Higher-Order Object Nets and Their Application to Workflow Modeling

Michael Löwe[1], Dietmar Wikarski[2] , Yanbo Han[1]

Abstract

As more and more office work gets automated or computer-aided, the demand for computer-aided cooperation and coordination emerges and increases. It becomes a more apparent challenge to electronically provide all necessary information and suitable software tools to the right person at the right time for the right job. A workflow management system is a software tool dedicated to such tasks. Instead of being implicitly embedded in the application programs, processes in a workflow system should be explicitly described, managed and enacted in order to support system prototyping and evolution, and to facilitate coordinative and cooperative communication. To methodically support system analysis and to manage the complexity and concurrency, a conceptual model is needed. As such a model, Petri nets have extensively been used in process-centered environments. For modeling workflows, however, Petri nets and their variants still show some weaknesses. For instance, due to their inflexible structure, net models are rather incapable to cope with evolutionary or exceptional dynamics. Openness in nets and dynamic configuration are almost untouched issues. The mechanisms supporting abstraction and compositionality are still a weak point of the most net models proposed so far.

In this paper, we will propose a new petri-net-based process modeling language: Higher-Order Object Nets that can better cope with some special modeling demands of workflow systems without loss of generality. After the introduction, the paper begins with definitions of workflow systems and workflow management systems and with a discussion about some modeling demands with which the traditional Petri nets have difficulties. Thereafter, the application of Higher-Order Object Nets is suggested and some further developments w.r.t. Higher-Order Nets [LöHa94] and Object Nets [Wika90] are presented. The connection of the object-oriented concepts to Petri nets is also going to be further investigated. To describe the static structural aspects of application domains, an object model and an organization model are incorporated into our methodology. Finally, based on the interpretation of Higher-Order Object Nets and DOC (Distributed Object Computing), a workflow management system architecture is proposed.

1) Computergestützte Informationssysteme(CIS), Technische Universitat Berlin
EN7, Einsteinufer 17, 10587 Berlin, Germany
loewe@cs.tu-berlin.de, yanbo@cs.tu-berlin.de
2) Fraunhofer-Einrichtung fur Software- und System-Technik (ISST)
Kurstr. 33, 10117 Berlin, Germany
dwikar@isst.fhg.de

Contents

0. Introduction

Nowadays, the designers of modern office information systems face many problems that are different from the ones met in designing traditional information systems. Business environment and conditions change very quickly. New and normally more powerful standard software components for office information processing appear continuously. Therefore the corresponding software systems must have a more flexible structure and more flexible interfaces to cope with evolutionary dynamics. Traditionally, working processes are not explicitly specified. The processes supported by an application system are not separated from the functional components and the database management component of the system. This results in an inflexible process structure and prevents the system from easy process evolution or redesign. On the other side, the demand for computer-aided cooperation and coordination of (groups of) persons who normally operate concurrently also increases as more and more single pieces of office work are either fully automated or computer-aided. In addition to analyzing data and functional data transformation, process aspects must also be explicitly considered now. In the meantime, "Business Process (Re)engineering (BPR)" has become a buzzword. It aims at the (re)design and optimization of an organization‘s business processes to gain more efficiency and more effectiveness. Here, process modeling and control are also playing a central role.

For the above-mentioned reasons, the management of working processes in organizations has become a hot topic and many software systems for process management have recently been developed. As working processes of an organization are often called workflows, these software systems are consequently called workflow management systems (WfMS). A workflow application system which is also simply called a workflow system (WfS) is composed of a WfMS, the real-world workflows being controlled by the WfMS, as well as the material, software and human resources relevant to the workflows. In this paper, workflows are considered as a special kind of processes that represent the execution orders and the interactions of activities within an organization. Therefore, we use the two words - workflow and process - as synonyms when we talk about office information processing and no confusion arises. But referring to the Higher-Order Object Nets described in this paper, we use the term "process model" instead of "workflow model" just to stress their generality. Although many research prototypes or commercial products are categorized as workflow management systems, their philosophy, emphasis, and functionality differ very much. In our opinion, a WfMS should put emphasis on the following premises in addition to the commonly-recognized functions such as provision of graphical user interfaces and database access to shared pieces of information.

- A WfMS should be based on a formal and executable process model in order to methodically facilitate system analysis, simulation and enaction and to separate processes from application programs so that the modification or evolution of processes and the sharing of process segments can be better supported.
- The application structure of a WfS, i.e. the arrangements and interactions of the software components and manual activities, should solely be determined by its process model. Workflow management systems should control workflows by interpreting the process model. Just as a database management system separates (shared) data from application programs, the workflow interpreter separates (interrelated) processes from application programs to reach a new level of application independence.
- A WfMS should be repository-based. Based on the workflow repository, a workflow management system should be able to manage the resources[1] relevant to the workflows, including information resources, software components[2], human resources, and reusable model segments.
- Fig.1-1 compares the system architectures of a traditional information system and a WfMS-based application system and highlights the main difference, namely, in WfS's the process information is separated from application programs and is represented by a modifiable process model. Each subsystem supports some specific activities and their activation and interaction are controlled by the workflow management system. In a WfS, interpersonal interaction is realized or supported by the interactions between software components. A natural requirement is that a WfMS should have an open and flexible application interfaces so that new standard or application-specific software components can be easily integrated.
- The behavior of an application system is determined by the internal behaviors and the composition of its components, as well as by the interactions between the components. The process supporting component compositions and interactions should support a decentralized strategy. To support interoperability of various distributed software components, a WfMS should incorporate mechanisms dealing with heterogeneity, autonomy, synchronization and consistency of these components in a distributed environment.

1. Compare subsection 1.1, the concept *resource* is defined there in the context of workflow systems.
2. Here, software components refer not only to plug-in software building blocks in the common sense but also to stand-alone independent software tools such as text editors. They are also called software services.

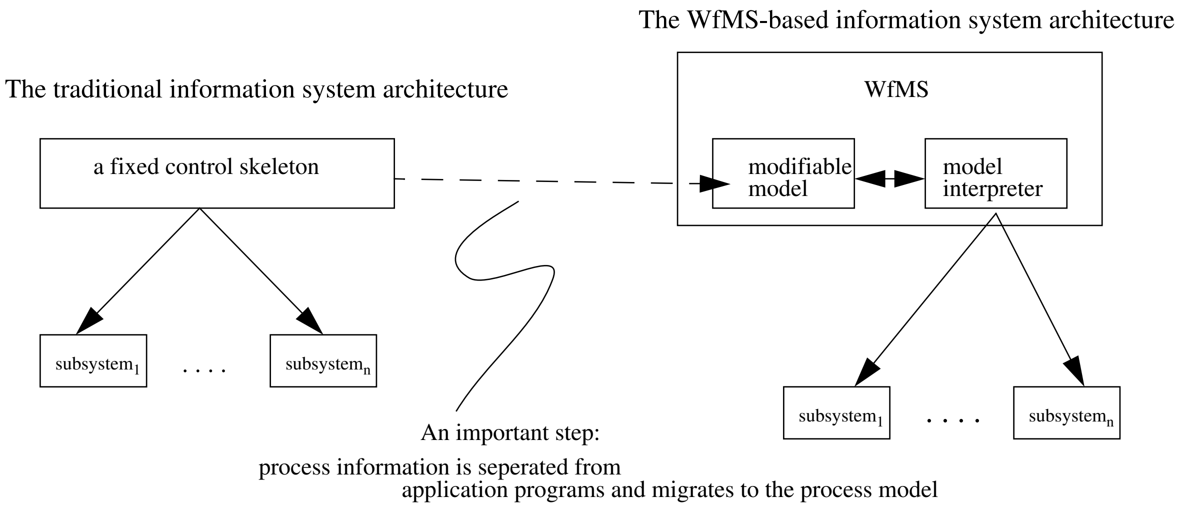


Fig. 1-1 Traditional and WfMS-based applicational system architectures

Although process description, documentation, analysis and simulation, as well as process monitoring all belong to features of a workflow management system, in this paper we will concentrate on a suitable process model and its interpretation, and on the resource management during the interpretation of the process model. Workflow models are far from being mature compared to, let's say, database models. We hope that our work can contribute a little to this field and that the process model proposed could find a broader application though it was originally developed for modeling workflows. A workflow interpreter which is based solely on the interpretation of the process model and is therefore application-independent is also proposed in this paper as the architectural core of a workflow management system.

Among the formal and semi-formal behavioral models, Petri nets [Pete81][Reis86][Baum90] are a very prominent one. On the one hand they have an intuitive graphical representation, and on the other hand they also have a well-defined semantics and a formal background which allows system analysis and simulation. Petri nets are based on the automata theory, but they allow local states and therefore allow true concurrency. They are very suitable to model concurrent and non-deterministic processes. Moreover, Petri nets are an executable model provided that the firing rules and the ways of conflict solution are specified. A short review of Petri nets and their applications to developing information systems can be found in section 2.

Petri nets are finite, bipartite and directed graphs with two disjointed sets of nodes indicating passive and active entities of the modeled system, for example, states and state-changing activities, or buffers and processors, or data and computation steps, and so on. State changes are graphically simulated by token movement. Since the introduction of Petri nets, a lot of modifications have been made in order to solve various theoretical and practical problems. High-level Petri nets, such as Colored Petri Nets (CPN) [Jens92], Predicate/Transition Nets (PrT) [GeLa81], and Algebraic High-level Petri Nets (AHL-nets) [EhPa93], have been introduced to allow individual and structured tokens. With the evolution of the net theory, attributes for edges and nodes become a constituent part of Petri nets. Practically, Petri nets have found a wide range of applications including communication protocols, distributed software systems, flexible manufacturing, software development process management, information system development, etc [Mura89][ScRe93]. Recently, the application of Petri nets to the modeling of workflows attracts increasing attention and again some variants of Petri nets have been proposed [Laus88][Gruh91][ElNu93][WoKr93].

For modeling workflows, Petri nets and their variants still show some weaknesses. For instance, one problem of such models is their incapability to cope with evolutionary and exceptional dynamics. System evolution is normally unavoidable because business processes evolve continuously caused by internal organizational reforms, external environmental changes, etc; new software components are continuously introduced whereas some old ones are discarded; and each business process has its own life cycle. On the other side, many exceptional situations in WfS's are unpredictable or unexpected "routines", but they are impossible or difficult to be captured in the modeling phase and may, therefore, cause run-time modifications to the application system structures. The evolutionary and exceptional dynamics raises a high demand on the structural flexibility of the conceptual models. Some inherent problems of Petri nets and most of their high-level variants become more apparent here because the net structure, firing predicates of transitions, as well as actions associated with transitions are all set down at the modeling phase and dynamic binding and dynamic (re)configuration can not be supported. Conceptually, most net systems are closed systems: except for the initial marking, state changes will not be influenced by external factors and all intermediate states and end results can only be derived from the initial marking. Run-time communication with the external environment is normally excluded. It is implicitly assumed that everything has been modeled and included in the net system. This assumption does not work well in workflow system where human factors dominate. The "closed world" hinders the workflow model from incorporating an effective exception handling mechanism supporting run-time user intervention and dynamic (re)configuration.

In reality, interfaces between a process model and its environment and interfaces between running instances of the process model need to be well defined. In Slang [BaFu93], tokens can be added manually to some special places to support user interaction, but external resources are not treated systematically and bidirectional interactions are not supported. In Process Nets [Wika95], a special kind of transitions (sensors) can detect triggering signals from the external environment, but the signals contain only control information in a predefined context. Resource management is not explicitly embodied in Process Nets. Some other related work will be discussed in section 2.

In [LöHa94], we had proposed Higher-Order Nets (HON) which generalize the data types that can be used in nets to function spaces and power sets. As a result, functions can be modeled as tokens. Higher-Order Object Nets (HOON) are a further extension where some other mechanisms, such as clustering and behavioral abstraction of nets as well as object-oriented techniques, are introduced into nets. The concept *meta type* as a multiset of types and the concept *meta place* as a container of normal places are introduced into the model. Meta places make the net graph more compact and generalize the interfaces between nets. Unlike in other petri-net-based models, each transition in HOON may own an interface place called *control place*. Tokens on control places are called *control tokens*. Through exchange of control tokens, a net

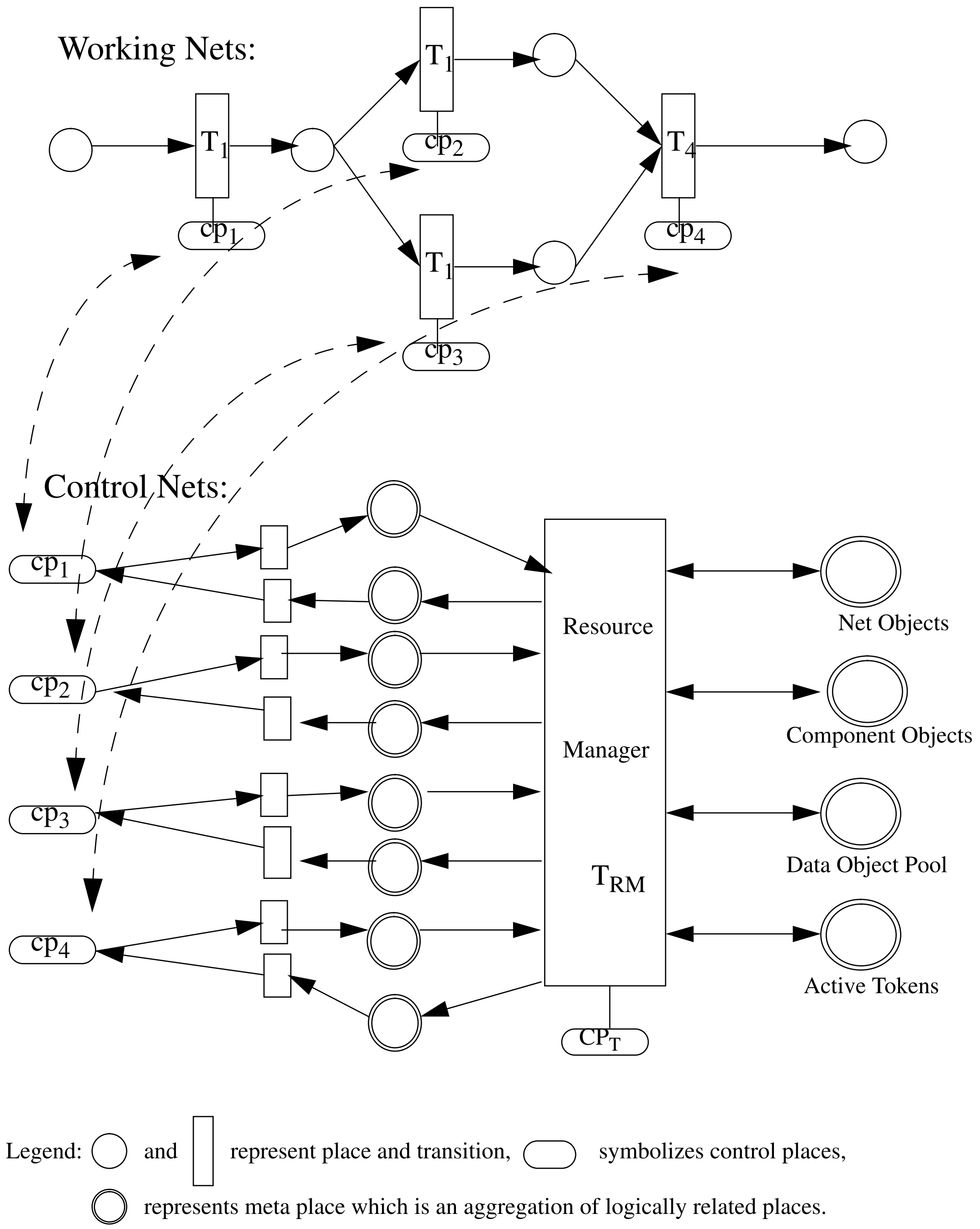


Fig 1-2 An example diagram of a HOON workflow model

can interact with the "external world". According to different control policies, there are many ways to provide tokens for control places. For example, tokens on a control place can be put manually, by a software tool or by another net. For modelling and controlling workflows, it is proposed that HOON consist of two classes of nets and a set of interfaces between the two classes of nets. One class corresponds to "normal" Petri nets consisting of an underlying net structure, simple data types, net inscriptions, etc, and can be used for modeling information flows focusing on the control structure, concurrency and synchronization aspects of the application. We call this class of nets *working nets* which are, not considering control places, comparable to CPN. The other class of nets is used to model parts of the "outside world" which are relevant to resource management (in the context of workflow management). This kind of nets is called *control nets* consequently. Our intention is the following: we separate resource management issues from workflow models. Only proxies of resources appear in the workflow model. Resources are managed by the control nets which are modeled independently and run in the background.

Fig. 1-2 shows an example diagram of a HOON workflow model. The diagram shows an example working net and indicates the generic control net structure which is not dependent on application problems being modeled and is not influenced by the hierarchies of the corresponding working nets. The interfaces between the working nets and the control nets are a set of control places which belong to both classes of nets. For the working nets, control places serve as a means to interface with workflow repository and the external world. For the control nets, runtime control and intervention to the workflows, including provision of suitable services, dynamic modification of the process structure, and dynamic binding of resources, can be well supported with the help of these open interfaces.

Onto a control place we can assign subnets encapsulated as a kind of objects, encapsulated software components as another kind of objects, persistent data objects as a third kind of objects, or actors being either persons or functional program codes which bear analogy to the jobs in Funsoft nets. In the given example, depending on the control tokens on the control place of the transition T_{RM}, a software tool or a subnet may be activated when T_{RM} fires. In this paper, as indicated in fig.1-2 and later in fig.3-5, we only associate a software tool - a resource manager - to the transition T_{RM}. Based on the OMG‘s specification for distributed object computing - Common Object Request Broker Architecture (CORBA)[OMG91], the above-stated resources can be encapsulated with the interface definition language (IDL). The resource manager can treat all resources available in a distributed environment as a commonly accessible collection of heterogenous objects.

A lot of people have strived to make Petri nets "object-oriented", see, for example, [BaBr88][BuGu91][Sibe93][Verk93][Lako94]. Object-orientation is characterized by the following principles or techniques: abstraction, encapsulation, classification and instantiating of classes, inheritance, and polymorphism. It is worth discussing whether it could be possible to define an object-oriented version of Petri nets in the sense of pure object-orientation. We doubt that because of some intrinsic distinguishing characteristics of Petri nets such as the intentional and explicit separation of states and state changes, the emphasis on the flow relation, and the special ways to model concurrency and nondeterminisms. Furthermore, it is difficult to define and apply class hierarchy and object inherientance at the net level. As a common practice proposed in the literature about "object-oriented" Petri nets, tokens are often used to represent objects in the sense of object-oriented programming. Such objects have their own life cycles, identities, and local memories. But in the Petri net world, tokens are not persistent. According to the standard Petri net definition, transitions do not hold any data and tokens are dynamically created or destroyed with each transition firing. Solving conflicts in nets also needs extra mechanisms which have no counterpart in the object-oriented technology and may prohibit the realization of message passing between objects (For this problem, suggestions have been made e.g. by [Wika90] to keep all potential conflicts within the objects which correspond to a special kind of subnets. This idea is also incorporated in HOON, compare the next paragraph). Of course, it is not important whether a Petri net variant can be called object-oriented or not. More important is that we add nice features and enhancements to Petri nets by introducing object-oriented techniques and other useful mechanisms.

As mentioned above, we distinguish objects at different levels of granularity, namely *net objects* at the net level, *component objects* at the subsystem level, and *data objects* at computation level. We use *objects* to generally represent all of them. Objects are represented by control tokens in HOON. Tokens on the normal places of the working nets can represent either normal data values as in CPN or messages which either serve as references showing the availability of certain objects or can directly model the messages passed between objects. Each object token may carry information about its activation conditions. Such information is expressed in form of predicates. Moreover, an object token has its own life cycle and is persistent, i.e. an object may exist longer than a single transition firing. On the contrary, the reference tokens or message tokens in the working nets are transient.

Net templates are structural abstractions of subnets, i.e. they are skeletons with topological and type information of the net elements. Net objects are instances of the net templates. When we assign control tokens to the control places of a net template, we obtain a net object. Net templates have well-defined public input and output interfaces. The internal states and internal state changes are determined by sets of internal places and transitions. Net objects may be activated by the "input" interface transitions (or interface places, in this case the places serve as "input buffers") and may act upon "output" interface places (i.e. input buffers of other net objects) by firing the "output" interface transitions. In the construction of net templates, some object-oriented concepts such as encapsulation, object identification, behavioral aggregation are applied in order to encapsulate certain, semantically connected states and transitions. Our construction method has, among others, the characteristic property that the transitions which are in a potential conflict, (i.e. they have a common preplace and, in case of nets with capacities, a common postplace), are kept within one net template [Wika90]. In contrast to component objects or data objects, the behaviors of the net objects are again determined by nets and can be described by its input (I) and output (O) ports and its $I \rightarrow O$ relations. When a net object token on a control place is unpacked, a net of one level lower will be instantiated. Some control places of the instantiated net may again contain net object tokens. In this way, an abstraction mechanism and a way of hierarchical net construction are introduced.

Component objects are encapsulated software components which have special functionality and provide services through defined interfaces. During a transition switching, a component object can be activated if its activation predicate is evaluated to true or if a definite triggering message arrives. The main differences between a component object and a piece of functional program code like, for example, the jobs in Funsoft nets are: (1) a component object has no definite association with a definite transition and certain input/output places and can be persistent over time, it corresponds to a subsystem having internal data and functions, and therefore has certain degree of autonomy. (2) the duration and results of activating a component object may be unpredictable because the component object may be "open-looped", that is, it may have its own user interfaces through which it may receive unpredictable control information. (3) a component object is encapsulated with an interface specification language, it interacts with its "outside world" only through the defined interfaces.

Data objects are data entities constructed and encapsulated with an object-oriented method. Depending on different methods employed, complete object-oriented concepts including class hierarchy, inheritance and polymorphism can be supported. In HOON, the structures and relationships of data objects are described separately in the accompanying object model. In nets, the differences between data object tokens and simple data tokens lie in that (1) data within the data objects are accessible only through the services of the data objects rather than by an arbitrary action associated with the corresponding transition. (2) data objects are neither pure passive entities nor pure active entities, therefore, can not be simply associated with transitions or assigned to places in CPN or in Funsoft nets. (3) data objects interact with messages carrying triggering information. In section 3, we will introduce the message passing scheme to facilitate data object interactions in nets.

Summarizing, we are striving to develop a process model which ultimately possesses the following properties:

- having a more flexible process structure without loss of generality
- being able to model both static and dynamic properties of a target system
- including a suitable abstraction mechanism for hierarchical model construction, supporting reuse and (run-time) composition of model segments
- being enactable, incorporating a flexible process interface for run-time communication with the running environment of the process model, supporting run-time user intervention and run-time binding of resources
- optimizing the work-load balance between model construction and model enaction
- being suitable for modeling workflows and helpful for building a flexible distributed workflow system architecture

After this introduction, we start the next section with definitions of workflows, workflow systems, and workflow management systems. We will also summarize some special modeling demands for structural flexibility of workflow models. In section 2, a short review of Petri nets and some existing workflow systems in which Petri nets have been used as a conceptual model is given. Beginning with the introduction of the main features, we present the petri-net-based process modeling language - Higher-Order Object Nets - in section 3. For designing and implementing complex information systems, use of different models is often mandatory because no single model is a panacea. Section 4 discusses about complementing HOON with an object model and an organization model. In section 5, we will propose a general workflow interpreter which is based on the interpretation of the process model. To manage heterogenous resources and support interoperability of the underlying software components, the de-facto-standard of distributed object technology - CORBA - is proposed as the basic architectural platform of the proposed WfMS. Finally, we draw a short conclusion in section 6.

1. Managing Workflows for Business (Re)Engineering

The coordination of distributed work becomes the main challenge in a large and complex information system. Much research effort has been initiated in recent years to develop concepts and methodologies in order to make distributed agents (including both people and machines) cooperate more efficiently. The new research field CSCW (computer supported cooperative work)[Grei88] has consequently established which has a very broad coverage from social and linguistic aspects to system specification techniques, implementation strategies and data sharing methods. Concretely speaking, CSCW is an information technology which can be used to help people work together more effectively and is concerned mainly with coordinated access to shared resources in a distributed environment. Workflow management systems are a special class of CSCW systems for office procedures. In this paper, according to [ElNu80], an office means the part of business that handles information, and office work consists of information-handling activities and communications within and between offices. In the following, we try to define workflow systems and workflow management systems. But first, because we talk a lot about resources in this paper and the central idea in HOON is that tokens can be used to represent instances of all kinds of resources, let us explain what is meant by "resources" in the following subsection.

1.1 Resources

The goal of business process (re)engineering is to (re)organize and (re)arrange the following basic entities optimally: a set of business objects, a set of business activities, the staff who performs the activities, programs that support the people, and the control and information flows between the activities. These concrete *physical resources* of an organization are further described hereinbelow:

- *Business objects* are work items to be handled within workflows. Custom forms, program documents, archive folders, purchase orders, delivery notes and bills can all be treated as business objects. A business object can be atomic or compound. An atomic business object can not be further divided meaningfully from a given viewpoint whereas a compound business object may consist of atomic business objects and/or other compound business objects. Some business objects may have an internal "memory" and may be governed by its own business rules.
- An *activity* is a unit of work stimulated by a request of service, notification of results or occurrence of some events. When an activity is performed, we call this "running" activity an *action*. An action acts upon business objects. As an action is an instance of an activity, one activity may correspond to many actions.
- The set of relations corresponding to the communication links between actions also belongs to the system resources.
- Activities are performed by actors. An *actor* can be either a manual actor who is a person or an automatic actor which is a piece of functional software code. Correspondingly, an action is attributed with manual, automatic or partly automatic when the corresponding activity is performed by a manual actor, an automatic actor or both.

- Each actor plays one or more roles in a workflow system. Roles correspond to positions in the organization structure. The actors are associated with the activities through roles. Moreover, roles can also be used to control the execution right and other privileges.

When the physical resources are mapped to the information processing world, they are normally abstracted to conceptual entities in forms of data or functions before any implementation takes place. It is these entities that a workflow model will be built upon. Here, we call them *conceptual resources*. In a broader context, we refer to both conceptual resources and their implementation-relevant forms as *resources*. Among conceptual resources, we differentiate passive entities, active entities, and object entities.

- *Passive entities* include simple passive data and passive data structures. By passive we mean that they are operands in the system. For instance, data of atomic data types like integer, boolean or string are simple passive data. Stacks, queues or database records are passive data structures.
- *Active entities* are operators in a workflow system. People and software tools are often modeled as active entities because they take actions by performing activities. Actions are naturally active because they manipulate business objects. In a broader sense, a workflow itself is an active entity.
- An *object entity* contains both passive entities and active entities. Except for those interfacing active entities, both the passive and the active entities are private to their object. The passive entities can only be manipulated by the active entities of the object and accessed from the outside through the interfacing active entities. Well-defined segments of workflows can be treated as a single entity having the above-stated characteristics of an object entity. In this sense, depending on different viewpoints, a workflow can also be modeled as an object entity.
- An entity can be hierarchically composed of an aggregation of several entities at a more fundamental level.

1.2 Workflow and Workflow systems

The terminology workflow is mainly used in the office information processing field. As a special kind of process, a *workflow* is a meaningful partially-ordered arrangement of the following elements: activities, business objects associated with the activities, as well as actors performing the activities. A workflow has normally an expected outcome, but deviations may occur which are called exceptions. As a matter of fact, a lot of real-world processes bear this common structure though they originated from different areas. What differs is contents, not approaches. Therefore, we regard software process as a special class of workflows. Warboys [Warb94] has discussed in detail about the differences between software processes and workflows. We believe that the difference lies in the business objects not in the process structure. Later in this paper, we will also review software process management systems MELMAC and SPADE [Gruh91] [BaFu93]. Conceptually, a workflow is made up of a collection of interactive autonomous active entities that act or execute sequentially or in parallel, consume and produce some passive entities or modifies states of some object entities.

A workflow system corresponds to a real-world organization. It consists of both the computer application system and the part of the real-world of concern. Given external stimuli as inputs, some definite outputs can be expected. Working processes dominate in such systems where a set of instantiated workflows are interwoven or run in parallel. A workflow progresses towards its defined outcome when its partially related actions take place as specified. The most actions need to utilize part or all of the results of their former actions, i.e. the generation of new passive entities or new states of passive entities or object entities.

Compared to processes in other fields such as technical processes, workflows have their own characteristics and therefore raise some special modeling demands, especially the evolutionary dynamics and exceptional situations caused by human factors and rapidly-changing business world. The structure of the workflow model must therefore be more flexible. Some of the main reasons are given below.

- A workflow system is normally distributed, and consists of many autonomous subsystems which run concurrently and interact with each other through exchange of control information, data or even software components. Each subsystem has its local states and autonomy. The communication connections of the subsystems must not be fixed because links between some components can only be determined at run-time.
- The main goal of a workflow system is to coordinate activities and people to complete a common task. Involving human factors is therefore a typical characteristic of such systems. People bring creative, uncertain, nonrepeatable, and unpredictable factors into the system. Therefore, a workflow system is subject to exceptional situations which are unpredictable and can only be encountered at runtime. It is rather difficult to reflect such situations in a process model which only supports a fixed topological configuration.
- A durable workflow system must be able to evolve in order to keep up with the changes caused by business environment, personnel change, and component updating. Even during the execution of a large workflow system, new business procedures may be created while some old ones are discarded. Sometimes, such changes must be dynamically captured and recognized by a "running" model.
- It is also very natural that some resources should be assigned to the actions dynamically, especially the scars resources or the resources subject to rapid changing.

In principle, Petri nets are an appropriate candidate for workflow modeling. But as we have mentioned above and in the introduction section, Petri nets suffer from some inherent problems. The main critic is their rather inflexible structure. Although high-level Petri nets can model the routinized part of a workflow system, they are inflexible when dealing with the above-mentioned situations.

1.3 Workflow Management Systems

A workflow management system is a special kind of software tool to assist the specification, analysis and enaction of workflows. It is a top-level application software between users and the software components responsible for specific tasks. Workflow management systems establish links between users, between software components, and between users and software components. The difference between traditional office automation systems and workflow management systems is the following: the traditional systems support only predefined routine work, and working processes are not systematically treated and explicitly managed, whereas the workflow management systems should be based on a formal process model and support routine work, unstructured creative work, cooperative communication between actions and actors, as well as exceptional situations which are both unavoidable and troublesome in real workflow systems.

Till now, there is no common agreement on a general definition of workflow management systems or workflow systems in general. Ellis and Nutt define a workflow management system as an application level software which helps to define, execute, coordinate and monitor the flow of work within organizations [ElNu93]. The Workflow Management Coalition (WfMC) has a similar definition: a workflow management system is a system that completely defines, manages and executes workflow processes through the execution of software whose order of execution is driven by a computer [WfMC94]. In this paper, we will not further deal with these "standard" functions and their realization, but concentrate on the workflow model and its interpretation.

The main focus of a WfMS is how to organize the object entities and active entities rationally and efficiently, and how to manage the use of the passive entities according to the need, priority, privilege, etc. The run-time snap of any workflow management system is nothing more than a hierarchical combination of object entities and active entities which manipulate passive resources. In the proposed methodology for developing WfMs‘s, emphasis is put both on the HOON process model and its interpretation and on the management of resources. A WfMS consists mainly of the following two parts: (1) the formal process model and the model interpreter, and (2) the resource management model and the resource manager. The formal process model provides a basis for system analysis and enaction. On the one hand, the process model must capture the most important static and dynamic properties of the workflows and satisfy the modeling demands of workflow systems, and on the other hand, it must be executable and be open to (run-time) modifications in order to use the model for control purposes besides understanding, analysis and simulation purposes. The behaviors of a WfMS are determined by the interpretation of its workflow model. Only proxies of resources appear in the process model and the resources themselves are managed according to the resource management model and by the resource manager. Although the two parts can be investigated and developed independently (a virtue of the proposed methodology), the model interpreter must cooperate with the resource manager to control real workflows.

In section 5, we will present an architectural diagram of a workflow management system which is based on our HOON process model. We will also discuss the interpretation of HOON and issues about repository support, user interface, and interface to software tools and application specific components.

2. Modeling Workflows with Petri Nets

2.1 Petri Nets

There are several standard types of Petri nets. Condition/Event (C/E) nets and Place/Transition (P/T) nets are normally referred to as the basic forms of Petri nets. We only give a definition of P/T nets here since C/E nets are special P/T nets whose place capacity is restricted to one. A P/T net $N = (P, T, F)$ consists of two sets of elements P and T, and a flow relation F between the elements of different kind. The P-elements are called places and the other kind of elements is called transitions. Places and transitions are disjunct, i.e. $P \cap T = \varnothing$. The flow relation $F \subseteq (P \times T) \cup (T \times P)$ corresponds to a subset of all arcs between P and T. For each transition $t \in T$, the preset of t, denoted as $°t$, contains the places "before" t, i.e. $°t = \{(p \in P) \mid (p, t) \in F\}$, and $t°$ denotes the successor places $t° = \{(p \in P) \mid (t, p) \in F\}$. $°t$ and $t°$ are also called preplaces and postplaces of t respectively.

The dynamic states of N are represented by net markings. A *net marking* is a P-indexed family of sets $M = (M_p)_{p \in P}$, where M_p denotes the set of tokens lying on the place p. Note that this conception of marking considers tokens as distinguishable entities. With this setup, the transition from P/T nets to high-level nets becomes much easier. The classical notion of P/T nets can be obtained from ours if we abstract from token names, i.e. if we pass from sets to isomorphism classes on sets (natural numbers). A *switching set* t^s for a transition t is a special marking satisfying $\left|t_p^s\right| = 1$ for each $p \in °t \cup t°$ and $\left|t_p^s\right| = 0$ otherwise. Here, $°t^s$ denotes the restriction of a switching set to the preplaces $°t$ and $t^s°$ to the postplaces $t°$. A *state transition* in N is defined as follows: the switching of a transition t is possible and changes the current state M to the state M', written as $M[t>M'$, if there is a switching set t^s and a marking $\overline{M}$ such that $M = \overline{M} \cup °t^s$ and $M' = \overline{M} \cup t^s°$. The tokens in $\overline{M}$ are the tokens both in M and in M', and are not affected by the current transition switching.

When all preplaces of a transition t are marked, t is said to have a concession or to be enabled. The transitions which are enabled behave according to the above-defined firing rule. Murata has summarized that the firing rule of transitions is the only rule one has to learn about Petri-net theory [Mura89]. In the standard definition, a transition may fire when it has concession, but must not do so immediately. In section 3, we will introduce some extra firing rules suitable for the modeling and enaction of workflows.

1.2 High-level Petri Nets for Modeling Information Systems

In P/T nets, the dynamic net markings are traditionally defined as a mapping $M = P \rightarrow \mathbb{N} \cup \{\omega\}$ indicating whether a place is marked or not and how many tokens lie on that place. However, different tokens at the same place can not be distinguished. For this reason, high-level Petri nets were introduced. High-level Petri nets refer to the nets which allow structured individual tokens in the net markings.

Petri nets have been considered as a promising tool to model information processing systems.

As a matter of fact, their first practical application was in the office automation field [Zism77][ElNu83]. The main focus of our research is to establish a suitable petri-net-based model which can satisfy the modeling demand of workflow systems on the one side and retain virtues of Petri nets, i.e. their graphic representation, their rigorous semantics, their analytical techniques on the other side. Several Petri net variants have been proposed in the past few years with the same or similar goals. However, some of the efforts have made a net model much more complex and therefore introduce much modeling overhead whereas some others touched only part of the problems. Funsoft nets [Gruh91][DeGu94] are among the successful ones. They have been developed as a workflow modeling language in the EUREKA software factory project ESF. Originally, our work is based on a careful analysis of Funsoft nets.

Funsoft nets are derived from Function nets which were developed at the Technical University of Berlin for the modeling of information systems [God83]. In Function nets, executable SIMULA-routines (called jobs) are attached to transitions and executed whenever their transitions fire. A job library can be defined for an application. Places contain typed tokens and can be accessed in different ways as a queue, a stack or a random array. Token types include boolean, real, and string. Data flows and control flows in the system are distinguished. The read-only form of data access can be modelled with a "copy edge". Time consumption is also considered. Furthermore, the firing behavior of Petri nets is modified to allow selective outputs. The job can determine the input and output behaviors of the transition, for instance, the transition can only produce a single token to one of many postplaces. Since their introduction, Function nets have been used e.g. in information system modeling, for the simulation of database machine architectures, in software process modeling and management, and as part of software development environments with automatic code generation.

In Funsoft nets, some concepts of Function nets are modified and extended. Funsoft nets provide a job definition language. The executable description of a job is a C-routine. Data types associated with places are also extended. A data type may be one of the following primitive data types: integer, real, boolean, string, text, and binary, or may be of complex data type which is an aggregation of the primitive data types. Other new concepts w.r.t. Function nets are: role concept, hierarchy concept, as well as predicates which define constraints on names and values of objects to be accessed during a transition firing. The semantics of Function nets is operationally defined while the semantics of Funsoft nets is defined in terms of PrT nets whose semantic definition is based on C/E nets - the very basic form of Petri nets.

High-level Petri nets, Function nets or Funsoft nets could all be thought of as decorated Place/Transition nets. In Function nets, the places are decorated with data types and the transitions are decorated with functions implemented with jobs. Functions themselves, however, could not be manipulated as "resources". They are bound to their transitions once and for all. This implies, among others, that all functions can be accessed arbitrarily and have unlimited copies. Problems will arise when we model, for instance, licensed or access-right restricted functions. These problems have motivated our first version of the process modeling language (HON).

In the following subsection, we summarize some other attempts of modeling workflow on the basis of petri-net-alike formalisms.

2.3 Some Related Work

- SCOOP

SCOOP [Ziss77] is an experimental system of workflow management developed by Zisman in the late seventies at the university of Pennylvania. The system was implemented with an execution monitor which is based on a kind of extended Petri nets with which offices are modeled as aynchronous concurrent processes. Transitions of the nets are augmented with production rules. SCOOP is one of the earliest application of Petri nets for the coordination of office work. Zisman emphasized that whole office procedures should be considered systematically rather than simply automating single office steps.

- DOMINO

DOMINO [WoKr93] is a very early german experiment on workflow systems intended to support structured business processes in organizations. Office communication is realized through message exchange rather than information sharing. A central mediator coordinates autonomous agents, each of which is responsible for a single step of the whole office procedure. The input/output relations of the office steps define an office procedure. DOMINO notifies the users about the action due, provides the information needed, and routs the results of the action to relevant actors.

A variant of Petri nets called CoPlanS has been used to describe office procedures. In CoPlanS, the basic Petri net structure is interpreted as a set of forms, a set of actions, a need relation between forms and actions, and a produce relation between actions and forms.

- MELMAC/ LEU

MELMAC is a software process management system which supports business process modeling, model analysis, and process model enaction [Gruh91][DeGr94]. The development of MELMAC was initiated in 1989 at the University of Dortmund, Germany. Funsoft nets is used as the modeling language in MELMAC and also in a commercial product LEU implemented by LION GmbH in Bochum, Germany.

In MELMAC, processes are managed by means of different views on the process model. The structure view contains information described by Funsoft net skeleton about process structure, relationships between activities, and relationships between data types and activities. The service view defines services such as tools that can be called from the process model. Project management view contains information about persons, time, and cost. Profile view specifies qualifications of the persons involved. Change view deals with some aspects of the evolutionary system dynamics. During process enaction, changes to the underlying process structure are possible. Modification of a subnet can be assigned as a job to a transition. After a subnet structure is modified, the old version is replaced by the new one. But before the modification, all instances of the subnet must terminate and during the modification no new instances of the subnet are allowed to be initiated. Since the action bodies associated with transitions are specified in C code, recompilation is needed if the code is modified.

- SPADE

Being developed in Milano, Italy, SPADE [BaFu93] is a process-centered software engineering environment where software processes can be modeled, analyzed, and enacted. The process modeling language SLANG is based on another extended version of high level Petri nets which is called ER-nets. SLANG supports dynamic evolution of process model by describing meta processes as part of the process model. Of course, this makes the process model more complicated. In SLANG, subnets are defined as activities. Tokens describing activities are kept in activity places and can be read and modified by transitions. It is possible to change the definition of an activity while some instances of this activity are still running. Compared with Funsoft method, SLANG has a more flexible modification mechanism, but the systematic management of other active entities such as software tools is not considered in SLANG.

- WAM

WAM is another workflow management system developed recently at the Fraunhofer Institute ISST, Germany. The WAM system supports asynchronous and synchronous distributed coordination of business processes. In WAM, work gets done by executing a task. A task is a meaningful work unit which is made up of ordered activities that work with the set of data organized in a folder explicitly associated with the task. A folder is a collection of documents including office files, access control, and a catalogue of software tools. Folders can be cut apart for distributed processing and merged afterwards. They are globally registered and controlled in order to assure that only one executor is allowed to physically update a folder each time. Each task is assigned to an executor or a group of executors who are responsible for its execution. Roles are attributes assigned to tasks, and they are linked to one or several positions in the organizational structure. At run-time, roles are replaced by responsible persons.

Workflows are modeled and controlled with Process Nets which are a variant of low level Petri nets where tokens are not distinguishable. Process Nets are a special type of Object Nets in the sense of [Wika90]. In WAM, Process Nets embody the control flows and a part of the real workflows (document flow). The major characteristics of Process Nets are summarized and commented below:

- Tokens are indistinguishable black dots and restrictions are imposed on the net structure to guarantee safeness and liveness. But much information is hidden in the black dots and complexity is shifted from nets to their interpreter and the execution environment.
- new types of transitions, namely event-transition, trigger-transition, sensor-transition, set-alarm-transition, clear-alarm-transition, are introduced to cope with external and internal events which are the main communication means in WAM.
- Each task corresponds to a special kind of subnet which interacts with its executing environment through events. Within a task, new subnets at a lower hierarchy may be activated. The structure is very suitable for a decentralized control strategy. However, shared resources like software tools and business objects must be centrally controlled.

- Mature and well-tested subnets called process modules are stored in a library for reuse. But the generalization of net modules and concepts about net template and class hierarchy are not considered.

3. Higher-order Object Nets

On the net graph, one can differentiate between active and passive elements, namely, between transitions and places. Traditionally, the active and passive entities in a system are naturally associated with their counterparts in the corresponding net model respectively. But this has brought us some disadvantages (see e.g. [LöHa94]). The central idea of HOON is that the domain of places may include both passive and active resources, as well as object-oriented entities. Hence, active and object entities can be dynamically bound to transitions, and a transition firing may link passive entities with active entities dynamically. As such we hope that some inherent problems can be solved.

First, we introduce the basic graphical elements of HOON. Then, some distinguishing features of HOON (compared to CPN or Funsoft nets) are summarized, so that the reader will get acquainted with HOON more quickly. Thereafter, the complete graphical language including the firing behaviors of HOON is described in the following subsections. The formal definition of HOON and the grammars will be described in subsequent publications.

3.1 Basic Elements of The Graphical Language

Besides the normal elements of Petri nets, some new elements are introduced in order to incorporate some new concepts like aggregation of places, late-binding of tokens, resource management and run-time user interaction into HOON. To establish a HOON model, we need a description language. Fig.3-1 illustrates all graphical notions of the modeling language. The elements of this language will be further explained bellow.

- Places

Places (P) are defined as containers of a multiset of distinguishable tokens. A data type can be explicitly associated with a place, and tokens on the place can only carry values of the corresponding data type of the place. Graphically, places are represented as a circle.

- Meta places

Meta places (MP) are containers of places and/or other meta places. A meta place may correspond to various combinations of aggregations of logically related places. The graphical representation of meta places is a nested circle.

- Control Places

Control places (CP) are an extension of the higher-order places of HON. Such places are also a special kind of meta places. Graphically, the control places are drawn as an oval. The connection edges between transitions and control places are drawn as dotted lines to indicate dynamic binding, borrow and return of resources. For conciseness, the arrows of these bidirectional connections are omitted. Each transition owns a control place exclusively. If no control information for resource management needs to be explicitly modelled, the corresponding control places can be omitted on the net graph. In order to show the hierarchical structure of HOON more clearly, it is recommended that net objects residing on a control place are drawn as black stars instead of black dots.

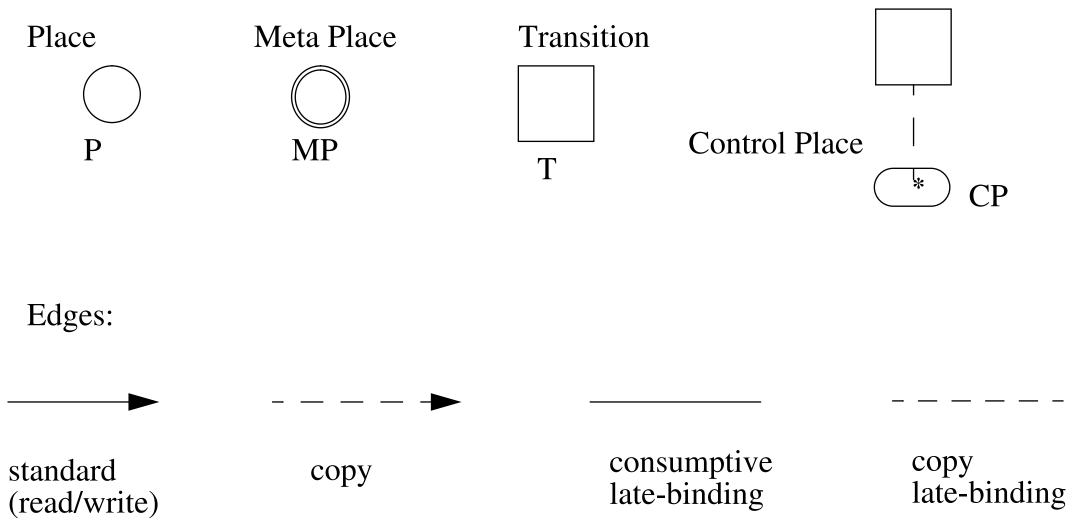


Fig. 3-1 Graphical notions of HOON

- Transitions

Transitions are the active elements in HOON. Their semantics is slightly extended in HOON. The firing rules will be discussed later in the subsection 3.5. They are graphically represented as rectangles.

- Edges

Edges in the net graph are directed connections between transitions and places (including meta places and control places). They can be either from places to transitions or from transitions to places. Beside the standard edge type of Petri nets, some new edge types are introduced in HOON in order to solve certain specific problems. They are copy type, consumptive late-binding and copy late-binding types. Late-binding edges allow that the corresponding tokens are chosen at a later stage (compare the paragraph "late binding mechanism in HOON " in the next subsection for details). A copy edge will ensure that tokens on the preplace are not destroyed, each transition gets a copy. This is under the assumption that the tokens can be copied arbitrarily. If this is not the case, copy late-binding edges should be applied which ensure that the tokens which have been taken away will be put back afterwards. Normal edges are drawn as solid lines with arrows which indicate flow directions. Copy edges are drawn as dotted lines with arrows. Consumptive late-binding edges are represented by solid lines without arrows and copy late-binding edges are represented by dotted lines without arrows.

3.2 Main Features of HOON

• Control Places in A Workflow Model: Providing Interfaces to The Outside World

For modelling and controlling workflows, we propose a HOON-based process model which consists of two classes of nets: working nets and control nets. The working nets are responsible for describing synchronization and control structure of the workflows with the help of information flows between workflow activities which are modeled with transitions. The information flows are expressed by the changes on the prestates and the poststates of the workflow activities. Different states are indicated by tokens on the related places. In the working nets, tokens on a place can represent (1) a data value of a fundamental data type such as boolean, string, etc. (2) a triggering message exchanged between component objects or between data objects. (3) a reference to an object, which is a special kind of message and only indicates the availability the real data object. The control nets are responsible for resource management and run-time user interaction. Control places which belong to both the working nets and the control nets serve as the interfaces between the two classes of nets. On the one side, tokens on control places can represent requests or notifications from the working nets to the control nets, and on the other side, four categories of resources can be represented as control tokens and can be assigned to the working nets via the control nets. The four categories of resources include: subnets encapsulated as objects (net objects), encapsulated software components (component objects), persistent data objects (data objects), and agents being either persons or unencapsulated functional program codes (actors).

Conceptually, if no explicit restrictions are imposed, all shared objects are accessible on every control place. But the corresponding transition may not be able to "see" all of them. Their visibility is controlled according to different strategies: the visibility of net objects and component objects is determined by the control nets through assigning or withdrawing the corresponding control tokens to or from the control places. The visibility of data objects to a certain transition is indicated by their reference tokens or message tokens on a preplace of the transition. Restrictions can be imposed to prohibit some data objects from being visible by limiting the data type of the container of the message tokens to certain object classes. When a unique resource or the last copy of a resource is occupied during a transition firing, it will become invisible to other control places. After the release of the resource, the visibility will be rearranged. This is one of the most important tasks of the resource manager in the control nets.

A promising feature of a HOON process model is that run-time user interventions and timely response to external events can be well supported. Some control tokens can be generated outside the working nets at run-time and be placed on the control places shared by the working nets. Such control tokens can then be consumed in the working nets. The working nets can also produce control tokens which will be consumed later in the control nets. Concretely speaking, the working nets can, on the one side, produce control tokens containing feedback information, requests for some resources or for exception handling. The request or feedback information from the working nets to the control nets indicates requests for new resources, notification of resource modifications such as new instantiating of a data object. On the other side, control tokens may be placed by the designer, system administrator or privileged users within the control nets. As a result, external dynamic modifications and dynamic configuration of the working nets can be realized by adding or withdrawing tokens to or from the control places in the control nets. For example, the run-time workflow system configuration can be modified by a new assignment of certain net object tokens.

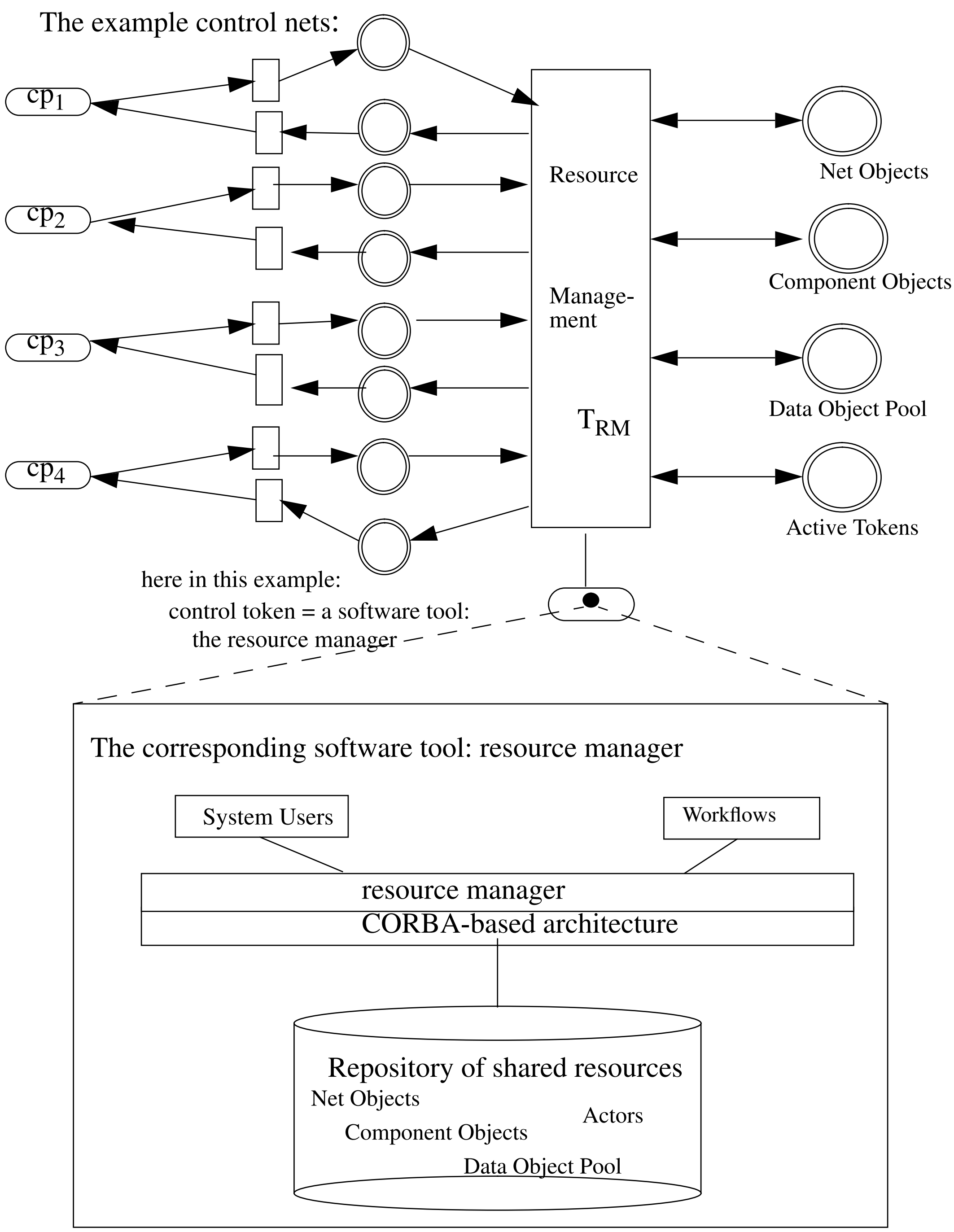


Fig. 3-2 Control nets with a resource manager as "higher-order" control token

Fig. 3-2 amplifies the control nets shown in fig.1-2. The two kinds of transition connected with each control place are used to model the resource-traffic. They stand respectively for feedback information or request from the working nets to the control nets, and resource assignment from the control nets to the working nets. The transition T_{RM} corresponds to an autonomous node of resource management. As an example, a control token representing a software component is now laid on the control place of the transition T_{RM}. Borrow and return of the shared and exclusive resources of each control place are registered by the software tool - resource manager - which normally has to be implemented in a heterogeneous distributed environment. The resource management issue and the distributed platform will be treated in section 5 once again.

The control net structure is application-independent and of generic nature, that is, the transitions may also own a control place. As such, a control net may again contain a working net and a "higher-order" control net. According to different strategies of resource management, control nets can also be constructed with several levels consisting themselves of "higher-order" working nets and "higher-order" control nets. But in this paper, we only consider the "first-order" control nets as indicated in fig.3-2.

- Net Objects, Component Objects And Data Objects: Introducing Object-Oriented Concepts into Petri Nets

As already discussed in the introduction, we do not think that it is appropriate to turn a Petri net model into an object-oriented schema. We advocate a hybrid modeling concept which adds new features to Petri nets by introducing some object-oriented concepts and maintains the old features of Petri nets at the same time. Several proposals to integrate object-oriented concepts into Petri nets have been made in recent years. The most popular approach is to claim that the data type of a certain place corresponds to an object class whose members are either data components constructed with an object-oriented language [FlLi93] or instances of a subnet [Laco94]. Both cases are supported in our approach because we have recognized that the object-oriented concepts can be applied to different levels, namely net level, component level and data level, independently and in varying degrees of completeness. Moreover, the traditional method and the object-oriented method may coexist in our Higher-Order Object Net model, leading to two sorts of tokens, i.e. normal tokens and object tokens. Comparing to normal tokens which carry concrete values, object tokens represent entities encapsulating structures with behaviors and having their own life cycles and even their own ways of interaction. In object-oriented methodology, objects are considered as instances of abstract data types defining common internal structures and behaviors. In Petri net world, data types are usually associated with places in most extensions, tokens represent instances of the data types. To be able to represent objects with tokens, the abstract data types have to be associated with some places. In HOON, a special class of places - control places - may carry such data types.

At net level, a set of commonly-used net structures form *net templates* bearing analogy with classes in the object-oriented programming terminology. Net templates are well-defined and abstracted subnets without assignment of any control tokens because at this level of abstraction we do not consider any resource management issues and only want to deal with flat nets. With the assignments of control tokens, net objects can be derived from a net template. Each net object has a unique name of identification. Although one net template may contain other net templates, no inheritance relation is defined, therefore there is no class hierarchy at the net level in HOON. All templates are equal in this context and belong to a unique meta type - *net*. This meta type can be directly associated with each control place, hence, net objects derived from different net templates can coexist on the same control place at the same time. Net templates can not directly be used as data types for places in HOON. They are used as reusable generic structures for the construction, modification and instantiating of net objects and are stored in the workflow repository.

In many variants of Petri nets, subnets or net modules[1] are introduced to improve the structure of net models. The remarkable differences between our net templates and the subnets or net modules proposed elsewhere are:

- Net templates are behavioral abstractions of subnets encapsulated with well-defined input and output interfaces. One net template may therefore correspond to many subnets since their instantiating results in net objects and the instantiating of the net objects leads to the execution of lower level nets. The net objects can be treated as single entities in nets. These entities can be modeled as tokens and can therefore be bound to transitions at run-time instead of at modeling time in a fixed manner. A structural diagram of net templates and net objects is given in fig. 3-6. Net objects have their own local states, state-changing elements, and public input/output interfaces which are also called ports when the boundary elements are transitions, i.e. the ports are transitions which interface with preplaces and postplaces of the neighboring nets. Each control token representing a net object can be wrapped with a predicate which examines the activation conditions of the net object. It is also possible to define an activation predicate for each net template or for each kind of net objects instead of for each control token if there is only a single way to activate the net object or the net objects being able to be derived from the net template.
- Due to the introduction of meta places and activation predicates, the interfaces between net objects are more general and flexible. Note that we can not abstract an arbitrary subnet as a higher-order token in the early version HON if the input and output interfaces of the subnet do not match those of the corresponding higher-order transition. This restriction is alleviated in HOON thanks to the meta places which will be explained in the following paragraph. Furthermore, not all member places on the input meta place need to be marked in order to activate a certain net object, some of them may be insignificant (see also the subsection 3.5).
- When executed for analysis, simulation or control, the traditional approaches replace the refined net elements by the corresponding net modules, obtaining a flat net which is normally very large. In contrast, no static flat net can be derived for HOON because net objects at different levels are activated and linked dynamically during transition firing. Each nonatomic net object may, therefore, correspond to many possible flat nets.

1. Subnets and net modules are not used as synonyms here. Comparing to programming terminology, subnets correspond to subprograms and net modules correspond to macros.

At component level, we talk normally about software subsystems or standalone software tools. Inheritance and classification are not taken into consideration at this level. Component objects are black-boxes with well-defined interfaces. Through the interfaces, services (also called functions or methods) can be selected and called. Each control token representing a component object is wrapped with a predicate which is similar to that of the net objects. A component object can be activated when its predicate is evaluated to true. There is a second way to activate a component object, namely through message passing.

Data objects are data entities constructed with object-oriented techniques (not necessarily with an object-oriented language) by encapsulating data with the services which manipulate the data. The encapsulated data are only accessible through their services. Data objects are treated as a special kind of data in HOON and they stay in the background. The management of all shared data objects belongs to the tasks of the resource manager which has been shown in fig.3-2. Only the proxies of data objects, namely reference tokens or message tokens, appear in the model. For data objects, HOON support full object-oriented concepts including inheritance and polymorphism.

The messages passed between objects are modelled as tokens of the message type. Message tokens reside either on a message place which is a normal place associated with the message type or on a meta place as a constituent member of the meta place. In the former case, the place is called *message place* consequently. Instances of the message type appear in form of tuples, for example, <object identity, service name, parameter list, addressee, sender, token access mode>. A single message type can be used both for message passing and for object referencing. Normally, a message token contains service names which can serve as triggers of activation for a certain object. If the service name is vacant, we call a message token a reference. A reference to an object only indicates the visibility and readiness of the object. The message/reference token will be ignored if the object being addressed is not available to the corresponding transition (i.e. no control token which represents the object exists on the control place).

For security and efficiency reasons, a control token representing an object can be wrapped with one or more definite services which are a selected subset of all available services of the object. As a result, the object can only be activated with the defined services. When only one service is defined as the "mask" for a component object, the activation of the control token is equivalent to the calling of a Funsoft net job. If no services are specified for the control token, all available services will be presented to the actor at run-time and run-time decision is left to the actor bound to the transition.

- Meta Places: Making Net Interfacing General And Keeping The Net Structure Stable

Different input places and different output places can be gathered together to form an input meta place or an output meta place respectively. A meta place has normally many possible edge expressions. According to different edge expressions, a meta place can be viewed as "marked" when the member places of various combinations are "marked". Meta places are introduced mainly to make the interfaces of net objects more flexible, but they can also be simply an aggregation of some closely related normal places just to make the net graph cleaner and more compact.

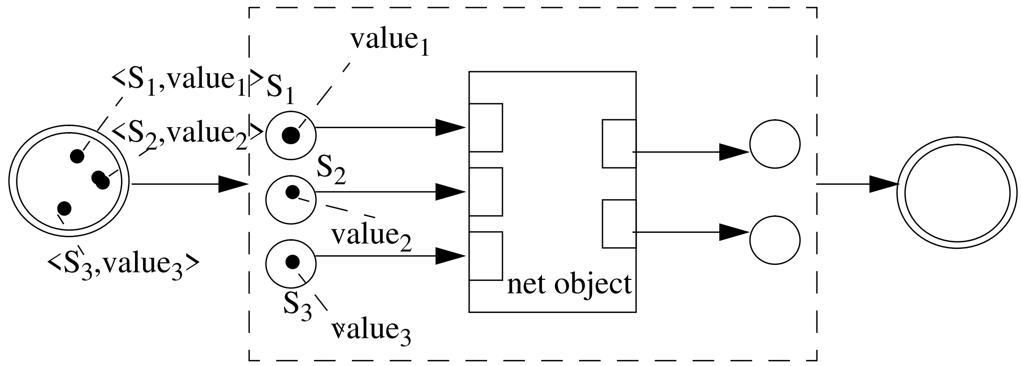


Fig. 3-3

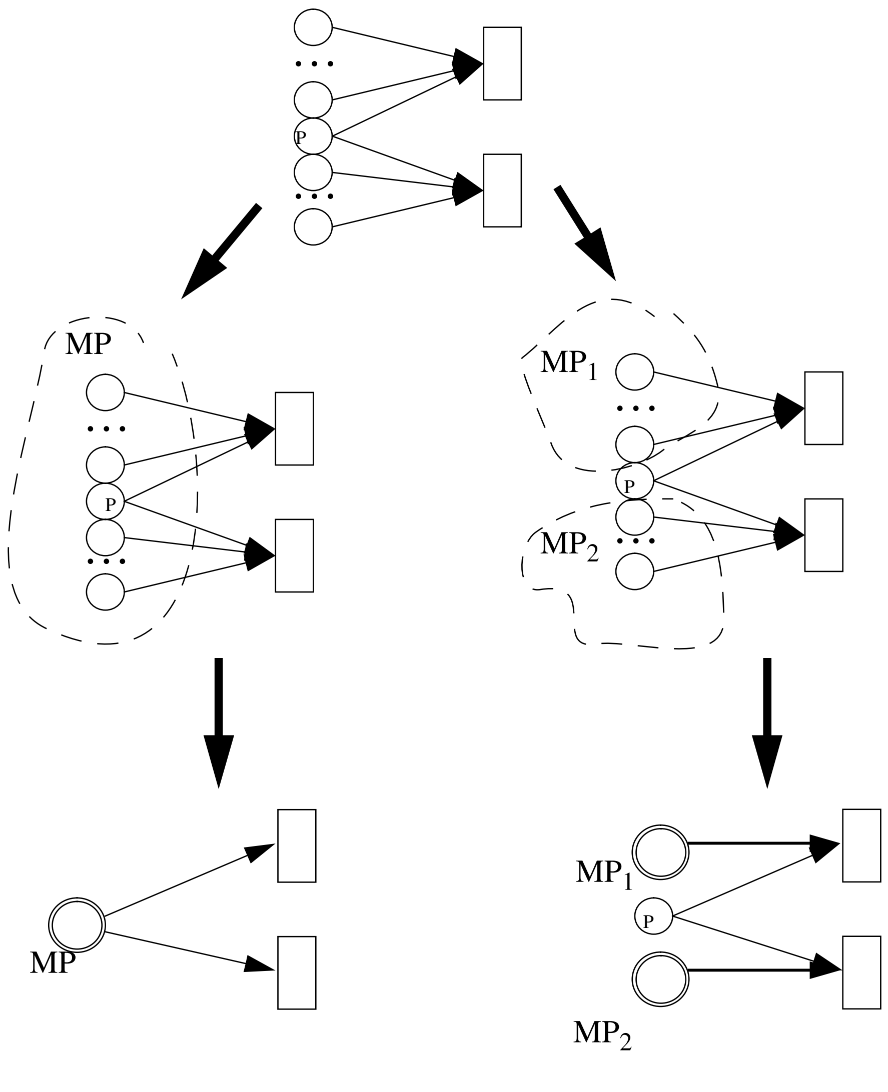


Fig. 3-4

With the help of meta places and control places, different net objects can be associated with the same transition. The activation of a net object is only dependent on one of all possible markings of the input meta place. In this way, we can activate net objects of different net templates within the same structure, i.e. with the same transition, only tokens on the input meta place differ. Because places are allowed to receive tokens sent or broadcasted to it dynamically by a transition which has no explicit connections to them (see also next paragraph), different net topologies can be realized at run-time with the same net structure. The overall net structure remains stable even though the interfaces of the same net object are modified. If the domain of an input meta place is defined as a power set of the member places, a limited number of net object interfaces (<= $2^n - 1$ where n is the number of the member places) can be realized and no explicit connections between the transition and members of the meta place are mandatory.

The data type associated with a meta place is the meta type whose intermediate instances are data types and whose ultimate instantiating results in concrete values. Therefore, depending on different viewpoints or different granularity, a token on a meta place may be interpreted as (1) a marked place or (2) a concrete token with group identity which indicates the member place to which the concrete token belongs. In this paper, as shown in fig. 3-3, tokens on a meta place are interpreted as concrete tokens with their group identities. In order to transfer tokens to the right input ports of net objects, tokens on a meta place must carry at least following information: group identity, token identity and token value. Group identities correspond to traditional place names whereas token identity and token value are the same as defined in Funsoft nets or in HON. Fig.3-3 shows conceptually how the parameter-matching is done during the unpacking of a net object. Although the first way of interpretation is not going to be further treated in this paper, it is worth investigation and may be very useful when one wants to dynamically create places for system (re)configuration. It is also a matter of taste how to express a conflict situation caused by the competition between two follow-up transitions for one or more member places of two preceding meta places. Two possibilities are shown in fig.3-4. On the left side, the potential conflicts are kept within a single meta place. Which member places serve as inputs for which transition firing will be determined by the predicates of the control tokens to be unpacked. On the right side, the place in contention (place p in fig 3-3) is modeled with a normal place explicitly. The meta places are always interpreted as a complete logical unit of resources.

- **Extension of Postplaces: Incorporating Token Broadcasting Scheme to Support Dynamic Object Interaction.**

In HOON, transitions may have implicit postplaces. Such postplaces are not predefined in the net structure, i.e. there are no corresponding graphical connections on the net graph. This mechanism contributes much to the flexible compositionality of net objects and promotes dynamic object interaction within nets. In HOON, places can receive tokens dynamically sent to them provided that the type compatibility and receiving constraints (type checking) are satisfied. To perform the type checking in HOON, class hierarchies and object relationships defined in the accompanying object model may have to be referenced at run-time.

There are two kinds of such dynamic links from transitions to places: send and broadcast. The first mode has a definite recipient which only becomes known at the moment the tokens are sent. One or more tokens will be sent exclusively to the recipient. In the broadcasting mode, tokens together with constraint information will be broadcasted, each place which satisfies the constraint will receive one or more tokens. Two different situations may be identified from the viewpoint of resource management: infinite deposit and finite deposit of resources. In the second case, tokens can only notify the existence of some concrete resources. There may be more places which have received a broadcasted token than the available concrete resources, conflicts may arise in consequence. In HOON, it is presumed that the solution of such kind of conflicts will be provided within the control nets, e.g. by an external resource manager or repository manager in a general sense.

• Late-Binding Mechanism in HOON

Tokens can be accessed according to one of the following modes: FIRST, LAST, USER, and LABI mode. LABI mode supports late-binding of tokens, i.e. one can dynamically decide, among the available resources, which tool or net object should be called or which data should be first processed, and so on. As indicated by the example given in fig.3-5, we can imagine that the firing process of the transition T with a LABI edge is equivalent to that of the compound transition T'. All connections between T_1 and its preplaces are of the copy type, i.e. the firing of T_1 has no influence on its preceding states. When T_1 fires, the internal edge expression EXP of the edge from P_3 to T_2 will be determined definitely. The tokens satisfying the edge expression EXP will contribute to the enabling of T_2 and will be consumed by the firing of T_2. In fact, the firing process of the transition T_2 can progress as if there were no LABI-edge. Tokens on P1 and P2 will be drawn away at this stage. The above-stated mode with the normal way of token movement is called *consumption late-binding*. Another possibility is provided by the *copy late-binding* mode. In copy late-binding mode, tokens will be put back after or during the transition firing. Therefore, two token access modes must be specified for such edges, for example, FIRST/FIRST means that the first token will be taken away and then be put back as the first member of the token array. The late-binding of control tokens is of the copy late-binding mode.

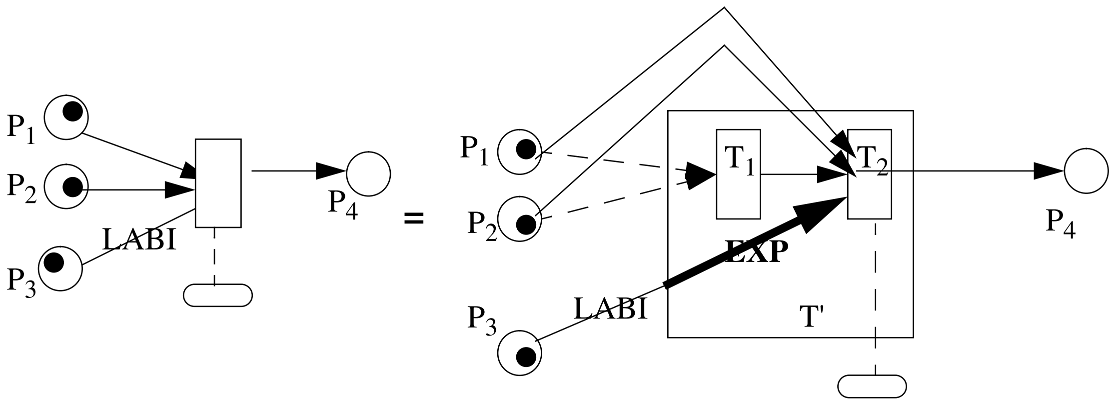


Fig. 3-5

• Three Stages of State-changing

In HOON, three stages must be undergone to complete a state-changing, namely *transition unlocking, transition enabling,* and *transition firing*. Furthermore, unlike in CPN or in Funsoft nets where a single predicate is employed as the firing "guard" of a transition, the guard is "distributed" to certain control tokens in HOON, leading to a slightly different firing process. A transition is unlocked when there are suitable tokens on its preplace(s) which satisfy the given edge expressions. The transition is immediately enabled after it unlocking when there are no late-binding places (The late-binding places are the places which are connected to their follow-up transitions with edges of the late-binding mode, see the subsection 3.3 for a more detailed explanation of the token access modes). Otherwise, the transition is enabled only when tokens on late-binding places can be bound to the transition. Note that control places also belong to the late-binding places and a control token can be bound to the transition only when its activation predicate is evaluated to true. After a transition is enabled, it may or must fire according to the defined firing rule. In the subsection 3.3, the three stages of state-changing will be further explained.

• Hierarchical Construction of Nets: Coping with Different Levels of Activity

Reusable subnet skeletons are encapsulated as net templates and stored in the HOON workflow repository which is very essential in the proposed methodology for developing model-based workflow systems. In HOON, nets are practically a hierarchical combination of various net objects. As single entities of the system resources, net objects can be statically or dynamically assigned as control tokens. In this way, we can obtain a very flexible, reconfigurable net topology. This new way of hierarchical model construction is shown in fig.3-6, together with a structural diagram of net templates and net objects.

Applying net objects, we can generically construct a net model and do not need other refinement mechanisms any more, such as transition refinement or place refinement. The process model consists of a set of net objects of different levels corresponding to different levels of activity. Note that the levels are not fixed and there is no predefined level information associated with any net template in the repository. Recursive model construction is not forbidden though we will not make any further investigation in this direction in this paper. One can build a net model either in a bottom-up way or in a top-down way.

• Role

In Funsoft nets, roles are used to denote a set of logically related actors. In HOON, role is a single meta type whose member types (called *role types*) are user-defined, for example, MANAGER or DESIGNER for people and PROGRAM for a functional software tool. The concrete actors like the staff of an organization or some functional software tools can be attached to the user-defined role types statically or at run-time. Although a concrete actor can be associated to different role types, the different assignments of the same thing are no more comparable. The correspondence, assignment, constraint and security control are managed by the resource manager on the basis of the organizational structure (compare also section 4). In HOON, the concrete actors are represented as control tokens. Conflict solution and constraint checking are done in the control nets. For an example of constraint, we can define that the staff member Schmidt may act as the actor for the role type A exclusively, participate in one to three tasks of the role type B, act as the actor for the role type C no more than two times at the same time, and if he takes part in tasks of both type A and type B, no more than two tasks of each type can be assigned.

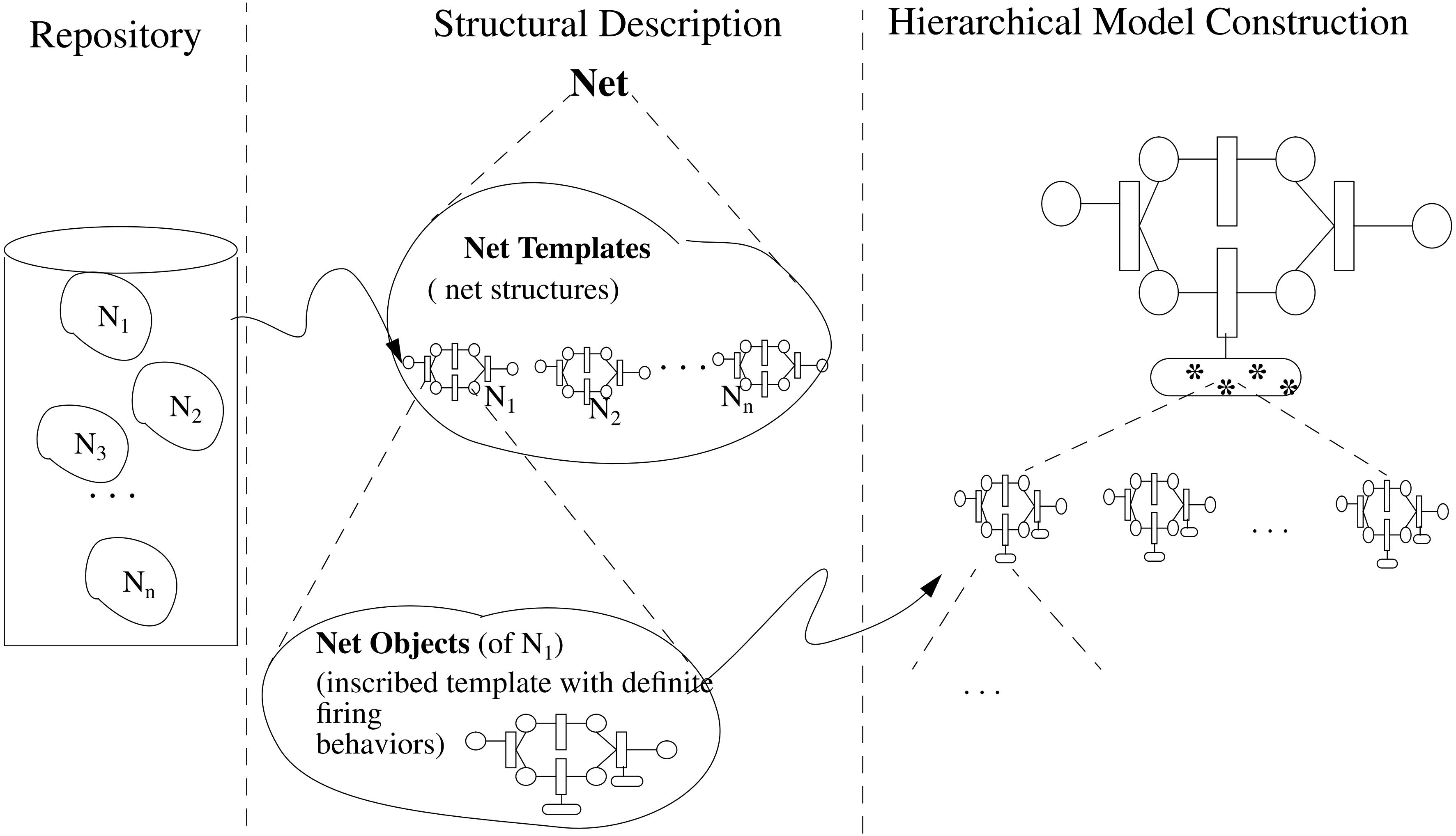

Repository
Structural Description
Hierarchical Model Construction
Net
Net Templates
(net structures)
N1
N2
N3
Nn
Net Objects (of N1)
(inscribed template with definite firing behaviors)

Fig. 3-6

3.3 Inscriptions

- Edge Expression

An edge expression specifies the collection of tokens being able to flow through the edge when its subsequent transition fires. An expression may be evaluated to a multiset of values. The edge weight in P/T nets is a special case of a general edge expression.

The edge expressions for meta places are obtained by connecting the edge expressions of the related member places of each combination with the concatenation operator AND. In case of late-binding edges, the edge expressions are determined at run-time, but an extra expression can be given to restrict the possible run-time candidates.

Several edge expressions (of different combinations) can be assigned to one edge at the same time. By this trick, a single place or a single meta place can virtually be viewed as several ones. Different edge expressions are connected by the alternative operator OR. In HOON, it is suggested to establish an automatic correspondence between the edge expressions and the activation predicates of control tokens (see also the paragraph about firing predicates).

- Token Access Modes

In HOON, tokens on a place are organized as an array. The order of the array corresponds to the creation order of the tokens or can be specified explicitly by the user. The two ways of access: read-access and write-access are related to the two kinds of the flow relation, namely from places to transitions and from transitions to places. The token array can be read or written with the following predefined modes:

- FIRST: when used for the read-access, the first token(s) of the array will be read; or for the write-access, one or more tokens will be added to the array at the beginning.
- LAST: when used for the read-access, the last token(s) of the array is going to be read; or for the write-access, one or more tokens will be appended to the tail of the array.
- USER: this access mode can be operationally specified by the user for a specific application. The default mode is the RANDOM mode which allows the token array to be accessed randomly.
- LABI (late-binding): it will be decided during a firing which one of the candidate tokens will be chosen. LABI is only valid for the read-access.

Different from in Funsoft nets, token access modes are normally associated with edges in HOON. The Funsoft net approach is a special case where an access mode is associated with a place, which means that all outgoing or incoming edges have the same specified mode. Moreover, a default access mode can be defined by the user (for example, FIRST) just to make the net model graphically cleaner.

Normally, tokens on a meta place need to be organized as a multidimensional array. For the edges connecting a meta place, multiple access modes can be specified for a single imaginary edge in the following form: <group-info, access mode>. The group-info shows which member place is concerned. Such a specification can be nested if a member place is again a meta place. It is also allowed that tokens from different member places are arranged in a single flat array. As such, only a single access mode is necessary for the edge connecting the meta place.

The copy edges in HOON correspond to the copy edges in Funsoft nets. In Funsoft nets, however, no rules are specified to avoid the following problem: some tokens may never be selected whereas others may always be chosen. To ensure that all tokens have equal chance to be selected and avoid unnecessary conflicts at the same time, each copy edge can have its own copy of the token array, i.e. each edge corresponds to a virtual place with the same content as the concerned place. In addition, it is also necessary to specify for each copy edge how the token array should be reorganized after each firing. The token access modes for copy edges have to be given as a tuple <read-mode, write-mode>. The read-mode and the write-mode can be any one of the above-listed access modes. The access modes for copy late-binding edges are specified in the same way.

- Data Types in HOON

Some primitive data types of Funsoft nets have been taken over as the primitive data types of HOON. They are boolean, integer, real, and string. Heuristic construction of complex data types is supported by means of the primitive types and the aggregation constructor. In addition, several new types are provided and some old ones defined in Funsoft nets are extended. The most of them have already been discussed in the subsection 3.2. These additional data types are listed below:

- Data Object
- Component Object
- Net Object
- Role
- Message
- Pseudo

Like the undefined type in Funsoft nets, Pseudo type can be replaced by any other data type at a later stage.

- Firing Predicates

Firing predicates express constraints on the transition firing by defining conditions on the names and values of the tokens to be read. Unlike in CPN or in Funsoft nets in which only one predicate can be assigned to a transition, several kinds of control tokens can be equipped with a predicate in HOON. The predicates can be evaluated to either true or false indicating wether a net object, a component object or a data object can be unpacked or wether a functional tool can be called. Indirectly, the predicates associated with the control tokens determine wether the transition can fire or not. The predicates are defined by logical formulae (the syntax and interpretation will be covered in our next publication). In HOON, there is a correspondence between edge expressions and the firing predicates. When a control token is assigned to a control place, the edge expressions of the edges connecting the preplaces and the transition will be automatically modified by adding extra terms to them. When a control token is withdrawn from a control place, the edge expressions of the edges connecting the preplaces and the related transition will be automatically modified by withdrawing the corresponding conditions.

3.4 Place Marking, Transition Unlocking And Transition Enabling

- Net Marking And Initial Marking

A net marking (which is called marking later on for short) is a possible legal distribution of tokens on the places of the whole net in concern. The initial states of a net can be defined through assigning tokens to certain places. The initial markings are the special initial states from which at least one subsequent net marking can be derived.

- Place Marking

A places P is marked for a transition T if there are enough tokens on P satisfying one of the edge expressions of the edge from P to T.

- Marking of Meta Places

A meta place MP is marked for a transition T if one of the edge expressions (connected with the alternative operator) from MP to T are satisfied. Conceptually, a meta place is a logical aggregation of resources and tokens residing on it can be considered as marked places. The marked places contain either marked places at a lower hierarchy or normal tokens which carry concrete values. Technically, we use group identity to identify a marked place. Graphically, there is only one edge connection to each associated transition though it can be imagined that for each member place there is an edge connection.

- Conflict

Note that, due to the association of token access modes to edges and the introduction of meta places, the semantics of conflicts in HOON is slightly different from that of Funsoft nets. For example, as we can see from fig.3-7, T1 and T2 are in conflict on the left side, but may no more be in conflict on the right side if the token access modes are assigned and there are more than one candidates on place P.

Fig. 3-7

- Transition Unlocking And Transition Enabling

When there are enough tokens residing on the preplaces of a transition and at least one expression of each edge is satisfied, the transition is unlocked. If there is not any late-binding place, the transition is also enabled at the same time. Otherwise, after a transition is unlocked, tokens on late-binding places can be evaluated and bound to the transition. If this step is successful, we say that the transition is enabled. An unsuccessful binding makes the transition being locked again. Control places are a special kind of late-binding places. The most control tokens can be wrapped with an activation predicate. The activation predicates are part of the control tokens and define unpacking conditions which specify constraints on names and values of the tokens residing on the preceding places and/or meta places.

- Partial Transition Unlocking And Enabling

There is a special case of transition unlocking and transition enabling. A transition may be unlocked even if not all preceding places are marked, that is, there exists at least one edge whose expressions are all evaluated to false. Such kind of transition unlocking is called *partial transition unlocking*. The places which have such privilege to activate a partial transition unlocking are inscribed with a special label Δ. The corresponding transition enabling is called *partial transition enabling*

3.5 Firing Behaviors of HOON

- Firing Rules

If a transition is enabled, it may or must fire according to a certain firing rule. Concerning the firing rules, we can distinguish the following three cases:

- progress free: when a transition is enabled, it may fire, but has never to do so. This mode is also called may-firing rule.
- progress prone: whenever a transition is enabled, it will fire or be disabled due to the firing of some other transition. But when the firing occurs is not confined definitely.
- immediate firing: if a transition is enabled, it must fire in the next step. This is also called must-firing rule.

- Firing Behaviors

The transition firing in HOON differs from the standard one in Petri nets mainly in the unpacking and activation of control tokens. Now, we discuss this essential aspect of HOON. On the basis of the above-stated firing rules, an enabled transition may or must fire. When the transition T fires, the following things will happen:

- Evaluation And Activation of Control Tokens

  One or more control tokens on the control place will first be searched, selected, evaluated, and activated. According to the edge expressions, different modes for selecting control tokens can be specified, for example, no control token, one control token for each group of member places at a time, or a complex mode. If a null value is specified, it is meant that the transition fires unconditionally, producing tokens specified by the edge expressions of the edges to the postplaces. The default mode is the second mode which specifies that, among all candidates of the control tokens of a given group, only one is selected according to a predefined access mode such as FIRST, LAST, etc. Control tokens from different groups will be unpacked mutually exclusively (compare next paragraph). The complex mode is defined by the user with a complex edge expression. For instance, it may allow that more than one control tokens from the same group are selected at each transition firing (compare fig. 3-8).

The evaluation of a control token is realized by evaluating the predicate associated with it. If the predicate is evaluated to true, the corresponding control tokens can be activated and the firing process may progress further, otherwise the unpacking of this token is disabled under current states.

When a control token is unpacked, it disappears on its control place. Other control tokens representing the same resource may also become "invisible" temporarily. Conflicts (if any) caused by the competition on the same resources are solved externally to the working nets (namely within the control nets). After the "show" of the unpacked control token comes to an end, the control token will be put back and/or a feedback signal will be sent to the control nets resulting a new assignment of certain control tokens, i.e. making them "visible" to the desired control places.

Among different groups of control tokens, the selection is made on the basis of different levels of granularity or of user-specified priority. The default order is given as follows:

a) Net objects are at the highest level of granularity. If there is any net object token, it will be evaluated first. If a net object is selected, the unpacking of the net object will start a subnet at a lower hierarchy.

b) Component objects are at the subsystem level. Each component object is responsible for a specific activity. As in Funsoft nets, activities can be assigned to a transition and will be performed each time the transition fires. One difference is that the encapsulated software components are modeled as tokens on the control place instead of being bound to the transition fixedly. In case of more than one candidates, choices of activities which can be performed on the basis of the input tokens will be presented to the user during the transition firing. According to the user decision, a certain activity will be performed. Another difference between Funsoft net approach and ours is that we do not confine the executable description of the activities to a piece of program code. No working programming language is defined in HOON. The executable descriptions are viewed as a standalone software package in binary code wrapped with an interface definition language. Herein, DOC and CORBA-based distributed object request brokers will play a key role. This issue will be further investigated in section 5. Normally, an activity only touches on a subset of all preplaces and a subset of all postplaces. Each activity can specify its own firing behaviors which determine the token movement dealing with its preplaces and postplaces.

c) Data objects are located at the computation level. Data objects accessible to a computation step (transition) reside on the control place of the transition in form of control tokens. Data objects are activated by messages or references. Only when both the message token and the data object token addressed by the message token are available, the corresponding data object can be activated.

Independent of the activation priority, the control tokens of the meta type Role will always be selected and then assigned to the transition dynamically.

- Token Consumption On Preplaces

  When the transition firing is not disabled at this stage, specified number of tokens satisfying the edge expressions which are specified explicitly or by the activation predicates of the control tokens are removed from the corresponding preplaces.

- State-changing Actions at Transition Firing

  Actions triggered by a transition firing are purely determined by the unpacking of the control tokens whose predicates are evaluated to true. As discussed above, the unpacking of control tokens results in either an action directly or a lower level net. In case that more than one control tokens are unpacked, the effect of one transition firing may be equal to many transition firings. An example is given in fig.3-8. The virtual transitions in fig.3-8 are activated sequentially, but can act in parallel after activation if no conflicts occur. The transition firing ends only when all actions come to an end. However, the output tokens can be produced asynchronously. This means that some of the subsequent transitions may fire even when the current transition firing is still in progress, introducing another level of concurrency.

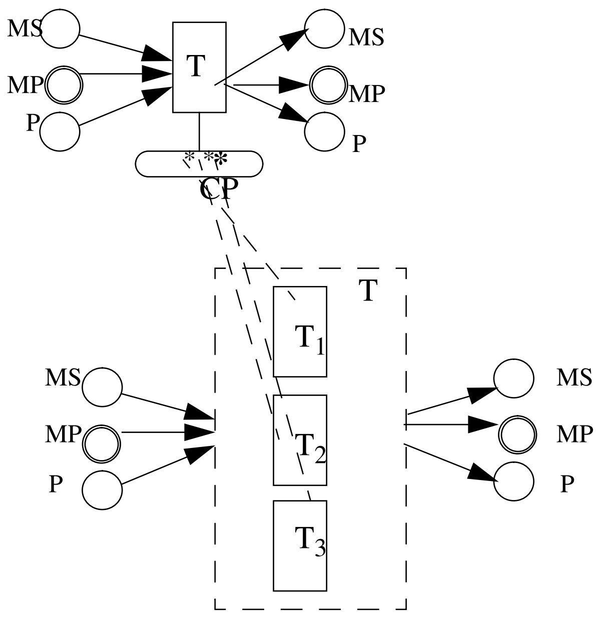
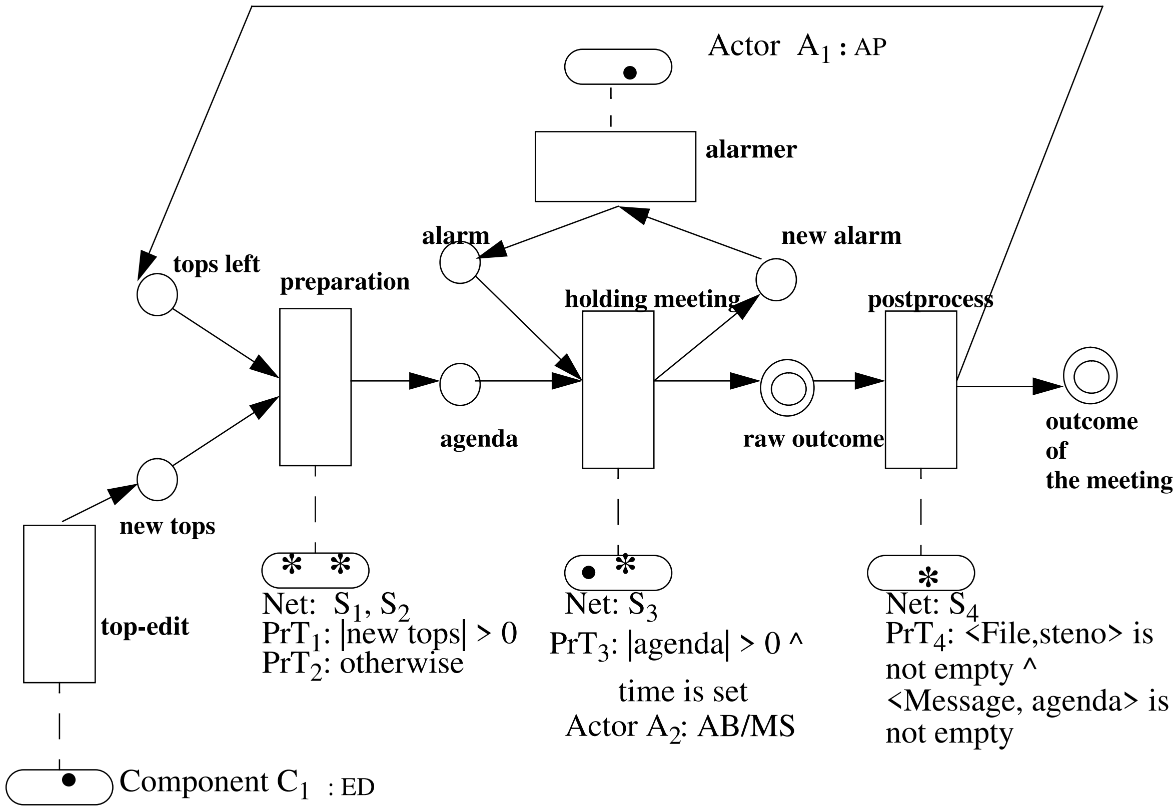


Fig. 3-8

- Production of New Tokens onto Postplaces

  In HOON, the production of normal output tokens is the same as in the traditional Petri nets. In addition, HOON support selected output and broadcasting of tokens. Selected output means that only some of all postplaces obtain tokens produced by the transition



Data types Inscriptions:
Net Object: S_1,S_2,S_3,S_4
Actor: A_1,A_2
Component Object: C_1
Data Object: tops, agenda
Message: tops left, new tops, agenda, agenda passed, agenda left
ED: topic editor
AP: alarm program
AB: Angelica Bell (director)
MS: Magie Sven (vice director)
Complex: File, Note
File: stenography, protocol, notes
Meta: raw outcome, outcome of the meeting
raw outcome: <File, steno>, <Note, notes>, <Message, agenda passed>, <Message, agenda left>
outcome of the meeting: <Message, tops left>, <File, protocol>
Boolean: alarm
Real: new alarm

Fig. 3-9 A Modeling Example

firing. HOON also allow that certain tokens are sent to specified places or specified classes of places no matter whether explicit graphical connections exist or not. Two functions are provided for this purpose: SENDTO and BROADCAST. SENDTO has a definite addressee and BROADCAST has a class name in the object model as recipient and send tokens to all members of the class (and, according to user specification, members of all derived classes). If more tokens are broadcasted than the available resources they represent, conflicts may arise. Except for unlimited simple data values, the broadcasted tokens do not carry concrete values. In another word, they are also of the type message. To obtain the concrete values, the corresponding control tokens representing public resources as encapsulated objects must be once more evaluated. The management of shared resources and conflict solution are provided within the control nets.

3.5 A Modeling Example

As a diploma thesis, the computer-aided coordination of a research team is going to be modeled and controlled with HOON. The team coordination is based on periodical group meetings. In this paper, only the net model of the group meeting part of the example is roughly presented. The complete modeling, including the object modeling and the organization modeling, is left as part of the diploma thesis.

Fig. 3-9 shows a rough top-level net model of the group meeting situations. New topics can be registered and entered into the system by individuals of the group at any valid time. During the preparation process, similar topics are merged, and the new topics are merged with the old topics left by former meetings. The agenda of a meeting will be produced shortly before the meeting is held. In this example, the preparation is performed by one of two net objects assigned as control tokens. The net for the actual meeting is an open manual process. The outputs of the meeting are modeled with meta places which are composed of a stenography, the agenda which has been run through, and a set of personal notes taken by the group members. They are inputs of the postprocessing process. The postprocessing stage stretches until to the next meeting. Based on the stenography, a protocol is written down for each meeting. Data objects like tops or agenda do not flow in this working nets. Resource management is done in the control nets which are not drawn here. In the diagram, predicates are given in quasi-plain language.

4. Complementing Higher-order Object Nets with Object and Organization Models

A model is only an abstraction of a subset of reality for a particular purpose or from a particular viewpoint. Building a large, complex system such as a workflow system, we need modeling support from different viewpoints or with different emphasizes. For example, when designing a database, the structural, static properties of the application system become more important. For controlling workflows, the behavioral aspects are more important. Although process models are central to workflow modeling, references to other models are still needed because the static structural system properties are difficult to be explicitly described by petri-net-based models. Moreover, modeling a complex object as a named token often causes information loss. Proposals concerning a true integration[1] of a dynamic model and a static model for information system modeling have been made since early 80's [Kung89][HePe93]. The resulting graphical representation of the integrated model is often very large and becomes difficult to be understood if the static structural aspects of the model are also taken into consideration. Here, we do not intend to combine all useful models of a workflow system into a single integrated one. Instead of integration, we recommend a coexist of three conceptual models: process model, object model, and organization model. The models complement each other and can consult with each other both statically and at run-time. In this section, only an initial proposal is given. The cross-reference of the models and the corresponding interfaces are the subjects of our further investigation.

4.1 Object Model

The object model in HOON (called HOM) captures the static properties of the modeled system. Static properties refer to the following: the possible states of the system, integrity constraints on the system states, objects that constitute system states, structure and attributes of the objects, and relationships between the objects. HOM incorporates the essentials of the E-R model[1], that is, the treatment of entities and relationships on an equal basis.

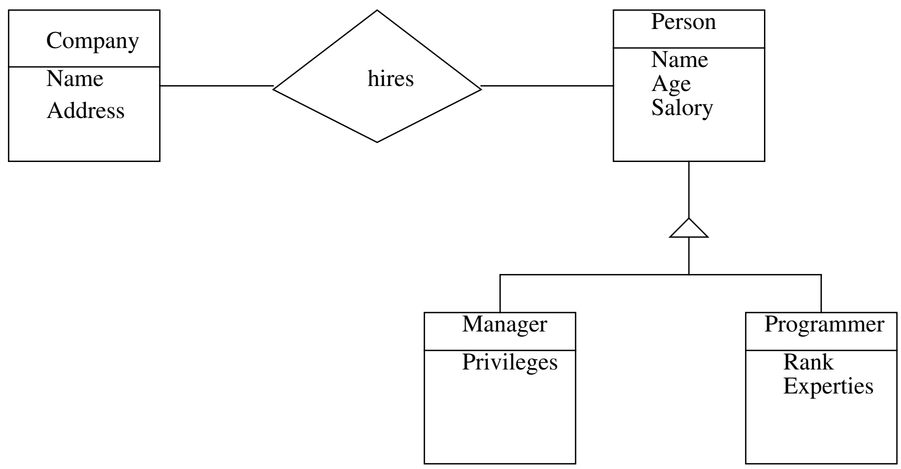


Fig. 4-1

In HOM, the basic graphical elements are classes which are containers of the objects which are similar according to certain criteria. Classes correspond to abstract data types and objects correspond to instances of the abstract data types. In the proposed methodology, the object model covers two important aspects: (1) object structures which describe class structures and the inherientance hierarchies of classes, and (2) object relationships which describe the links between different kinds of objects. The world is viewed as a container of various classes having various relationships with each other. The difference between the objects here and the entities referred by Chen is that objects have autonomy, may determine their relationships to other kinds of objects, and may, therefore, be characterized as both active and passive. The most important relationships which can be applied for constructing an object model are: aggregation,

1. By true integration it is meant that the links between the static model and the dynamic model are explicitly expressed.
1. Here, it is assumed that the E-R model [Chen76] is familiar to the readers.

generalization, and association. Aggregation corresponds to the "a-part-of" relationship in which objects representing components of something are assembled to a higher-level aggregate object. Generalization represents the "is-a" relationship and association corresponds to the "member-of" relationship. An object class defines a structure into which objects with common behavior and common relationships to other objects are associated together. An object has data attributes defining its properties and may have private functions presenting the object to the others and keeping its private information hidden. Since classes are viewed as types of objects, they may have data types as attributes. In HOM, we combine the notions of OMT [RuBl91] with the E/R model to represent class structures and class relationships graphically. Fig.4-1 is a simple example diagram of HOM.

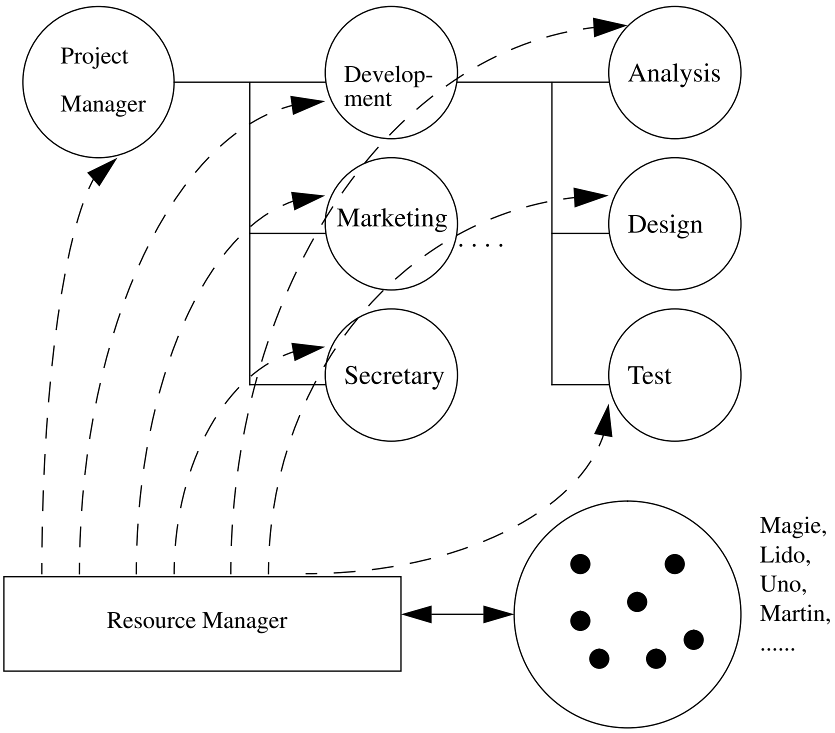


Fig. 4-2

4.2 Role Concept and Organization Model

For the staff of an organization, roles are status of persons and specify their functions, privileges, responsibilities, etc. For the organization, on the other side, roles also correspond to positions in the organization structure. Therefore, roles combine the personnel with the organization structure. In HOON, it is allowed that the staff members are not bound to the organization structure fixedly but can be dynamically assigned to each workflow activity (transition). We adopt the hierarchical organization structure as shown in fig.4-2. In the diagram, the dynamic assignment of the staff members is also demonstrated. For the running efficiency, default values can be declared in the process model, i.e. some workflow activities may have default actors associated with them. Corresponding to the reality, one person may hold many positions and some positions may be held by more than one persons.

In the working nets of HOON, role is a single meta type whose user-defined member types (role types) are able to be assigned to the control places. The concrete actors like the staff of an organization or some functional software tools can be attached to the user-defined role types. The resource manager shown in fig.4-2 is responsible for the dynamic dispatch of the actors.

5. Interpreting Higher-order Object Nets

A model is an abstract and synthetical description of one or more concrete objects. The conceptual interpretation of the model gives semantics to the model and the concrete interpretation of the model results in changes and movements of the objects. Terminologies like model execution or model enaction are sometimes also used to mean concrete model interpretation. But there is a subtle difference between the term enaction and execution in this paper. The enaction of a process model results in concrete processes and the execution of the concrete processes results in the changes and movements of real things. A good terminological clarification can be found e.g. in [Lonc93]. The word interpretation that appears in this paper always means concrete model interpretation. In this section, we describe how a workflow management system can manage workflows by interpreting the corresponding Higher-order Object Nets.

Places, transitions, edges, and markings are mathematical concepts. When used for controlling workflows, the process model must interpreted in terms of the targets of the real world being modeled or first in terms of a simpler and more tangible model. As a matter of fact, high-level Petri nets are conceptually interpreted in terms of the Place/Transition nets. In HOON, a net object corresponds to an independent skeleton of workflow segments, normally common to several workflows. Transitions correspond to autonomous workflow activities and places are interpreted as buffers between the autonomous nodes. According to the data types, tokens on a place in the working nets are interpreted as simple workflow data, references, or messages. Tokens in the control nets represent e.g. encapsulated software components, workflow segments, or actors like human resources. An edge corresponds to a communication connection or a information channel. Control nets are responsible to supply and to maintain shared software tools, business objects and other shared resources for the workflow activities. An edge expression determines which token(s) can be passed through the information channel. The firing of a transition in the model corresponds to preparation and execution of one or a series of actions in the reality. The real processes progress in accordance with the interpreted process model.

For controlling real workflows with the aid of computers, the process model must be interpreted with computer software. The software tool which fulfills this job is called *workflow model interpreter*. In a workflow system, several instances of the interpreter may run in parallel. The instances are called *workflow engines*. Each time a net object is activated, a new workflow engine will be started. Centered on the interpreter of workflow models, a workflow management system can be constructed as shown in fig.5-1. The workflow interpreter should possess the following features:

- The interpreter should be application-independent. Normally, the design of a traditional office information system is influenced by the application-specific aspects and the running environment. The use of HOON should contribute to the application-independence of a workflow management system. To be application-independent, all application relevant things must be reflected in the workflow model and the application-specific resources must be recognized by the interpreter through an interface. Although different applications have different workflow models resulting different processes, the workflow model interpreter remains the same.

- With the same interpreter, it should be possible to switch back and forth between workflow simulation and control of real workflows.
- Only proxies of resources appear in the process model and the resources themselves are managed by the resource manager. Therefore, the model interpreter must cooperate with the resource manager to control real workflows

Fig.5-1 shows a diagram of a workflow management system. Both model designers and users interact with the workflow management system through user interfaces. Workflow models are developed by the model designers. The workflow structure and model segments are managed by the workflow model manager supported by a workflow repository. The workflow model interpreter controls workflows by interpreting the workflow model. During the interpretation, resources including both human resources and software tools may be assigned or activated (with the help of the resource manager). The activation of active resources controls one or more work steps of the real workflow. The resource manager manages the workflow resources and the interfaces to the external world including the interfaces for user interaction. The workflow repository stores information about the application, meta information about business objects, software components, and segments of the workflow model.

The resources in a HOON system are encapsulated as objects of different granularity. To support object interaction in a distributed heterogenous environment, use of the OMG's specification for distributed object computing - CORBA - is suggested. CORBA provides a foundation for higher level distributed object collaboration and therefore provides an infrastructure to integrate distributed application components. The integration and incorporation of distributed application components and resources are of key interest for a workflow management system. Based on a CORBA-implementation, the resource manager can treat all encapsulated resources available in a distributed environment as a commonly accessible collection of objects.

A great portion of workflow activities must deal with exceptional situations. By exceptions we mean both erroneous and acceptable situations which are unpredictable. As consequences of exceptional situations, reconfiguration of the workflow structure, rescheduling of activities, reassignment of personnel or software tools are sometimes unavoidable. As indicated in Fig.5-1, the exceptions are reflected by feedbacks from the services, the resource manager captures the feedbacks, and so on. Finally, the feedbacks are reflected on the workflow model through creation or removal of certain control tokens.

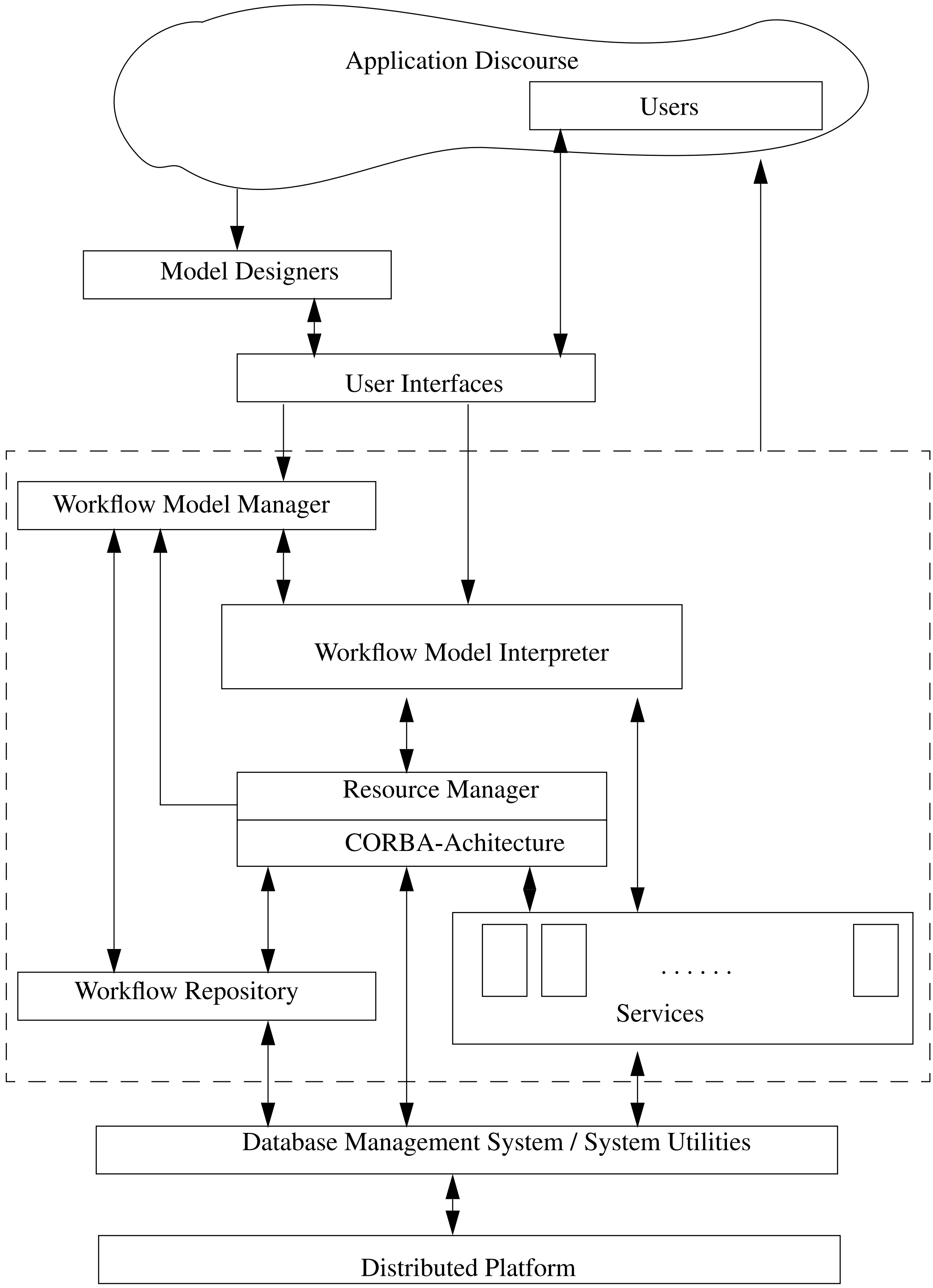


Fig.5-1 Diagram of A Workflow Management System

6. Conclusion

In this paper, we proposed a petri-net-based process modeling language - HOON. HOON-based models have a very flexible process structure and can, therefore, better cope with the evolutionary and exceptional dynamics. In our method, human resources, data objects, software components as well as well-defined subnets are modeled as special tokens. As a result, they are "live" in the process model. Moreover, for modeling and controlling workflows, it is proposed that the control structure representing synchronization and dependency of workflows and the management of the related resources are modeled with two distinct but closely related subsets of the process model: working nets and control nets. As such, there is a clear boarder between standard workflows and the resource management activities, as well as between foreseeable structured parts and unforeseeable creative parts. The interfaces between the two classes of nets are the control places introduced in HOON. We also hope that the work loads of workflow modeling and system implementation can be in a better balance. In this paper, the main promising concepts have been described. However, most of the features need to be further validated and evaluated.

Object-oriented concepts are introduced to the different levels at different grades of completeness. Some problems need also to be further investigated. For example, the Petri net rule of local effects of transitions is sometimes violated dealing with persistent objects. If we allow a net object to have local memory, it becomes more difficult to decide when the "show" of a net object comes to an end.

In this paper, analytical issues are avoided intentionally. A complex model is not easy to analyze (if not impossible). Till now, the analysis of Petri nets is confined to the elementary or limited net classes. To make analysis, high-level nets are transformed into elementary nets. This kind of transformation is also called unfolding. Funsoft nets, for example, has been unfolded to PrT nets in order to use the analysis algorithms of PrT nets. Although we can derive equivalent Funsoft nets for HOON at any fixed point of time, the run-time combination of net objects in a large workflow model is nearly impossible to track. By now, the analysis of HOON is still an open question to be answered. One possibility is to use a hierarchical divide-conquer approach. Each atomic net object is first analyzed, then net objects at one level higher are analyzed where the analyzed atomic net objects are treated as single entities, the process continues till the top level is reached.

As future directions of our research, the following interesting points are going to be investigated with the help of several diploma theses: interfaces between running instances of a process model and interfaces between processes and resource management at an operational level, as well as issues about repository support. Then, in order to validate the virtues of HOON, the general interpreter as proposed in section 5 is going to be implemented. In this paper, HOON are not formally defined. Based on the same foundation of the HON-definition as presented in [LöHa94], the formal definition of HOON is also part of the future works.

References

[BaBr88] Baldassari M. and Bruno G.: <<An Environment for Object-Oriented Conceptual Programming Based on PROT Nets>>, Advances in Petri Nets 1988, LNCS340, Springer Verlag, pp1-19

[BaFu93] Bandinelli S.C., Fuggetta A, Ghezzi C.: <<Software Process Model Evolution in the SPADE Environment>>, IEEE Tran. on Soft. Eng., Vol.19, No.12, Dec 1993

[Baum90] Baumgarten B: <<Petri-netze : Grundlagen und Anwendungen>>, Wissenschaftsverlag,1990

[BuGu91] Buchs D. and Guelfi N.: <<CO-OPN: A Concurrent Object Oriented Petri Nets Approach>>, Application and Theory of Petri Nets 1991

[Chen76] Chen P. P. S.: << The Entity-Relationship Model: Toward A Unified View of Data>>, ACM Tran. on Database System, Vol.1, No.1, pp 9-36, 1976

[CuKe92] Curtis B, Kellner M, Over J : << Process modeling>> , Comm. ACM, Vol.35, N0.9, Sep,1992

[DeGr94] Deiters W, Gruhn V: << The FUNSOFT Net approach to software process management>>, Int. Journal on Software Engineering and Knowledge Engineering, Vol. 4, No.2, 1994

[DoNe91] Dowson M, Nejmeh B, Riddle W : << Concepts for Process Definition and Support>>, Proc. of 6th International Software Workshop, 1991

[EhMa85] Ehrig H.; Mahr B. : <<Fundamentals of Algebraic Specifications 1>>, EATCS Monographs on Theoretical Computer Science 6, Springer-Verlag, 1985

[EhPa93] Ehrig H, Padberg J, Ribeiro L: << Algebraic High Level Nets: Petri Nets Revisited>>, Tech. Report 93-6, TU Berlin,1993

[ElNu80] Ellis C., Nutt G. <<Office Information Systems and Computer Science>> Computing Surveys 12(1), pp.27-60, 1980

[ElNu93] Ellis C., Nutt G.: <<Modeling and enaction of Workflow Systems>>, Application and Theory of Petri Nets 1993, LNCS691, Springer-Verlag, pp1-16

[Fisc95] Fischer U.: << Prozessmodellierung mit Petrinetzen: Eine Fallstudie der Vor- und Nachbereitung vob Besprechungssituationen>>, Studienarbeit, TU Berlin, 1995

[FlLi93] Fleischhack H., Lichtblau U., Sonnenschein M. and Wieting R.: << Generische Definition Hierarchischer Zeitbeschrifteter Hoeherer Petrinetze>>, Berichte Nr. AIS-13, Universitaet Oldenburg, 1993

[Gen86] Genrich H. J.: <<Predicate/Transition Nets>>, LNCS254, 1986

[GeLa81] Genrich H. J., Lautenbach K : <<System modeling with High-level Petri Nets>>, Theor. Comp. Science 13(1981), pp109-136

[God83] Godbersen, H. P.: <<Funktionsnetze>>, Ladewig-Verlag, 1983

[Grei88] Greif I. (ed.) : <<Computer-Supported Coorperative Work : A Book of Readings>>, Morgan Kaufmann Publishers, Inc., 1988

[Gruh91] Gruhn V.: <<Validation and Verification of Software Process Models>>, Diss. Universität Dortmund, 1991

[Gruh93] Gruhn V.: <<Entwicklung von Informationssystemen in der LION-Entwicklungsumgebung>> in [ScRe93]

[Holt88] Holt A. W.: <<Diplans: A New Language for the Study and Implementation of Coordination>>, ACM Tran. on Office Information Systems, Vol.6, No.2, April 1988, pp109-125

[Jens92] Jensen K.: <<Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical Use>>, Vol. 1, Springer-Verlag, 1992

[Klug86] Kluge W.: <<Reduction, Data Flow and Control Flow Models of Computation>>, LNCS256, 1986

[Kung89] Kong C. H.: <<Conceptual Modeling in the Context of Software Development>>, IEEE Tran. on Software Engineering, Vol.15, No.10, 1989, pp1176- 1187

[Lako94] Lakos C. A.: <<From Coloured Petri Nets to Object Petri Nets>> Tech. Report, TR94-9, Uni Tasmania, Australia, 1994

[Laus88] Lausen G.: <<Modeling and Analysis of the Behavior of Information Systems>>, IEEE Tran. on Software Engineering, Vol.14, No.11, 1988, pp1610-1620

[Lonc93] Lonchamp J.: << A Structured Conceptual and Terminological Framework for Software Process Engineering>>, IEEE Computer, 1993

[LöHa94] Löwe M., Han Y.: <<Process Modeling and Control with Higher-order Nets>>, Tech. Report 94-34, TU Berlin, 1994

[Mura89] Murata T.: <<Petri Nets: Properties,Analysis and Applications>> Proc. of IEEE 77(1989), pp541-580

[OMG91] Object Management Group: <<Common Object Request Broker: Architecture and Specification >>, Document 91.12.1, OMG 1991

[Pete81] Peterson J. L.: <<Petri Net Theory and the Modeling of Systems>>, Prentice-Hall, 1981

[Reis86] Reisig W.: <<Petrinetze>> , 2.Auflage,Springer-Verlag,1986

[Reis91] Reisig W.: <<Petri Nets and Algebraic Specifications>>, Theor. Comp. Science 80 (1991), pp1-34

[ScRe93] Scheschonk G., Reisig W.: <<Petri-Netze im Einsatz für Entwurf und Entwicklung von Informationssysteme>>, Springer-Verlag, 1993

[Sibe93] Sibertin-Blanc C.: <<Cooperative Nets>>, Application and Theory of Petri Nets 1993, LNCS691, Springer-Verlag, pp377-396

[Trüm88] Trümner H.: <<Functionsnetze: Ein auf Baustein basierender Ansatz zur Software-Konstruktion>>, Diss. TU Berlin, 1988

[Verk93] Verkoulen P. A. C.: <<Integrated Information System Design: An Approach Based on Object-Oriented Concepts and Petri Nets>>, PhD Thesis, Technical University of Eindhoven, the Netherlands, 1994

[Warb94] Warboys B.: <<Reflections on the Relationship between BPR and Software Process Modeling>>, Entity-Relationship Approach - ER'94, LNCS881, Springer Verlag, 1994

[WfMC94] Workflow Management Coalition: <<Glossary - A Workflow Management Coalition Specification>>, 1994

[Wika90] Wikarski D.: <<Object Nets - A Canonical Class of Models for Behavior Simulation and Structure Systhesis of distributed Systems?>>, Informatik - Informationen, reporte 6, 12, Berlin, 1990, pp91-100

[Wika95] Wikarski D.: <<Nutzer-Handbuch fuer Prozessnetze>>, ISST - Intern, 3, 1995

[WoKr93] Woetyel G., Kreifelts T.: <<The Use of Petri Nets for Modeling Workflow in the DOMINO System>>, Workshop on CSCW, Petri Nets and Related Formalisms, Int. Conf. on Appl. and Theory of Petri nets, 1993

[Zism77] Zisman M. D.: <<Representation,specification and automation of office procedures>> Ph.D. dissertation, Wharton School, Univ. Pennsylvania,Philadelphia, Pa., 1977