviour of communicating information systems. The structure is specified by the concepts 'message type', 'transaction protocol', 'procedure', and 'step'. The behaviour is specified by the concepts 'message', 'transaction', and 'job' respectively. The concept 'step' has a relation with the concept 'transaction protocol' that we discuss lateron.
We first define the concepts 'message', 'message type', 'transaction', and 'transaction protocol'. Secondly, we define the concepts 'procedure', 'step', and 'job'.

Definition *message, message type, transaction, transaction protocol*
A message is a unit of information exchanged between a sender and a recipient. A message type is the set of messages that have the same characteristics. A transaction is a sequence of messages. A transaction protocol is a set of allowed sequences of message types.

A transaction protocol is decomposed in an initiation protocol and a confirmation protocol. The initiation protocol supports a negotiating mechanism between a superior and a subordinate with respect to (the execution of) a task. Depending on the co-ordination level, the confirmation protocol supports either the willingness to execute a task (strategic and tactical level) or the results of the execution of a task (operational level).
To be able to support the co-ordination levels and the contractual or incidental relations of actors, we distinguish between the following transaction protocols:

- *contract protocol*
  The contract protocol is a specific transaction protocol used to reach agreement on the specification of the tasks and the related services, and to bind resources of a subordinate for the execution of the services. The contract protocol supports the strategic level.
- *planning protocol*
  The planning protocol is a specific transaction protocol used to exchange more detailed information regarding a task that is to be executed one or more times and is agreed upon during the contract protocol. A transaction of the planning protocol enables a subordinate to bind resources for the execution of one or more services. The structure of the task and the matching service of the operational level is specified. The planning protocol supports the tactictal level.
- *execution protocol*
  The execution protocol is a specific transaction protocol by which a task of a superior that is agreed upon as part of the contract or the planning protocol is executed once by a subordinate. The execution protocol supports the operational level.

A contractual relation is supported by the contract protocol, the planning protocol, and the execution protocol. An incidental relation is only supported by the contract and the execution protocol for a standard service. The execution protocol can also be used to specify the communication between the information system of an actor and a task in the business process of that actor.
A superior or a subordinate must be able to cancel an action or an activity respectively. Therefore, we introduce the *rollback protocol*: a specific transaction protocol used to cancel a transaction of another transaction protocol.

A transaction can be initiated both by a superior or a subordinate. Depending on the role of the actor that initiates a transaction, the related transaction protocol is different. For instance, if a subordinate initiates the execution of a task (i.e. he offers

to execute a task), he must execute that task if the response of the superior is positive. If a superior initiates the execution of a task, he is allowed to cancel the execution even after subordinate has given a positive response.
A transaction has to conform to the following properties (we use the term activity of a transaction as the execution of a service by a subordinate):

- *atomicity*
  A transaction has to be *atomic* from the viewpoint of a superior. This means that the service that is controlled by that transaction is executed completely and without any interference, or not at all. A complete execution of a service is defined as the consumption of all input objects and the production of output objects by that service. The input and the output objects are represented by the information exchanged in the messages of a transaction. From the viewpoint of the superior, a partial execution of the service is not allowed.
- *consistency*
  The information concerning an activity has to be *consistent* for both the superior and the subordinate. This means that after completion of the activity the information related to the activity that is stored by the superior and the subordinate is exactly the same.
- *isolation*
  The activity of a transaction between a superior and a subordinate has to be *isolated* of other activities in other transactions between the same or other superiors and the subordinate. This means that from the view of a superior its result is independent of the result of other services that the subordinate executes at the same time.
- *durability*
  The activity of a transaction between a superior and a subordinate has to be *durable*. This means that the result of an activity can only be altered by initiating a new transaction. The result of an activity is the production of output objects by that action.

By binding resources, a subordinate has the capability to execute the service of a transaction in *isolation* of the execution of the execution of other services or the same service. The allowed sequence of the message types of a transaction protocol has to support the properties *consistency*, *atomicity*, and *durability*. Consistency of information can be reached by allowing the exchange of updates to previously sent information, e.g. one or more messages can be exchanged to report on the result of an action.

Based on the properties of transactions and to allow the initiation of a transaction by a superior or a subordinate, we specify the following basic message types:

type	abbreviation	definition
request	r	a request is a message from an actor to another actor requesting (the execution of) a task
response	s	a response is a message from an actor to another actor negotiating the task or the action
confirm	c	a confirm is a message from an actor to another actor confirming the willingness to execute the task or the results of the execution of the task

Table 2.1: Basic message types

We make these basic message types specific to a transaction protocol (table 2.2).

type	contract pr.	planning pr.	exec. pr.	rollback pr.
request	cr	pr	er	rr
response	cs	ps	es	rs
confirm	cc	pc	ec	rc
exception	–	–	ee	–
exception resp.	–	–	ep	–

Table 2.2: Abbreviation of message types per protocol

Hereafter, the name of a message reflects the type of that message, e.g. a confirm is a message of the type 'confirm'. An 'exception' and an 'exception response' message type is added to the execution protocol to support the exchange of information regarding changes in the action of a transaction that occur during the execution of that action. Changes can occur due to disturbances in the business process, e.g. a traffic jam or a flat tyre. The execution protocol can also be used for the communication between the information system of an actor and a task in the business process of that actor.
The message types listed in table 2.2 are generic in the sense that in practice they have other names. For instance in external logistics, an execution request and an execution confirm are identical to a shipping instruction and a proof of delivery respectively.
The allowed sequence of messages in a transaction protocol is specified by the following rules:

- a request can always be given before a transaction is completed;

- a response can only be given after a request has been received;
- a response can refer to one or more requests;
- a response always refers to the last request that is processed for constructing the response;
- only a subordinate can send more than one confirm, independent of the transaction protocol;
- messages must be processed by the receiving actor in the sequence of sending;
- as part of the contract or the planning protocol, a confirm can be given by a superior after either a response or a confirm to a former request has been received from a subordinate;
- a confirm of a superior in the execution protocol is the agreement of the superior with the result of an action;
- as part of the execution protocol, a superior can only give a confirm after the execution of the task is completely confirmed by the subordinate;
- a confirm of a superior in the contract or the planning protocol is the agreement of the superior with the response of a subordinate;
- a confirm can only be given by a subordinate after either a request or a confirm has been received from a superior;
- as long as the execution of a task of a superior is not completed, a subordinate can exchange exceptions;
- a superior gives a response to each exception;
- a rollback confirm can follow a contract request, a planning request, or an execution request.

Definition *step, procedure, job*
A *step* is an elementary unit of work in an information system. A *procedure* is a specific ordering of steps that has a beginning and an end, and is used to manage the execution of a service. A *job* is the execution of a procedure.

A step in an information process is similar to a task in a business process. A procedure is similar to a service. A business process can have many services, which implies that an information process can have many procedures. Whereas actors have to be flexible to offer new services, the information process has to be flexible to support new procedures.

A procedure consists of steps. These steps have a relation with a transaction protocol. A transaction protocol has to be executed by a superior and a subordinate. A job executes a transaction protocol of a subordinate. The execution of steps is the execution of a transaction protocol of a superior.
To be able to specify the relation between steps and a transaction protocol, we introduce outgoing and incoming transactions. *Outgoing transactions* are transactions that are initiated by a superior to execute a service of a subordinate. We distinguish two different steps to manage the action of an outgoing transaction: one step to support the initiation of a transaction and another step to support the

confirmation of that transaction. A job is initiated by a transaction of a superior that we call an *incoming transaction* of the subordinate. We distinguish two different steps to manage the action of an incoming transaction: one step to send a response and another step to send a final confirm. An incoming transaction that initiates a job can only be confirmed after all outgoing transactions have been confirmed by their subordinates. In terms of messages, a final confirm of the incoming transaction can only be exchanged after the confirms of all outgoing transactions have been received. Therefore, a procedure ends in this case with a final confirm step. Examples, in which a procedure ends with a final confirm step are the ordering of articles, the instruction for transport, and the handling of insurance claims. There are other cases, where the final confirm step is not used, e.g. a request that is received is only initiating a job. An example is the ordering of articles if the number of articles is below a certain stock level. The incoming transaction consists only of one request that can be generated by, for instance, an inventory control software package. In such a case, the procedure does not contain a final confirm step.
A subordinate can send several confirm messages according to the rules of the execution protocol. A choice must be made between the following options:

- only the final confirm of an outgoing transaction initiates a confirm of the incoming transaction. This option is supported by the final confirm step;
- each confirm of a subordinate initiates the sending of a confirm of the incoming transaction. This option is supported by the initiation step;
- if several confirms are exchanged, only the first and the last confirm of a subordinate initiate the sending of a confirm of the incoming transaction. This gives the superior information of the incoming transaction at the starting and the end of an action. This option is supported by the initiation step.

One of these options can be selected at the time a procedure is defined for a service.

A subordinate may for some reason cancel an action. If so, the superior must be able to either initiate a new action instead of the cancelled action or cancel the job and start a new job. If a new action cannot be initiated (the procedure does not contain steps for the control of this new action), a new job has to be started. To be able to cancel a job, none of the actions of the job may already be started or completed. Starting a new job can give changes in the completion time of an activity. These changed times have to be exchanged with the superior.

2.4.3 Modelling the concepts of communicating information systems

The concepts are modelled as follows:

- a *procedure* is modelled as an open net with a special structure;
- a *job* is modelled as the firing sequence of an open net;
- a *step* is modelled as a non-elementary processor. We distinguish between the following non-elementary processors as steps: a start, an end, an initiation, a confirmation, a final confirm, a response, a rollback, and an end-rollback processor. The start and the end processor can be used as a start and an end of

parallel steps respectively. The rollback and the end-rollback processor support the rollback protocol;

- a *transaction protocol* of an incoming transaction is modelled by an open net modelling a procedure and a transaction protocol of an outgoing transaction is modelled by an initiation and a confirmation processor modelling a step. Because we distinguish different transaction protocols, there are different initiation and confirmation processors per transaction protocol;
- an *incoming transaction* is modelled by one or more tokens in an open net modelling a procedure. An *outgoing transaction* is modelled by a token in an initiation or a confirmation processor. The data structure of these tokens is specified in chapter 3;
- a *message type* is modelled as an attribute in a data structure;
- a *message* is modelled as a token of a complex class that models a *message type* (chapter 3).

Thus, the structure of the initiation and the confirmation processor is specific to a given transaction protocol. Because the processing of a message may take some time, the response to that message will be received after some time. However, a response must be received in time, because other actions of the same activity may have to be initiated. Therefore, we specify that an initiation processor has the ability to retransmit a request if a response has not been received within a time period that we will call the *retransmission time*. Retransmission of the request can be repeated a number of times. We call the maximum number of retransmissions of the same message the *retransmission count*. If the maximum value of the retransmission count is reached and the response is not yet received, the outgoing transaction cannot be initiated and an alternative outgoing transaction is to be started. The retransmission time and the retransmission count are parameters that can be set during procedure design.

The set of allowed sequences of message types is a transaction protocol. Each elementary processor of a step can produce a message of a specific type. Therefore, we can define a function f from a processor of a step to a message type. A protocol is the projection under f of all firing sequence of these elementary processors in an initiation and a confirmation step. For instance, if a step consists of elementary processors a, b, and c, and $f(a)=m_1$, $f(b)=m_2$, and $f(c)=m_1$, then the firing sequence 'abbacbaac' is transformed in the transaction '$m_1m_2m_2m_1m_1m_2m_1m_1$'.

The definition, the selection, and the execution of procedures is handled by the *Business Transaction Management System* (BTMS). The BTMS is modelled as an open net. Procedures that are also modelled as open nets, are tokens in the BTMS. Thus an open net is a token in another open net.
Several instances of the BTMS can communicate with each other via a composite place called *network*. They can also communicate with other processors of the information system of an actor; we will call these latter processors the *in-house processors* (figure 2.10). Their specification is outside the scope of this monograph. Examples of in-house processors are resource planning software packages (e.g.

route planning) and object type engineering software packages (e.g. Computer Aided Design (CAD) software).

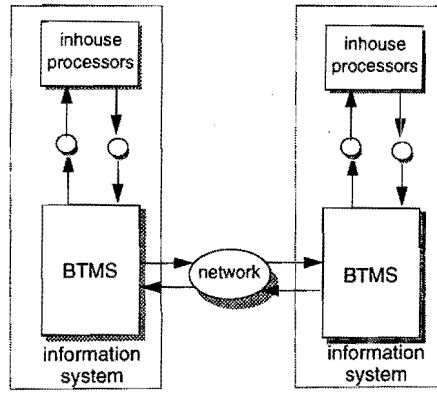


Figure 2.10: Two communicating Business Transaction Management Systems

The BTMS is decomposed as follows (figure 2.11):

- *Service Designer*
  The Service Designer supports a person in the design of new services on the basis of existing tasks. Those tasks can be services of other actors. The output of the Service Designer is a token called 'service'.
  The specification of the Service Designer is outside the scope of this monograph. A tool that can support service design is ExSpect (1990).
- *Transaction Protocol Designer*
  The Transaction Protocol Designer supports a person in the specification of the steps of a procedure. The output is a token called 'step'. The specification of the Transaction Protocol Designer is outside the scope of this monograph. We specify the processors that support the execution protocol to illustrate protocol design.
- *Procedure Designer*
  The Procedure Designer supports a person in the design of procedures for a particular service. The input to the Procedure Designer is a service and steps and the output are tokens called 'procedure'.
- *Procedure Selector*
  The Procedure Selector selects a procedure that is to be executed upon reception of a request. The selected procedure may already be in execution for one or more other incoming transactions. The input of the Procedure Selector is a message and a procedure. The output is a token called 'selected procedure'.
- *Procedure Interpreter*
  The Procedure Interpreter executes procedures. The input of the Procedure Interpreter is a selected procedure and a message. The output is a message.

- *Exception Handler*
  The Exception Handler executes the rollback protocol either for the complete job or for one action of a job, selects an alternative procedure to be executed, and handles exceptions in the execution of an action. Alternative actions for an action that is rolled back are part of a procedure. The rollback protocol is supported by a rollback procedure for a service.
- *Message Handler*
  The Message Handler consumes tokens from the composite place 'network'. If the token is a request, it is produced for the Procedure Selector. If it is a rollback request of the superior, an exception, or a response to an exception, the token is produced by the Message Handler for the Exception Handler. Otherwise, the token is produced by the Message Handler for the Procedure Interpreter. This detail of specification of the Message Handler is sufficient for this monograph.

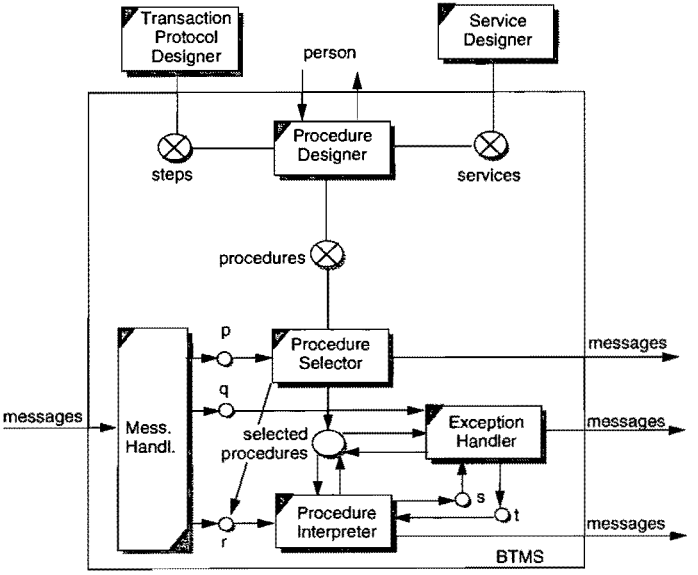


Figure 2.11: Decomposition of the BTMS

A detailed specification of the steps, the Procedure Selector, and the Exception Handler is given in chapter 4 of this monograph. The procedures are tokens in the store that is shown as 'procedures', or in the place 'selected procedures' if they are selected for execution. The data structure of the tokens is specified in chapter 3.

2.4.4 An example of a procedure

An example of a procedure that supports a transport service from a region in the south of the Netherlands to a region in Louisiana (see figure 2.7) is shown in figure 2.12. The start and the end processors are left out in this example.

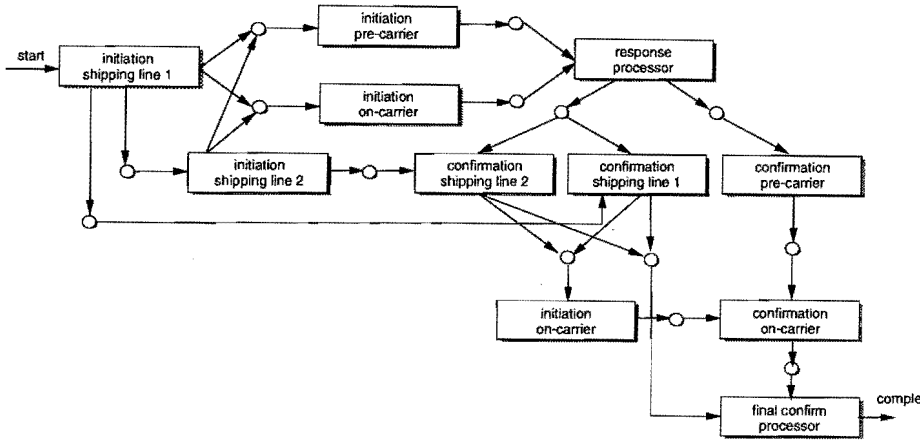


Figure 2.12: An example of a procedure of a forwarder

Figure 2.12 shows the steps that initiate (initiation) and confirm (confirmation) an outgoing transaction and with each step is indicated which actor can execute the service of that outgoing transaction. For instance, an outgoing transaction for pre-carriage to a port is managed by the initiation and the confirmation pre-carrier processor.

The transport by sea of the actor shipping line 2 is an alternative action to the transport by sea of the actor shipping line 1. Only one of these actors can give a confirmation of the execution of its action. Therefore, if one of them is completed successfully, it produces a token for its corresponding confirmation processor. The pre-carriage requires information on the port of loading of the transport by one of the shipping lines. Each of the steps of the procedure in the example can receive messages from or send messages to other actors. The initiation shipping line 1 can, for instance, send requests to the actor shipping line 1 and receive responses from that actor.

The procedure shown in figure 2.12 can lead to several time sequence diagrams, e.g. each outgoing transaction can be confirmed with one or more confirm messages. Figure 2.12 also shows one procedure to manage the service. Other procedures are also possible, e.g. pre-carriage can be initiated before a transaction with a shipping line is initiated. Thus, different procedures and different time sequence diagrams per procedure are possible.

3 Data structures

3.1 Introduction

In the previous chapter we have described the concepts of interorganizational systems. Part of the information system of an actor is the BTMS. The BTMS is modelled as an open net that can consume and produce tokens. Examples of these tokens are messages that are handled by the BTMS, and procedures that describe how to handle the messages. In general, tokens are complexes of a complex class (annex 1). The data representation of a business process, which we call the *business process data structure*, is the basis for the structure of the tokens that can be present in the BTMS. From the business process data structure we derive the following data structures:

- *message data structure*
  The message data structure is the data structure of the input and the output tokens of the BTMS.
- *internal data structure*
  The internal data structure is the data structure of the tokens that can be in the internal place 'selected procedures' of the BTMS.

We will call the rules for the derivation of the message data structure out of the business process data structure the *message modelling rules*. We call the rules for the derivation of the internal data structure from the business process data structure the *internal modelling rules*. Additionally, we introduce the *transaction data structure* meaning the data structure that encompasses all messages in a transaction protocol.

We will start by specifying the generic part of the business process, the transaction, the message, and the internal data structure (sections 3.2 unto 3.4 respectively). We will end this chapter by specifying the data structures of the tokens that can be contained in the BTMS (section 3.5).

3.2 The business process data structure

***Definition** business process data structure*
The business process data structure is a data representation of the structure and the behaviour of business processes.

Because our modelling is based on Petri nets, the business process data structure is a data representation of a Petri net. The business process data structure is specific to a business system (e.g. logistics and health care). We distinguish between elements that are common to all business processes and elements that are specific to certain business processes. We call the first set of elements the kernel of the business process data structure. The kernel is described in this section. The second set of elements is a specialization of the first set. A specialization for external logistics is given in chapter 5.

The components of the data structure that specify the structure of a business process are:

- places are modelled by the entity 'place';
- business processes and tasks are modelled by the entity 'task';
- the decomposition of a business process in its tasks is modelled by a function of the task entity to itself;
- a connector of a place to a business process or task is modelled by an association between the task entity and the place entity. One business process or task may have connectors with several places, whereas each place may have connectors with several business processes or tasks. A distinction is made between an input and an output connector;
- the value and the identity of objects that can be present in a place are modelled by attributes of the entity 'object type'. There is a function from the place entity to the object type entity to model the relation between the object types that can be present in a place.

The behaviour of a business process is modelled as follows:

- activities and actions are modelled by the entity 'action'. Each activity and action have a unique identification and an execution time. There is a function from the action entity to the task entity;
- a specific object is modelled by the entity 'object'. There is a function from the object entity to the place entity. The availability time and the identity of an object is an attribute of the object entity.

The hierarchy between an activity and its actions is not modelled explicitly. It can be derived via the hierarchy of a business process and its tasks and the function from the activity entity to the business process entity. The entities, the associations, the functions, and the attributes of the kernel of the business process data structure that are common to all of its instances, are shown in figure 3.1.

One of the hierarchy functions between the task entity and the place entity represents the decomposition of a composite place, whereas the other function relates a place to a higher decomposition level.
To represent the real world, places and object types have attributes that specify them in more detail. Examples of the identification of places are 'city name', 'region', 'bank account number', and 'in progress'. The latter identification of a place is a stage in the processing of objects. Examples of the identification of objects types are 'container size and type', 'article number', and 'product number'. An

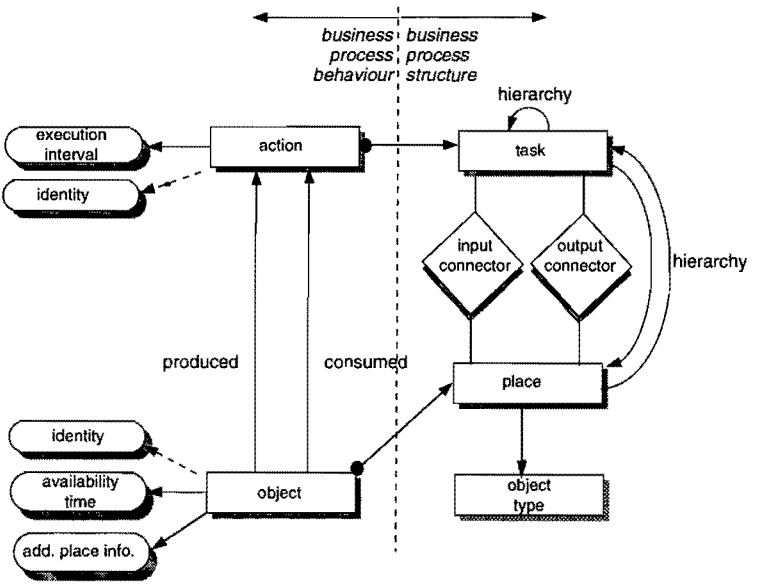


Figure 3.1: Kernel of the business process data structure

object is either in a place or consumed by an activity or an action. Additional information of the place in which the object actually is or has to be, can be specified in more detail focussing on the behaviour, e.g. a place at which the goods are loaded is a city in a certain region. Therefore, objects have an additional attribute 'additional place information', giving more detail of a place specified by business process structure.

In the real world, certain input objects that are to be consumed to be able to produce certain output objects, e.g. a steel plate and an engine can be part of a car. In our model, such a relation between object types is modelled by a firing rule of a task. The concept task is modelled by an elementary processor that has one or more firing rules (annex 1). A particular activity selects a specific firing rule of each task.

Resources are modelled in the same way as objects. A resource has a certain capacity that is available in different time periods to one or more activities. The amount of capacity that is required by an activity is modelled by a firing rule of the corresponding business process or task.

The following constraints are given for the business process data structure:

- an object that is consumed by an action or activity can only be of an object type that can be present in a place that has an input connector to the business process of the action or activity;
- an object that is produced by an action or activity can only be of an object type that can be present in a place that has an output connector to the business process of the action or activity.

3.3 The transaction and the message data structure

> ***Definition*** *transaction data structure, message data structure*
> The *transaction data structure* is the structure of the information that can be exchanged between two actors regarding a service or the execution of that service. The *message data structure* is the structure of the information that can be exchanged between two actors as part of a transaction.

A processor of a procedure can consume and produce tokens that have the message data structure. We derive the message data structure from the business transaction data structure. We make a distinction between a business definition transaction data structure, a business operation transaction data structure, a business definition message data structure, and a business operation message data structure. A *business definition transaction data structure* models the exchange of information concerning the structure of a service; a *business operation transaction data structure* models the exchange of information concerning the execution of a service. In a contractual relation, a business definition transaction data structure is the data structure of the information that can be exchanged by all messages of a transaction of either the contract or the planning protocol and the business operation transaction data structure is the data structure of the information that can be exchanged by all messages of a transaction of the execution protocol. In an incidental relation, it is the information that can be exchanged by all messages of a transaction of the contract and the execution protocol.

The *business definition transaction data structure* is the structure part of the business process data structure extended with an entity representing the characteristics of a transaction and replacing the task entity by a 'service' entity.
The *business operation transaction data structure* is the behaviour part of the business process data structure, extended with an entity representing the characteristics of a transaction of the execution protocol, and replacing the functions between the behaviour and the structure part of the business process data structure with attributes.
The 'transaction' entity has the following attributes (the name of the attribute is given in brackets):

- the identification of the actor that initiates a transaction (init id);
- the identification of the responding actor (resp id);
- the transaction identification (trans id);
- the sequence of transfer of the superior and the subordinate (sup sequence and sub sequence respectively).

A transaction is uniquely identified by the transaction identification and the actor that initiates a transaction (key constraint).
To allow the exchange of information regarding one object that can be in different places at different times, an 'availability' entity is inserted in the business operation

transaction data structure. It has a function to the 'object' entity and a 'consumed' and a 'produced' function to the 'activity' entity.
A transaction of the execution protocol contains one activity and zero, one, or more actions (e.g. a logistic service from the Netherlands to the United States uses sea transport from the port of Rotterdam to the port of New Orleans). An activity is distinguished from an action by the function that an action has to the activity.
The input objects of a service can be contained in the output object of that service. In a business process, objects can be distinguished from each other (e.g. packages can be distinguished from a container in which they are packed). In an information system, a relation between objects must be stored explicitly and a transaction is used to exchange amongst others such a relation. Therefore, the object entity in the business operation transaction data structure has a function to itself. This function is of particular interest if a resource is to be used (e.g. packaging material like boxes, crates, and containers). If a function between two objects exists, both objects have to be in the same output place (constraint).
Figure 3.2 shows the business operation transaction data structure. The business definition transaction data structure can be shown in a similar way.

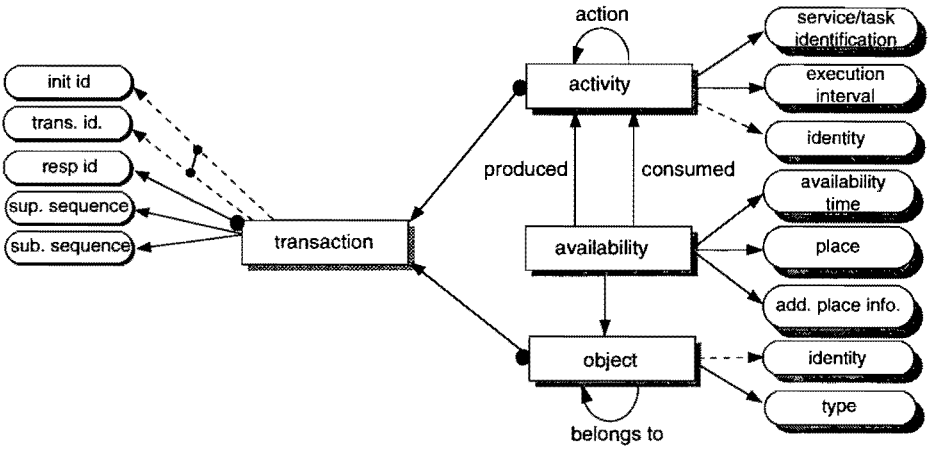


Figure 3.2: The kernel of the business operation transaction data structure

We also make a distinction between a business definition and a business operation message data structure. Both are derived from the business definition transaction and the business operation transaction data structure respectively, by replacing the transaction entity with a message entity. That entity has the following attributes:

- the role of the actor that is initiating the transaction (role). A superior or a subordinate may both initiate a transaction (section 2.4);
- the identification of the sender that is either the superior or the subordinate (sender);

- the identification of the recipient that is either the superior or the subordinate (recipient);
- the identification of the transaction to which the message belongs (trans id);
- the message type;
- the sequence of transfer, which is either the superior sequence or the subordinate sequence depending on the message sender (my sequence);
- the sequence of transfer of the last message that has been received from the other actor and has been processed (your sequence). This attribute only has a value for a response message. It contains the sequence number of the request that has been processed before sending the response.
- the identification of the message (message id).

A message is uniquely identified by the sender, the recipient, the transaction identification, and the message identification (key constraint). The sender or the recipient is the identification of the actor that initiated the transaction. Therefore, the transaction id is unique for a combination of the sender and a recipient. The message id is unique per transaction id. Figure 3.3 shows the business operation message data structure.

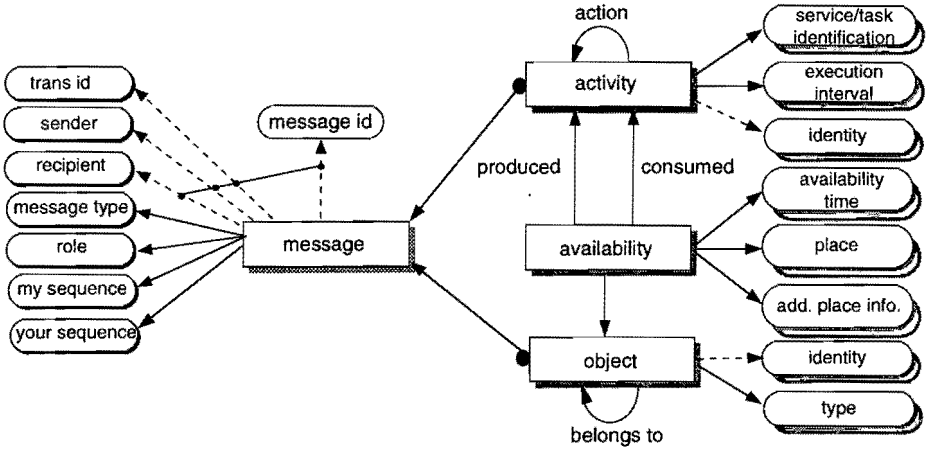


Figure 3.3: The business operation message data structure

3.4 The internal data structure

Definition *internal data structure, business definition data structure, business operation data structure*
The *internal data structure* is the data structure of a token that can be present in the place 'selected procedures' of the BTMS. It consists of the business definition data structure, the business operation data structure, and a data representation of procedures. The *business definition data structure* is a data representation of the services of an actor. The *business operation data structure* is a data representation of the information that an actor has concerning the execution of its services.

In the real world, a service can be supported by one or more procedures. We restrict ourselves in this monograph to one procedure per service. A task that is part of a service of an actor, can be a service of another or the same actor. If the latter is true, a job of an actor initiates another job of the same actor.
The relation between the business definition structure and the business operation structure is identical to the relation between the structure and the behaviour of the business process as modelled in the business process data structure.
Information regarding the same objects can be exchanged by messages that belong to one or more business operation transactions. The status of the objects can differ per business operation transaction (the status of a particular object is the value, the identity, and the availability of that object (annex 1)). We make a distinction between the following three types of 'availability' entities:

- the 'required' entity that represents the *required* status of objects, which specifies that certain objects are required to be available at a certain place and time. The required status is given by the superior;
- the 'planned' entity that represents the *planned* status of objects, which specifies that certain objects are going to be available at a certain place and time. The planned status is given by the subordinate;
- the 'confirmed' entity that represents the *confirmed* status of objects, which specifies that certain objects are actually available at a certain time and place. The confirmed status is given by the subordinate after execution of a service. The confirmed status is represented by the 'confirmed' entity.

If the objects are specified in terms of a quantity (e.g. number of objects of a type and the weight and the volume of objects), then quantity is part of the status information. The status information is modelled by the three entities mentioned before, that have a function with an activity or an action and a function with objects. Whereas in the business process data structure the object entity has a function to the place entity, that function is via the required entity in the business operation structure. The planned and the confirmed entity do not contain a function with a place. The functions between an object and a transaction is via the action.

Both in the real world and in our model of the real world, there is a difference between the availability time and the execution time. Therefore, we also differentiate the following execution intervals:

- the required execution interval;
- the planned execution interval;
- the confirmed execution interval.

These intervals are attributes of the 'action' entity. They are not shown in figure 3.4.

An actor entity is part of the internal data structure to store the identity of superior and subordinate actors once for all transactions in which a particular actor has a role. The 'service' entity has also a function to the 'actor' entity to store information concerning the actor that is able to execute a task of the service, and information concerning the actor that offers a service.

The business operation and the business definition data structure of the internal data structure are shown in figure 3.4. The procedure data structure is shown separately in figure 3.5. The function between the business definition structure and the procedure data structure is shown in figure 3.4.

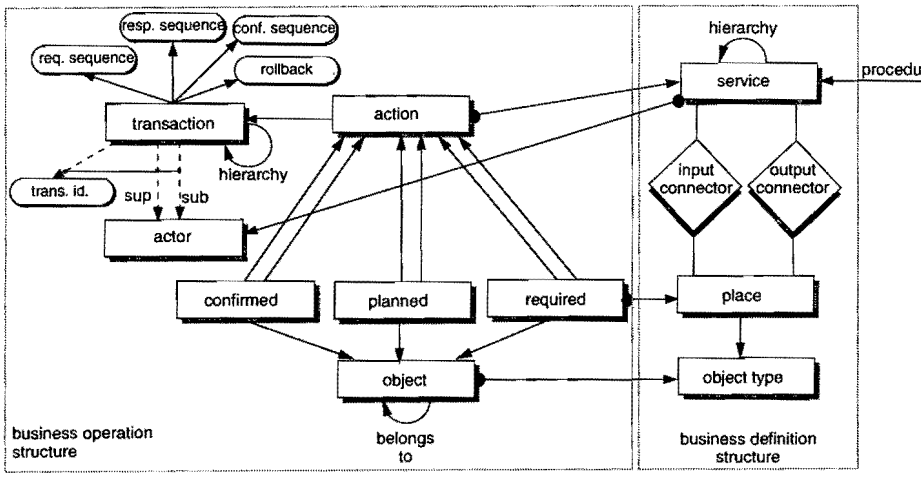


Figure 3.4: The kernel of the internal data structure

The 'rollback' attribute of the 'transaction' entity is used to store whether a transaction is in execution (the value of this attribute is false) or is in rollback (the value of this attribute is true).

Outgoing transactions are related to an incoming transaction by means of the selected procedure of a service. We specify a tree constraint (a tree constraint is explained in Van Hee (1994)) for the internal data structure. The incoming trans-

action is the root simplex. All outgoing transactions can be reached via the incoming transaction.
The structure of the tokens that can be consumed and produced by and can be present in a procedure, is given by the message data structure and the data types respectively. The data types can be derived from the business operation structure of the internal data structure.

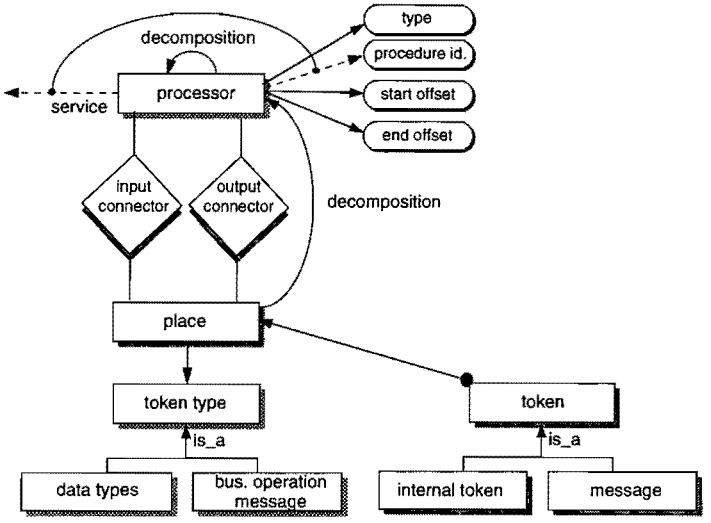


Figure 3.5: Data structure modelling structure and behaviour of procedures

A procedure is uniquely identified by the service and the value of the 'procedure id.' attribute. The start and the end offset are specific to a step in a procedure. The 'type' attribute is a composite attribute that can represent whether it is a normal or a rollback procedure for the strategic level, the tactical level, or the operational level (N for normal procedure, R for rollback procedure, S for strategic level, T for tactical level, and O for operational level).

3.5 Data structures of the tokens of the BTMS

The data structures of the tokens that can be present in the BTMS are derived from the data structures that we have specified thus far. The places of the BTMS are shown in figure 2.11. The data structures of the tokens that can be in those places are:

place	data structure
step	procedure data structure (without the decomposition function)
service	business definition data structure
procedure	business definition data structure and procedure data structure
selected procedures	internal data structure
p	message data structure in which the 'type' attribute can have a value of the set {cr, pr, er}
q	message data structure with (type = rr and role = superior) or (type has a value of the set {ee, ep})
r	message data structure where type has a value that is not in the set {cr, pr, er, ee, ep, rr}
s, t	data type

Table 3.1: Places and data structure of tokens of the BTMS

The internal tokens of a procedure and the tokens that can be in the places s and t are of a data type. We distinguish two data types:

- a data type that represents one or more keys of a transaction. One of these keys identifies an incoming transaction, whereas the other keys identify outgoing transactions in the 'selected procedures' place. Tokens of this data type can only be in the places to which the processors modelling the steps of a procedure are connected with the exception of those places that can contain messages. There can be zero, one, or more tokens with the key of an incoming transaction in a procedure. If the key of an incoming transaction is not present as part of a token in a procedure, one or more steps of the procedure are in execution for outgoing transactions;
- a data type that represents the key of a transaction. A token of this type identifies an outgoing transaction and can only be in the internal places of the processors modelling the steps of a procedure. There is only one token with the key of an outgoing transaction in a procedure if a token with that key is present in the place 'selected procedures' of the BTMS.

The state of an incoming transaction is represented by a set of tokens. This set consists of the tokens in the places between the steps of a procedure containing the key of the incoming transaction, and the tokens in the internal places of the steps containing a key of an outgoing transaction that has a function to the particular incoming transaction. The state of an outgoing transaction is represented by the place of a step that contains the token representing the key of the particular outgoing transaction. In, for instance, the example of section 2.4.4 tokens with the key of an incoming transaction can be in both places between the 'initiation of shipping line 1' and the 'initiation pre-carrier' and the 'initiation on-carrier' processors.

4 Structure and behaviour of the BTMS

4.1 The components

Chapter 2 presents the concepts of business systems and communicating information systems. The behaviour of the information system of an actor is realized by the execution of procedures. The basic elements of procedures are steps that are modelled by non-elementary processors. The BTMS supports the definition, the selection, and the execution of procedures. The decomposition of the BTMS is already shown in figure 2.11 of chapter 2. The data structures of the tokens that can be contained in the places of the BTMS are specified in chapter 3.
We restrict ourselves in this chapter to the specification of the processors of procedures and of the BTMS that support the execution protocol initiated by a superior. The processors to support the other protocols can be specified in a similar way. The structure of procedures can be different for the other protocols, e.g. a protocol initiated by a subordinate is supported by procedures that support only the execution of one protocol. The structure and the behaviour of the BTMS will also be different for the other protocols, e.g. the Message Handler has to process 'request' messages with the role of 'subordinate'.

First of all, we specify the Procedure Designer (section 4.2). Secondly, we specify the steps of procedures (section 4.3). The other sections give a specification of the other components of the BTMS:

- section 4.4: the Exception Handler.
- section 4.5: the Procedure Selector.

At the end of this chapter we will present some conclusions with respect to the verification and validation of the specification, the realization and the implementation, and the visualization of protocols (section 4.6).

A detailed specification of the Message Handler and the Procedure Interpreter is not described in this monograph. The functionality of the Message Handler is briefly discussed in section 2.4.3. The Procedure Interpreter has to execute one or more procedures, whereas one procedure can be executed for one or more incoming transactions. If the Procedure Interpreter can consume a token, it selects a procedure from the 'selected procedure' place. The token is offered at one or more connectors of the procedure and zero, one, or more processors of the procedure are executed. Functionality of the Procedure Interpreter to execute one procedure at a time is already implemented by the software product called ExSpect (Executable Specifi-

cation Tool, ExSpect (1990), Van der Aalst (1992), and Van Hee (1994)). Therefore, we will not specify the Procedure Interpreter in more detail.

In this monograph, we present a specification of the components of the BTMS and the steps of a procedure in structured English. Each processor is specified by a pre condition and a post condition. A pre condition specifies the conditions under which a processor fires and a post condition specifies the result of firing. Pre conditions are printed in italics, whereas post conditions are printed in a normal font (Times Roman).
We will specify the pre and post conditions of the elementary processors of the initiation processor in more detail than we will specify the pre and post conditions of the other processors. The specification that we give can be translated in a formal specification language like Z (Spivey, 1988).

4.2 The Procedure Designer

We make a distinction between two types of procedures: a *normal procedure* and a *rollback procedure* (we will use 'procedure' to refer to 'normal procedure' in this monograph). A rollback procedure supports the rollback of the actions of a service. The Procedure Designer supports a person in the design of procedures for a specific service by selecting and structuring the steps of that procedure. Both a service and a procedure are Petri nets. Figure 2.7 shows an example of a Petri net of a service, whereas figure 2.12 shows the corresponding procedure. The basic functionality of the Procedure Designer is the transformation of a Petri net that represents a service into a Petri net that represents a procedure.
The decomposition of the Procedure Designer is shown in figure 4.1.

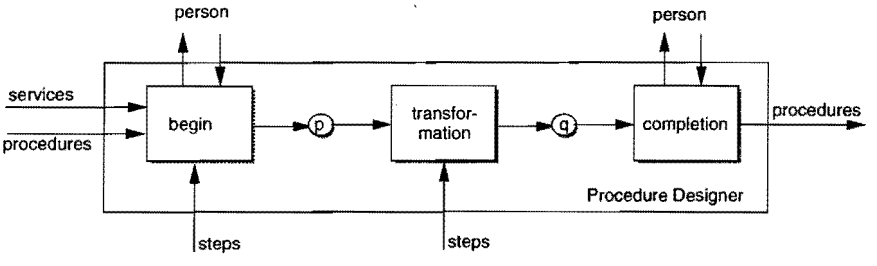


Figure 4.1: Decomposition of the Procedure Designer

The labels of the connectors of the Procedure Designer represent the name of the place to which they are connected. The data structure of the tokens that can be contained in place q is identical to the structure of the tokens that can be contained in the place 'procedures'.

The data structure of the tokens that can be contained in place p is identical to the data structure of a service extended with one or more additional attributes per task. Each of these attributes consists of a sequence number that identifies a leaf in a tree, has a reference to a step to be used in a procedure, and has a reference to a node in a tree. The tree represents a sequence of steps in a procedure. The nodes are aggregations of the tasks of a service. Each node has a sequence number and a reference to a higher parent node, with the exception of the root of a tree. The root does not have a sequence number nor a reference to another node. The steps that are referred to by leaves that are children of the same node can be executed in parallel in a procedure if these leaves have the same sequence number. If leaves that are children of the same nodes have a different sequence number, the steps can be executed in sequence in a procedure.

An example. Figure 4.2 shows a service. The tasks of the service can be aggregated. At the highest level of aggregation, a sequence number is given by a person to the aggregated tasks. Subsequently, the sequence number is given to all tasks of an aggregated task, and a person assigns a reference to a step of each task. The value of the sequence number and the corresponding tree are shown in figure 4.2. The sequence number is shown in brackets behind a character that identifies a task. The reference to a parent node is given by the structure of the tree. The reference to a step is not shown. Furthermore, the places between the processes are not shown.

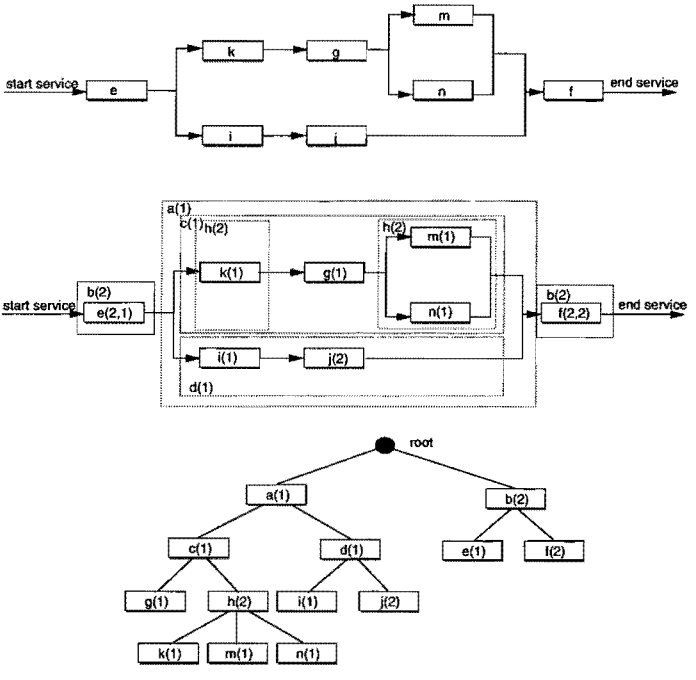


Figure 4.2: An example of a service and its tree

The tree structure is only presented in figure 4.2 to show the hierarchy of the leaves. A token with a tree structure is never available as a token in the Procedure Designer. In the example, nodes or leaves that have identical parents can have identical or different sequence numbers. For instance, leaves h, m, and n have the same sequence number. Therefore, their corresponding steps can be executed in parallel. The leaf g and the node h have different sequence numbers, which means that their corresponding steps can be executed consecutively.

The *begin processor* of figure 4.1 is to support a person in setting up a procedure in reality. This means that a sequence is determined in which the tasks of a service have to be initiated and confirmed in a procedure. The pre-condition for the initiation of procedure design is a service in the place 'services' and specific procedures are available. The service is available, because it is selected by a person. One or more of the following combinations of pre and post conditions are valid:

- *the procedure has to support the strategic level, a relation with a subordinate actor is contractual, the initiation and the confirmation of the contract protocol are present in the place 'steps', and each task in the procedure has received at least one identification of a leaf and a the type of step that is to be used*
  a token is produced in place p
- *the procedure has to support the tactical level of a contractual relation, a procedure to support the strategic level is present, the initiation and the confirmation processors of the planning protocol are present in the place 'steps', and each task in the procedure has received at least one identification of a leaf and a the type of step that is to be used*
  a token is produced in place p
- *the procedure is to support the operational level of a contractual relation, a procedure to support the tactical level or a procedure to support the strategic level is present, the initiation and the confirmation processors of the execution protocol are present in the place 'steps', and each task in the procedure has received at least one identification of a leaf and a the type of step that is to be used*
  a token is produced in place p
- *the procedure has to support the operational level of an incidental relation, the initiation and the confirmation processors of the contract and the execution protocol are present in the place 'steps', and each task in the procedure has received at least one identification of a leaf and a the type of step that is to be used*
  a token is produced in place p

The *transformation processor* has to transform a net of a service into a net of a procedure by using the value of the attributes that are present with a task:

- *there is a token available in place p, where each task has at least one value of the attribute discussed earlier, and the steps that are referenced to by the token in place p, are available in the 'step' place*
  a token is produced in place q

A token in place q can be constructed in several steps. A brief description of an algorithm for the transformation processor is given hereafter. A procedure can be initialized with a start and a final confirm connector that are connected via a place. Each node in a tree can be initialized with a start and an end processor connected via a place. If the procedure is initialized, a processor is inserted for every child of the root of a tree between the start and the final confirm processor. If this child is a leaf, the processor is the step that is referred to by that leaf. If this child is a node, the node is initialized and serves as the root of a new tree. The sequence of the processors that are inserted has been discussed earlier. The procedure that is produced by this algorithm can be optimized further by deleting combinations of start and end processors that have only one output and one input connector respectively.

The *completion processor* is used to complete the procedure by inserting one or more response processors, selecting confirm options, inserting the steps to support the execution of alternative tasks, and generating a rollback procedure:

- *a token is present in place q, the response, the rollback, and the end-rollback processors are present in the place 'steps', and one or more locations of the response processor are known*
  two tokens are produced in the procedures place and related to the service for which they have been designed

A person can insert a response processor to the procedure after an end processor, after all initiation processors, or after both. It is not useful to add a response processor in-between the confirmation processors, unless it is preceded by one or more initiation processors.
A person may decide to send the result of the execution of a service by several confirms. The following options are possible:

- the confirmation processor of each task produces one confirm for a superior of an incoming transaction after having completed its execution;
- all confirms that are received from a subordinate are reproduced for a superior of an incoming transaction by the confirmation processors;
- the confirmation processor of the first task in the service produces one confirm for a superior of an incoming transaction after having completed its execution.

The initiation processor of an alternative task is connected to the same output places in which the initiation processor of the task can produce a token. An initiation processor of a task can produce a token that can be consumed by the initiation processor of an alternative task. An initiation processor can also produce a token that can be consumed by the confirmation processor of the same task. The confirmation processor of the alternative task can consume a token from and produce a token in the same place out of which and in which respectively the confirmation processor of the task can consume and produce a token.

A rollback procedure is generated on basis of the previously designed procedure. A rollback procedure is initialized with the 'start' and the 'end-rollback' processor. Per subordinate that can execute a task of a service, a 'rollback' processor is inserted

between the 'start' and the 'end-rollback' processor. These 'rollback' processors can be executed at the same time (the 'start' processor is able to produce a token in the input place of each 'rollback' processor). A 'rollback' processor can produce a token in either a 'reject' or a 'confirm' place that serve as the input places of the 'end-rollback' processor. An example of the structure of a 'rollback' procedure that can support the rollback of the execution of two tasks is shown in figure 4.3.

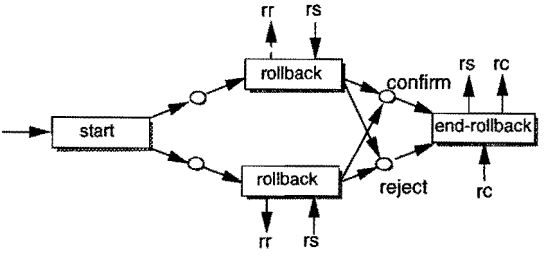


Figure 4.3: An example of a rollback procedure

An example. We will illustrate the design of the procedure that is shown in section 2.4.4. We will show the value of the tokens for a given service in place p and q and the output of the Procedure Designer in the place 'procedures'. Figure 4.4 shows the service and the tree that is constructed by a person via the begin processor.

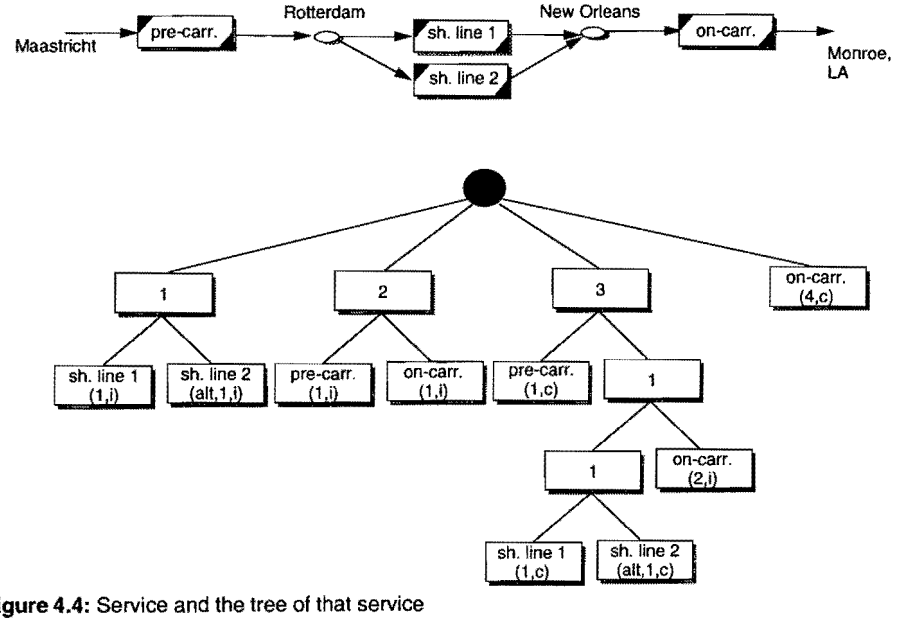


Figure 4.4: Service and the tree of that service

The tree shown in figure 4.4 can be read as follows. Initiate first of all sea transport by shipping line 1. Secondly, initiate pre- and on-carriage. Thirdly, wait for confirmation of pre-carriage and sea transport and send an update of the initiation to the actor executing on-carriage. Finally, wait for the confirmation of the on-carriage.
Figure 4.5 shows the value of the token in place q that is generated on the basis of the tree shown in figure 4.4. Each processor of a procedure and a rollback procedure has a connector to a store. This store and these connectors are not shown in this section. Furthermore, all connectors of the initiation and the confirmation processor are specified in section 4.3. We do not show all connectors in this section.

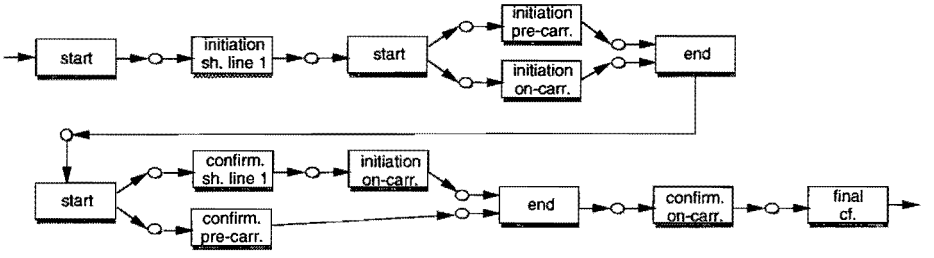


Figure 4.5: The value of the token in place q generated on basis of the value of the input token of figure 4.4

The value of the token that is produced by the completion processor depends on the decision made by a person and is determined by the alternative tasks. In this example, we assume that a person inserts a response processor in the structure of the procedure after the initiation processors of the pre-carriage and the on-carriage. The value of the token that is produced by the completion processor is shown in figure 4.6.

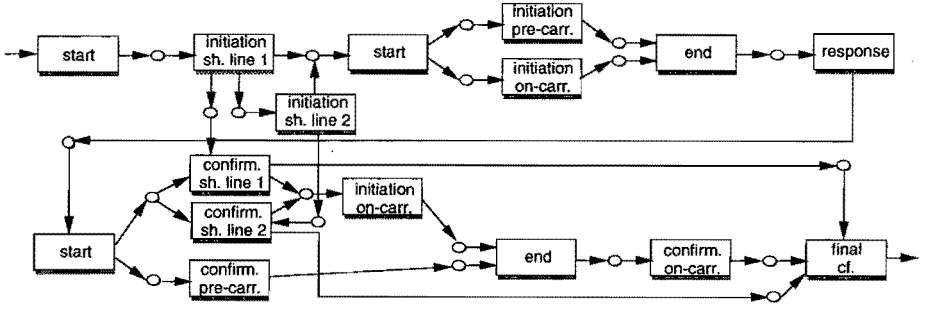


Figure 4.6: The procedure of the service shown in figure 4.5

As one may see, the example of a procedure shown at the end of chapter 2 is simplified with respect to the example given here. The start and the end processors are not shown in chapter 2.

4.3 The steps supporting the execution protocol

4.3.1 High level Petri net of the initiation and the confirmation processor

As we have explained in section 4.2, the initiation and the confirmation processor that support the execution of a transaction of a specific task, are connected to each other via places and have connectors to the environment and to other initiation and confirmation processors in a procedure. Figure 4.7 shows the high level Petri net of the initiation and the confirmation processor of the superior initiated execution protocol. The names that are given to each connector, identify the places to which these connectors are connected. The initiation and the confirmation processors are decomposed further in subsection 4.3.2 and 4.3.3 respectively.

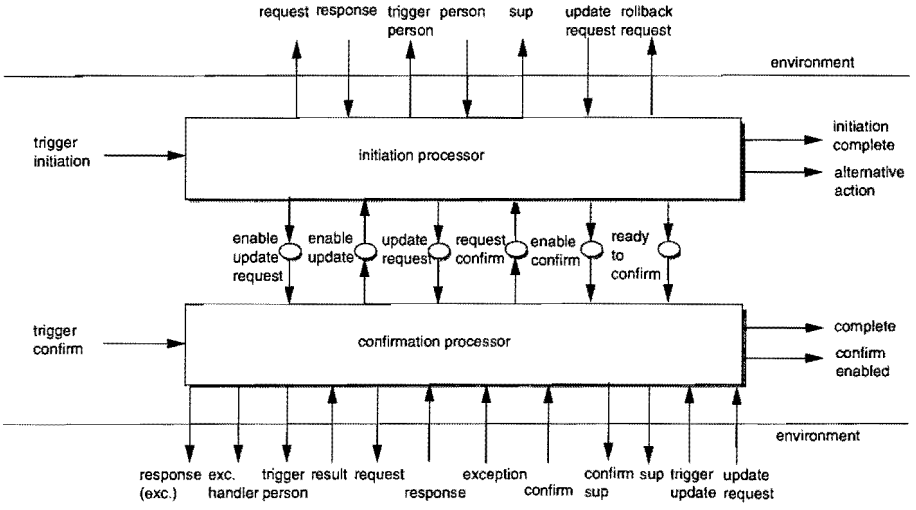


Figure 4.7: High level Petri net of the initiation and the confirmation processor

The places identified as 'request', 'response', 'exception', and 'confirmation' can contain tokens with the message data structure of the corresponding message type. The place identified as 'trigger initiation' can contain tokens with the data type identifying an incoming transaction. All other places can contain a token with the data type identifying an incoming transaction and an outgoing transaction.

The places identified as 'trigger initiation', 'initiation complete', 'alternative action', 'confirm enabled', and 'complete' are connected to other processors of a procedure at procedure design. The 'confirm enabled' place is connected to the final confirm processor. The Exception Handler and a person have input connectors with the 'person' and the 'result' place. The Exception Handler has output connectors to the places 'rollback request', 'response (exc.)', and 'exc. handler'. A person has also input connectors to the 'trigger person' place. The Procedure Selector (see figure 2.11) is the environment that produces tokens in the other places of the initiation and the confirmation processor. The tokens that are produced in the remaining places of an initiation or a confirmation processor can be consumed by the composite place called 'network' (see figure 2.10).

4.3.2 Decomposition of the initiation processor

The functionality that is specified in this section, is based on message tokens produced by a superior that contain only information concerning one activity. An incoming message token can also contain information concerning actions of an activity. The specification can be easily adjusted to support both the processing of an activity and one or more actions in message tokens produced by a superior. A processor of the initiation processor can only fire if the attribute 'rollback' has the value 'false'.
The structure of an initiation processor is shown in figure 4.8.

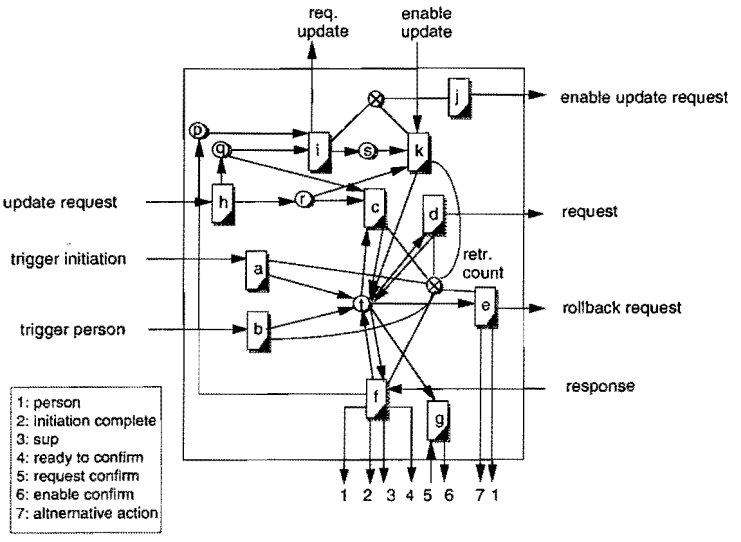


Figure 4.8: Structure of an initiation processor

One token can be left in the place 'enable update request', after the transaction has been completed. The confirmation processor can have produced a token in the 'complete' and the 'confirm enabled' place already, whereas the initiation processor is still trying to send an update to a request. This request can never be send. It will be consumed by a local error detection process that does not communicate with another information system.

Instances of the 'required' and the 'object' entity and functions between these instances have to be computed by an initiation processor. It concerns the computation of the value of the 'required time' attribute of the instances of the 'required' entity, the function of the 'required' entity to the 'place' entity, and the instances of the 'object' entity (figure 3.4). This computation is supported by several processors of the initiation processor. Therefore, we specify it once. We first specify the computation for the values of the 'required time', secondly for the function between the 'required' and the 'place' entity, and finally for the instances of the 'object' entity.

There are several firing rules for performing this computation. The condition for firing is printed in italic and we will identify each condition by a character and a digit. If a condition is decomposed, a digit is added (e.g. condition A.3.2). If a condition is the same for the 'required time', function between 'required' and 'place' entity, and instances of 'object' entity, the condition is not repeated. It is referred to by the character and the digit. For instance, condition B.1 appears several times. This prevents us from repeating the condition several times.

Computation of 'required times'.
We use the 'starting time' and 'completion time' to refer to the availability time for consumption and production respectively of objects. The required times are computed by one of the following firing rules:

A. *If the initiation processor is the first processor of the procedure*
the starting and the completion time are computed either by a backward or a forward control concept (chapter 2).

B. *If the initiation processor is preceded by one or more initiation processors and*
B.1 the preceding processors initiate the execution of tasks that precede the task triggered by this particular initiation processor.
The fact that the tasks are preceding this task, is retrieved from the internal store on basis of the structure of the service. The planned completion times of the preceding actions are retrieved via the identifications of the outgoing transactions of the previous actions and the one with the highest value is the required starting time of the particular action. The required duration is used to compute the required completion time.
B.2 the preceding processors initiate the execution of tasks that are to be executed just after the task that is triggered by this particular processor.
This information is again retrieved via the structure of the service. The planned starting times of the succeeding actions are retrieved via the

identification of the outgoing transactions and the one with the lowest value is the required completion time of the particular action. The required duration is used to compute the required starting time.

B.3 the tasks of the preceding processors do not precede or succeed the task of the initiation processor directly.
This information is again retrieved via the structure of the service. Firing rule A is applied.

C. The initiation processor is preceded by one or more confirmation processors.
The confirmed completion times of the outgoing transactions that were executed by these processors, are retrieved from the internal store and the one with the highest value is assumed to be the required starting time of the incoming transaction for this particular initiation processor. Firing rule A is applied.

Computation of function between 'required' and 'place' entity.
The functions between the instances of the 'required' entity and the 'place' entity are computed by one of the following firing rules. The character and the digit in this list refer to the condition given by the same character and digit in the list given for the computation of the times. In some cases, the condition is further refined:

A.1 The task of which the execution is initiated, is the first task of the service.
The input places of the incoming transaction that are retrieved via the identification of the incoming transaction, are the input places of the outgoing transaction. The input places have to be places that can serve as input places of the task of which the execution is initiated by this processor. The output places of the outgoing transaction are the output places of the task of which the execution is initiated by this processor.

A.2 The task of which the execution is initiated, is the final task of the service.
The output places of the incoming transaction that are retrieved via the identification of the incoming transaction, are the output places of the outgoing transaction. The output places have to be places that can serve as output places of the task of which the execution is initiated by this processor. The input places of the outgoing transaction are the input places of the task of which the execution is initiated by this processor.

A.3 The tasks of which the execution is initiated, is neither the first nor the last task of the service.
Both the input and the output places of the outgoing transaction are the input and the output places respectively of the task of which the execution is initiated by this processor.

B.1 The input places are the output places of the preceding actions. The input places have to be places that can serve as input places of the task of which the execution is initiated by this processor. The output places of the outgoing transaction are the output places of the task of which the execution is initiated by this processor.

B.2 The output places are the input places of the succeeding actions. The output places have to be places that can serve as output places of the task of which the execution is initiated by this processor. The input places of the outgoing

transaction are the input places of the task of which the execution is initiated by this processor.

B.3 Both the input and the output places of the outgoing transaction are the input and the output places respectively of the task of which the execution is initiated by this processor.

C. The output places of the task of which the execution is confirmed, are handled as the input places of an incoming transaction for this particular initiation processor. Firing rule A.1, A.2, or A.3 can be applied.

Computation of instances of the 'object' entity.

The instances of the 'object' entity are computed by one of the following firing rules. The reference to the condition is the same as before:

A.1 The input objects of the incoming transaction are the input objects of the outgoing transaction. The instance of the 'required' entity that contains the required starting time and the input places, receives a function with the input objects of the incoming transaction. These objects must be of a type that can be present in the input places of the task of which the execution is initiated by this processor. The output objects are computed on basis of the firing rule of the task of which the execution is initiated.

A.2 The output objects of the incoming transaction are the output objects of the outgoing transaction. The instance of the 'required' entity that contains the required completion time and the output places, receives a function with the output objects of the incoming transaction. These objects must be of a type that can be present in the output places of the task of which the execution is initiated by this processor. The input objects are computed on basis of the firing rule of the task of which the execution is initiated.

A.3 *This option is only valid if, at the design of the service, one of the following options is selected:*

A.3.1 the action of the initiation processor is part of the incoming transaction.
Firing rule A.1 is applied if the incoming transaction has a function between the 'required' entity for the input to the action. Firing rule A.2 is applied if the incoming transaction has a function between the 'required' entity for the output of the action.

A.3.2 all preceding tasks are of the type 'movement' and the action of this initiation processor has to consume all input objects of the incoming transaction.
Firing rule A.1 is applied.

A.3.3 all succeeding tasks are of the type 'movement' and the action of this initiation processor has to produce all output objects of the incoming transaction.
Firing rule A.2 is applied.

B.1 The input objects are the union of the output objects in a particular output place of the preceding tasks.

B.2 The output objects are the union of the input objects in a particular input place of the succeeding tasks.
B.3 This option is identical to firing rule A.3.
C. The union of the output objects of the task of which the execution is confirmed, is seen as the input objects of an incoming transaction for this particular initiation processor. Firing rule A.1, A.2, or A.3 is applied.

The behaviour of each processor of an initiation processor is specified as:

- ***processor a***
  Processor a consumes a token from the 'trigger initiation' place.
  One of the options listed before for the computation of the estimated times, the places, and the objects is executed and an instance of a 'transaction' and an 'action' entity is inserted in the internal store. The 'action' entity has a function to the task to which the initiation processor of the procedure has a function. An identification of the transaction is generated and a 'sup' function is inserted from the new instance of the 'transaction' entity to the instance of the 'actor' entity that has a 'sub' function with the incoming transaction. A 'sub' function is inserted from the instance of the 'transaction' entity of the outgoing transaction to the instance of the 'actor' entity that has a function with the task selected by the function of the executed initiation processor. The value of the attributes 'request sequence', 'response sequence', and 'confirm sequence' of the instance of the 'transaction' entity of the outgoing transaction is set to zero.
  The 'retransmission count' of the outgoing transaction in the store 'retransmission count' is set to zero. The availability time of the outgoing transaction is set to the current time.
- ***processor b***
  Processor b consumes a token from the 'trigger person' place.
  The token contains an instance of an outgoing transaction. The identification of the outgoing transaction is used to retrieve the value of the retransmission count from the 'retransmission count' store. One of the following cases occurs:
  - *if the value of the retransmission count is 0*
    the consumed token at the trigger person connector is deleted (the first request is not yet send)
  - *if the retransmission count is not yet present*
    the consumed token is also deleted
  - *if there is a retransmission count with a value that is greater than 0 and a token is present in place t for the outgoing transaction*
    those attributes of the token that is consumed at the trigger person connector and have a value, overwrite the value of the attributes of the instances of the entities that are part of the complex 'transaction' of the outgoing transaction in the internal store

  In all cases, the value of the retransmission count is set to zero and the availability time is set to the current time.

- ***processor c***
  Processor c consumes a token from place q and place r if the transaction identification of the outgoing transaction in place r is identical to a transaction identification in place q and both transaction identifications are identical to a transaction identification in place t.
  The instance of the 'required' entity of the outgoing transaction that is selected by the transaction identification in the internal store, is updated according to the rules that are executed by processor a. The value of the retransmission count selected by the transaction identification in the 'retransmission count' store is set to zero. The availability time is set to the current time.
- ***processor d***
  The 'retransmission count' store contains a token of which the availability time is equal to the current time, the value of the retransmission count is less than the maximum allowed value, and place t contains a token that is selected by the transaction identification of the token in the 'retransmission count' store.
  The result is:
  - the value of the attribute 'request sequence' of the selected instance of the 'transaction' entity is increased with one;
  - a token is produced in the 'request' place. It consists of an instance of the 'action' entity and the instances of the 'object' entity that have a function with the instance of the 'transaction' entity selected by the transaction identification of the token in place t. The instances of the 'action' and the 'object' entity in the token have a function with an instance of the 'message' entity. The value of the attributes 'service/task identification', 'execution time', 'place', and 'type' of the instances of the 'action' and the 'object' entity of the 'message' entity are copied from the functions that these instances have in the internal store. The value of the attributes 'time' and 'place' of the instances of the 'availability' entity of the message token are the value of the attributes 'time' and 'place' respectively of the 'required' entity. The functions of the instances of the 'availability' entity are copied from the instances of the 'required' entity. The attributes of the instance of the 'message' entity are computed as follows. A value of the attribute 'message id' is generated. The value of the attribute 'trans id' is the value of the transaction identification. The value of the attribute 'sender' is the value of the actor identification that is identified by the 'superior' function of the selected instance of the 'transaction' entity. The value of the attribute 'recipient' is the value of the actor identification that is identified by the 'subordinate' function of the selected instance of the 'transaction' entity. The value of the attribute 'role' is superior. The value of the attribute 'my sequence' is the value of the attribute 'request sequence' of the selected instance of the 'transaction' entity. The attribute 'your sequence' does not have a value.
  - the value of the retransmission count is increased with one;
  - the availability time is set to the current time increased with the retransmission time.

- ***processor e***
  There is a token for a particular outgoing transaction in place t of which the retransmission count has reached its maximum value and the retransmission count is available again.
  A token is produced in the 'rollback request' place.
- ***processor f***
  A token is available in the 'response' place of which the value of the attribute 'trans id' selects an instance of a transaction in place t and a retransmission count in the 'retransmission count' store with a value larger than zero, the value of the attribute 'my sequence' is one higher than the value of the attribute 'response sequence' of the selected instance of the 'transaction' entity, and the value of the attribute 'place' of each instance of the 'availability' entity of the message token is equal to the function between each instance of the 'required' entity that has a function with the instance of the 'object' entity identified by the function of the instance of the 'availability' entity with an instance of the 'object' entity of the message token. Furthermore, if the value of the attribute 'your sequence' of the message token is equal to the value of the attribute 'request sequence' of the selected instance of the 'transaction' entity to which the instances of the 'required' entity have a function, also have instances of the 'planned' entity that have a function with these instances of the 'object' entity and:
  - *for all input objects: (required starting time – starting boundary) ≤ planned starting time) ≤ (required starting time + starting boundary), and*
  - *for all output objects: (required completion time – completion boundary)) ≤ planned completion time) ≤ (required completion time + completion boundary).*
    (The value of the starting and the completion boundary are the values of internal parameters of processor f.)

  The result is:
  - first of all, the instances and the functions of the 'availability' entity are copied to instances of the 'planned' entity. Secondly, the value of the attribute 'response sequence' of the selected instance of the 'transaction' entity is equal to the value of the attribute 'my sequence' of the message token;
  - the starting and completion times of instances of the 'object' entity in the response message are stored in instances of the 'planned' entity that have a function with those instances of the 'object' entity. A token with the identification of the incoming and the outgoing transaction is produced in the 'initiation complete' and the 'ready to confirm' place. If the 'sup' place exists, a message token is produced in the 'sup' place. The instances of the 'action', the 'planned', and the 'object' entity and their functions of the outgoing transaction are copied to the message token. The instance of the 'activity' entity of the outgoing transaction is copied with a function to the instance of the 'action' entity of the incoming transaction. The latter instance is also copied. An instance of the 'message' entity is created with the value of the attributes similar as specified for processor d (e.g. 'my sequence' is equal to

the 'response sequence' that is increased and the value of the 'sender' and the 'recipient' attribute are selected via the 'sub' and the 'sup' function respectively between the instance of the 'transaction' entity of the incoming transaction and the 'actor' entity).
If a token is produced in the 'sup' place, the value of the attributes 'response sequence' of the incoming transaction is increased with one and a token with the identification of the outgoing transaction is produced in place p.

Otherwise, if the planned times are not within the boundaries
An error occurs and a token with the identification of the incoming and the outgoing transaction is produced at the 'person' connector.
If the value of the attribute 'your sequence' of the message token is not equal to the value of the attribute 'request sequence'
A token is produced in place t and the availability time of the token in the 'retransmission count' store identified by the identification of the outgoing transaction is set to the current time plus the retransmission time.
If the 'places' of the message token differ from the 'required' places
An error occurs and a token is produced at the 'person' connector.

- ***processor g***
  A token is consumed from the 'request confirm' place with a transaction identification that selects a token in place t.
  The token selected in place t is produced in the 'enable confirm' place.
- ***processor h***
  All tokens are consumed from the 'update request' place.
  These tokens are produced in the places q and r.
- ***processor i***
  A token is present in both place p and q with the same identification.
  A token is produced in the 'request update' place with the identification of the tokens consumed from the places p and q, the token consumed from place q is produced in place s, and a timer is set for waiting for the enable update token (the timer token is a data type that can contain a value of an identification of an outgoing transaction and a waiting time).
- ***processor j***
  The timer for an 'enable update' token is exceded and a token is available in place s, that is identified by the transaction identification of the timer.
  The token consumed from place s is produced in the 'enable update request' place.
- ***processor k***
  A token with a transaction identification is available in the 'enable update' place and a token with the same identification is present in place s.
  The timer identified with the transaction identification of the token consumed from place s, is deleted, the value of the transaction that is identified with the transaction identification of the token consumed from place s in the internal store, is updated with the value of the token consumed from place s according to one of the rules given earlier, and the value of the retransmission count selected

by the transaction identification in the 'retransmission count' store is set to zero and the availability time is set to the current time.

4.3.3 Decomposition of the confirmation processor

A confirmation processor is capable of processing incoming confirmation messages. The structure of the confirmation processor is shown in figure 4.9.

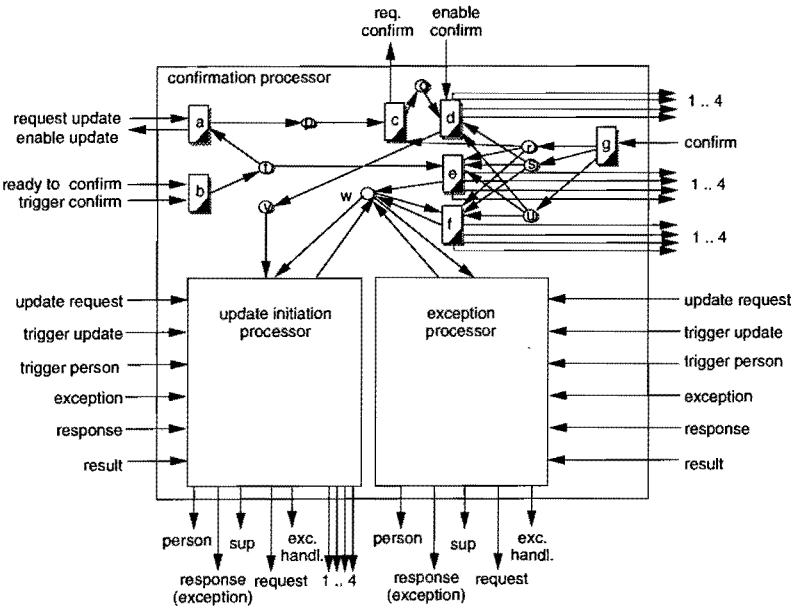


Figure 4.9: Structure of the confirmation processor

The connectors 1 to 4, shown in figure 4.9 as '1..4', are connected to the 'complete' place, the 'confirm sup' place, the 'confirm enabled' place, and the 'exception handler' place respectively. Tokens consumed from the 'exception' place are produced by a person in this place. Depending on the result of the processing of these tokens, either a response is given to the subordinate or a token is produced in the 'exception handler' place. A processor of the confirmation processor can only fire if the attribute 'rollback' has the value 'false'.

We distinguish between three situations for the processing of a confirm message:

- the initiation processor has requested the possibility to process an update of a request. This situation is supported by processors a, c, and d;
- the first confirm message can be processed (the value of the attribute 'confirmation sequence' is zero). This situation is supported by processor e;
- an update to a confirm is to be processed (the value of the attribute 'confirmation sequence' is greater than zero). This situation is supported by processor f.

If processors d, e, and f process the final confirmation, a token is produced in either the 'complete' place, the 'confirm sup' place, the 'confirm enabled' place, or a combination of these places. The production of a token in these places is specified by the structure of the procedure as described before. An overview of the functionality of the processors d, e, and f is given below. There are two possibilities:

- *The confirm message is the final confirm message (a confirm message is the final confirm message, if all objects of the outgoing transaction are consumed and produced according to the requirements and the firing rule of the task to which the action is related).*
  For all objects that are present in the token consumed from place q, s, or u, an instance of the 'confirmed' entity is constructed in the internal store with a function to the action and the concerning object. The time and the place of the object in the token of place q, s, or u is stored in the instance of the 'confirmed' entity.
  A token is produced in one or more of the following places: the 'complete', the 'confirm sup', and the 'confirm enabled' place.
  Additionally,
  - *the confirmed starting or completion time is not in the interval ((planned starting or completion time – starting or completion boundary) ≤ confirmed starting or completion time ≤ (planned starting or completion time + starting or completion boundary)).*
    A token is produced in the 'exception handler' place.
  - *the value of the place of the 'confirmed' entity is not equal to the value of the place of the 'required' entity.*
    A token is produced in the 'exception handler' place.
  The confirm message is not the final one.
  For all objects that are present in the token consumed from place q, s, or u, an instance of the 'confirmed' entity is constructed in the internal store with a function to the action and the concerning object. The time and the place of the object in the token of place q, s, or u is stored in the instance of the 'confirmed' entity.
  A token is produced in place w by processors e and f and in place v by processor v.

As long as a token is in place w of the confirmation processor, an update of a request or an exception can be processed by the update initiation and the exception processor respectively. The behaviour of these processors is explained lateron in this section.

The functionality of each elementary processor of a confirmation processor is briefly specified. If two tokens are consumed from two places, they have the same identifications.

- ***processor a***
  A token is available in place t and in the 'request update' place.
  A token is produced in the 'enable update' place and in place p. Thus processor a enables the initiation processor to process an update of a request.

- ***processor b***
  A token is available in the 'trigger confirm' and the 'ready to confirm' place.
  A token is produced in place t with the identification of the tokens that are consumed.
- ***processor c***
  Place p and place r contain a token and the value of the attribute 'my sequence' of the token in place r has the value 1.
  The token that is consumed from place r is duplicated to place q, and a token is produced in the 'request confirm' place.
- ***processor d***
  A token is available in the 'enable confirm' place and in the places q, s, and u.
  The processing is as indicated before.
- ***processor e***
  A token is available in the places r, s, t, and u and the value of the attribute 'my sequence' is one for the token available in the places r, s, and u.
  The processing is as indicated before.
- ***processor f***
  A token is available in the places w, r, s, and u, the value of the attribute 'my sequence' is identical to the value of the attribute 'confirmation sequence' plus one.
  The processing is as indicated before and the value of the attribute 'confirmation sequence' of the outgoing transaction is increased with one.
- ***processor g***
  A token is available in the 'confirm' place.
  The token is copied completely to places r, s, and u.

The decomposition of the update initiation processor is shown in figure 4.10.

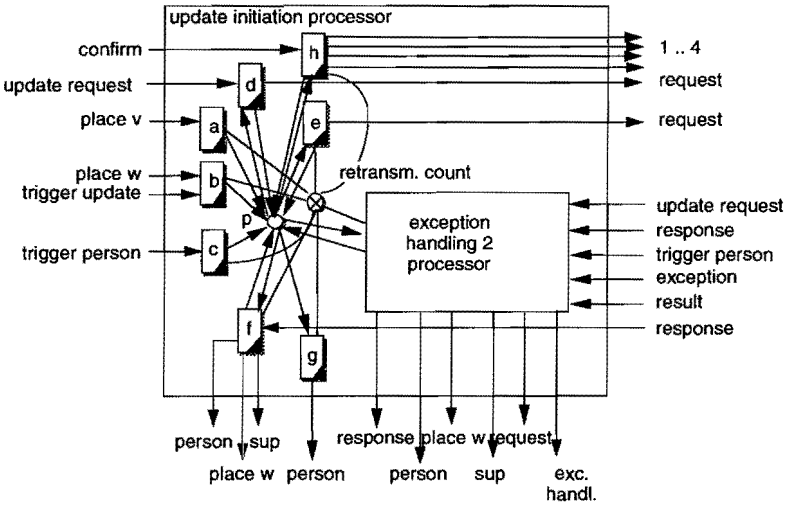


Figure 4.10: Structure of the update initiation processor

Processor h of the update initiation processor can produce a result at the connectors 1 unto 4, according to the behaviour of processor d, e, and f of the confirmation processor (see before). The functionality of the other processors, with the exception of processor a, is identical to processors of the initiation processor. However, the place from which the input token is consumed, has a different name. Table 4.1 lists the functionality of the initiation processor that is supported by the update initiation processor. Note that a rollback is not given after the retransmission count reaches its maximum (there is not a connector with the 'cancel' place). In that case, only a message is given to a person.

initiation proc.	update initiation proc.
a	b
b	c
c	d
d	e
e	g

Table 4.1: Similar behaviour of processors of the initiation and the update initiation processor

Processor a can consume a token from place v of the initiation processor. If so, a request has been given and the superior is waiting for a response. The retransmission count and the retransmission timer are set to their initial value (see the specification of the initiation processor).

The decomposition of the exception handling 2 processor of figure 4.10 is shown in figure 4.11.

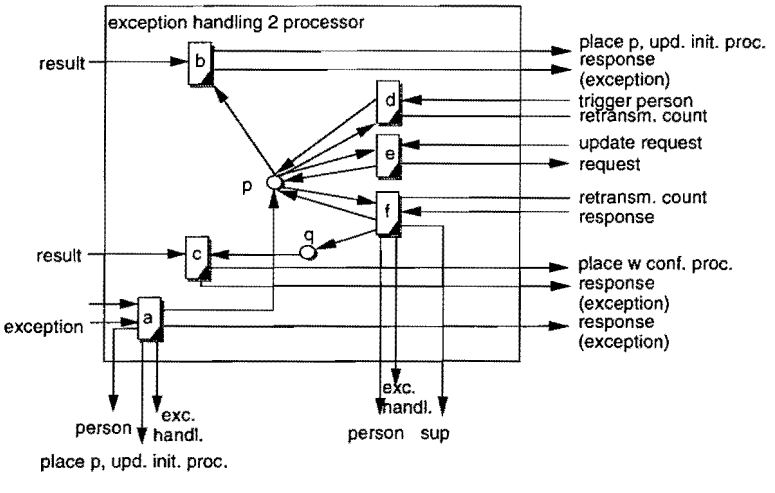


Figure 4.11: Structure of the exception handling 2 processor

The functionality of a number of processors of the exception handling 2 processor is similar to the functionality of processors of the initiation processor (table 4.2). However, the input and the output places have different names.

initiation proc.	except. handl. 2 proc.
c	e
b	d
f	f

Table 4.2: Similar behaviour of processors of the initiation and the exception handling 2 processor

The main functionality of exception handling is given by processor a. Its functionality is similar to the functionality of the processors d, e, and f of the confirmation processor. If an exception does not cause changes in the execution of an action in comparison with the planning, the response can be given directly. Otherwise, the exception is either handled by a person or by the Exception Handler. At the implementation of the confirmation processor, the choice can be made to send exceptions either to a person or to the Exception Handler. If a person handles the exception, the result is processed by processor b or c, depending on the state of the execution protocol (a token with the proper identification is either in place p or in place q of the exception handler 2 processor), and a response is given to the exception. The result can never cause the rollback of the transaction, because the subordinate did already start the execution of the action. It can only give changes in times and possibly also in places.

The decomposition of the exception handling processor of figure 4.9 is shown in figure 4.12.

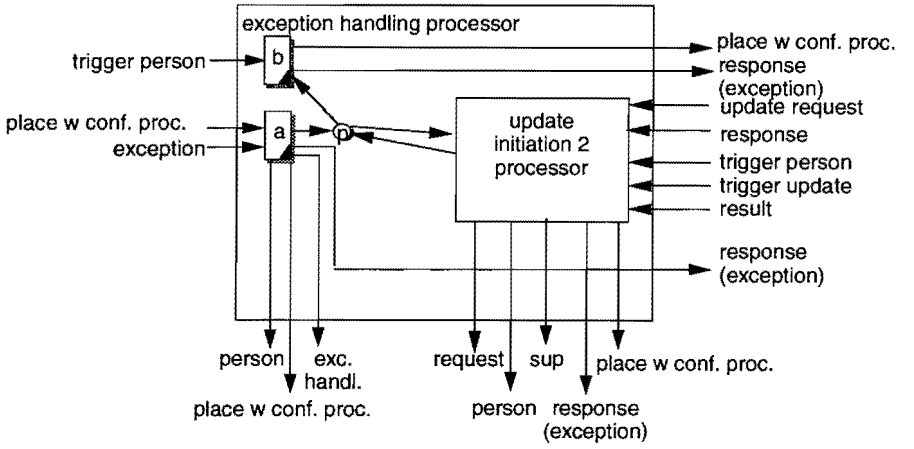


Figure 4.12: Structure of the exception handling processor

The exception handling processor consumes exception messages and produces responses to these exceptions. The functionality of processor a and b is similar to the functionality of processor a and b, respectively, of the exception handling processor 2.

The decomposition of the update initiation 2 processor of figure 4.12 is shown in figure 4.13.

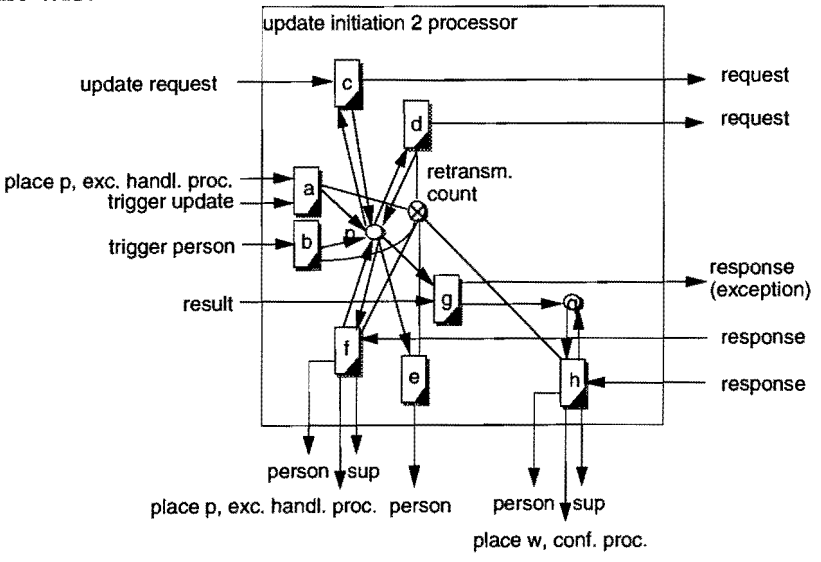


Figure 4.13: Structure of the update initiation 2 processor

The functionality of the processors of the update initiation 2 processor is similar to the functionality of the processors of the initiation processor (table 4.3). However, the place from which the input token is consumed, has a different name. Note, that a rollback is not given after the retransmission count reaches its maximum. In that case, only a token is produced that can be consumed by a person.

initiation proc.	update initiation 2 proc.
a	a
b	b
c	c
d	d
e	e

Table 4.3: Similar behaviour of processors of the initiation and update initiation 2 processor

The functionality of processor g is similar to the functionality of processor b of the exception handling processor, with the exception of the input connectors at which the tokens are consumed, and the output connectors, at which the tokens are produced.

4.3.4 Decomposition of the remaining processors of procedures

The remaining processors of procedures are the 'start', the 'end', the 'response', the 'final confirm', the 'rollback', and the 'end-rollback' processors.

The 'start' processor is an elementary processor that is able to consume an input token and produce the same input token in one or more output places.

The 'end' processor is also an elementary processor that is able to consume one token from each input place, whereas these tokens all have the same identification of an incoming transaction, and produce one token in its output place with the union of the value of the consumed tokens.

The 'response' processor is capable of consuming a token in the 'end complete 1' place and produce that same token in the 'end complete 2' place after having produced a token in the 'sup response' place. The latter token contains the instances of the 'activity' and the 'object' entity of the incoming transaction and their functions. The instances of the 'availability' entity are the instances of the 'planned' entity of the first and the last actions of the activity of the incoming transaction, that have a 'consumed' and a 'produced' function respectively with their instance of the 'activity' entity. If there are more actions that are the first of the activity of the incoming transaction, all instances of the 'planned' entity that have a 'consumed' function with these actions, are given in the token that is produced. The same is applicable to the instances of the 'planned' entity of the last actions. An action is the first action of an activity, if one or more instances of the 'required' entity that has a 'consumed' function with the action, have a function to a place that is an input place of the activity of the incoming transaction. A similar rule can be given to identify the last actions of an activity.

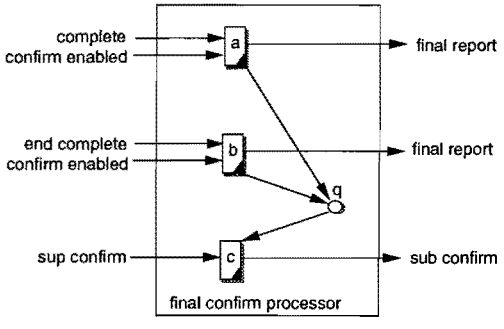


Figure 4.14: Decomposition of the final confirm processor

The 'final confirm' processor is decomposed in two elementary processors and an internal place (figure 4.14).

Depending on the structure of a procedure, a 'final confirm' processor has either an 'end complete' or a 'complete' place. If a token is available in one of these places and a token can be consumed from all 'confirm enabled' places, a token is produced in the 'final report' place. The value of the token is specified in the same way as the value of the token that is produced by the 'response' processor in the 'sup response' place.

The structure of the 'rollback' processor is shown in figure 4.15.

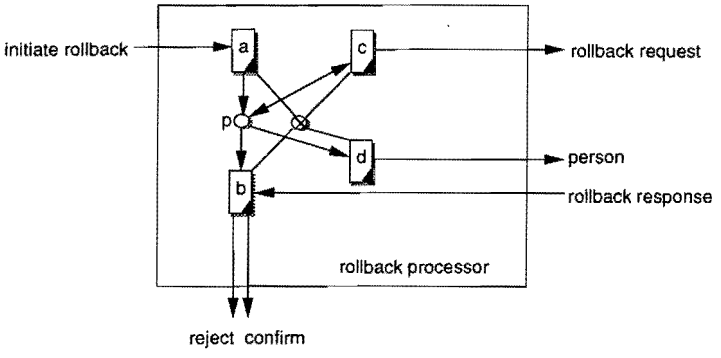


Figure 4.15: Structure of the 'rollback' processor

All tokens that can be consumed and produced by processors a and b of the 'rollback' processor contain at least the transaction identification of the action that is to be rolled back. On the consumption of a token from the 'initiate rollback' place, a token is produced in place p and a timer is set with a retransmission time equal to the current time and a retransmission count equal to zero. If the value of the retransmission time is equal to the current time and the retransmission count is not equal to a maximum, processor c produces a token in the 'rollback request' place. Furthermore, processor c sets the value of the retransmission time to the current time increased with the waiting time and increases the retransmission count with one. Processor b fires if it can consume a token from the 'rollback response' place that identifies a token in place p. If the rollback is confirmed, processor b produces a token in the 'confirm' place. Otherwise, processor b produces a token in the 'reject' place.

The structure of the 'end-rollback' processor is shown in figure 4.16.

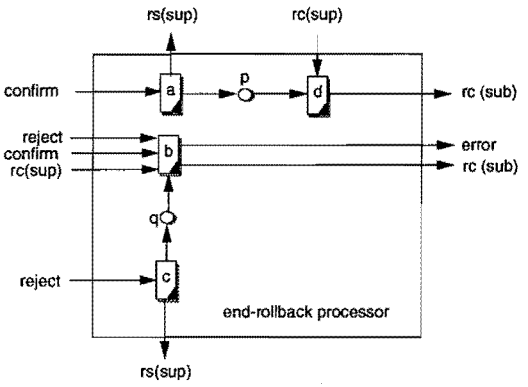


Figure 4.16: Structure of the 'end-rollback' processor

As soon as processor c can consume a token from the 'reject' place, a token is produced in the 'rs(sup)' place with a rejection of the rollback. Furthermore, processor c produces a token in place q with an availability time that is the retransmission count times the retransmission time. Processor c sets the value of the 'rollback' attribute of the incoming transaction to 'false'. This allows all other subordinates to reject or to confirm the rollback within the agreed time. If the token in place p is available to processor b, that processor is able to consume all remaining tokens from the 'confirm' or 'reject' place that refer to the incoming transaction. If the number of tokens consumed by processor b at these two connectors is not equal to the number of outgoing transactions, that were rolled back, minus one, a token is produced in the 'error' place. Processor a can only fire if it is capable of consuming a given number of tokens from each 'confirm' place. The number is equal to the number of outgoing transactions that is to be rolled back. After firing, processor a has produced a token in the 'rs(sup)' place with a confirm of the rollback, and a token in place p. Processor d waits for the confirm of the rollback by the superior (a token is available in the 'rc(sup)' place), produces a confirm of the rollback to each subordinate that had confirmed the rollback, and consumes the identification of the incoming transaction and the identification of all its outstanding outgoing transactions from the 'normal' procedure. The token in place p contains the identifications of all outgoing transaction that have confirmed the rollback.

4.4 The Exception Handler

The Exception Handler can be triggered by tokens that are produced by:

- a superior. These tokens are messages of the type 'rollback request'; or
- an initiation processor. These tokens are produced when the retransmission count reaches its maximum value and the retransmission timer is expired; or

- a confirmation processor. These tokens are produced if the confirmed times are not within the boundaries of the planning, the confirmed places are not the required places, an exception is reported by a subordinate and the planned times are not in the boundaries, or an exception is reported by a subordinate and the planned places are not equal to the required places.

These three cases are specified separately.

A 'rollback request' token of a superior either triggers a rejection or selects the 'rollback' procedure for execution. The structure shown in figure 4.17 supports the processing of a 'rollback request' token.

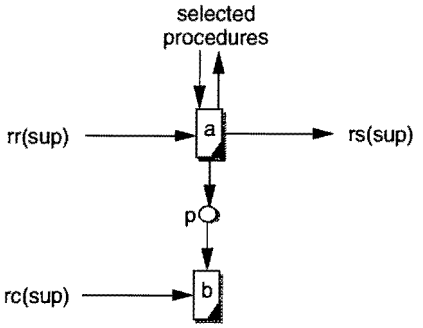


Figure 4.17: Structure to process a superior rollback request

If processor a consumes a token from the 'rr(sup)' place, the 'normal' procedure that is identified via the identification of the incoming transaction and is in execution, is inspected. If a token is in one of the 'confirm enabled' places of the procedure and the token has an identification of an outgoing transaction referring to the particular incoming transaction, processor a produces a token in place p and in the 'rs(sup)' place. The latter token contains the rejection of the rollback.
If there is not a token in one of the 'confirm enabled' places and one of the 'confirmation' processors has a token either in place w, in place p of the 'update initiation' processor, in the place p or q of the 'execution handling 2' processor, in place p of the 'exception handling' processor, or in places p or q of the 'update initiation 2' processor and the token has an identification of an outgoing transaction referring to the particular incoming transaction, processor a produces a token in place p and in the 'rs(sup)' place.
Otherwise, the value of the 'rollback' attribute of the incoming transaction is set to 'true', the 'rollback' procedure is selected for execution, and the identification of the incoming transaction is offered to the 'start' processor of the 'rollback' procedure.
The 'rollback' procedure contains a 'rollback' processor to support the rollback of the execution of each task of the service supported by the 'normal' procedure. However, only those tasks are to be rolled back, of which the execution is already

initiated. These task can be selected by querying the places p and t of the 'initiation' processors and place t of the 'confirmation' processors. If these places contain a transaction identification of an outgoing transaction with a reference to the incoming transaction that is to be rolled back, those outgoing transactions have to be rolled back. Therefore, the 'rollback' procedure of the selected service is duplicated from the 'procedures' store to the 'selected procedures' place. Those outgoing connectors of the 'start' processor of the 'rollback' procedure are removed, that are connected to a place out of which a 'rollback' processor that is not to be executed, can consume a token. The number of outgoing transactions of which a confirm or reject is to be received, is entered as the value of an internal parameter of the processors a and c of the 'end-rollback' processor.

If the 'initiation' processor produces a token in the 'rollback request' place, this token can be consumed by the following net.

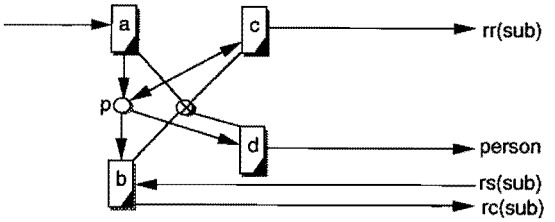


Figure 4.18: Net structure for processing a cancel request of a subordinate

The firing of these processor is identical to the behaviour of the processors of the 'rollback' processor with the exception of processor b (section 4.3). We will not specify them in further detail.

If the 'confirmation' processor produces a token in the 'exception handler' place, two situations occur that are processed by the same net structure. The situations are:

- there is a difference in confirmed and required times and places;
- there is a difference in planned and required times and places.

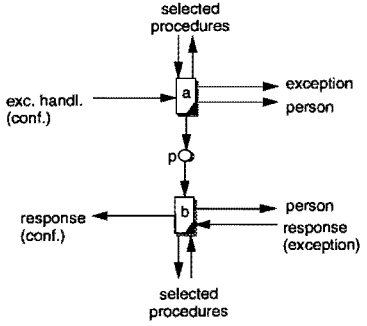


Figure 4.19: Net structure for processing an exception of a confirmation processor

We specify only the processing for the differences in the confirmed and the required times and places. The processing of the differences in the planned and the required times and places is basically the same. An exception is in the rollback.
If processor a consumes a token from the 'exception handler (conf.)' place, two or bothe options are possible:

- *There is a difference in completion times.*
  In this case, the planned completion time of the incoming transaction is computed on the basis of the duration of the other tasks. The planned completion times of the incoming transaction is the planned completion time of the concerning task plus the duration of the succeeding tasks. The latter duration is the summation of the maximum duration of each sequential task succeeding the concerning task. If the planned completion time of the incoming transaction is within a defined boundary and the next tasks in sequence did not yet confirm their execution, the changes in the completion time has to be exchanged with actors that execute the remaining tasks of the service. If the planned completion time of the incoming transaction is not within the defined boundary, a token is produced in the 'exception' place.
- *There is a difference in places.*
  This can only be a difference in output places. In this case, the execution of the succeeding tasks that are already initiated, is to be rolled back and a token is produced in the 'person' place. The 'rollback' procedure is duplicated in the 'selected procedures' place. Those outgoing connectors of the 'start' processor of the 'rollback' procedure are removed, that are connected to a place out of which a 'rollback' processor that is not to be executed, can consume a token. The number of outgoing transactions of which a confirm or reject is to be received, is entered as the value of an internal parameter of the processors a and c of the 'end-rollback' processor.

4.5 The Procedure Selector

The Procedure Selector is able to consume message tokens with the value of the attribute 'message type' being 'request' and the value of the attribute 'role' being 'superior'. We make a distinction between processing first request messages (value of the attribute 'my sequence' is one) and updates to this first request message (value of the attribute 'my sequence' is greater than one).

First request message.
In case of a first request message, a procedure is to be selected from the internal store. The instance of the 'activity' entity of the request becomes a function with a service that is identified by the value of the attribute 'service/task identification'. In a similar way, the instances of the 'availability' and the 'object' entity become a function with instances of the 'place' and the 'object type' entity respectively (instances of the 'availability' entity become instances of the 'required' entity). The instances of the 'service' entity selected by instances of the 'activity' entity in the

request message that have a function with another instance of the 'activity' entity (the 'main-activity'), must have a function with another instance of the 'service' entity that is selected by the 'main-activity'. If this is not the case, a cancel confirm token is produced with the superior as a recipient. Also, a new instance of the 'transaction' entity is created with a 'sup' and a 'sub' function to instances of the 'actor' entity. The 'sup' function must be with that instance of the 'actor' entity, that has a function from the instance of the 'service' entity selected by the 'main-activity'. The value of the attribute 'request sequence' is set to one. Only one procedure has a function with the selected instance of the 'service' entity of the 'main-activity'. If that procedure is not in the place 'selected procedures', it is copied to this place together with all the information of the incoming transaction and the selected instances of the 'service' entity. A procedure is not in the place 'selected procedures', if that place does not contain a token with a processor that has an identification of the selected procedure. A token with the identification of the incoming transaction is produced in a place of the procedure that is connected to the first processor of that procedure. The latter action is executed as soon as a procedure is in the place 'selected procedures'. The Procedure Interpreter will execute the procedure.

Updates of request messages

Updates of request messages are tokens of which the value of the attribute 'my sequence' is greater than one. The value of this attribute has to be equal to the value of the attribute 'request sequence' of the incoming transaction incremented with one. Furthermore, the 'sup' and the 'sub' functions of the instance of the 'transaction' entity selected by the transaction identification of the request message in the internal store have to be equal to the value of the attributes 'sender' and 'recipient' respectively, and the value of the attribute 'rollback' of the incoming transaction has to be 'false'.

First of all, a procedure is selected as described before. Secondly, the following conditions have to be met:

- the selected procedure must have a function with an instance of the 'service' entity that is selected by the 'main-activity' of the incoming transaction;
- the input and the output places of the 'main-activity' of the incoming request have to be identical to those of the incoming transaction in the internal store;
- the instances of the 'object' entity of the request have to be identical to the instances of the 'object' entity in the internal store;
- the completion times minus the starting time of the request has to be greater than or equal to the duration of the instance of the 'service' entity that is selected by the 'main-activity';
- if the request contains actions that are already present in the internal store with a function to the 'main-activity', the input and the output places, the instances of the 'object' entities, and the difference between completion and starting time is checked.

If these conditions are met, the starting and the completion time of the instances of the 'required' entity with a function to the 'main-activity' and the actions are

updated. If the request contains actions that are not yet present in the internal store, those actions, their object instances, and the availability instances of these object instances are inserted in the internal store.
If one of these conditions is not met, an error occurs. The following actions are taken:

- different procedure selected: the 'old' procedure is rolled back and the 'new' procedure is activated. The 'old' procedure can only be rolled back if the confirmation processors in the procedure either do not contain a token or have a token in place t, and the confirmation processors that do not contain a token, have not yet produced a token in the 'confirm enabled' place of the 'confirmation' processor. Otherwise, the 'old' procedure cannot be cancelled and a person has to get in touch with the superior;
- different places: the 'confirmation' processors of the procedure are inspected. If the condition of cancellation of the 'old' procedure given under the error 'different procedure' is met, the incoming transaction is updated and a token with the identification of the incoming transaction is put in a place of the processor that is connected to the first processor of that procedure. If the condition is not met, a person has to contact the superior to solve the error;
- different duration: the selected service is not capable of supporting the required activity. Another service between the same input and output places with the proper duration is to be selected. If such a service is not available, a person has to handle the error;
- differences in the actions: the above mentioned solutions are given for the places and the duration of the action.

If updates of requests have to be given, only those subordinates have to receive an update, that have not yet completed their confirmation (they have not yet produced a token in the 'confirm enabled' place). If none of the subordinates has completed its confirmation, a token is offered to the start of the procedure. If one or more of the subordinates has completed its confirmation, the following actions are invoked:

- the procedure of the selected service is completely duplicated (it receives a new identification). Only those internal tokens of the procedure are put in the new procedure, that contain the identification of the incoming transaction or refer to the identification, that is to be updated;
- those connectors, that are connected to places out of which the initiation or the confirmation processors of the outgoing transactions that have completed their confirmation, can consume a token, are not duplicated;
- a token is offered to the start of this duplicated procedure.

4.6 Conclusion

Aspects that require further study are:

- *validation*
  The validation of the specification of the BTMS. Validation can be carried out by making a prototype. A prototype can be developed using tools that are

available in, for instance, a PC environment, or by using a Petri net prototyping tool like ExSpect (Executable Specification Tool). The first type of prototype can be presented to potential users of the software, demonstrating the functionality. The second type of prototype can also be used to simulate the functionality.

- *verification*
  The specification needs can be formally verified in terms of correctness and completeness. Rambags (1993) has given an algorithm to formally verify a specification of regular protocols (annex 1). The verification uses the concepts of 'regular expression' and 'deterministic automaton'.
- *realization and implementation*
  The specification can be realized either by developing software to support a specific business system or by generating (part of) the specification. Implementation is concerned with the acceptance of the functionality by persons.

Validation using ExSpect requires a more formal specification of the BTMS and the steps of procedures. An example is given by Koop (1992).

Verification by means of an algorithm like the one presented by Rambags (1994) requires a representation of the protocols by specifying a deterministic automaton of the steps that support a protocol. Figure 4.20 shows a finite automaton of the initiation and the confirmation processors that support the superior initiation execution protocol.

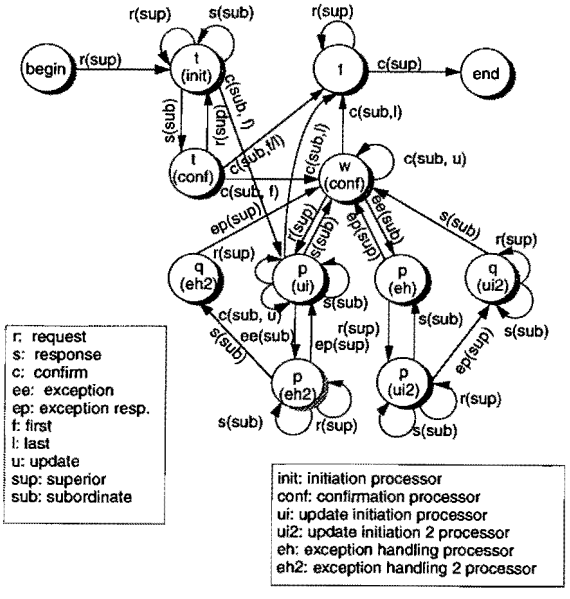


Figure 4.20: A state transition diagram of a finite automaton to support the superior initiation protocol

The finite automaton shown in figure 4.20 is non-deterministic. It can be made deterministic by, for instance, specifying each firing rule of a processor as a state or by assigning a unique message type to each non-deterministic state transition. The latter approach is most often taken in practice. The states of the finite automaton shown in figure 4.20 are identical to the places of the initiation and the confirmation processor, with the exception that retransmissioning cannot be modelled by the finite automaton and the places that are required to model the communication between the initiation and the confirmation processor. State 1 of the state transition diagram of figure 4.20 is part of the Petri net of a procedure.
Although we are able to transform a protocol into a deterministic automaton, the application of a verification algorithm like the one of Rambags is subject to further study.

Regarding *realization*, two considerations are of importance :
- realization versus generation;
- process approach versus data approach.

It is possible to realize the BTMS for a specific business system. Another approach is to realize generic components that can be configured to support a specific business system. In fact, the Transaction Protocol Designer (TPD), the Service Designer (SD), and the Procedure Designer (PD) can be realized independently of any business system. We have already realized a part of the Transaction Protocol Designer in a software product called EDIT (1993). EDIT can support the modelling of messages on the basis of the concepts that we have specified, and the mapping of these models to the EDIFACT syntax (EDI For Administration Commerce and Transport).

The TPD, the SD, and the PD are the components that configure the BTMS. In this way, the BTMS can also be made generic and software is generated by the TPD to support steps in procedures. Additionally, a user interface and an interface to in-house processors can be generated for the BTMS to improve acceptance of the software by persons.

We have taken a process approach to the specification of the BTMS, which means that procedures that are modelled by a Petri net are the parameters to the BTMS. However, we could have taken also a data approach for realizing the BTMS. The steps of a procedure can, for instance, be realized by a data structure that supports state transition diagrams. Each time a transition can be executed, a process will be executed. The internal data model as shown in figure 3.4 is extended with an attribute 'state' of a transaction. A functional data model of a state transition diagram is shown in figure 4.21.

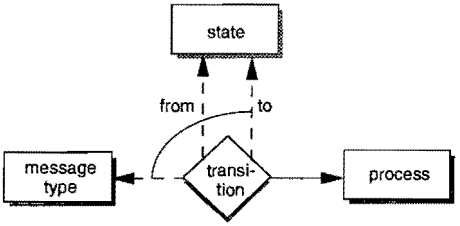


Figure 4.21: Data structure of a state transition diagram

As a transaction is in a certain state, a message type can trigger a transition that has a function with a process. To be deterministic, each transition must uniquely identify the 'from' and the 'to' state and the message type. A data structure of a procedure can be given by a set or rules similar to a script in Workflow Systems (Ellis and Nutt, 1992). A rule can activate a certain outgoing transaction, can wait for the result of the initiation and the confirmation respectively of an outgoing transaction, and can send a response and a final confirm.

Implementation aspects of the BTMS are related to the acceptance of the functionality by persons. The acceptance has to do with, for instance, the (presentation of the) functionality, the flexibility that is offered, and the costs of the software. Acceptance may be improved by offering different software products that help to solve specific problems of actors. Examples of such products are, for instance, the automatic handling of routine transactions and the ability to interface different in-house processors with each other. We believe that the costs of the software can be less than the costs of conventional software, because the software can be easily adapted to different business systems and different business processes.
Implementation aspects of the other components are related to the actors that provide the configuration services of the BTMS. One may distinguish actors that develop messages and actors that support business process (re-)engineering. A distinction can be made between management domains as discussed in the Open EDI Reference Model (ISO, 1993). To be able to support these management domains, the configuration software products have to communicate with each other.

5 External logistics

5.1 Introduction

In the previous chapters we have presented the concepts of interorganizational systems and have given a specification of the components of the information system and the steps of procedures. The concepts can be applied to various business systems and the information system can support procedures of several business systems. The selection of a procedure by the BTMS and the steps of procedures can be made specific to a particular business system. This chapter describes the application of the concepts and the information system to external logistics. First of all, an example of external logistics is presented in section 5.2. This example will be used to clarify the concepts of external logistics that are given in section 5.3. Section 5.3 also models the concepts of external logistics. Section 5.4 specifies the interorganizational information system to support external logistics and in section 5.5 the modelling is applied to the example of section 5.2. Finally, section 5.6 discusses the application of the concepts in general.

5.2 An example of external logistics

COSY (Computer Supply) is a non existing world-wide supplier of computers, instruments, and medical equipment. COSY has a number of production units all over the world, national sales offices, and three warehouses. The warehouses serve as the inventory function for products that are to be delivered to customers located in a certain continent, Europe, Asia, etc. They have separate organizational units that control the stock level, co-ordinate the delivery to the warehouse, and sell to customers. The products of COSY are subdivided in four groups. The logistics and the information exchange is different for each group. We will specify the logistics of printers and PC (Personal Computer) configurations in detail. The other groups are briefly described. COSY can make a distinction between different packaging types, e.g. small boxes for the spare parts and larger boxes for the computers. The logistics are specified as follows:

- *printers and PC configurations*
  Printers and screens for PCs are produced by two production units, one of them in North America and the other in Asia. They are transported from the production units to the warehouses and are stored in those warehouses. The transportation from North America to the warehouse in Europe is by air and takes at least 8 hours and at most 9 hours. Printers and screens are transported in ULDs (Uniform Load Devices) from the production units to the warehouses. Those ULDs are

stripped and their contents is stored in a warehouse.
Printers that are part of a PC configuration consist of the following parts: a laser writer, a specific cable to connect to the computer, a diskette with fonts and printer drivers, and a manual in a specific language. PC configurations can be of several types, depending on the CPU (Central Processing Unit). One PC configuration may, for example, consist of: 1 printer, 3 cables, 2 manuals, 1 keyboard, 1 mouse, 1 SVGA screen, 1 CPU (586, 66 MHz, 4Mb, 240 Mb), MS-DOS 6.0, and MS Windows 3.1. The PC configurations are not tested before shipping. The despatching of a complete PC configuration takes at least 15 minutes and at most 30 minutes. The transport from the warehouse to a destination in one of the EC countries takes at most 24 hours. The transport to other destinations takes at least 24 hours.

- *computers*
  The computers, and also other expensive products like medical equipment, are produced on the basis of a customer order. They are transported either directly to the customer or through one of the warehouses. If the delivery is through the warehouse, the warehouse assembles the main product with other products, e.g. special cables, a printer and its manuals, and specific manuals for the operating system of the product. Furthermore, each installation that is sold to a customer is tested in the warehouse during assembly.
- *spare parts*
  The spare parts are meant for servicing the products installed at the site of a customer. Spare parts are kept in stock in the warehouses. They are produced if the stock is below a minimum stock level. The service people pick up the parts they need to perform certain repairs. They order the products from the warehouse.
- *small parts or consumables*
  The small parts or consumables are products that are used by other products and have to be replaced regularly (e.g. plotter pens). These products are kept in stock and are produced if the stock is below a minimum stock level. On the basis of a customer order, different parts can be delivered in one consignment to that customer.

The logistics control of each of these product groups is different and the way of exchanging information between the actors involved is different for each product group. The actors involved are the organizational units of COSY, the suppliers, and the customers. As mentioned, we will only illustrate the information exchange for the PC configurations. We make a distinction between the supply to the warehouse and the delivery to the final destination.
Each national sales unit of COSY receives its customer orders. These orders are passed on to the stock control unit of a warehouse out of which the PC configurations are delivered to a specific location. The information that is contained in the customer order is checked against reference information of COSY, and the following information is added to the order by the national sales organization: .

- the details of the customer are validated against the customer reference tables (e.g. name and address);
- the details of the product are validated against the product reference table that contains all types of products that can be sold;
- allowances and charges that are applicable to the specific customer are added;
- other relevant reference information is added to the order.

A final check is made on the financial position of the customer and relevant contract information is added to the order by the sales unit (e.g. restriction on delivery dates). If these checks are positive, the delivery process can be started. The order is given to the stock control unit, which produces an order response to the sales unit if the amount in stock for delivery is sufficient. The stock controlling unit passes a despatch instruction to the warehouse. After receipt of the despatching schedule, a transport instruction is transmitted to a carrier that produces a transport plan as a response. The transport plan can invoke updates to the despatch instruction. The stock control unit produces a delivery schedule to the sales unit that is passed on to the customer.

PC configurations can be delivered if there are sufficient parts in stock. If the stock of certain parts of a PC configuration is below a certain level, extra parts have to be supplied. The order is sent by the stock controlling unit and passed on to the production unit that produces the parts. After production, the parts are shipped in boxes with a unique identification number for each box. A box list shows all the identification numbers of a shipment from the production unit to a warehouse. The production unit sends an invoice to the stock controlling unit. The invoice is entered in the ordering system and the forwarding system of the stock controlling unit. The box list is used by the warehouse to accept the boxes.

The message flow or scenario for the supply and the delivery of PC configurations through a warehouse is shown in figure 5.1. The invoices are left out in figure 5.1.

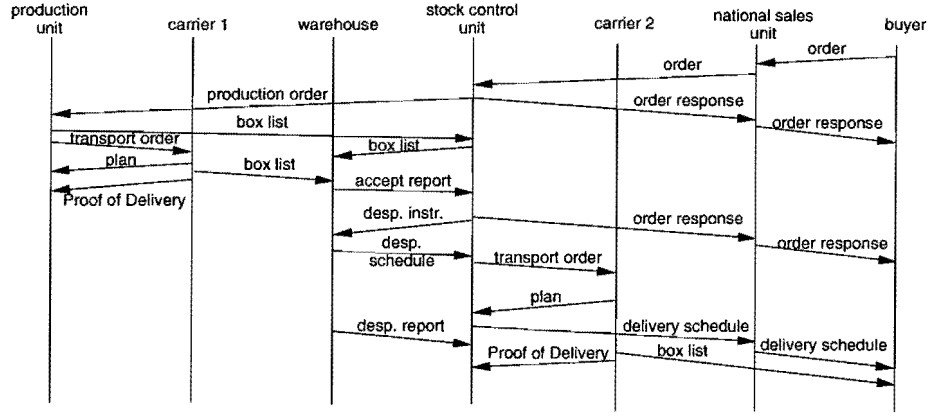


Figure 5.1: Scenario to support the supply and the delivery of PC configurations

Figure 5.1 shows the exchange of messages if the parts are out-of-stock and a production order is to be given to a production unit. In case there are sufficient parts in stock, the messages to support the supply of parts are not exchanged for a specific customer order.

5.3 Concepts of external logistics

5.3.1 A definition of external logistics

The example of COSY shows both the external logistics interorganizational business system and the interorganizational information system. The external logistics interorganizational business systems consists for instance of transport, acceptance of boxes in a warehouse, and despatch of the products from the warehouse by a carrier to the customer. Other forms of the external logistics business system can be envisaged. To be able to specify them, we need a description of external logistic processes.

Several definitions of logistics in general are found in literature:

- Bowersox (1986) defines a logistical process as a system that links an enterprise with its customers and suppliers. A distinction is made between a requirements information flow from customer to supplier and a value-added inventory flow from supplier to customer. The logistical processes between supplier and customer are subdivided into purchasing, manufacturing support, and physical distribution (figure 5.2).

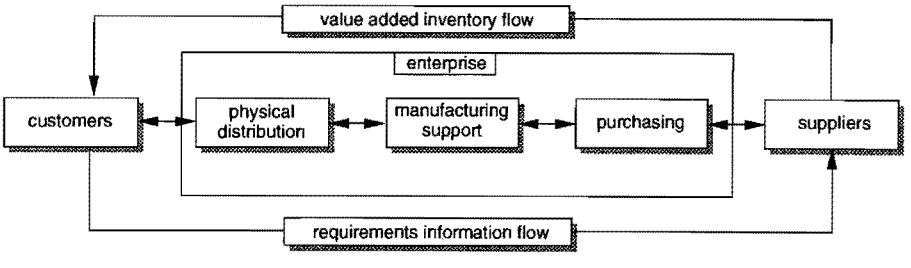


Figure 5.2: Logistics system according to Bowersox

- Ballou (1987) uses the term business logistics: business logistics deals with all move and store facilities that facilitate product flow from raw material to final consumption, as well as with the information flows that set the product in motion for purposes of providing adequate levels of customer service at reasonable cost. Ballou subdivides business logistics into material management and physical distribution. Physical distribution is further subdivided in transportation, inven-

tory maintenance, order processing, production scheduling, protective packaging, warehousing, materials handling, and information maintenance.

- Van Goor et al. (1989) define logistics as the business function that controls the goods flow and the related information flow. Logistics comprises material management and physical distribution, whereas physical distribution in its turn comprises transport, groupage/degroupage, transshipping, storage, and handling.

These definitions of external logistics differ from each other. Bowersox defines for instance three subsystems (physical distribution, manufacturing support, and purchasing), whereas Ballou and Van Goor define two subsystems (physical distribution and material management). Furthermore, Bowersox introduces an enterprise that performs the subsystem, but he does not define an enterprise. We assume that an enterprise can have the role of both a customer in purchasing products and a supplier in selling other products, performing manufacturing support in between (figure 5.3). Van Goor et al. further specify material management as purchasing and production, which is also done by Ballou. However, Ballou does not discuss production any further, whereas Van Goor et al. do. Tilanus (1990) points out that purchasing and physical distribution have great similarities and, therefore, may be defined as external logistics (figure 5.3).

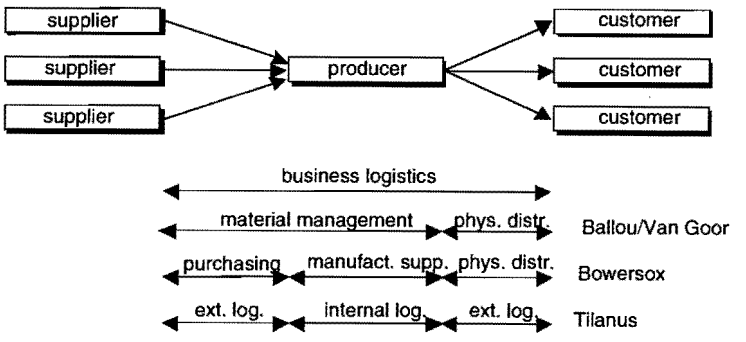


Figure 5.3: Definitions of logistics

At this point, we will present our definition of external and internal logistics. *External logistics* is the interorganizational business system that consists of business processes mainly of the type movement (chapter 2) including handling, that consume and produce objects of a certain type, and the interorganizational information system that stores the structure and controls the behaviour of that interorganizational business system.

Internal logistics is the business system with business processes of the other types (e.g. transformation) consuming or producing the same object types that are

produced or consumed respectively in external logistics, and the interorganizational information system that stores the structure and controls the behaviour of that business system.

Amongst others the object types define external logistics. Object types in external logistics are, for instance, articles, container types, and packaging material. If we define also persons as object types in external logistics, the business system is extended.
According to these definitions, production (or manufacturing support as it is called by Bowersox) is part of internal logistics. External logistics also includes physical distribution and purchasing. External logistics may include the transport of cargo, but also the handling of articles in a public warehouse (Bowersox, 1986). Based on these definitions, the boundaries of external logistics are defined by:

- generic tasks (e.g. transporting, arrival, or assembling). The same generic task can appear in different business processes with different values of certain parameters, e.g. transporting can be present in the business process of several carriers with a different value of the maximum weight that can be transported;
- the object types (e.g. packaging types, container types, or product types). The object types are common to the business processes of different actors;
- the actors involved.

Based on these boundaries, payment is not part of external logistics. Payment can be modelled in a similar way as external logistics. However, a model of payment will be based on other object types with other generic tasks and other actors involved. Regulations (e.g. customs clearance regulations, agricultural regulations, and port authority regulations) may have consequences for the behaviour of the interorganizational business system, e.g. certain types of dangerous cargo are not allowed to be transported together. Other regulations require a view of the objects that differs from the view defined in external logistics (for instance an object in external logistics may be split into several objects for customs clearance, depending on the material of which the object is made, and the customs clearance procedures). Only those regulations that influence the behaviour of external logistics are specified with attributes in this monograph.

5.3.2 Generic tasks in external logistics

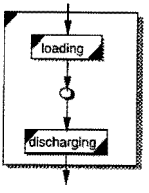
The example in section 5.2 illustrates the use of some of the generic tasks of the business system of external logistics. For instance, transport of boxes is a task in this example. Other tasks in the example are the storage of printers in a warehouse and the packaging of a printer together with a manual and other products prior to despatch to a customer. In other examples, the same tasks may occur, possibly with different naming and different values of certain parameters. We specify the following generic tasks in external logistics:

- *loading*
  Loading is transferring objects into another larger object. Normally, the larger object is an object that can move by itself (e.g. a vessel).

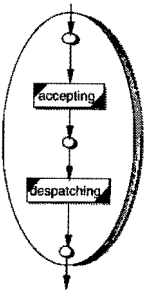
- *discharging*
  Discharging is transferring objects from another larger object. Normally, the larger object is an object that can move by itself.
- *accepting*
  Accepting is the arrival of objects at a certain location. The location may be a warehouse or a transshipping location.
- *storing*
  Storing is putting objects at a certain location. The objects are said to be in stock. When objects are stored, they may be broken down into smaller objects.
- *picking*
  Picking is taking objects out of a certain location. The objects can only be picked if they are stored at the location from which they have to be picked.
- *despatching*
  Despatching is grouping objects and putting those objects at a specific place to have them available as a group of objects.
- *handling*
  Handling is reshaping objects in such a way that the input objects of handling can be obtained again out of the output objects. Examples of handling in external logistics are assembling, packaging, and marking of articles. A large number of synonyms is available for handling, e.g. customization, kitting, packaging, reconditioning, and value-added services.

Please note that we define handling different from the way it is used in literature. Normally, handling is known as those tasks that are performed in a warehouse (Bowersox, 1986). In the example in section 5.2, handling is, for instance, the packaging of a printer, the cables, and the proper manual before despatching.
The generic tasks are the basis for generic composite tasks. These generic composite tasks reflect reality and are not exclusive. Both a definition of each generic composite task and its modelling are presented:

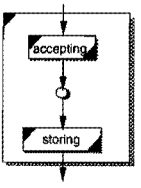
- *transporting*
  Transporting is the physical movement of objects from one location to another in an agreed period of time.



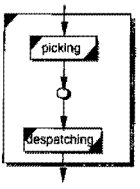
- *transshipping*
  Transshipping is accepting objects at a location and despatching the same objects, in principle, from the same location.



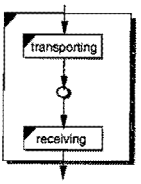
- *receiving*
  Receiving is accepting objects at a given time and storing those objects at a location in a warehouse.



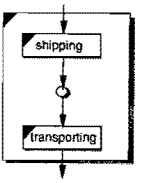
- *shipping*
  Shipping is picking objects and despatching those objects.



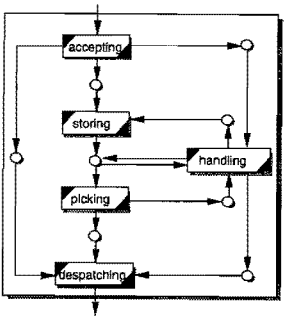
- *supplying*
  Supplying is transporting objects from a location to another location and receiving those objects at that other location.



- *delivering*
  Delivering is shipping objects and transporting these objects to another location.



- *warehousing*
  Warehousing is a specific order of accepting, storing, picking, despatching, and handling. The order is given by a Petri net. We call the place in-between storing and picking a *decoupling point*.



A decoupling point is defined in literature as a location at which the articles or products are not client specific (van Goor, 1989). Using timed, coloured, hierarchical Petri nets, the articles of products still can be client specific in a decoupling point. In principle, with a transshipping task all objects that are accepted at the same time have to depart all at once at a different time. Therefore, transshipping is a specialization of the combination of accepting and despatching. During transshipment, several objects can also be grouped together in a resource like a container (a task type that is generally known as *stuffing*). Resources and their contents can be the output objects of a task. They can be separated from each other by another task (a task that is generally known as *stripping*). Stuffing and stripping are known as the tasks that are performed for grouping objects into a larger object and taking

smaller objects out of a larger one respectively (e.g. the stuffing of pallets into a container and the stripping of that container). We use the generic tasks loading and discharging to represent stuffing and stripping respectively. However, the larger object is not able to move by itself in case of stuffing and stripping.
Up to this part, we have defined the (composite) generic tasks that can be used to structure business processes. We will now give the definition of a consignment as part of the behaviour.

A *consignment* consists of one or more objects that are at the same time offered by one actor to another actor for the execution of a service, and will be together at another time the output objects of that service. Consignments are not known between accepting and despatching.

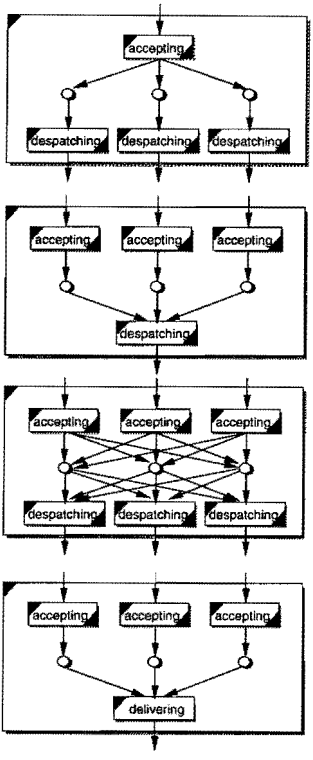
In the example of COSY, several parts are used to make up a PC configuration. These parts are not assembled, but are packed together. This is called consolidation in the literature (Bowersox, 1986). There are other services that can be executed in a warehouse. Using the definition of a consignment, we can define services like breakbulk, consolidation, distribution assortment, in transit mixing, and manufacturing support (Bowersox, 1986, page 239). For visualization, the figures show one place per consignment and one task for consuming or producing a consignment. In practice, one task may of course consume or produce many consignments.

- *breakbulk*
  Breakbulk is accepting objects of an incoming consignment and despatching the same objects in two or more outgoing consignments.

- *consolidation*
  Consolidation is accepting objects of two or more incoming consignments and despatching the same objects in an outgoing consignment.

- *in transit mixing*
  In transit mixing is a combination of breakbulk and consolidation.

- *manufacturing support*
  Manufacturing support is accepting objects of one or more incoming consignments and delivering those objects in a specific sequence to a production unit.



An example of breakbulk is accepting an incoming consignment of meat that is split into several outgoing consignments that have to be delivered to customers.

Using the generic tasks, external logistics can now be modelled by a Petri net (figure 5.4). Figure 5.4 shows a generic model of external logistics. Figure 5.4 does not show that several incoming consignments can be combined into one outgoing consignment, nor that one incoming consignment can be split into several outgoing consignments. In practice this model can be applied by several actors by assigning a value to parameters.

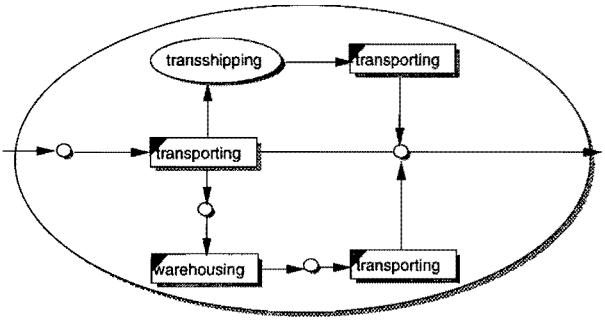


Figure 5.4: A generic model of external logistics

The generic model of the interorganizational business system of external logistics is a composite place, thus showing that external logistics can be viewed upon as a pipeline or a 'logistical channel' according to Bowersox (1986).

Additional to the parameters of a task as specified in chapter 3, a task in external logistics is characterized by the value of the following parameters:

- *minimum and maximum number, weight, and volume*
  These are the minimum and the maximum number, the weight, and the volume of objects of a certain type that can be consumed or produced by a service or a task. The value of these parameters is specific for both the input and the output object types. In case of transport and transshipment, the value of these parameters is identical for the input and the output object types. The cost of a service or a task can be related to the value of these parameters, e.g. parcels below 5 kilogram have another tariff than heavier parcels for transport.
  Only one of these parameters has a value for a specific task, e.g. only the minimum and maximum weight are specified in the example of parcels below 5 kilogram.
- *physical characteristics* and regulations
  An object type may require special handling, for instance, a container is to be stored at a certain temperature or dangerous cargo is not to be grouped with harmless cargo. If an object of the particular type is to be produced or consumed,

the value of temperature or a dangerous cargo identification has to be exchanged between actors.
Regulations in countries will have impact on the mode of transport and/or the time required for transportation. For instance, heavy transport through Switzerland is limited by government regulations. Combinations of modes of transport may also have to be used, e.g. huckepack transport (road vehicle transported by a train).
Other types of regulations are for instance customs regulations and agricultural regulations.

- *cost*
  The cost is the amount of money that is to be paid by a superior for the execution of a task. We will relate the cost only to a task. In practice, the cost can be related to both a task and to object types. However, the costs related to object types are the costs for consuming or producing objects of these types.
  The cost will determine the resources to be used and thus, for instance, the mode of transport to be applied, e.g. air transport will be more costly as compared with sea transport. In relation to cost, the duration of a task is also of importance.
- *season*
  The season of the year may influence the regions to be reached or crossed by a means of transport, e.g. a region with mountains is difficult to cross during winter. Therefore, the execution of a task can depend on the time of the year.

Each task is connected to at least two places: the place of acceptance and the place of delivery. Those are the places at which the objects can be accepted or delivered respectively. A place may be a location, a postal code range (begin and end range), a state, a province, a region, or a country. Synonyms of place of acceptance and place of delivery are the port of loading and the port of discharge respectively, the pick-up place and the delivery place respectively, and the railway station of acceptance and the railway station of delivery respectively.
Delivery conditions like 'ex works' or 'free delivered' determine the role of actors with respect to the interorganizational business system. They are specific to a consignment and are not part of the structure of a business process of an actor.

5.3.3 Objects and object types in external logistics

In practice, the objects can have a variety of appearance in external logistics (see before). Objects are for instance products, packaged cargo, containerized cargo, and bulk cargo. Products can be packaged for transporting and transshipping in for instance boxes, pallets, or containers. Bulk cargo is a product that is not packaged for transporting or transshipping. Examples of products are the printers and the computers of COSY. In general, cargo is defined as products packaged in such a way that they facilitate (physical) transport (Royal Nedlloyd N.V., 1989). In case the packaging material is a container, the cargo is known as containerized cargo, otherwise it is known as packaged cargo or bulk cargo, depending on the characteristics of the products (Royal Nedlloyd N.V., 1989). The distinction between

containerized and packaged cargo is made because the packaging material 'containers' is owned by an actor and has to be tracked by that actor. They are identifiable resources of that actor that can be re-used. Other packaging material like trays and boxes are resources that can be consumed and not reproduced. These resources may not be identifiable separately. Objects have to be transported from one place to another by using a resource that can move autonomously (a definition of such a resource is given in for instance Royal Nedlloyd N.V., 1989, and TDID, 1991). Examples of these resources are a vessel and a truck. We call these resources self-moving units. A self-moving unit can also be loaded onto another self-moving unit, e.g. a truck can be loaded on a train. In the latter case, the truck is the cargo of the train. Depending on the view of an actor, an object is either a resource or part of the cargo.

We define the following object types and objects in external logistics:

- *product type and product*
  A product type is an object type that can be exchanged between a buyer and a seller. A product is an object of a certain product type. A product type is for instance specified by its weight, description, and components. A product type is identified by a product identification (e.g. in case of consumer products an article number and in case of bulbs a combination of 'measure' and 'cultivar'). A specific product is either identifiable or not, e.g. consumer products and bulbs cannot be identified separately, whereas cars as products can be identified by their chassis number (for other purposes, cars can be identified by their licence plate).
- *packaging type and cargo*
  Packaging type is an object type that can be used to facilitate transporting and transshipping of products of a certain type or cargo of a packaging type. Cargo is an object of a packaging type either with or without any contents. Container types and pallet types are examples of packaging types. A specific container or a specific pallet is cargo, although it can also be a product for the actor producing and selling it. A packaging type can be a resource type.
  Packaging types are for instance specified by their size and type (e.g. a Europallet, a Uniform Load Device, and a twenty foot container), their (tare) weight, their maximum gross weight, and their volume. Cargo refers to a packaging type by the size and type. Cargo is specified by for instance the number of objects, their weight, and their volume. A packaging type is uniquely identified by its size and type. Cargo is either identified by the unique identifications of the packaging material (e.g. a container number) or by other readable information like shipping marks.
- *self-moving unit type and self-moving unit*
  A self-moving unit type is an object type that is able to move from one place to another autonomously with or without the use of an engine. A self-moving unit is an object of a self-moving unit type. A self-moving unit is specified by, for instance, a size and type (e.g. a truck, a vessel, and a barge), the mode of transport for which it can be used, the (tare) weight, the maximum gross weight, and the

volume. A self-moving unit type is uniquely identified by its size and type. A self-moving unit is identified by, for instance, a licence place (trucks) or a Radio Call Sign (vessels).

In external logistics we distinguish between the time of acceptance and the time of delivery: the time stamp of the objects at the place of acceptance is known as the *time of acceptance*, the *time of delivery* specifies the availability time of the objects at the place of delivery.
Objects can be packaged into other objects. We have called stuffing a task that can perform packaging (see before). In general, not all objects can be packaged in other objects. The following constraint can be formulated: objects can only be contained in other objects if those objects have a larger volume, have sufficient free volume, and if the total gross weight is less than or equal to the allowed maximum gross weight. Additional constraints relate to, for instance, physical characteristics like the temperature setting (reefer cargo, for which sufficient reefer facilities have to be available) or dangerous types (e.g. chemical products may not be packaged with flammable products). It is beyond the scope of this monograph to list all these characteristics.

5.3.4 Actors

In this monograph, we distinguish only two roles for each actor: superior and subordinate. In external logistics, the actor type is related to the activity to be performed or to the regulations to be taken care of. Examples of actor types are a shipper, a consignee, a carrier, a stevedore, a liner-agent, a forwarder, and a warehouse operator. The actor type is usually specific to a mode of transport, e.g. liner-agent is specific to sea transport. Each of these actor types can be mapped to the two roles that we distinguish.
The transactions between actors can control the execution of tasks or composite tasks, which means that a service can be identical to a (composite) task. For instance, a principal and a warehouse operator can have a transaction for each (composite) task that can be executed by that operator.

5.4 The interorganizational information system in external logistics

We have presented a generic model of external logistics, the parameters that determine the execution of a task, and the object types of external logistics. The parameters are either related to an object (e.g. weight and physical characteristics) or are specific to a country or area (regulations and season). The value of these parameters is exchanged during the behaviour of a business process to select a firing rule of a task. The parameters are modelled as attributes of the association entities that model the input and the output connectors. For instance, the minimum and the maximum weight specify the minimum and the maximum weight of the objects

that can be consumed for an input connector. Because external logistic consists mainly of movement tasks, the value of these parameters is not given at the output connectors.
The constraints of object types (e.g. a smaller object cannot contain a larger object) are also part of the firing rules of a task. The place of acceptance and the place of delivery are the input place and the output place respectively. In the real world, the place of acceptance can for instance differ from the place at which the goods have to be loaded (normally called the pick-up place). The information concerning the difference between the place of acceptance and the pick-up place, and between the place of delivery and the delivery location can be exchanged between actors by exchanging information concerning an activity and one or more actions. The time of acceptance is the availability time for consumption of objects by a task, and the time of delivery is the time of production of objects.

The specification of the data structures of the tokens, the process model of the Procedure Selector, and the steps of procedures as given in the chapters 3 and 4 must be detailed to support external logistics.
The data structure of the tokens is based on the kernel of the business process data structure. The kernel is extended by modelling the object types and the objects of external logistics (figure 5.5).

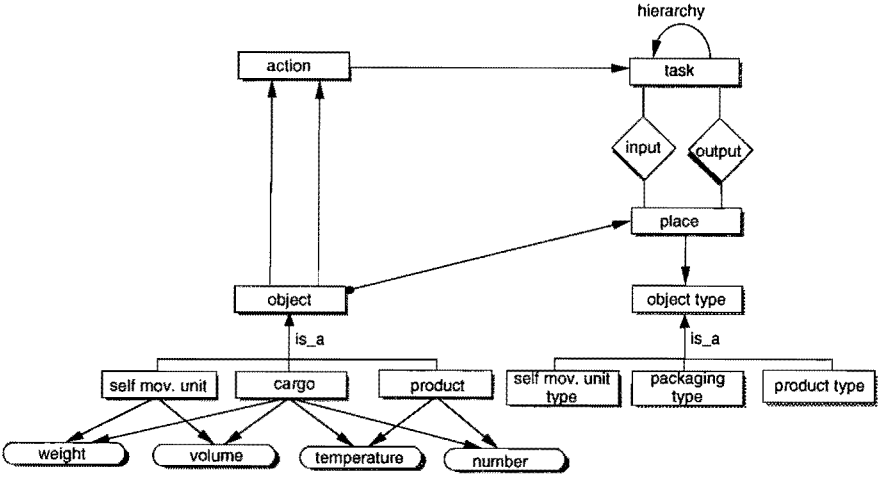


Figure 5.5: Business system data structure of external logistics

As figure 5.5 shows, only the object types and the objects are further specialized. The physical attributes of the objects that are relevant for the selection of a service of the business process are shown by the attributes 'weight', 'volume', 'number', and 'temperature'. To be able to arrange the execution of a service, actors have to

exchange more information. Additions to the business system data structure for arrangement of the execution are for instance:

- *dangerous goods*
  A special object type called 'dangerous goods types' and a special object called 'dangerous goods' can be added as specializations of object type and object respectively. The 'dangerous goods types' object type models the dangerous goods classifications of a product type. The classification is specific to a certain mode of transport (sea, air, rail, and road).
- *customs procedures*
  A special object type called 'commodity type' and a special object called 'customs item' can be added as specializations of object type and object respectively. The 'commodity type' object type models the commodity code of a product type for customs purposes.
- *cost*
  The value of the objects can be of relevance to the selection of a service (see before).
- *resource ownership*
  A relation between an actor and a packaging type can be used to store the ownership relation of a resource.

A product type can have an association with a dangerous goods type and a commodity type. In the interorganizational information system the dangerous goods and the commodity type is related to cargo by the function 'belongs to' with a product.
A resource type and a resource, e.g. a container type and a container, are modelled as an object type and an object respectively.

The data structures of the other tokens in the BTMS and procedures to support external logistics can be derived from the business system data structure of external logistics (chapter 3).

The specification of the processors of the steps in chapter 4 is to be extended to support external logistics. The value of the attributes that are specific to objects in external logistics has to be processed. For instance, one of the conditions to check completeness of an action or an activity is specified as follows:

- $\Sigma_{perf.}$ number = $\Sigma_{req.}$ number
- $\Sigma_{perf.}$ weights = $\Sigma_{req.}$ weights
- $\Sigma_{perf.}$ volumes = $\Sigma_{req.}$ volumes

This condition is part of the pre condition of, for instance, the processors d, e, and f of the confirmation processor (section 4.3.3, figure 4.9). These conditions are depending on the value of other parameters, e.g. when it rains, the weight of sand in an open inland barge increases.

Furthermore, the conditions to select a procedure in the Procedure Selector of the BTMS are extended with the following:

- min. $number_{service} \leq \Sigma$ number of objects related to the activity $\leq$ max. $number_{service}$, and
- min. $weight_{service} \leq \Sigma$ weight of objects related to the activity $\leq$ max. $weight_{service}$, and
- min. $volume_{service} \leq \Sigma$ volume of objects related to the activity $\leq$ max. $volume_{service}$.

5.5 Modelling the example

Section 5.2 gives an example of external logistics. In this example we distinguish between the following actor types: sales units, stock control units, warehouses, carriers, customers, and production units. We will model the supply of parts for PC configurations to a warehouse, the delivery of PC configurations to a customer, and the procedures of the stock control unit for the supply of parts and the delivery of PC configurations.

The business process of COSY can be modelled by the tasks of external logistics. We will model the transport to the warehouse, the acceptance of computer parts by the warehouse operator, the despatch of PC configurations, and their transport to the final destination. The transport to the warehouse is by air transport. The transport to the final destination is by road transport. Figure 5.6 shows the tasks for the supply of parts to a warehouse and the delivery of PC configurations to their final destination. Figure 5.6 does not show that the small parts and the printers can also be used for other purposes than PC configurations. Nor does figure 5.6 show that several warehouses are used by COSY. Furthermore, the transport from the production units to the warehouses and from the warehouses to the customers may be decomposed into several transporting processes that are interconnected through a composite place for transshipping. This is also excluded from figure 5.6.

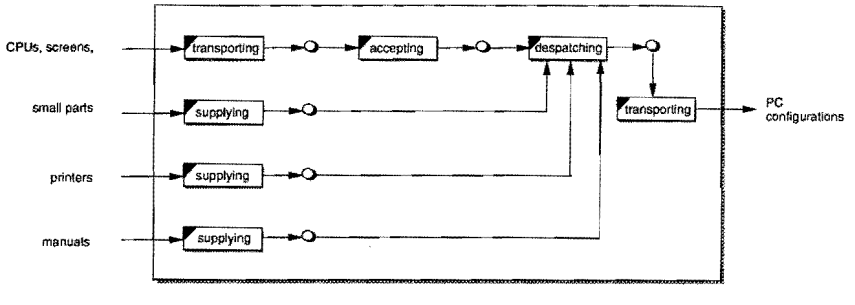


Figure 5.6: External logistics of COSY for delivering PC configurations via a warehouse

In principle, the transporting and accepting of CPUs and screens shown in figure 5.6 can also be combined to supplying. However, we decompose supplying for these products, because we want to illustrate the value of the parameters for transporting and accepting.
In the example of COSY, a PC configuration is for instance a product type and the boxes are the packaging types. Other product types of COSY are printers, small parts, computers, and spare parts. The places out of which the input objects can be consumed by the external logistics of COSY are the production units. For instance, CPUs and screens are consumed from a production unit in North America. The places at which the PC configurations are produced are locations in the EC countries. The internal place, e.g. between transporting and accepting, are places inside the warehouse. The associations that represent the input and the output connectors of the tasks are specified by the objects that can be consumed and produced respectively.
The value of the parameters of the tasks and the input and the output connectors for the supply of parts to the warehouse and the delivery of a certain PC configuration to the final destination is given in table 5.1.

parameters	air transport to warehouse	accepting	despatching	transport to final destination
service id	KL123	1	2	3
type of service	flight KLM number 123	accepting	despatching	24 hours delivery
minimum duration	8 hours		15 minutes	
maximum duration	9 hours	30 minutes	30 minutes	24 hours
consumed object types	ULDs	1 ULD	1 laserprinter, 3 cables, 2 manuals, 1 keyboard, 1 mouse, 1 SVGA screen, 1 CPU (586, 66 MHz, 4Mb, 240 Mb), DOS 6.0, Windows 3.1 (all in boxes)	pallets, PC configurations
produced object types	ULDs	10 SVGA screens, 1 empty ULD	1 PC configuration type 20	pallets
min. weight consumed/ produced				25 kg
max. weight consumed/ produced				50 kg

Table 5.1: The parameter values of the tasks and the connectors for COSY to deliver PC configurations

The procedure of the stock control unit of COSY to support the delivery of a certain PC configuration in the EC is shown in figure 5.7. The procedure to support the other tasks of COSY and the other actors involved can also be specified.

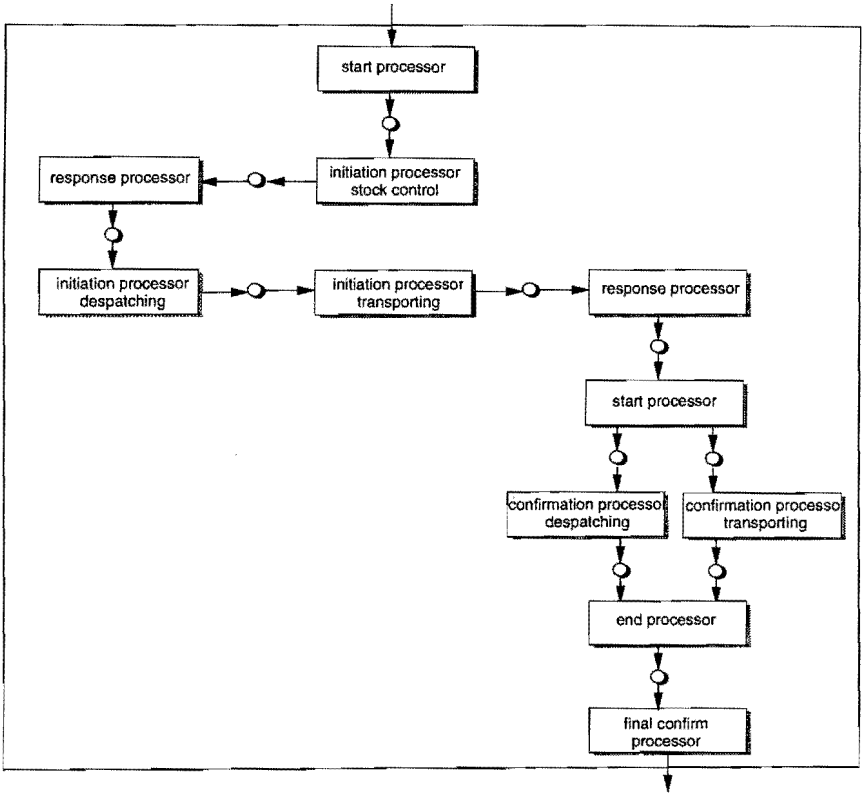


Figure 5.7: Procedure to deliver PC configurations

The 'initiation stock control' processor is to bind sufficient objects to constitute the required PC configuration. If there are sufficient objects, a response is given to the sales unit and the despatch and transportation task can be initiated. Once the responses of despatching and transporting are received, a response can be given again to the sales unit. The contents of the messages and the business operation transactions can be derived from the value of the object types and the characteristics of the tasks and the service.
If there are no sufficient resources that can be bound by the initiation stock control processor, the production is to be initiated and sufficient parts have to be supplied to the warehouse (figure 5.1). This part of the procedure is not shown in figure 5.8.

5.6 Application in general

In this chapter, we have applied the generic concepts to external logistics by specifying the business system in more detail. We have shown the application of

this generic model of external logistics to a part of the business process of COSY. In this section, we will discuss the following:

- the application of the concepts to different business systems (subsection 5.6.1);
- the application of the concepts to different business processes in a business system (subsection 5.6.2).

5.6.1 Application to business systems

We can learn from the previous sections that the application of the concepts to external logistics requires modelling by the following components of the business system of external logistics:

- tasks that can be executed in external logistics (e.g. loading, discharging, and accepting);
- places for the physical locations. Places can have different names, e.g. an input place can be called a place of acceptance and an output place a place of delivery;
- object types that can be consumed and produced by the tasks;
- parameters that determine the firing rules of the tasks.

The firing rules that we have modelled in external logistics are the consumption of input objects and the production of the same objects as output objects. Therefore, the parameters that specify the firing rules are defining the object types that can be consumed and produced. In other business systems, the relation between the input and the output object types can be more complex, e.g. several steel plates, an engine, and other object types are used to produce a car.

These rules can also be applied to other business systems. For instance, in health care a task can be a cure, an object type can be a patient, and a place a stage in the recovery of a patient. The parameters that determine the firing rules of a cure, are specific to that cure.

Another example is the application to finance. A task in finance is, for instance, payment, a place is a bank account number, and money is an object type. The parameters that determine the firing rules are for instance the minimum number of objects that is allowed at a certain bank account number and the interest that is paid by the bank or the customer that uses the bank account number.

In a similar way, other business systems can be specified. By specifying those business systems, we also specify the steps that can be used to construct procedures and the message data structure that is used to exchange information between actors. We can conclude that the concepts defined in this monograph can be applied to different interorganizational systems.

5.6.2 Application to business processes

Within a given business system, each actor has engineered or is engineering its business process. This includes the determination of the sequence in which the messages can be exchanged for the supply of parts to a warehouse and the delivery of PC configurations to a customer.

The business process and the scenario as specified for COSY can serve as an example to modelling the processes of other actors. In fact, certain business processes that serve as a reference to business process (re-)engineering are given in literature (e.g. lean production as for example specified by Womack et al., 1991). These business processes can be applied by many actors. They have to be made specific to the business process of an actor. Examples of such business processes are, for instance, the following in logistics (Bertrand, 1990):

- *assemble-to-order*
  Assemble-to-order (figure 5.8) usually refers to the situation where one or more product families (a product is identical to an object) exist with a considerable variety of final products in each family. The customer is offered a variety of choices leading to a unique identification of a desired object which is to be assembled from standard parts. The standard parts may be available in stock or have to be supplied by an actor. If they are available in stock, an internal procedure may have to be started by the seller to keep the stock level above a minimum. If there is no stock of parts, the procedure for processing a customer order includes the supply of parts.

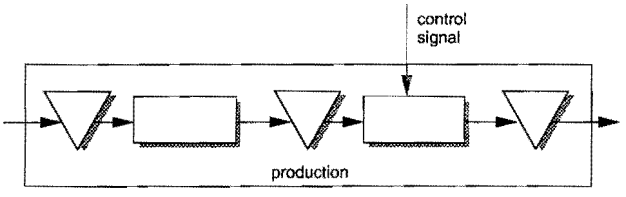


Figure 5.8: An example of a business system modelling assemble-to-order

Figure 5.8 shows that the information system of the superior has to produce the control signal of the last process of its business process. This control signal is produced on using information exchanged between a buyer and a supplier.

- *capacity selling* (make-to-order):
  In the make-to-order situation (figure 5.9), the product families and the products have to be produced completely by a supplier on the basis of the incoming parts of that supplier. Production and assembly is triggered by an order of a customer. In fact, the supplier sells capacity. External logistics is an example of capacity selling.

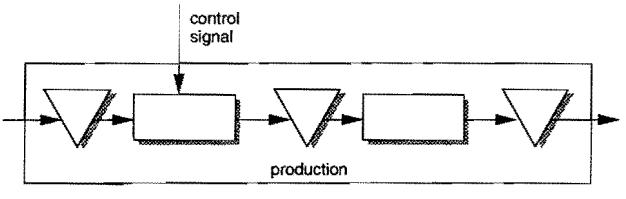


Figure 5.9: An example of a business system modelling make-to-order

Figure 5.9 shows that information is exchanged between a customer and a seller to produce the control signal of the first process of the production process of the supplier.

- *engineer-to-order:*
  In engineer-to-order, the production process (i.e. the tasks and their firing rules) are known. However, the service that can be performed on the basis of the production process of the seller is not yet known. As part of engineer-to-order, the product types that are produced by a supplier have to be engineered on the basis of the existing production process. The rules presented in the GARM (General Architecture, Engineering, and Construction Reference Model) can be applied to specify the characteristics of the product types. As part of the request of a customer, the functional specification can be given; as part of the response or confirm, a technical solution is presented. The technical solution can be confirmed by the customer. Nieling (1988) describes the rules for the GARM, whereas Nederveen (1991) presents the application of these rules to compose a data model for building purposes.

Assemble-to-order and make-to-order are modelled by a technique that can be mapped to timed, coloured, hierarchical Petri nets (annex 1). The distinction between assemble-to-order and make-to-order is vague. In practice, assemble-to-order consists of one process that produces the required products, whereas make-to-order consists of two or more processes (figure 5.8 and 5.9).
The production process of engineer-to-order is similar to the production process shown for make-to-order. Each of these examples can be supported by a standard scenario and standard procedures. The message types can be specific to a scenario, however, they can be mapped to the message types that we have defined in chapter 2. For instance, a box list that is exchanged between a production unit and a stock control unit in the example of COSY (see section 5.2) is a confirmation of delivery. Thus the reference business processes, the scenarios, the message types, and the procedures can be mapped to our concepts. Such a reference situation has to be made specific to certain actors, which means that the business system, the scenarios, the message types, and the procedures have to be made specific.
Both the generic and specific scenarios can be visualized by a tree. The root of the tree is the superior that has transactions with several subordinates. Each node is an actor that is both subordinate and superior and the leafs are subordinates. We will call such a tree a transaction tree. Two transaction trees can for instance be given for the example of supply and delivery of COSY.
In a similar way, reference business processes and scenarios can be specified for other business systems.

We can conclude that given a (reference) business process, the procedures, the steps, and the BTMS can be used to support these business processes. Once a business process is engineered, our concepts can be applied to the information system.

6 Other approaches

6.1 Introduction

In the previous chapters of this monograph, we have presented a conceptual approach of interorganizational systems and transaction management, and have specified a component of the information system of an actor that manages transactions. Furthermore, we have given an instance of the concepts, the steps, and the BTMS for external logistics.

The basic concept of the BTMS is a 'procedure'. Automation of procedures is the primary objective of Workflow Management Systems (WFMS). The business opportunities of EDI and WFMS seem to be similar. They can be found in, for instance, the EDI literature. We will first discuss the business opportunities of EDI (section 6.2). Secondly, we take a closer look at the underlying concepts of interorganizational systems, business process and transaction engineering, and Workflow Management Systems (section 6.3). Finally, technical aspects of distributed transaction processing and EDI are discussed (section 6.4).

6.2 Business opportunities

"Levi Strauss & Company is a $3 billion manufacturer of apparel, with headquarters in San Francisco, California. It is best known for its denim trouser, or blue jeans, which it has made for over 100 years. Levi Strauss sells its clothing products to approximately 17,000 retailers operating over 200,000 stores. In 1986, Levi Strauss recognized that their business was serving the retailer, and that service could be improved. They found that rapid, accurate replenishment of merchandise was not happening, and that relationships were sometimes termed adversarial. So Levi Strauss, using available computer technology, developed what is named a quick response program to maintain a competitive edge, improve the way it does business, and develop better trading relationships."

The I/S Analyzer of August 1989 starts with this example of applying communication between different organizations. The article mentions that the lead time of products of Levi Strauss has been cut from 9 days to 3 to 5 days after having automated the ordering and the shipment. Inventories are replenished almost daily. Levi Strauss is able to maintain lower inventories, while avoiding that popular products are out of stock. Other examples of the advantages of applying interor-

ganizational communication can be found in, for instance, van der Vlist et al. (EDI in trade and EDI in industry).
Technically, this communication concerns the computer-to-computer exchange of structured data, normally known as EDI (Gifkins et al., 1988, Sokol, 1989, Stone, 1988, and Hofman, 1989).

The most important business opportunity of interorganizational systems lies in the improvement of trading partner relations (Sokol, 1989). The process of structuring the information and reaching agreement on the semantics of the information requires much co-operation between trading partners. Sokol mentions that in order to reach agreement between parties with respect to providing accurate information, information processing, and acting on data received, it is important to agree on business procedures, data requirements and usage, communication methods, and testing schedules.

Another opportunity mentioned by for instance Sokol is that receiving more accurate and complete information will eliminate key-entry errors. Suppliers can be assured of making more accurate and timely shipments to their customers, as well as eliminating the need for charges associated with return shipments of rejected products and reducing emergency orders.
Furthermore, Sokol divides the business opportunities of EDI into opportunities that are a direct result of implementing interorganizational systems, and opportunities that are derived from making effective use of interorganizational systems. According to Sokol, the business opportunities of interorganizational systems are the following:

- reduced order cost associated with handling of business transactions: key entry of information and errors due to rekeying may be eliminated;
- reduced cost for materials and services to support paper transactions;
- reduced order-pay-cycle period: the time needed to process an order will be reduced, the buying company may order new products later and can reduce its stock;
- improved planning and forecasting.

Other authors (Gifkins et al., 1988) describe the early benefits of interorganizational systems as being efficiency benefits. The benefits are primarily cost savings associated with automating highly repetitive tasks. However, some authors mention that interorganizational systems should be of strategic importance to organizations (Stone, 1988). Stone also notes that few interorganizational system's efforts today have strategic focus; most organizations approach interorganizational systems tactically.
Another reason for introducing interorganizational systems of which the potential benefits are doubtful (EDP Analyzer, 1987), is being forced to introduce these systems. For example, in 1984, General Motors sends a letter to most of its suppliers, urging them to do business with GM via computer communication: 'No EDI, no business'. Forcing trading partners is, in general, only possible for companies with a large market share. Companies can also be forced to use

interorganizational systems if their competitors, having enough market share, decide to use interorganizational systems.

Other potential business opportunities can be as follows (Hofman, 1989):

- the acquisition of new markets, the retention or enlargement of existing market shares;
- offering a better customer service by decreasing the delivery time of products. This can also reduce the stock if supplier and buyer mutually adjust their stocks;
- improvement of the customer service by reducing errors in handling the information and by direct access to the (status of the processing of the) information. For example, an intermediary, which provides a service for tax declaration, can improve its customer service by applying external communication with the Tax Department;
- faster delivery of goods or services can lead to more timely invoicing and payment by customers;
- faster delivery can be achieved because errors in rekeying or processing the information are reduced. Delays and cost caused by handling those errors can thus be avoided;
- misunderstandings concerning the meaning (the semantics) of information will be avoided, because the structure and the meaning of the information is known to both the sender and the recipient of the information;
- availability and processing of information will be improved;
- the handling of information internally in a company (or process) can be improved, because the information is received more accurately.

Strategic business opportunities can be found in larger market shares, reduced stock, improved planning, forecasting, and improved trading partner relations. Ideally, an organization should strive for an enterprise-wide interorganizational system to support flows that are required for the organizational processes. The information systems department (Porter, 1985) can be seen as the department which is able to see the link between various departmental systems accounting, engineering, manufacturing, and so forth and is thus well positioned to develop a useful strategy with respect to interorganizational systems (I/S Analyzer, 1989). Companies like Levi Strauss created a new department, EDI Services, positioned between the information systems department and the user organizations. Such a department can be charged with providing strategic directions for interorganizational systems.

In general, it can be said that using interorganizational systems helps to improve 'customer service' (Stone, 1988). Customers can reduce their inventory position and shipments can be delivered faster. Highly automated computer communication systems have high fixed cost and relatively low variable cost. They can provide competitive advantage to those businesses having such systems. Opportunities as mentioned by for instance Sokol will be common to all organizations once standard software components like the BTMS are available. With the aid of interorganizational systems, business transactions that are processed according to procedures

can be handled automatically by means of the BTMS. For example, a payment can be made without reception of an invoice. Efficiency can be gained by automating highly repetitive tasks and making information available to other departmental systems in an organization. The contribution of employees can be shifted from administrative to more value-adding services.
Increase of productivity and enhancement of customer service in information processing companies is seen as the most important advantages of WFMS (see for instance documentation on existing WFMS products).

6.3 Business process related concepts in literature

In this section, we will relate the concepts introduced in this monograph to the concepts of interorganizational systems, business process engineering, and Workflow Management. First, theory on interorganizational systems is discussed and secondly, the underlying concepts of business process engineering are described. Finally, Workflow Management concepts are discussed.

6.3.1 Interorganizational systems

A number of definitions of interorganizational systems is given in other publications, such as:

- Barret et al.(1982) and Cash et al.(1985):
  "In the broadest term, an interorganizational system consists of a computer and a communication infrastructure that permits the sharing of an application, such as programs for making reservations or for ordering supplies. The players in a system are either participants or facilitators. An interorganizational system participant is an organization that develops, operates, or uses an interorganizational system to exchange information that supports a primary business process. Participants can be competitors, organizations in the buyer-supplier chain, or a combination of these. An interorganizational systems facilitator is an organization that aids in the development, operation, or use of a network for the exchange of information among participants. The supporting products or services are a part of the primary business of the facilitator."
- Suomi (1989):
  "..a system in which two or more independently managed organizations communicate in a computer memory-to-memory fashion without the transfer of physical media."
- Wierda (1991):
  "Interorganizational information systems are information systems that are jointly developed, operated, and/or used by two or more organizations that have no joint executive."

Barret et al. and Cash et al. identify on the one hand a computer and a communication infrastructure, and on the other hand participants and facilitators. They even

describe the sharing of applications, thus implying that one application is central to the actors. Furthermore, Barret et al. and Cash et al. do not say anything about the boundaries between the primary businesses of the participants and the facilitators. They only state that the products or services used to support the development, operation, or use of the interorganizational system are part of the primary business of the facilitators. Therefore, the primary business of the facilitators differs from the primary business of the participants.
The components that are distinguished can be provided by separate facilitators, e.g. an actor may operate a network, whereas another actor provides the conversion software. To apply the separate components in an interorganizational system, it is either required to have standard interfaces between the different components or to have one actor monitoring the development, the realization, and the implementation of the components. Actors that have a multi-national nature like IBM can be capable of performing such a monitoring role in their interorganizational system. Other actors like banks can have outsourced such a role to a third party. The third party performs the monitoring on behalf of these other actors. To create an open environment where every actor can communicate with another actor using components provided by different actors, standardization of the protocols and the interfaces between those components is required.
The definition of Suomi is a technical definition, defining an interorganizational system as the communication between computer memories. In this definition, the physical aspect of an interorganizational system is emphazised, whereas the conceptual aspect is lacking in the definition. However, Suomi does not say anything about the use of software and computers in an organization.
The definition of Wierda focuses on the absence of 'joint executives' of the organizations involved, which is the same as the 'independently managed organizations' of Suomi. Therefore, this definition is similar to the other definitions.

With respect to interorganizational systems, the lack of an accepted or validated theoretical framework or structure for applying the concepts of interorganizational systems is stressed by King et al. (1989) and Wierda (1991). The definitions of interorganizational systems only tell us about the components of an interorganizational system. However, they do not teach us the protocols and the interfaces between these components. The protocols and the interfaces specify the function of each component, they do not describe other aspects like the organizational, the cultural, or the social aspects of interorganizational systems.
We have given a theoretical framework of the technical aspects of interorganizational systems by conceptually modelling business processes and specifying the BTMS. A distinction between actors using and providing the BTMS, and the network by which the BTMS implementations can communicate, is as mentioned earlier outside the scope of this monograph.

6.3.2 Business process and transaction engineering

We investigate the literature of business process and transaction engineering, because it may offer concepts of business processes and activities. Business process engineering is discussed by various authors like Davenport (1993) and Hammer (1993). Creemers (1993) looks at linking business processes of different organizations by defining transaction and he defines transaction engineering instead of business process engineering.

Davenport (1993) gives the following definition of a business process: a business process is a specific ordering of work activities across time and place, with a beginning, an end, and clearly identified inputs and outputs. According to Davenport, a business process is a structure for action. Davenport distinguishes between the following aspects:

- the responsibility and the reporting relationships (hierarchy), and a dynamic view on how the organization delivers value (vertical structure);
- a process approach focuses on how work is done instead of focusing on which specific products or services are delivered to customers. Successful organizations must do both;
- product versus process innovation should occur;
- a process approach implies adopting the customer's point of view. Processes are the structure by which an organization does what is necessary to produce value for its customer;
- a clearly defined owner must be identified to be responsible for design and execution and for ensuring that customer needs are met (process ownership);
- there are cross-functional processes e.g. Research and Development, marketing, and manufacturing.

Hammer (1993) gives another definition of a business process: "a business process is a collection of activities that takes one or more kinds of inputs and creates an output that is of value to the customer".

Creemers (1993) combines the value chain approach of Porter (1985) and transaction cost economics (Williamson, 1975) as a basis for business process design. He identifies three chains within one actor:

- a direct chain representing transaction-oriented ordering;
- two side chains, one for planning and another for collecting. These are the group-oriented actions.

Furthermore, he identifies four points where the different types of chains interfere. Two of those points have to do with the connection and the disconnection of the planning and the collection side chain and are named (Group Connection and Group Disconnection Point respectively). The other two points have to do with communication between chains and are the Interaction Connection and the Interaction Disconnection Point.

A number of direct chains of several actors in cascade supports client specific actions that provides input to a planning side chain. The input is either physical or

is abstract. Creemers shows examples of these cascades in which the actor that executes the actions of the planning chain, triggers the first direct chain in the cascade. It seems as if there is no communication between the actors executing the direct chains or between the actor executing the side chain and other actors than the one executing the first direct chain.
The basic concept of transaction engineering is the 'transaction' itself. A transaction is defined in two domains:

- *the governance domain*
  A transaction is an agreement between parties concerning the transfer of goods or the delivery of services. The structural aspect is the governance structure (contract organization) and the process aspect is the actual observance of the agreement;
- *the business administration domain*
  A transaction is a sequence of client-specific actions. The structural aspect is the description of the actions that have to be taken, and the process aspect is the performance of these actions.

These two definitions use amongst others the following concepts:

- a *service* is an effort, a sequence of actions, to the delivery of which a person or organization(al unit) can commit him/her/itself;
- an *action* is a predetermined task, that contributes to the production and transfer of a good or the delivery of a service;
- a *business process* is an ordering of actions to achieve a defined result.

Whereas Davenport and Hammer focus on an approach to business process engineering, Creemers gives a number of concepts before giving an approach to transaction engineering. The definitions of 'business process' given by these three authors are a mixture of structure and behaviour.
Using the definition of 'business process' given by Davenport implies that a process can only be executed once with specific inputs and producing specific outputs. Davenport makes a distinction between a hierarchy of responsibilities and a vertical structure of the business system. In this monograph the hierarchy is supported by the procedures that can be processed by the BTMS. The vertical structure of the business system is not clearly specified by Davenport. It can be seen as the service that is added by an actor to the underlying tasks. We explicitly specify the process ownership by the actor that is able to execute a service.
Mapping the definition of 'business process' given by Hammer to our concepts, we come to the following conclusion: one or more object types are consumed by a collection of actions to produce a specific object that has value to the customer. As we can see, the definition is a mixture of structure and the behaviour. Whereas Davenport defines a beginning and an end of a business process, this is not mentioned by Hammer. Hammer, however, explicitly adds the value of the output object to a customer, whereas this value is mentioned by Davenport only as an aspect.

A business process as defined by Creemers can be decomposed in actions. Such a business process achieves a defined outcome. Creemers does not specify a 'defined outcome'. It seems as if the outcome is achieved only once (behaviour). Implicitly, the outcome is 'the production and the transfer of a good or the delivery of a service'. The definition of 'action' given by Creemers contains both structural and behavioural elements.
Similar to these definitions of 'business process', one can say that a service is also a behavioural element. However, a service can be used more than once, thus it becomes a structural element.

In addition to Davenport (1993), Hammer (1993), and Creemers (1993) we distinguish between the information system and the business system. This distinction allows us to define clear concepts of interorganizational information systems. These concepts reflect the interorganizational business system that is to be controlled. Once business process re-engineering or transaction engineering has been completed, our concepts can be applied.
Let us give an example. For instance, Creemers distinguishes between two interaction points. The Interaction Connection Point as defined by Creemers can be identical to the control signal of a business process or a task as defined in this monograph. The Interaction Disconnection Point can be identical to the report signal. In our approach, several messages can be exchanged to control the execution of business processes or tasks and the control and the report signal are internal to an actor. After transaction engineering has been completed, the interaction points can be replaced with these internal signals. Furthermore, the new structure of the business system is supported by new procedures that can be entered in the BTMS. If the business system that is re-engineered, is the information system, the actions performed by persons are possibly reduced to specifying the procedures and executing those tasks that cannot be supported by procedures of the BTMS. The latter is exactly what happened in the IBM Credit example given by Hammer.

We have discussed some of the definitions given by Creemers already. Creemers gives two definitions of a transaction that contain structural and process aspects. The structural aspect of the governance domain seems identical to the contract and the planning protocol. In this monograph, we have not specified a process aspect for observing the behaviour of the agreement. The structural and the process aspect of the business administration domain are identical to our concepts 'task' and 'action' respectively. Regarding transactions, we make a distinction in 'protocol' (structure) and 'transaction' (behaviour). Furthermore, we define the concept 'transaction' in the interorganizational information system, whereas Creemers defines it as part of the interorganizational business system.
As we can learn from literature, different aspects of the same problem are dealt with. There are similarities between the approaches, but also differences. Differences can occur if one wants to describe aspects of the current situation that cannot be changed easily. Such aspects are for instance organizational structures or existing information systems. In the latter case, one has to convert between, for

instance, the software that can be developed using our concepts and the existing information system. However, these differences can give problems when one wants to develop new software that is to manage business processes. In such a case, clear, unambiguous definitions are required. Because communication between these systems is becoming of great importance, our concepts can be used to come to flexible communicating information systems.

6.3.3 Workflow Management

Within document processing environments like the inland revenue and insurance companies, the control of the flow of documents is becoming a major issue. If an actor is required to provide the status to the customer of its document, that actor has to have possibilities to track the document. Workflow Systems provide such insight. Workflow Systems can give advantages in environments, where the customers are private persons. These private persons are mostly not automated and are, therefore, not capable of giving structured information. However, one can also see that for instance insurance companies can receive structured information if they work with agents. Also the inland revenue is capable of getting the information in a structured way by providing low cost software to private person.

According to Ellis and Nutt (1992), Workflow Systems are intended to assist groups of people in executing work procedures. They have defined the following concepts:

- *procedure*
  A procedure is a set of work steps and a partial ordering of these steps. A step consists of a header and a body.
- *activity*
  An activity is the body of a work step. An activity is either an elementary activity or a compound activity. A compound activity contains another procedure.
- *workflow system*
  A workflow system contains a computerized representation of the structure of procedures and activities.
- *script*
  A script is a specification of a procedure, an activity, or an automatic part of a manual activity. The composition or building of this script from available building blocks is called scripting.
- *job*
  A job is the locus of control for a particular execution of a procedure. In some contexts, the job is called a work case. If a procedure is modelled by a Petri net, then a job can be considered as a token flowing through the net.

Others define the following concepts (van der Aalst et al., 1993):

- *task*
  A task is a piece of work to be done by one or more resources in a pre-determined time interval. A task is atomic, which means that it cannot be split into smaller tasks.

- *procedure*
  A procedure is a (partially) ordered set of control activities, pairs of tasks and sets of resource classes, and (sub)procedures. A control activity specifies the routing of the work within the procedure and the synchronization of tasks.
- *job*
  A job is a process modelling the execution of an amount of work according to a given procedure.
- *workflow*
  Workflow is a partially ordered set of jobs. A Workflow Management System is a computer system (or a software package) that manages workflows.

As we can see from the definitions, there is a large similarity with our concepts. A WFMS as defined by Ellis and Nutt (1992) and Van der Aalst et al. (1993) can be considered a specialization of our BTMS for two reasons. First of all, the objects are documents. In existing applications of WFMS, the concept of a dossier is added: a *dossier* is a number of related documents. A dossier is identified by for instance a dossier number and the identification of the initiator of the first document of the dossier. Each document is identified with a dossier number, an initiator identification, and a document type. We have made a distinction between dossier and document, thus allowing several documents of the same type in one dossier. In our concepts, a dossier is similar to an incoming transaction and is an implementation of a job.
Secondly, the tasks that are to be executed are part of the procedures contained in a WFMS in van der Aalst. Therefore, there is a difference with the way we treat tasks. In van der Aalst (1993), the concept of the control of actions and the concept action are part of a procedure in a WFMS. Ellis and Nutt (1992) did not treat the concept of a task separately.

6.4 Technical related concepts in literature

Technical aspects of interorganizational systems refer to the concepts that are used to develop, realize, implement, and operate the software components. There are different approaches to the communication between computer systems:

- distributed databases: a number of databases that communicate with each other are viewed as one large distributed database.
- international standardization of messages (UNSMs: United Nations Standard Messages). The relation between the conceptual approach and the current and future message development is investigated and existing UNSMs are related to the conceptual approach.

6.4.1 Distributed databases

We have introduced the concept of a business transaction. In this section, we will compare this with a database transaction. Definitions of a database transaction

mention a unit of work, e.g. Date (1983) specifies a database transaction as"a unit of work", the ISO (ISO/DP 10026-1,2,3, 1989) specifies a transaction as "a unit of work characterized by four properties: atomicity, consistency, isolation, and durability", and Encarnacao et al. (1987) specify a database transaction as a "comprehensive unit of work characterized by the properties consistency, atomicity, persistency, and separability". A database transaction is further specified in Date (1983) as consisting of the execution of an application-specified sequence of operations, beginning with a special 'begin transaction' and ending with either a 'commit' operation or a 'rollback' operation. Each separate database transaction can be specified by a standard sequence of operations. When a database transaction is implemented, the standard sequence of operations can be made application specific, whereas each transaction may involve the execution of only a limited set of the chosen operations. If this is the case, various options have to be specified to reach either a successful ('commit') or unsuccessful ('rollback') ending of a database transaction.
Özsu et al. (1991) present an overview of definitions of a transaction (page 259). Intuitively, they claim that a transaction takes a database, performs an action on it, and generates a new version of the database. They maintain that a transaction is made up of a sequence of read and write operations on the database, together with computation steps. Their formal definition of a transaction is a partial ordering over its operations together with a termination condition (the operations are either read or write).

Özsu et al. define three dimensions for specifying different types of transactions: application areas, duration, and structure. With respect to the application area they make a distinction between local and distributed data. With respect to duration they distinguish between conversational, short-life, and long-life transactions. We must note that conversational transactions tell us something about the actor performing the database transaction, whereas short-life and long-life transactions can be performed by a person or a software program. With respect to the structure, they make a distinction between a flat and a nested transaction.

Based on these concepts, both Özsu et al. and Encarnacao et al. present an architecture for distributed databases. The architecture of Özsu et al. is shown in figure 6.1 (next page).

The Transaction Manager is responsible for co-ordinating the execution of database operations on behalf of the application, and the scheduler is responsible for the implementation of a specific concurrency control algorithm. A third part, which is not shown, is the local recovery manager that exists at each site. As the figure shows, communication between Transaction Managers is not allowed. This is based on the concepts of concurrency control. They can be based on locking, on time stamp ordering, or a combination of both. The locking mechanisms are as follows:

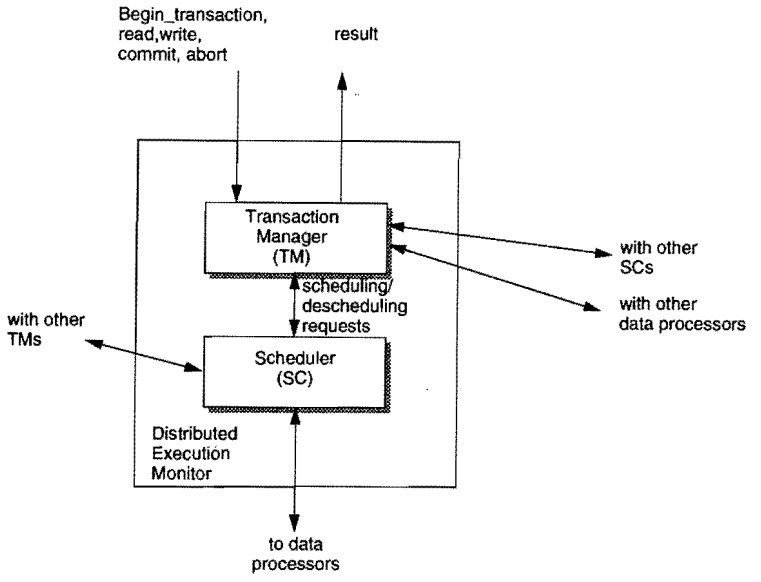


Figure 6.1: Architecture of distributed databases (Özsu et al.)

- centralized locking: one of the sites is the primary site where tables are locked for the entire distributed database. One of the Transaction Managers is a co-ordinating TM;
- primary copy locking: if a lock unit is replicated at several sites, one of the sites is selected as the primary site. All database transactions that desire to access that lock unit obtain a lock unit at the primary site. The primary site is the co-ordinating TM for a specific lock unit;
- decentralized locking: the lock management duty is shared by all the sites of the distributed database. Each site is the co-ordinating TM for its lock units.

In locking-based concurrency control, the scheduler is a lock manager. Özsu et al. have extended the locking to two phases (two-phase locking or 2PL) to allow the serialization of transactions.

Let us discuss these concepts briefly. The unit of work of a business transaction is an action from the view of a superior. That action is supported by a procedure of the subordinate of the business transaction. The concept of a 'database operations' is identical to the concepts of a 'step'. A Transaction Manager as specified by Özsu has similar functionality as the BTMS and the Scheduler can be seen as a special type of an in-house processor (e.g. a Resource Manager defined by van der Aalst et al., 1993). Özsu specifies different ways of locking. In our case, locking is identical to binding resources. One of Özsus locking mechanisms is called two phase locking. This concept is in our description two phase binding of resources to support the action of a business transaction in an incidental relation. The first phase is the execution of the contract protocol and the second phase is the execution of

the execution protocol. It seems as if the concepts of distributed databases can be considered as a specialization of the concepts specified in this monograph: resources in distributed databases are tables and in business processes they are physical or abstract objects. Steps in a procedure are similar to database operations of a standard sequence of operations.

6.4.2 Available messages

We have presented the kernel of the business operation message data structure in chapter 3. Amongst others, messages to support trade and transport have already been standardized internationally. These messages are documented in the EDIFACT syntax (EDI For Administration Commerce and Transport, see ISO 9735, 1988). This syntax specifies the structure in which information can be exchanged. It consists of so-called steering and control segments that are generic to all messages, and segments that have a meaning per message. A new version of the syntax makes a distinction between batch- and interactive EDI (ISO CD 9735:1993(E), 1993). We will first map the message attributes specified in this monograph to EDIFACT, and secondly discuss the structure of the trade and transport messages.

The relation between the message attributes and the EDIFACT elements is given in table 6.1.

message attribute	segment	data element
transaction id.	message header	common access reference
message id.	message header	message reference
sender	interchange header	sender
recipient	interchange header	recipient
message type	message header	message type
role	–	–
my sequence	message header	sequence of transfer
your sequence	–	–

Table 6.1: Message attributes in EDIFACT (–: no element available)

The attributes 'role' and 'your sequence' that are required to exchange the initiator of a transactions and the request sequence number respectively, cannot be mapped to existing data elements of the EDIFACT service segments. They have to be exchanged by segments between a message header and a message trailer segment.

An interchange is a set of messages from one sender to one recipient. Therefore, the sender and recipient attributes can be mapped to the interchange steering segment (interchange header). The message types that we have specified have to be mapped to existing values of the data element 'message type'. Besides these elements, the steering segments of batch EDI contain additional data elements, e.g. data elements specifying the syntax version and a test indicator. The steering segments of interactive EDI (ISO CD 9735:1993(E), 1993) contain a dialogue reference, a dialogue identifier, a transaction reference that is identical to our attribute 'transaction id', and a scenario identifier. We do not have a requirement to use the data elements for dialogues and scenarios (section 6.4.3).

The following table shows a number of the standard trade messages:

message group	standard messages	message types
catalogue	price/sales catalogue	contract request
order messages	purchase order	execution request
	order change	execution request
	order response	execution response
delivery messages	despatch service	execution confirm
	delivery just in time	execution request
	delivery schedule	planning request

Table 6.2: Trade messages

The order messages can be used after the price/sales catalogue has been exchanged. Actors do not have to conclude a contract as long as they adhere to the standard conditions exchanged by a catalogue. The despatch advice can be given as a response to the order messages, whereas the other delivery messages are requests of the superior.

The basic structure of the order messages contains the following elements:
- dates and times, e.g. despatch and delivery dates and times;
- locations at which the products are to be delivered or are going to be delivered;
- means of transport to be used;
- packaging of the products;
- equipment that is used as packaging;
- the products that are ordered, with for each product the possibility to specify dates and times, locations, packaging, means of transport, and equipment.

Equipment, products, packaging, and means of transport are object types. The functions between these object types are implicitly specified in the message by the nesting of the EDIFACT representation of the object types. The function between

packages and equipment, between packages and means of transport, and between equipment and means of transport cannot be exchanged by the order messages.
The despatch and the delivery locations and the times can also be given for each product or for all products Thus an actor has the possibility either to group the products per location and time, or give for each product the locations and the times.
The basic structure of the price/sales catalogue message is similar to the structure of the order messages. All details can now be given for a complete pricing group or for an individual pricing item of a pricing group. Also, the basic structure of the despatch advice is similar, giving the ability to exchange equipment details and a consignment packing sequence. The relation between the consignment packing sequence and the equipment is implicitly given by the message structure. Within a consignment packing sequence, packaging and product information can be exchanged. Per product, packaging information can be given. The consignment packing sequence is used to distinguish between hierarchical packaging levels (e.g. a pallet with boxes containing printers).
The just-in-time delivery message gives the ability to specify a sequence in which products have to be available at a certain time at a location. The delivery schedule message allows a sender to specify either per location the delivery of products, or per product the locations and times at which they have to be delivered.

The following table shows a number of transport messages that have been standardized:

message group	standard messages	message types
single consignment	provisional booking	contract request
	firm booking	contract request
	booking confirm	contract response/confirm
	instruction	execution request
	contract status	execution response
	arrival notice	execution request
	schedule and availability	planning request
multi-consignment	multi-consignment	execution request
	multi-modal status report	execution confirm
resource related messages	bayplan occupied and empty	
	locations	
	bayplan total numbers	

Table 6.3: Transport messages

A single consignment message is used to exchange information of one activity and a multi-consignment message to exchange information of one or more activities. Therefore, the multi-modal status message can also indicate the status of one activity. The single and the multi-consignment messages have the following basic structure:

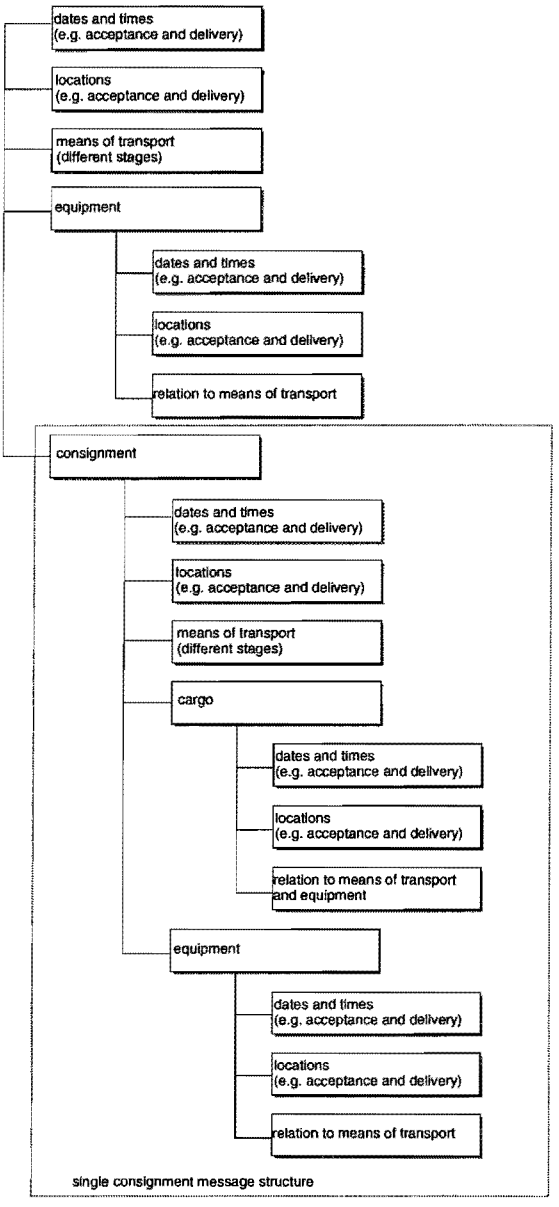


Figure 6.2: Basic structure of single and multi consignment messages

The means of transport in a single or in a multi-consignment message can also be a part of a total transport activity, e.g. the road transport part. In this case, the road transport is an action of an activity. The bayplan messages can contain information of the physical means of transport and, for each location of that means of the transport, the cargo or the equipment.

We will discuss the following aspects regarding the mapping of the kernel message data structure to the standard trade and transport messages:
- the representation of the entities of the kernel message data structure in the EDIFACT messages;
- the representation of associations in the EDIFACT messages;
- the representation of sub- and supertypes in the EDIFACT messages.

We will not go into details of mapping an entity or an attribute to segments or (composite) data elements of EDIFACT. Such details can be found in EDIT (1993).

In most messages the kernel message data structure is present as follows (figure 6.3).

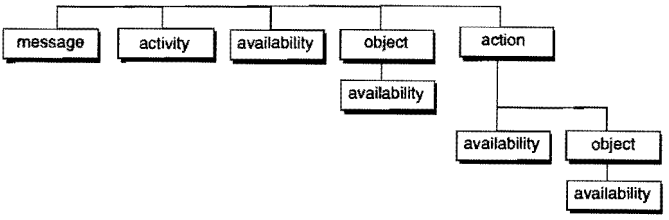


Figure 6.3: Message data model as in existing trade and transport messages

Figure 6.3 shows the entities in a hierarchic message structure. The mapping to EDIFACT segment is not shown. Only the transport messages contain instances of the 'action' entity. In the single consignment messages, an action is represented by the so-called 'transport details' segment group. The multi-consignment message can contain several instances of the 'action' entity with their availability and the objects. One of these instances of an action is the 'transport details' segment group. Other instances represent single consignments.

An association can be represented in the EDIFACT messages by association segments or by a hierarchic nesting of objects. The following transformation options are possible to represent an association (figure 6.4):
- *association segments*
  Associations or entities, which are a domain with two or more non-exclusive functions with one or more ranges, are transformed in association segments at the message level. The functions are data elements of the association segment.

- *hierarchic nesting*
  Each association or entity that has two or more non-exclusive functions is becoming part of the segments representing one of the entities to which it has a function. The foreign key of the other entity is represented by a data element of the segment group representing the entity under which it is nested.

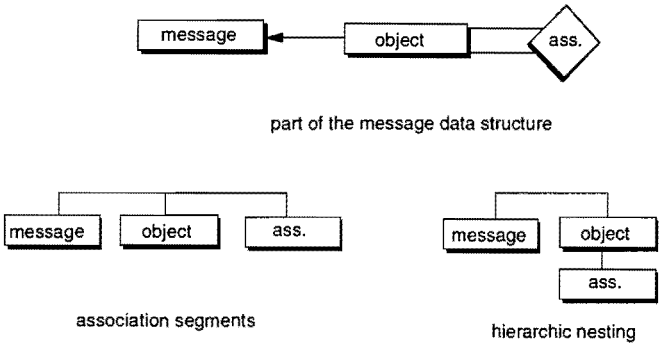


Figure 6.4: Transforming associations in an EDIFACT structure

The association segment approach shows that each association in the message data structure is transformed to one segment or a group of segments. Using the association segment approach, the segments can be exchanged in arbitrary order if the condition is valid that a unique relation exists between entities or between associations and segments. However, in the existing trade and transport messages the hierarchic nesting approach is taken. A reference segment can be used in the trade messages to represent foreign keys of objects. The representation of the association segment in the transport messages depends on the object type which can be referenced (e.g. split goods placement represents an association between cargo and containers).

We use the specialization rules of functional data modelling to specify object types. We identify two possibilities for mapping super- and subtypes (figure 6.5):

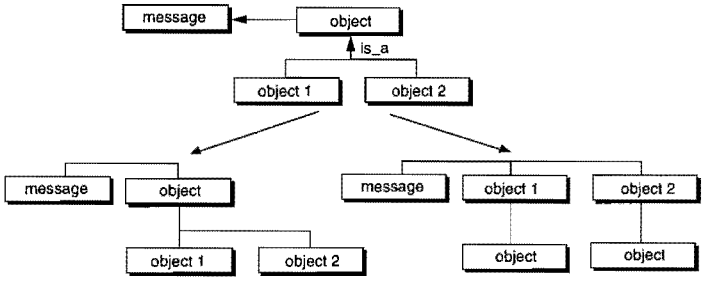


Figure 6.5: Mapping of super- and subtypes

- each subtype is mapped to a separate segment group;
- each supertype is mapped to a separate segment group, in which the segments of the subtypes are nested.

In the trade and transport messages standardization bodies have taken the approach to create a segment group per sub type.

We can conclude that although the specification of the EDIFACT trade and transport messages is not unambiguous, they contain the elements of the kernel message data structure. Furthermore, realization options have been selected when mapping of functional requirements to the EDIFACT syntax is done. It seems better to make these options explicit and select an option when mapping to the EDIFACT syntax.

6.4.3 Concepts of international message standardization

Currently, messages are specified using the EDIFACT syntax. There are three developments that start with modelling the functional requirements and map these requirements to the EDIFACT syntax:

- Interactive EDI (UN/ECE WP.4 GE.1, 1992);
- Business and Information Modelling Guidelines (UN/EDIFACT, 1993);
- Open-EDI Reference Model Standard - Working Draft (ISO/IEC JTC 1/WG 3 N255, 1993).

The development of interactive EDI and the Business and Information Modelling Guidelines is an activity of the United Nations. The specification of the Open-EDI Reference Model Standard is an activity of the ISO. We will discuss these three developments in more detail.

"*Interactive EDI* is a series of exchanges of information between the applications of independent parties in order to accomplish a joint task, where subsequent exchanges may depend upon the results of previous exchanges. Strict timing requirements frequently apply. Applications which are inherently interactive, include airline reservation systems and remote automated teller machines of banks" (UN/ECE WP.4 GE.1, 1993). The following concepts are defined to support interactive EDI (UN/ECE WP.4 GE.1, 1993):

- *scenario*

  The document on interactive EDI presents three definitions of a scenario:

  1. a scenario is a model of a set of real-world rules governing a real-world task - called a transaction - to be accomplished by independent organizations working together.
  2. a scenario is a formal description of a class of business activities which defines, among other things, the roles which can be played by role players (including the events which need to take place) to achieve a particular business objective.

3. a scenario is a formal description of all business rules and information flows of a type of business transaction among two or more parties.

- *transaction*
  A transaction is an instance of a scenario.
- *dialogue*
  A dialogue is a two-way conversation between co-operating role players within a transaction. It is formally composed of a pair of interchanges and is an instance of a dialogue type.
- *dialogue type*
  A dialogue type describes a set of actions between two role players within the context of a scenario. It is implicitly defined by the pair of roles and the scenario.
- *interchange*
  An interchange has the property of being an instance of the information flow in one direction within a dialogue.

We have already referred to a syntax to support interactive EDI (ISO CD 9735:1993(E), 1993). This syntax specifies the establishment of a connection and disconnection by means of the interchange header and the interchange trailer respectively. In-between, several messages can be exchanged (figure 6.6).

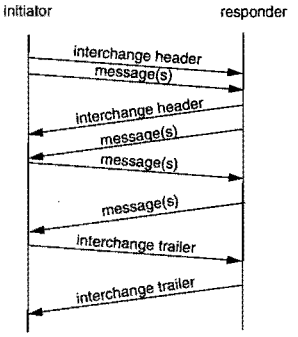


Figure 6.6: An example of a time sequence of messages in interactive EDI

The objective of the *BIM Guidelines* is to give certain rules for the application of modelling techniques to support message design (UN/EDIFACT, 1993). The guidelines specify three phases and their deliverables. Two of these phases consist of 'information models' and 'activity models' that seem to be similar to data modelling and process modelling respectively. The third phase consists only of 'information models'. It is the realization of messages in the EDIFACT syntax using standard EDIFACT elements that are present in the so-called EDIFACT repository (figure 6.7).

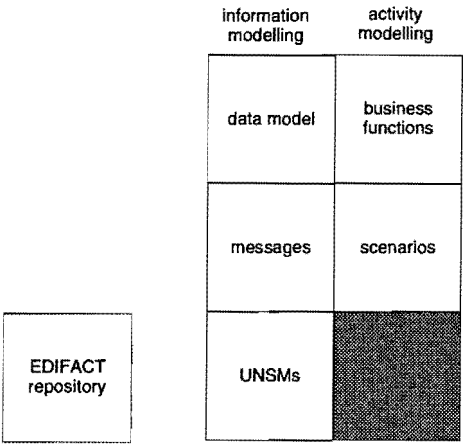


Figure 6.7: The so-called BIM-boxes (UN/EDIFACT, 1993)

The deliverables of the phases one and two, business analysis phase and EDI requirements phase respectively, are based on the following concepts (we only include the concepts of 'activity models'; the concepts of 'information models' are similar to the ones we have presented in annex 1 of this monograph concerning functional data modelling):

- *party*
  A party is a participant in an EDI transaction.
- *EDI transaction*
  An EDI transaction is a set of information flows between parties wanting to perform business processes with common goals. An EDI transaction is initiated by a specific party and terminated upon recognition of one of the agreed conclusions by any party involved.
- *scenario*
  A scenario is a formal specification of a class of EDI transactions which are identified from the functions defined in the business analysis phase.

The *Open-EDI Reference Model Standard* aims at providing a framework for standardization of scenarios and creating a so-called Open-EDI environment. Such an Open-EDI environment is required to reduce the costs of coming to an agreement in short-term partnership.
The model consists of the following two views (figure 6.8):

- *Business Operational View*
  This view addresses the business aspects of interoperability between the Open-EDI systems, including business conventions, agreements, and rules between business participants. Standards related to this view provide a specification of how to model an Open-EDI scenario. There is a requirement for generic and specific scenarios.

- *Functional Service View*
  This view addresses the information technology aspects of interoperability between Open-EDI systems. Several physical and organizational configurations have to be supported by standards called the Open-EDI Support Services (OeSS). The OeSS is a component that identifies groups of functions, each of which has been separately distinguished for the purpose of service, protocol, and API (Application Programming Interface). OeSS may include one or more OSI Application Service Elements. Several Management Domains can be recognized that use the OeSS components.

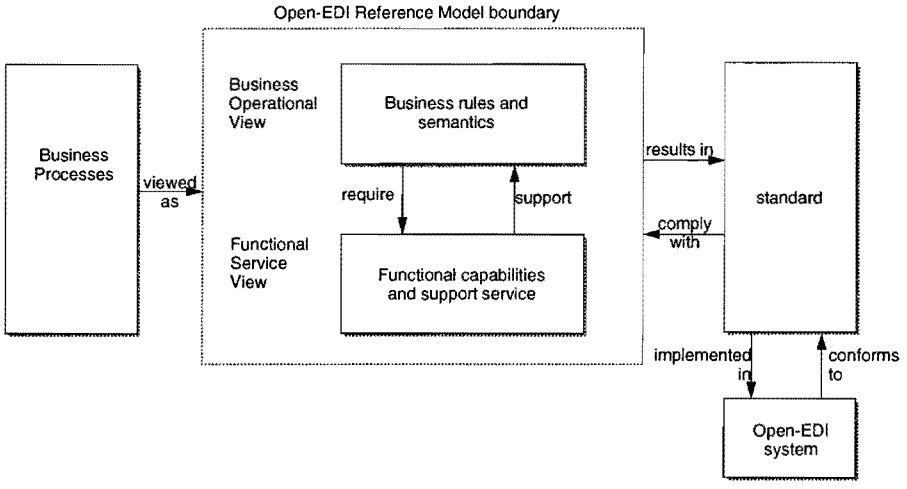


Figure 6.8: The Open-EDI Reference Model Standard views

These three documents present concepts that we also use, e.g. transaction and scenario. Furthermore, a number of concepts presented in these documents seem similar to our concepts. We will discuss them in more detail.
With respect to interactive EDI and the BIM Guidelines, we can distinguish between structure and behaviour elements in the concepts. The structure elements of interactive EDI are scenario, dialogue type, and role. The corresponding behaviour elements are transaction, dialogue, and role player respectively. The concept 'interchange' is also a behaviour element. The concepts 'dialogue' and 'dialogue type' are identical to the concepts 'transaction' and 'transaction protocol' respectively as defined in this monograph. The business objective that is to be achieved is specified to be supported by a scenario. Intuitively, a business objective seems to be identical to the concept 'activity'. However, there is no explicit definition given in UN/ECE WP.4 GE.1 (1992). In general, structure elements are defined in the BIM Guidelines by adding 'class' to a concept.

The definitions of 'transaction', 'EDI transaction', and 'scenario' are not the same in both documents. There are three different definitions of a scenario in interactive EDI. The first and the third definition of a scenario and the definition of a transaction refer to each other. The second definition is not complete and refers to business activities and business objectives that are not defined. The definition of 'scenario' in the BIM Guidelines refers to functions that are identified in the business analysis phase and is a class of EDI transactions. Therefore, a scenario is a structure element.
In this monograph, the concept of scenario is only used as a visualization of the time sequence of messages in a transaction. Therefore, in this monograph a scenario is a behaviour element. Because we focus on the concepts that model an information and a business system, we do not require 'scenario' as a separate concept.
Whereas a transaction is simply an instance of a scenario in interactive EDI, the definition of an EDI transaction given in the BIM Guidelines is more complex. The latter definition contains elements that are part of the concept 'scenario' of interactive EDI. The definition of 'EDI transaction' leaves us with some questions:

- how are the common goals defined and by whom are they defined;
- how is a conclusion defined, how is it agreed upon, and which of the parties involved may recognize it as a valid conclusion.

In both interactive EDI and the BIM Guidelines, a scenario can include two or more functions (or roles in interactive EDI). The sequence of events (interactive EDI) or the information flow (BIM Guidelines) is defined as a structure element. According to our concepts, the sequence of events or the information flow is given by the structure concept 'transaction protocol'. In this monograph, each actor specifies its own role by its services and its corresponding procedures. Therefore, depending on the procedures of all actors involved in the execution of their business processes, an instance of an '(EDI) transaction' is given.
With respect to interactive EDI, we can also make an analogy between distributed databases and the sequence of messages. The interchange header and trailer are identical to the begin and the end transaction respectively. The messages are the database operations. A difference is that the messages can be exchanged in two directions, which means that an initiator and a responder can perform operations at each others database in a transaction. The concepts of interactive EDI are also similar to the concepts of the ISO/OSI Basic Reference Model (section 1.2.4). The interchange header is used for connection establishment and the interchange trailer for orderly connection release. The messages are the data-PDUs. However, by layering the communication (annex 1) the ISO has specified that several transaction can exist during one connection, whereas interactive EDI specifies one transaction per interchange.

Regarding interactive EDI, the BIM Guidelines, and the Open-EDI Reference Model, our main conclusion is that the aim of these approaches is to standardize the behaviour in terms of scenarios. These approaches can be used to solve problems, where there is for instance no clear hierarchy between actors. An example is that a transaction passes several actors, whereas the last actor reports to

the actor that initiated that transaction. It may also be feasible to standardize behaviour that serves as a reference for a specific interorganizational system. Actors of such a specific IOS have to adapt the standard scenarios to their specific situation, because the interorganizational business system will also be specific. If these standard scenarios do not fit a business process, new scenarios can be specified using the concepts defined in this monograph. Additionally, we can conclude:

- there is a distinction between structure and behaviour in the document on interactive EDI;
- the concepts contain intuitive elements. Some of these concepts seem to be identical to the concepts defined in this monograph;
- the concepts do not focus on information systems of actors. Consequently, they do not give actors flexibility to specify, select, and change their internal procedures;
- the Open-EDI Reference Model adds the concept of management domains to support standardisation by different actors;
- concepts in these three documents are similar to concepts of distributed databases and of the ISO/OSI Basic Reference Model.

7 Conclusions and further study

7.1 Conclusions

In this book we have defined concepts that can be used to specify interorganizational systems. We have made a distinction in concepts that specify the structure and concepts that specify the behaviour. We have decomposed interorganizational systems in an interorganizational business system and an interorganizational information system and given concepts that can be used to specify both systems. An interorganizational business system itself can be decomposed in different business systems and an interorganizational information system in different information systems. Table 7.1 shows an overview of the concepts that specify a business system and an information system.

	structure	behaviour
business system	business process/service task object type	activity action object
information system	procedure transaction protocol message type	job transaction message

Table 7.1: Overview of the concepts

We have used timed, coloured, hierarchical Petri nets to model the concepts of business systems. Furthermore, we have specified a generic software component of the information system of an actor. We have called this generic component the Business Transaction Management System (BTMS). Procedures, transaction protocols, and message types are parameters of the BTMS. The BTMS is modelled as a timed, coloured, hierarchical Petri net and its parameters are modelled with a functional data model. In concrete practical situations verification of the conceptual model is still to be performed.

7.2 Achievements

Regarding our problem definition, we can first of all conclude that we have been able to develop a conceptual model for interorganizational business systems. The

conceptual model can be applied in practice for several types of business systems, like health care, transport and insurance. Based on the performed research, we can conclude that we have succeeded in giving a conceptual model of a BTMS that controls the execution of interorganizational business systems and that the software for such a BTMS can be realized. Furthermore, achievements can be found in business process engineering, message development, and message implementation. We will discuss these achievements in more detail.

There are a number of parameters that influence business process engineering. Such parameters are, for instance, information systems, business systems, and organizational structure.

We can say that the availability of the BTMS makes it possible to re-engineer business processes independent of information systems. Business process re-engineering results, for instance, in changes in the physical lay-out of the business processes and the organizational structure of an actor. The output of business process re-engineering is a set of procedures that can be a parameter to the BTMS. Thus, the BTMS is generic software capable of processing information independent of the structure of business processes. The BTMS can easily be adapted to a business system and, consequently, an actor can easily and less costly change his business system.
This means that actors are capable of reacting flexible to changing requirements of their customers by offering new services. Such flexibility is of great importance in every industry sector. Especially in industry sectors, where one is capable of changing its business system rapidly, the implementation of the result of business process re-engineering can be performed fast. Such industry sectors are, for instance, transport, finance, and insurance. These industry sectors have in common that the business system is either of an abstract nature, or they make use of an infrastructure like a network of roads offered by other actors. In other industry sectors, where the business system can not be changed rapidly and the most important aspect of the output of the business system is a physical object, e.g. in production, business process re-engineering will still take some time. When, however, in such cases an actor combines its products with services of an abstract nature, re-engineering can be less time consuming, like in the example of a lease contract with the possibility to lease a car of a car manufacturer, with or without an insurance contract and with or without the option to buy the car after the lease contract has ended.

Regarding organizational structures, Womack (1994) looks at organizations as a set of autonomous units. Each of these units can offer its services to other units. In this monograph, we have called these units 'actors'. Current information systems reflect the organizational structure. Thus, it is costly and time consuming to change the organizational structure. The BTMS, however, is process oriented and in that sense independent of an organizational structure. It requires a specification of a business process and it can support several organizational structures, especially

autonomous units. Organizational structures can change without having effect on the information system.

Let us discuss message development and implementation in more detail. Currently, EDI message development and implementation is a costly and time consuming activity. There are special groups developing standard messages for their own use and the use of others. These standard messages are documented in the syntax in which the information is going to be exchanged (the EDIFACT syntax). Actors that are going to implement these messages, have to interpret them and have to make choices with respect to the elements they are going to use. The interpretation can lead to different usage of the same message, thus leading to closed EDI environments. To come to a common interpretation, actors have raised so-called EDI organizations (Hofman, 1989) that have to come to a standard interpretation aligned with interpretations made in other countries. However, these EDI organizations make specific EDI implementation guides for specific groups of actors that have a common business system. It is our experience, amongst others in the transportation industry, that these aspects still prevent actors from implementing EDI. Another aspect of message implementation is the mapping of the messages to internal applications. One can use so-called EDI translation software for the translation between EDIFACT messages and database instances. Not all of these translators offer the functionality required by the actors and most of them are still costly.

In this book we have used functional data modelling and have specified a kernel business process data structure that serves as a basis for message development. First of all, we, and also others (UN/EDIFACT, 1993) are of the opinion that data modelling for EDI message development has several advantages. For instance, it improves the quality of the standard messages, it is less time consuming, and one can develop consistent sets of related messages. Secondly, using the kernel business process data structure and specializing the object types, has the advantage that one has a common view on the business system. Using this approach, the actors that are going use the messages in practice, do not have to be involved heavily during message development. Message development can be performed by specialized persons. Thus, because only a limited number of persons with special skills is involved in message development, it can be performed faster and cheaper. Standard messages can be seen as products that have to be sold. The result of message development will be better documented messages that are faster available for implementation (EDIT, 1993).
We have practical experience by developing a complete set of messages to support claims management in insurance within a period of half a year. In a similar way, we have developed message data models to support external logistics within a very limited time scale and with limited resources in comparison with resources used for message standardization.

We have mentioned two aspects that are of importance to message implementation: message interpretation and translation. Using the business process data structure

makes message interpretation easier. It does not lead to different interpretations and the actors do not require knowledge of the EDIFACT syntax. They can focuss more on aspects of their business that have to be represented by a business process data structure. Regarding translation, an actor can implement a complete business process data structure, map between this data structure and the internal database structure, and make its services known to others. In this way, an actor is able to communicate with other actors that have done the same, without having to make specific message implementation guides. The messages have to be seen as carriers of information. The information that is to be carried, depends on the service that is to be executed. One does not necessarily require a BTMS for this purpose. In practice a software product that translates on the basis of data structures will be adequate. However, a BTMS can offer additional advantages like offering flexibility in doing business. When our concepts are implemented, EDI organizations can focuss more on selling EDI messages and developing new messages if required.

7.3 Further research questions

The following aspects are for further research:

- *internal structure of organizations*
  Womack (1994) looks at organizations as a set of autonomous units. Current software cannot support these separate units. Introduction of the BTMS provides the same software to each of these units. The feasibility of such autonomous units using the BTMS and the effects on organizations and skills of persons needs further study.
- *trading relat*ions
  Using the kernel business process data structure, it is not required to make agreements on the usage of messages. This can reduce transaction costs between actors. The real benefits of this approach for trading relations are for further research.
- *engineer*-to-order
  The term 'engineer-to-order' is most often used in production. It means that the routing through a business process is determined per order. Similar situations can also be found in other business processes, for instance in health care. Selecting services on basis of parameters in messages can be also seen as some sort of engineer-to-order. The specification of the BTMS that we have given only supports the selection of services. Engineer to order is still a subject for further research.
- *tools*
  Business process engineering, message development, and message implementation can be improved if the proper tools are used by all interested actors. These tools have to communicate with each other. The development and standardization of the interfaces between those tools are a subject for further research. Furthermore, the realization of the concepts in existing tools like ExSpect (ExSpect, 1990) and EDIT (EDIT, 1993) is of interest.

Other aspects like the use of a relational syntax instead of a hierarchical syntax are subject for further research.
We have also discussed achievements that can be obtained for business process engineering. Actual cases are required to illustrate these achievements.

Annex: Modelling techniques

A.1 Introduction

We require modelling techniques to model the concepts of the interorganizational business system and the data and process structure of the BTMS that is part of the interorganizational information system. In general, a *system* can be decomposed into smaller parts, which can be called *components* (Aerts, 1991). Each component may be a system in itself. The environment of a system can be seen as a component as well. A system is called *closed* if the environment is a component of the system. In that case, the environment is specified as a black box. If the environment is not part of the system, the system is called *open*. Every open system can be closed by adding the environment as a component to the system. The components of a system can have a relation with each other and with the environment. This relation is modelled by the objects that are exchanged between two components. Objects in a model are, for instance, containers or messages.
A system is specified by its input and outputs, a set of *states*, and the possible *state transitions*. An input can trigger a state transition producing one or more outputs. The set of states of a system is called the *state space* of that system. The state space of an information system, which is the static part of that system, is specified by a *data model*. A *process model* describes the behaviour of a system by specifying the state transitions or sequences of states that are allowed.
Various *specification methods* exist for both data modelling, e.g. the entity-relationship model (Chen, 1976), the binary relationship model (Nijssen, 1989), and the functional data model (Aerts, 1989), and process modelling (Van Hee, 1991). In this monograph we have chosen the functional data model for data modelling (section 2) and the timed, coloured, hierarchical Petri net theory (Jensen (1990) and Van der Aalst (1992)) for process modelling (section 3). Timed, coloured, hierarchical Petri nets have been chosen because the theory is supported by diagramming techniques for description, analyzing techniques for evaluation, and tools for prototyping and simulation. A formal definition of the concepts of Petri nets as described in the next pages is derived from Van Hee (1994). The theory of timed, coloured, hierarchical Petri nets is an extension to the original theory of Petri (1962). Principles of Petri nets are discussed by Reisig (1985).
The concepts of layered communication are introduced in this annex to explain the concept of 'service' and 'protocol' and to show how these concepts can be specified by Petri nets.

A.2 Timed, coloured, hierarchical Petri nets

The basic structure of a Petri net is a directed bipartite graph with two kinds of nodes called *places* and *processors*. Usually, 'processors' are called 'transitions', however we prefer the first term because the term 'transitions' is already used for state transitions. The nodes are connected by directed labelled arcs, called *connectors*. A connector can only connect a place to a processor or a processor to a place. One or more connectors are possible between one place and one processor. Places can contain *tokens* and processors can consume and produce tokens. Each place is shown as a circle and each processor as a rectangle (figure 1).

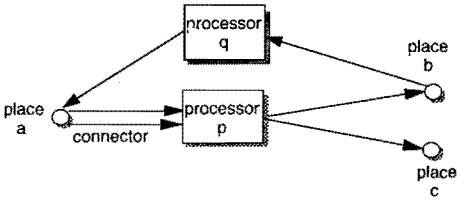


Figure 1: A net structure

Figure 1 shows a net structure of two processors, three places, and six connectors. Place a is called an *input place* of processor p since there exists a connector from a to p and place b is called an *output place* of processor p since there exists a connector from p to b. Place c is only an output place and the places a and b are both in- and output places. Places can represent locations or stages (we will not use the word 'state', because that word already has a specific meaning). For example, in transport a place can represent a location and in sales a place can represent the stage of the sales process of an article. The possibility that a place can represent a location or a stage provides us with a powerful theory to model both physical systems (e.g. external logistics) and abstract systems (e.g. finance and insurance). A processor is used to model the transformation of physical or information tokens, or the occurrence of an event. A transformation of physical tokens may alter the location or these stage of tokens. An example of such a transformation is the transport of containers from one location to another. A transformation of information tokens is, for example, the computation of a volume of a package on basis of the length, the width, and the height of that package. An occurrence of an event is for example the arrival of an aeroplane or the reception of a message. Processors are the system components. The communication between two processors is modelled by the places that are connected to both of them.

In hierarchical Petri nets, processors and places can be decomposed in other processors or can be aggregated to form a new processor. A Petri net as shown in figure 1 is called a *flat net* if the processors and the places of that net cannot be decomposed into other nets. A processor that cannot be decomposed and is without

memory is called an *elementary processor*. Consequently, the processors of a flat net are only elementary processors. An elementary processor is a function that can produce output tokens by consuming input tokens. In the classical theory of Petri nets, all processors are elementary (Petri, 1962). A processor that can be decomposed is called a *non-elementary* or *composite* processor. Such a processor is a net in itself and may contain memory in the form of places with tokens. A processor with memory can always be decomposed in places and processors.
A flat net is formally defined by a finite set of places (L), a finite set of processors (P), a finite set of connectors (C), a function I that assigns to a processor a set of input connectors, a function O that assigns to a processor a set of output connectors, and a function M that assigns connectors to places. A processor does not necessarily require an output connector (function O is not specified for such a processor) and processors may have unconnected connectors (function M is not specified for one or more connectors). If the latter is true, such a net is called an *open net*. If M is specified for all connectors (they are said to be connected), the net is called a *closed net*.
A decomposition of a net creates a hierarchy of decomposition levels, called a *hierarchical net*. A net at a lower level specifies further detail of a processor that is one level above. The aggregation of lower level nets to a higher level net can be represented by a function that assigns lower level processors to a higher level processor until no further aggregation is possible. The lowest level of a hierarchical net is a flat net. At the top only one processor exists. At all levels, with the exception of the lowest level, at least one processor can be decomposed into another net. A processor that can be decomposed and is not the top level processor is called a *subnet*.
Places can also be decomposed in more detailed subnets. These subnets eventually produce all previously consumed tokens and the connectors of these subnets with the environment are connected to places in the subnet. These subnets are called *composite places*. Composite places are useful to represent large networks or to structure a large number of places, connecting one or more processors. Composite places are modelled by an oval.
Figure 2 (see next page) shows an example of a hierarchical net, in which processor p of figure 1 is decomposed, and an example of a composite place. Processor p of figure 1 is the top level processor in figure 2. It is a subnet in figure 1.

As mentioned before, a place can contain tokens. A token can represent a physical object (e.g. an article or a person), an abstract object (e.g. a job), or an information object that refers to either a concrete or an abstract object (e.g. a message). In Petri nets, each token has the following characteristics:
- a *place* in which it resides;
- an *identity*.

In timed Petri nets, each token has also a *time stamp* that tells when the token may leave a place. In coloured Petri nets, each token also has a *value* which in itself can be a complex data structure. The value of a token is called its colour.

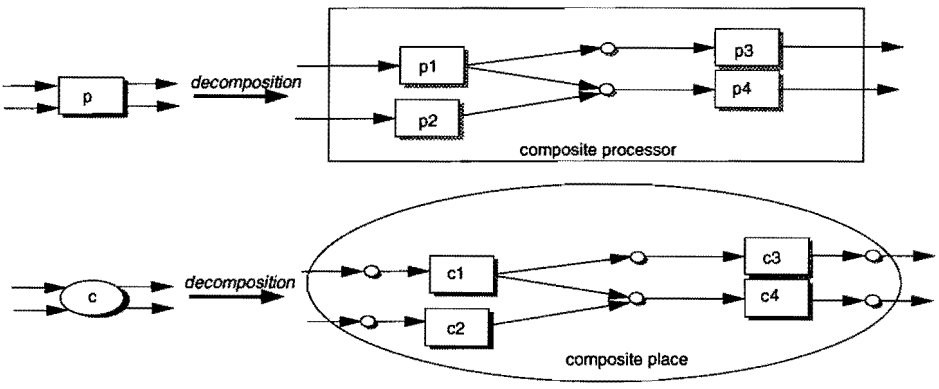


Figure 2: Examples of decomposition

The *state* of a net is specified by the set of all tokens in that net. The *initial state* of a net is defined as the state in which that net starts. The *state space* of a net is the set of possible states, including the initial state. A processor may produce output tokens on the basis of input tokens. An elementary processor executes by consuming one token via every input connector. In Petri net terminology, the execution of an elementary processor is called a *firing*. A *firing sequence* is a sequence of firings. A processor has one or more *firing rules*. A firing rule has the following properties:

- a firing rule is executed as soon as one token can be consumed at every input connector;
- the pre-conditions for consumption of those tokens are satisfied;
- the maximum time stamp of the consumed tokens is minimal under all possible firings;
- for every output connector one token is produced at most with the following characteristics:
  - the newly produced tokens have an identity that differs from that of all tokens that remain;
  - the time stamp of the produced tokens is greater than or equal to the time stamp of the consumed tokens;
  - the value of a produced token is a function of the values of the consumed tokens.

Firing rules may be executed simultaneously or consecutively. If two firing rules are executed at the same time, they will be executed in sequence at that time. The sequence is non-deterministic. If two firing rules can be executed by consuming a token from the same input place and that place does not contain sufficient tokens, one of the firing rules is executed. The firing rule that is executed, is chosen in a non-deterministic way.

The execution of a firing rule changes the state of a net. The state of a net is changed by all firing rules that execute at the same time. A *state transition* is the change of the state of a net. A sequence of states starting at an initial state is called a *trace*,

e.g. a possible trace is $0, s_1, s_2, s_3, s_4, s_5$ in which s_i represents the i-th state and s_0 the initial state. A *process* is the set of all possible traces.
Firing rules are only specified for elementary processors. However, a non-elementary processor will have the behaviour of an elementary processor, if it has one input processor, one output processor, and is only able to fire its input processor again after having fired. We will call such a non-elementary processor a *complex processor*. A complex processor can have one or more input connectors, zero, one, or more output connectors, and memory. By means of a feed-back mechanism, no firing rule of the input processor is enabled before an output token is produced. We can use complex processors for modelling services (section 5). An example of a complex processor is shown in figure 3.

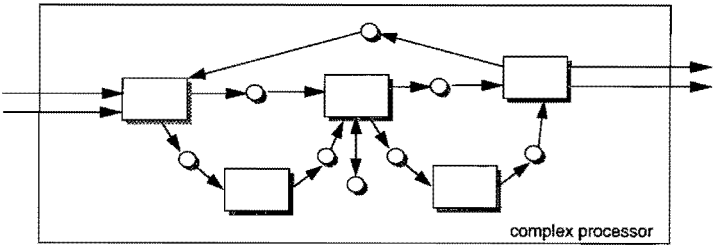


Figure 3: An example of a complex processor

A place is called a *store* if it is both an input and an output place of the same processor and has always a token available. Figure 3 shows one store that is connected to the processor in the middle only. A store may be connected to two or more processors.
In practice, processing will take time, which means that the execution of a firing rule takes a certain time. In Petri nets, all elementary processors fire timeless and we must simulate the duration of processing via the time stamp of a token. A processor with a simulated duration is called a *time-consuming processor*. Such a processor consists of an input processor, an output processor, and one internal token. The input processor can produce the internal token for the output processor after having consumed input tokens. The output processor can only produce output tokens if the internal token is available. After having produced its output tokens, the output processor produces the internal token for the input processor. The value of the time stamp of the internal token specifies the duration of a time consuming processor. Figure 4 shows the structure of a time consuming processor. Initially, place q contains a token.

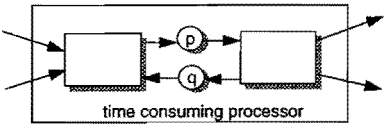


Figure 4: A time consuming processor

A special type of Petri net is a *finite automaton*. In this type of nets, besides the connections to the environment, every processor is connected to only one input place and only one output place. The number of tokens in the net is only one; initially the token is in a certain place, thus representing the initial state of that finite automaton, and after execution the token returns to that place (figure 5).

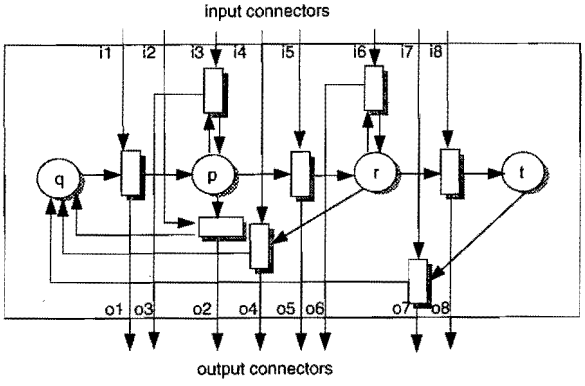


Figure 5: An example of a Petri net of a finite automaton

Place q may initially contain the token. Because a finite automaton contains only one token, the state of a finite automaton can be represented by the place in which that token resides. Therefore, a trace of a finite automation is a sequence of places in which the token resides. All traces can be represented by a *state transition diagram*, in which each place is represented by a *state* and each processor by a *state transition*. A state is shown as a circle and each state transition as a directed labelled arc. State q in figure 6 can be the initial state of the state transition diagram representing the Petri net of figure 5. The label of each arc in figure 6 refers to an input and an output connector of the finite automaton of figure 5, thus showing all the information of a finite automaton that is also shown in the figure of the corresponding Petri net.

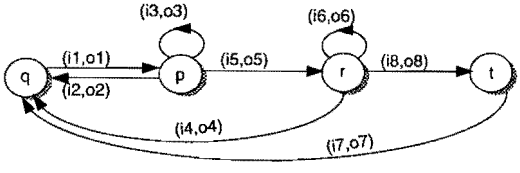


Figure 6: A state transition diagram of the Petri net of figure 5

We will use the following conventions for the graphical representation of a net:

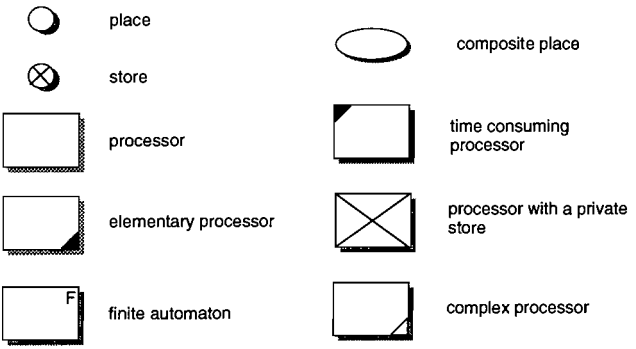


Figure 7: Graphical conventions for Petri nets

A.3 Data modelling

A functional data model consists of objects. We make a distinction between simple objects, called *simplexes*, and complex objects, called *complexes*. An example of a simplex is a specific person 'Jelle' or a specific job 'consultant' that are of the type 'person' or 'job' respectively. An example of a complex is a message of the type 'order'.

A *simplex* has the following characteristics:

- it is atomic, i.e. we do not consider any internal structure in them;
- it is distinguishable;
- it belongs to a type, called *class*;
- the classes can be named by a noun, e.g. 'person' and 'job'.

A simplex is called an *instance* of a simplex class. Simplexes are mutually related by functions, e.g. Jelle can be a consultant. The simplex classes and the functions are graphically represented by a *schema*. In a schema, a rectangle denotes a simplex class and a directed arc denotes a function f from one simplex class to another (figure 8).

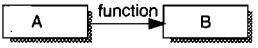
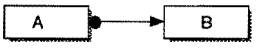


Figure 8: A schema with two simplex classes and one function

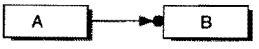
In general we do not consider all possible instances of a schema, but only instances that satisfy specific constraints. There are constraints that occur frequently and others that occur less frequently. The less frequently occurring constraints can be

specified by predicates that should be true for every instance. Frequently occurring constraints are characterized by properties of functions. They also should be true for every instance. To be able to specify the properties of functions, we use the following conventions. Let f be a function defined on a subset D_f of simplex class A where for every simplex in D_f the image is in simplex class B. D_f is called the domain of f and the set of all images in B of all simplexes in D_f is called the range of f (R_f). Clearly, R_f is a subset of class B. Both the range and the domain may be subsets of the same class.
We distinguish the following three properties of functions and use a graphical notation to represent them:

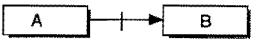
- *total*
  A function f is said to be total if D_f is equal to A.



- *surjective*
  A function f is said to be surjective if R_f is equal to B.

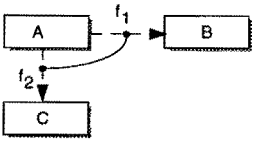


- *injective*
  A function f is said to be injective if every element of R_f is an image of exactly one element of D_f.

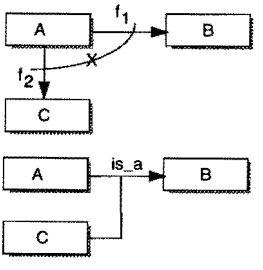


Combinations of these properties may also appear, e.g. a total injective function between the simplex class 'employees' and the simplex class 'person', which specifies that every employee is a person and that every person can be at most one employee. Other constraints that are not necessarily properties of functions can occur frequently and have a graphical notation. Those are key constraints, exclusion constraints, and inheritance (is_a) constraints. They are defined for two or more functions, but are only shown for two functions:

- *key*
  A key is a set of total functions f_1, .., f_n with D_{fi} identical for all i, such that every element of D_{fi} is uniquely determined by these functions, i.e. if f_1, .., f_n form a key for simplex class A then in every instance we have for all pairs of simplexes a_1 and a_2 of A that if $f_i(a_1)=f_i(a_2)$ for i an element of {1,..,n} then $a_1=a_2$.



- *exclusion*
  Two or more functions f_1, .., f_n with their domain in simplex class A form an exclusion if the intersection of the domains of f_1, .., f_n is empty.
- *is_a relation*
  An is_a relation is a total, injective function. Simplex class A is called a *subtype* and simplex class B a *supertype*.



A *complex class* **C** of a schema **S** is a schema that is obtained from **S** by deleting some of the simplex classes and the functions. An *instance* of a complex class is called a complex (note that simplexes are unique within a complex). Figure 9 shows an example of a schema with a complex class called 'cargo'.

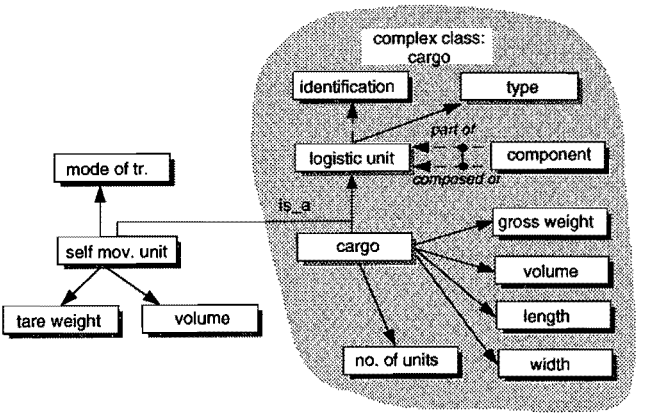


Figure 9: An example of a schema with a complex class

Up to this point, each simplex class is represented by a rectangle. Those simplexes that only occur as range are called *attributes*. In figure 9, attributes are for instance 'no. of units', 'length', and 'gross weight'. Simplexes that occur only as a domain and have functions that are the key of such a simplex are called *associations*. An example of an association is 'component' in figure 9. The remaining simplexes are usually called *entities*, e.g. 'cargo', 'self moving unit', and 'logistic unit'. We will use the following graphical notation in the rest of this book:

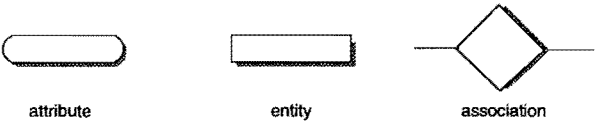


Figure 10: The graphical notation of simplexes

Using this notation, we will not show the key constraints of an association in a schema. Figure 11 shows the schema of figure 9 using the notation shown in figure 10.

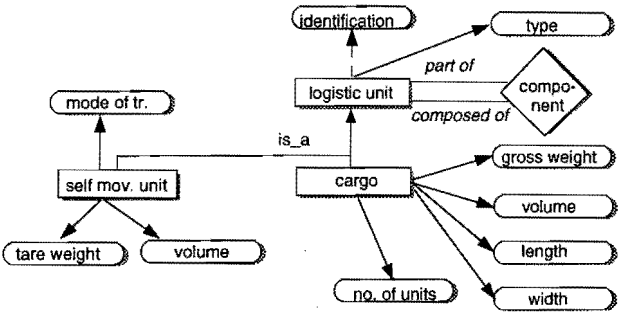


Figure 11: The schema of figure 9 using the notation of figure 10

A.4 Coloured Petri nets and functional data modelling

In the description of coloured Petri nets we have noticed that one of the characteristics of a token is its value, i.e. its value. The value of a token in a Petri net that models an information system is a complex, i.e. has a data structure. A state of a net is a set of tokens. It is allowed that in one state a simplex (i.e. an instance of an entity, an association, or an attribute) occurs in two tokens. The relation between the state space, a state, a token, and a complex is shown in figure 12.

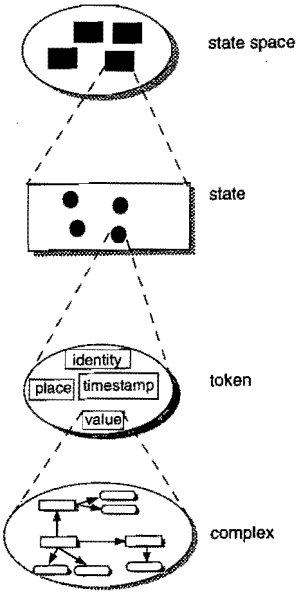


Figure 12: States, tokens, and complexes according to Van Hee (1994)

A.5 Layered communication

The ISO/OSI Basic Reference Model presents a layered approach to the modelling of the communication between two computer systems (we will use the notation of the ISO/OSI Basic Reference Model in this monograph). The principle of layering aims at hiding functionality provided by a lower layer to a higher layer. Two basic concepts are of importance (ISO 7498):

- *service*
  The service of layer N is the capability of that layer and the layers beneath it, that is provided to (N+1)-entities at the boundary between the (N)-layer and the (N+1)-layer.
- *protocol*
  The protocol of layer N is the set of rules and formats which determines the communication behaviour of (N)-entities in the performance of (N)-functions.

(N) stands for the number of the layer in the ISO/OSI Basic Reference Model (ISO 7498). In this context, an entity is defined as an active element in a subsystem, whereas peer-entities are entities in different information systems in the same layer. An entity as defined by the ISO/OSI Basic Reference Model can be modelled as a processor in Petri nets. In figure 13 the OSI layering concept has been modelled with Petri nets.

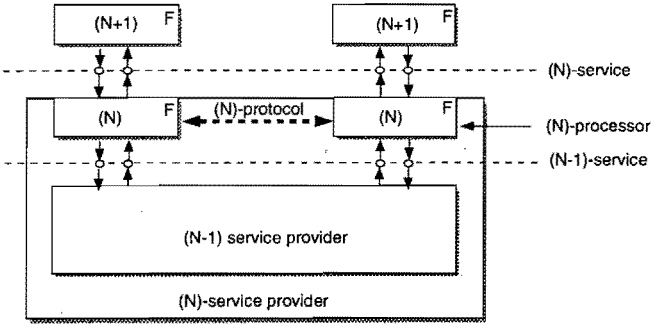


Figure 13: Service and protocol

Figure 13 shows that the (N-1)-service is provided to the processors in layer N. A service provider of layer N is modelled as a (N-1)-service provider and two finite automata (note that this is defined as a specially structured processor) that are communicating with the (N-1)-service provider. Each (N)-processor is in its turn a finite automaton. The places between an (N)-processor and an (N+1)-processor are called *service access points*. The realization of those places in software is called an *interface* by ISO. The interfaces between an (N)-processor and an (N+1)-processor may be different for each software supplier. However, the service offered at a certain interface must be the same for each software product.

According to the conventions of the ISO/OSI Basic Reference model, a service consists of four basic primitives:

- *request*
  An (N)-processor sends a request to its service provider, requesting a service.
- *indication*
  An (N)-processor receives an indication from its service provider as the result of a request of its peer-processor.
- *response*
  An (N)-processor reacts to a received indication by sending a response to the service provider.
- *confirm*
  An (N)-processor receives a confirm from its service provider as the result of a response of its peer-processor.

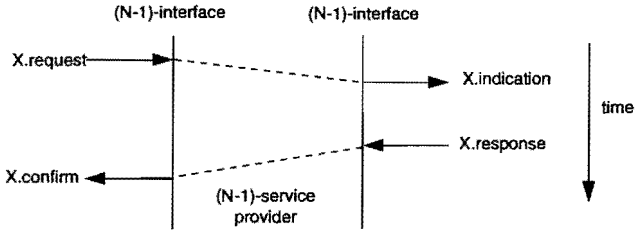


Figure 14: Conceptual primitives of a service

Figure 14 shows the time sequence diagram of the exchange of the service primitives of type 'X' between two (N)-processors. X stands for the service primitives defined for the service offered by layer N. Examples of X are the connect, the disconnect, and the data service primitives of a specific layer. Not all service primitives use the basic primitives, e.g. a data service primitive only uses the request and the indication. The time sequence diagram of figure 14 can be transformed into a processor by replacing the two lines by the processor of the service provider. The rules by which the sequence of service primitives is specified, are part of the service provider (figure 15).

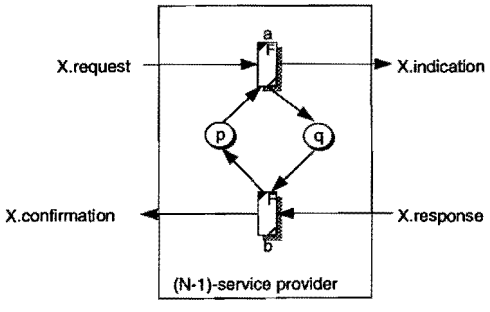


Figure 15: A processor to represent a simple service

In case each request is to be confirmed before a next request can be exchanged, the places p and q together contain not more than one token. In the initial state only place p holds a token. A service primitive is represented by a connector that is connected to a place that can hold that service primitive. The value of the token consists of a command with parameters and data. The data is transparent to a service provider and is a complex.
An (N)-service provider consists of two time consuming processors. According to the concepts of layered communication, an (N)-service provider and two processors of layer N+1 that communicate with the (N)-service provider, compose an (N+1)-service provider (figure 16).

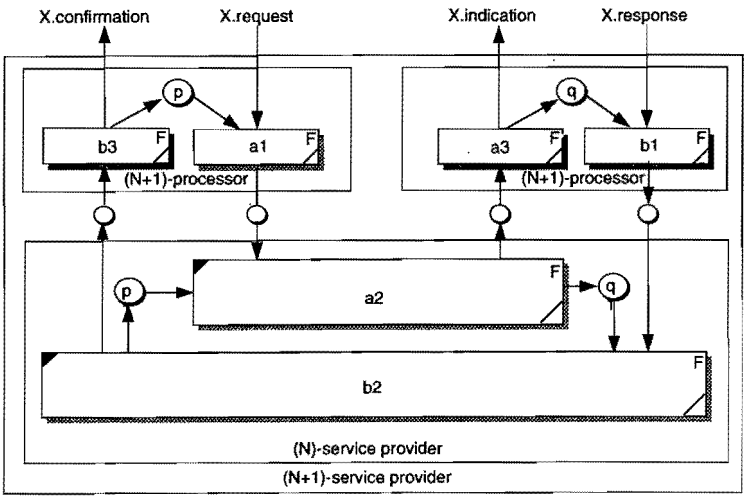


Figure 16: Composition of an (N+1)-service provider

The (N+1)-service provider can be composed of two processors a' and b' like the (N)-service provider, where a' is a composition of a1, a2, and a3, and b' is a composition of b1, b2, and b3.
Figure 16 shows that the processors in layer N+1 pass each service primitive from layer N to layer N+2, or vice versa. However, one service primitive from layer N+2 may result in one or more service primitives to layer N. Such is specified by the (N+1)-protocol between two (N+1)-processors. The processors in a layer and their communication may be more complex than assumed thus far. An example is given by the flow control mechanism of the transport layer (N=4): an acknowledgement of receipt of data is to be received if for example the number of data requests is equal to a given number (in ISO/OSI terminology called the window size). If, for example, the window size is four, a processor can send at maximum four protocol data units to its peer-processor, wait for an acknowledgement, and send again four protocol data units.

The proof of the fact that two peer-processors in a layer and their underlying service provider compose a new service provider is outside the scope of this book. Protocol verification of regular protocols in Petri nets is discussed by Rambags (1993 and 1994). Whereas most literature discusses the verification of specific protocols, e.g. Suzuki (1990) and Genrich and Shapiro (1992), Rambags (1994) and Bochmann (1977) discuss unified methods for protocol verification. Rambags defines a protocol as a regular protocol if a set of strings over an alphabet can be described by a regular expression or if it can be accepted by a finite automaton. He specifies two algorithms to verify such a regular protocol. We will show that the protocols specified in this monograph, can be described by deterministic automata. Therefore, the algorithms of Rambags can be used for automatic verification.

A.6 Other modelling techniques

We will use timed, coloured, hierarchical Petri nets and functional data modelling to model the concepts and to specify the BTMS. Other modelling techniques can also be used. More informal techniques are most often used to show business processes (Bertrand, 1990). These informal techniques are primarily drawing techniques for the visualization of the structure of business processes. They can be mapped to existing formal techniques like timed, coloured, hierarchical Petri nets. Figure 17 shows the elements of a modelling technique, using triangles, rectangles, and arrows.

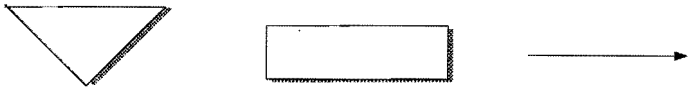


Figure 17: Elements of a modelling technique

A triangle is normally used to represent stock. It can be mapped to a place of a timed, coloured, hierarchical Petri net. A rectangle is used to represent a process and can be mapped to a processor. An arrow is identical to a connector.

We have discussed the relation between timed, coloured, hierarchical Petri nets and state transition diagrams and their application to the specification of protocols. A specification technique based on state transition diagrams and data modelling techniques is given by for instance the ISO (Estelle, 1988). State transition diagrams are a means to visualize protocols before they are specified in more detail using a formal technique.

Glossary

This glossary contains the concepts and their modelling components of timed, coloured, hierarchical Petri nets. The definition of the modelling components is given in Van Hee (1994).

concept	**description**	**modelled as**
action	an action is the execution of a task	firing of an elementary processor
activity	an activity is the execution of a service with uniquely identified input and output objects	firing sequence of a net
actor	an actor is an organization, an organizational unit, or a person operating an information system or a business system, an information on its own, or an information system that is operating a business system by exchanging signals with that business system	non-elementary processor
bound resource	a bound resource is a resource that is reserved by the information system of an actor for an activity or an action	
business process	a business process is the set of services that a business system can provide	open net
business system	a business system is a synonym for an interorganizational business system	
business process data structure	the business process data structure is a data representation of the structure and the behaviour of business processes	data structure
business transaction	a business transaction is a synonym of transaction	

concept	description	modelled as
business transaction data structure	the business transaction data structure is the data structure of the information that can be exchanged between two actors regarding a service or the execution of that service	data structure
contractual relation	a contractual relation is a relation between two actors where they, first of all, make agreements on the structure of an interorganizational business system at strategic and tactical level, and, secondly, co-ordinate the behaviour of that interorganizational business system at operational level	
control signal	a control signal is an information object that is used to initiate an activity or an action	token
incidental relation	an incidental relation is a relation between two actors for the execution of a standard service of a subordinate	
internal data structure	the internal data structure is the data structure of a token that can be present in the place 'selected procedures' of the BTMS	data structure
interorganizational business system	an interorganizational business system is the aggregation of the business processes of two or more actors	non-elementary processor
interorganizational information system	an interorganizational information system is the aggregation of two or more information systems	non-elementary processor
interorganizational system	an interorganizational system is defined in two ways: it is the aggregation of an interorganizational information system and an interorganizational business system; it is the aggregation of two or more actors	non-elementary processor
job	a job is the execution of a procedure	firing sequence of an open net

concept	description	modelled as
message	a message is a unit of information exchanged between a sender and a recipient	token
message data structure	the message data structure is the structure of the information that can be exchanged between two actors as part of a transaction	data structure
message type	a message type is the set of messages that have the same characteristics	attribute
object	an object is a physical thing (e.g. a container), an abstract concept (e.g. beauty), or a piece of information (e.g. a message)	token
object type	an object type is the set of objects with similar features	complex
procedure	a procedure is a specific ordering of steps that has a beginning and an end, and is used to manage the execution of a service	open net
report signal	a report signal is an information object that is used to represent the result of an activity or an action	token
resource	a resource is an object that can be used to facilitate an activity or an action	token
service	a service is a specific ordering of tasks that has a beginning and an end	open net
step	a step is an elementary unit of work in an information system	non-elementary processor
subordinate	an actor is called a subordinate with respect to another actor if the first actor is able to execute a service on behalf of that second actor	non-elementary processor
superior	an actor is called a superior of another actor if the first actor can outsource the execution of a task to that second actor	non-elementary processor

concept	**description**	**modelled as**
task	a task is an elementary unit of work that is capable of consuming clearly defined input objects and producing clearly defined output objects on the basis of a control signal, possibly using resources, and producing a report signal	elementary processor
transaction	a transaction is a sequence of messages	two tokens: one of the superior and the other of the subordinate
transaction protocol	a transaction protocol is a set of allowed sequences of message types	four communicating non-elementary processors

References

Aalst W.M.P. van der, *Time coloured Petri nets and their application to logistic systems,* Master thesis Eindhoven University of Technology, 1992.

Aalst W.M.P. van der, Hee K.M. van, Houben G.J., *Modelling workflow management systems with high-level Petri nets,* internal note Technical University of Eindhoven, 1993.

Aerts A.T.M., Hee, K.M. van, *Modelling with a Functional Datamodel,* Informatie 31, 12, p. 941 - 956, 1989 (in Dutch).

Aerts A.T.M., Alblas G., Hee K.M. van, *Conceptual modelling of systems,* Academic Service, 1991 (in Dutch).

Ballou R.H., *Basic business logistics: transportation, materials management, physical distribution,* Prentic Hall Inc., New Jersey, 1987.

Barret S., B.R. Konsynski, *Interorganizational Information Sharing Systems,* MIS Quarterly, Special Issue, pp. 93 - 104, Fall 1982.

Bertrand J.W.M., Wortmann J.C., Wijngaard J., *Production control: A structured and design oriented approach,* series Manufacturing research and technology volume 11, Elsevier Science Publishers B.V., 1990, ISBN 0-444-88122-0 (Vol. 11).

Bochmann G.V., Gecsei J., *A unified method for the specification and verification of protocols,* in B. Gilchrist, ed., Information Processing, volume 77, pages 229-234, IFIP, North-Holland, 1977.

Bowersox D.J., Closs D.J. and Helferich O.K., *Logistical management, A systems integration of physical distribution, manufacturing support and materials procurement,* third edition, Macmillan Publishing Company, 1986.

Cash J.I., Konsynski B.R., *IS redraws competitive boundaries,* Harvard Business Review, pp. 134 - 142, march-april 1985.

Chen P.P., *The Entity-Relationship Model - towards a Unified View of Data,* ACM Transactions on Database Systems 1 p. 9 - 36, 1976.

Creemers M.R., *Transaction Engineering: process design and information technology beyond interchangeability,* 1993 (ISBN 90-9006253).

Date C.J., *An introduction to database systems,* volume 2, Addison-Wesley Publishing Company, 1983.

Davenport T. H., *Process innovation: reengineering work through Information Technology,* Harvard Business School Press, Boston Massachusetts, 1993.

Ediforum, *National EDI-guide 1992/1993,* Ediforum, Leidschendam, 1992 (in Dutch).

EDIT, *EDI Development and Implementation Tool,* version 2.4, User Manual, Bakkenist Management Consultants, 1993.

EDP Analyzer, *The rise of 'cooperative' systems,* Volume 25, No. 6, june, 1987.

Ellis C.A., Nutt G.J., *Modelling and Enactment of Workflow Systems,* in Proceedings Petri Net conference, Springer Verlag, 1992.

Encarnacao J.L., Lockemann P.C. (Eds.), *Engineering databases, connecting islands of automation through databases,* Springer-Verlag, 1987.

Estelle, ISO 9074, *Information Processing Systems - Open Systems Interconnection - Estelle - A formal description technique based on an extended state transition model,* ISO, 1988.

ExSpect, *User Manual,* Technical University of Eindhoven, 1990.

Genrich H.J., Shapiro R.M., *Formal verification of an arbiter cascade,* in K. Jensen, ed., Application and theory of Petri nets, volume 616 of Lecture Notes in Computer Science, pages 205-223, Springer-Verlag, June 1992.

Gielingh ir W.F., *General AEC Reference Model (GARM),* TNO Building and Construction Research, BI-88-150, okt. 1988.

Gifkins M., Hitchcock D., *The EDI Handbook,* Trading in the 1990s, Blenheim Online, 1988.

Goor A.R. van, Ploos van Amstel M.J., Ploos van Amstel W., *Physical distribution,* Stenfert Kroese, Leiden/Antwerpen, 1989 (in Dutch).

Hammer M., Champy J., *Reengineering the corporation, a manifesto for business revolution,* Nicholas Brealy Publishing Limited, 1993.

Heck H.W.G.M. van, *Design management of EDI systems,* Samsom Bedrijfsinformatie, 1993.

Hee K.M. van, Verkoulen P.A.C., Data, *Process and Behaviour Modelling in an Integrated Specification Framework,* Technical University of Eindhoven, concept, february 21, 1991.

Hee K.M. van, *Information Systems Engineering: a formal approach,* Cambridge University Press, 1994.

Hofman, W.J., *EDI handbook, electronic data exchange between organizations,* Tutein Nolthenius, Amsterdam 1989 (in Dutch).

ISO 7498 Information Processing systems - *Open Systems Interconnection* - Basic Reference Model, ISO, 1984.

ISO 9735, *Electronic Data Interchange for administration, commerce and transport (EDIFACT)* - Application level syntax rules, ISO, 1988.

ISO CD 9735:1993(E), *Electronic Data Interchange for administration, commerce and transport (EDIFACT)* - Application level syntax rules, United Nations, 1993.

ISO/DP 10026-1,2,3, *Information Processing Systems - Open Systems Interconnection - Distributed Transaction Processing (part 1: model, part 2: service definition, part 3: protocol specfication),* 1989.

ISO/IEC JTC 1/WG 3 N255, *Open-EDI Reference Model Standard* - Working Draft 1993.

I/S analyzer (formerly EDP Analyzer), *The strategic value of EDI,* Volume 27, No. 8, August 1989.

Jensen K., *Coloured Petri Nets: a high level language for system design and analysis,* published in: G. Rozenberg (Ed.): Advanced Petri Nets 1990, Lecture Notes in Computer Science, Springer-Verlag.

King W.R., Grover V., Hufnagel E.H., *Using information and information technology for sustainable competitve advantage,* Information and Management, 17 (1989), pp. 87-93, 1989.

Koop C., *ExSpect prototype of an actor in external logistics,* Technical University Eindhoven, december 1992 (in Dutch).

Kreuwels C.M.A., *Integration of external logistics based on EDI, to a multi-level supply control between supplier and buyer,* Kluwer Techniek, Deventer, 1994 (in Dutch).

Nederveen ir. G.A. van, *A building data model excercise using the GARM approach,* A contribution to COMBINE Report, Working Draft, TNO Building and Construction Research, May 1991.

Nijssen G.M., Halpin T.A., *Conceptual Schema and Relational Database Design,* Prentice Hall, 1989.

Özsu M.T., Valduriez P., *Principles of distributed database systems,* Prentice-Hall International Editions, 1991.

Petri C.A., *Kommunikation mit Automaten,* PhD thesis, Institut für Instrumentelle Mathematik, Bonn, Germany, 1962.

Petri C.A., *Introduction to general net theory,* in W. Brauer, editor, Net theory and applications: Proceddings of the Advanced Course on General Net Theory, Processes and Systems, volume 84 of Lecture Notes in Computer Science, pages 1-20, Springer-Verlag, 1980.

Porter M., *Competitive advantage: creating and sustaining superior performance,* The Free Press, New York, 1985.

Rambags P.M.P., *Automatic verification of Regular Protocols in P/T Nets,* Computing Science Note 93/42, Eindhoven, December 1993.

Rambags P.M.P., *Decomposition and Protocols in High-level Petri nets,* PhD thesis, Eindhoven University of Technology, 1994. To be published.

Reisig W., *Petri Nets: An introduction,* Prentice-Hall, 1985.

Royal Nedlloyd N.V., *Nedlloyd Transport and Logistic Glossary,* Bureau Nedlloyd Standards, version 1, 1989.

Sadwhani A.T., *Electronic systems enhance JIT operations,* Management Accounting, 1987.

Schultz J.F.H, *EDI: game of chance, game of power, or cooperation,* Research to the influence of environmental and group processes on EDI development projects, Samsom Bedrijfsinformatie, 1994 (in Dutch).

Sokol P.K., *EDI, the competitive edge,* Intertext Publications, McGraw-Hill Book Company New York, N.Y., 1989.

Spivey J.M., *Understanding Z: A specification language and its formal semantics,* Cambridge Universtity Press, 1988.

Stone B.K., *One to get ready, How to prepare your company for EDI,* Published by The CoreStates Banks Philadelphia National Bank, Hamilton Bank and New Yersey National Bank, 1988.

Streng R.A.G.J., *Dynamic modelling to asses the value of EDI,* A study in the Rotterdam port community, 1993.

Suomi R., *What to take into account when building an interorganizational information system,* Turku School of Economics and Business Administration, Turku, 1989.

Suzuki I., *Formal analysis of the alternating bit protocol by temporal Petri nets,* IEEE Transactions on Software Engineering, 16(11):1273-1281, November 1990.

TDID, *Trade Data Interchange Directory,* version 91.1, United Nations, 1991.

Tilanus C.B., *External logistics and external constraints,* reprint Bdk/310, Transport, jrg. 3, 7 en 21 juli, 2 en 18 augustus, 1990 (in Dutch).

UN/ECE WP.4 GE.1, *Recommendation on Interactive EDI,* version 3.1, October 1992.

UN/EDIFACT, *Business and Information Modelling Guidelines,* Working draft version 2.2a, july 1993.

Vlist P. van der, *Telematic Networks,* Tutein Nolthenius, Amsterdam, 1987 (in Dutch).

Vlist P. van der, et al., *EDI in trade,* Samsom Bedrijfsinformatie, Alphen aan den Rijn, 1992 (in Dutch).

Vlist P van der, et al., *EDI in industry,* Samsom Bedrijfsinformatie, Alphen aan den Rijn, 1992 (in Dutch).

Wierda F.W., *Developing Interorganizational Information Systems,* Delft, 1991.

Williamson O.E., *Markets and Hierarchies: analysis an antitrust implications,* Free Press, New York, 1975.

Womack J.P., Jones D.T., Roos D., *The machine that changed the world: the story of lean production,* Harper Perennial, New York, 1991.

Womack J.P., Jones D.T., *From lean production to the lean enterprise,* Harvard Business Review, March-April 1994.

Index

Samenvatting

Actoren, bijvoorbeeld ondernemingen of afdelingen van ondernemingen, gaan in toenemende mate over op het klantgericht leveren van produkten en diensten. Elektronische informatieuitwisseling en automatische verwerking van die informatie speelt daarbij een steeds grotere rol. Dit proefschrift richt zich op het ontwikkelen van concepten voor bedrijfsprocessen en een conceptueel model voor de uitwisseling en verwerking van informatie. Het doel is het ontwikkelen van generieke software voor de informatie-uitwisseling ter ondersteuning van bedrijfsmatige processen. Er is een conceptueel model gemaakt van deze generieke software en dit conceptueel model kan gebruikt worden als formele specificatie voor de bouw. Er is een eerste prototype gemaakt waarmee de haalbaarheid is aangetoond.

De bedrijfsmatige processen zijn gemodelleerd als 'timed, coloured, hierarchical Petri nets'. De generieke software is ook gemodelleerd met 'timed, coloured, hierarchical Petri nets', waarbij de kleuring van de Petri netten is weergegeven met het functionele datamodel. De principes voor het modelleren van de interactie zijn afgeleid van de ISO/OSI standaard voor gedistribueerde transactieverwerking. Verder is gebruik gemaakt van de gelaagde decompositiemethode van het ISO/OSI Basic Reference Model voor de ontwikkeling van transactieprotocollen.

Het conceptuele model leidt tot de ontwikkeling van nieuwe generieke software (BTMS: Business Transaction Management System) die als uitbreiding van de tools op platforms (tools zijn bijvoorbeeld een DBMS (Data Base Management System) en een UIMS (User Interface Management System)) gezien kan worden. De functionaliteit van een BTMS is in de huidige systemen specifiek voor een applicatie.
Een BTMS werkt op basis van procedures voor het afhandelen van onder handen werk. Een procedure ondersteunt een dienst van een actor. Als zodanig kan een BTMS de afhandeling van werkzaamheden door een actor ondersteunen.
Er is een grote economische behoefte aan dergelijke software, aangezien applicatie specifieke software te duur is en de samenwerking tussen bedrijven van grote economische waarde wordt geacht. Deze samenwerking is echter pas haalbaar met een generieke BTMS.

A conceptual model of a
Business Transaction Management System

STELLINGEN

behorende bij het proefschrift

A conceptual model
of a
Business Transaction Management System

van

Wout Hofman

Eindhoven, 13 september 1994

I

Een Business Transaction Management System maakt het mogelijk informatiesystemen met relatief weinig inspanning en in korte tijd aan te passen aan nieuwe of vernieuwde bedrijfsprocessen en organisatiestructuren [1].

[1] hoofdstuk 7 van dit proefschrift.

II

Een Workflow Management System zoals gedefinieerd door Ellis en Nutt [1], is een instantiatie van een Business Transaction Management System voor het besturen en beheren van documentstromen [2].

[1] Ellis and Nutt, *Modelling and Enactment of Workflow Systems*, in Proceedings Petri Net conference, Springer Verlag, 1992.

[2] hoofdstuk 6 van dit proefschrift.

III

De integratie van een Business Transaction Management System en CASE-tools werkt kostenbesparend.

IV

De toepassing van een Business Transaction Management System zal de rol van EDI-organisaties structureel veranderen.

V

De directe baten van het vervangen van documentstromen door elektronische berichten zijn voor de verzender hoger dan voor de ontvanger [1], [2].

[1] Stichting UTC, *Een verkenning van kosten en baten van EDI bij verladers en vervoerders*, 1994.

[2] Meer P.C.M. van der, *EDI met de toeleveranciers*, in documentatie Nationaal EDI congres 1993, Stichting Ediforum, 1993.

VI

De werkgroep Business and Information Modelling van de EDIFACT Board van de Verenigde Naties [1] dient zich meer rekenschap te geven van de onderliggende structuur in de communicatie tussen organisaties [2]. Alleen dan is het mogelijk stabiele standaarden te ontwikkelen.

[1] UN/EDIFACT, *Business and Information Modelling Guidelines*, Working draft version 2.2a, july 1993.

[2] hoofdstuk 6 van dit proefschrift.

VII

Zoals de toepassing van EDI in alle sectoren economische voordelen geeft [1], [2] en [3], zal EDI ook aanzienlijke voordelen geven bij de ontwikkeling, de invoering en het beheer van het berichtenverkeer zelf [4].

[1] Hofman, W.J., *EDI handbook, electronic data exchange between organizations,* Tutein Nolthenius, Amsterdam 1989 (in Dutch).

[2] Vlist P van der, et al., *EDI in trade,* Samsom Bedrijfsinformatie bv, Alphen aan den Rijn, 1992 (in Dutch).

[3] Vlist P van der, et al., *EDI in industry,* Samsom Bedrijfsinformatie bv, Alphen aan den Rijn, 1992 (in Dutch).

[4] Shepherd Alan, Hofman Wout, *EDI Management - Its application in Automotive supply,* in Proceedings of the 5th World Congress of EDI users, 1994.

VIII

Alleen met duidelijke doelstellingen voor telematicabeleid in het verkeer en vervoer is het mogelijk om de resultaten van een dergelijk beleid te toetsen [1].

[1] Ad-hoc Commissie Telematica Verkeer en Vervoer, *Telematica in verkeer en vervoer,* Raad voor Verkeer en Waterstaat, 1993.

IX

De praktische bruikbaarheid van formele methoden en technieken is groter naarmate men relatief meer tijd besteedt aan de toepassing van die methoden en technieken in concrete situaties dan aan het vergelijken met andere methoden en technieken.

X

De creativiteit nodig voor het schrijven van een proefschrift en aanverwante stellingen is maximaal op het moment dat men bezig is met volledig andere activiteiten.