Published in: "Practical Aspects of Knowledge Management", Proceedings of the First International Conference, October 30-31, 1996, Basel, Switzerland

Management of Workflow Resources to Support Runtime Adaptability and System Evolution

Yanbo Han[1], Jürgen Himmighöfer[1], Thorsten Schaaf[1], Dietmar Wikarski[2]
1. CIS, Technical University Berlin, EN7, Einsteinufer 17, 10587 Berlin, Germany
yanbo@cs.tu-berlin.de, jh@cs.tu-berlin.de, tschaaf@cs.tu-berlin.de
2. Fraunhofer Institute for Software Engineering and Systems Engineering ISST, Kurstr. 33, 10117 Berlin,
Dietmar.Wikarski@isst.fhg.de

Abstract

This paper proposes some ideas for the enhancement of the flexibility and adaptability of workflow systems. The flexibility and adaptability problem is discussed from two perspectives, namely design and evolution of workflow software from a system perspective as well as redesign and runtime on-the-fly modification of workflow models from an application perspective. We argue that based on a systematic management of workflow resources including workflow models, software components, persons and roles, as well as data and documents, workflow systems can be engineered from these two perspectives convergently.

A new process modeling language, called Higher-Order Object Nets (HOON), is discussed. The intended uses of HOON are not only to directly describe real-world business processes but also to serve as an overall control skeleton of the underlying software components supporting individual business activities. Besides inheriting the basic features of Petri nets, HOON incorporate mechanisms which enhance the structural flexibility of Petri nets through dynamic binding of resources, and support reuse and composition of submodels in an object-oriented way.

Concerning systems design and evolution, a process-model-driven framework for developing configurable workflow systems is presented. The framework, which is based on HOON and the distributed object computing technology CORBA, can serve as the system core for a variety of workflow systems and remain stable in the process of system evolution, in which software components and process models are all subject to modification and substitution.

1. Introduction

In a large number of organizations, such as public administrations, bank and insurance branches, or even software development projects, the daily operation is governed by a set of cooperative dynamic business processes in which intensive group-oriented human interactions are involved. A business process is an ordered set of business activities. Workflow models refer to formally described business processes in these organizations. Workflow models and workflow software (i.e. the software that supports the workflow models and workflow activities) constitute a workflow management system. The term workflow system generally refers to a whole system consisting of a workflow management system, the real-world processes, as well as the material, software and human resources relevant to the workflows. Note that, in the context of this paper, we use workflow and (business) process as synonyms.

Traditionally, workflows were directly implemented in their corresponding workflow software leading to an inflexible whole. Newer methods separate process information from application programs and migrate the embedded information to process models. The process models function here as "very-high-level" programs and are interpreted by a process model interpreter.

Recently, the importance of the structural flexibility and system adaptability in workflow systems has become increasingly apparent. From the application perspective, process models are subject to two different forms of modifications: off-line and on-line modifications. The rapidly changing business climate forces corporations to adopt business process reengineering. However, it is not enough to radically redesign the business processes once and for all. Between fundamental redesigns, corporations must frequently remodel or adjust their business processes and then adapt their software systems quickly to the modified processes. This kind of model modifications are referred to as off-line modifications. In fact, nearly all business processes consist of two parts: the structured part which defines the baseline frames of the business processes and enforces discipline, and the unstructured part which handles unpredictable events and often requires human interventions. At runtime, the flexibility and adaptability of process models are also indispensable in order to handle unpredictable situations. When handling non-routine events, on-line model modification or even dynamic model reconfiguration is frequently required.

In this paper, we will discuss the flexibility and adaptability problem with a concrete scenario close to the one suggested by the workshop organizers, namely process support for clinical therapeutic treatments. In this example, the baseline structure of the process to handle patients is relatively fixed: the passing on of a case from a disease specialist to the laboratory for blood analysis, to technicians for actual treatments, and so on. However, while the baseline process can allocate responsibility for subprocesses department by department, it can less easily handle the unpredictable schedules of hospital staff and the uncertain availability of diagnostic and treatment equipment. Ad-hoc extensions to the baseline process are often needed. This concrete scenario will be modeled and discussed in more detail later.

From a systems design and evolution perspective, workflow software is confronted with a twofold evolution problem: coming from process model changes and from software component updating. To cope with the dynamics of business process remodeling, the workflow software must have a generic structure and flexible interfaces to adapt to the changes in workflow models. On the other side, the software components that are responsible for individual business activities and that operate normally on heterogeneous platforms are subject to frequent updating. These software components include standard software, subsystems designed or tailored for specific tasks, and the so-called legacy systems that are old but still useful. As an integrated federation of these individual software components, workflow software also undergoes constant evolution. Although much standardization effort, such as CORBA-specification[1], has recently been made to facilitate abstraction and interfacing of components and to cope with system heterogeneity and integration problems, suitable higher-level mechanisms are still needed to be

1. CORBA (Common Object Request Broker Architecture) defines a standard to support interoperability and portability of distributed objects [OMG91].

able to directly represent and manipulate distributed objects at the process model level.

Attempts have been made to redesign business processes by modifying only the process models, leaving the underlying software structure untouched. But this may cause inconsistency and mismatches. In fact, the mismatches between business engineering and software engineering have generally been recognized [Tayl95]. The emphasis of our current work is on a systematic support for engineering workflow systems from the application and the software perspectives convergently. A new approach is proposed which advocates the integration of process models and their software systems. The key concept of the approach is the systematic management of workflow resources and the dynamic binding of resources in process models. The key technique of the approach is to establish a set of open interfaces between models and their external environments and to provide technological support to these open interfaces. In this paper, both conceptual and informational entities are treated as workflow resources. These entities include workflow models, software components, roles, informational representation of persons, as well as data and documents. Workflow resources like software tools and human resources are often characterized as active, and those like data and documents are characterized as passive. The encapsulated entities containing both passive resources and active resources are called resource objects. Of course, the active and passive resources can be considered as special forms of resource objects.

In our approach, executable process models are used not only to directly reflect the real-world business processes but also to function as the overall control skeleton of the underlying software components. Therefore, one of the central topics of the paper is to present a new process modeling language, called Higher-Order Object Nets (HOON) [LöWi95]. Furthermore, a process-model-driven framework which can serve as the generic system core for a variety of workflow systems and remain stable in the process of system evolution is also briefly presented in the paper.

2. Constructing Flexible Workflow Models with Higher-Order Object Nets

2.1 Workflow Modeling with Petri Nets

In recent years, many new and established formalisms have been used for modeling workflows, for instance, rule-based [PeSc92], speech-act-based [Wino89], [MiGr93], and Petri-net-based approaches [BaFu93], [DeGr94], [FaWi96], [Wika96]. As a conceptual formalism, Petri nets [Reis86], [Mura89] have been used extensively in many application fields. Petri nets have an intuitive graphical representation, a well-defined semantics and a formal background which can provide better support for system analysis and simulation.They are particularly suitable for modeling concurrent and non-deterministic processes. Most importantly for process control, Petri nets are an executable model provided that the firing rules and the ways of conflict solution are appropriately specified. Due to the above-stated virtues, we have chosen Petri nets as the candidate for our purposes stated above. However, extensions have to be made because, mainly due to their fixed structures, Petri nets and the executable extensions known to us are not flexible enough to cope with evolutionary or exceptional dynamics at the model level, especially at runtime. The next paragraph discusses some concrete problems.

On the net graph of standard types of Petri nets, one can differentiate between active and passive elements, namely, transitions and places. Usually, the active and passive resources as discussed earlier in a modeled application system are naturally associated with their counterparts in the corresponding net model in such a way that they are either buried implicitly and fixedly inside transitions or associated to typed places at modeling time. Consequently, the runtime consumption of certain resources cannot be properly reflected. For example, in many executable variants of high-level Petri nets such as Function Nets [Godb83] or Funsoft Nets [Gruh91], executable files or routines are attached to transitions and are called whenever their transitions fire, under the assumption that these active resources are unconditionally available. In fact, some resources like commercial software tools may be licence-restricted or shared in a limited way. Moreover, runtime communication of a Petri net with its environment and dynamic configuration of net models are almost untouched issues. Here, rearrangement of certain resources, such as exchange of software tools, modification of submodels, or redeployment of staff, normally means structural modifications to the overall net model. To sum up, when used for modeling and controlling workflows, Petri net models have to be extended to incorporate dynamic resource provision[1], especially in cases where support to the ad-hoc part of business processes is strongly needed.

2.2 Informal Description of Higher-Order Object Nets

An important motivating factor which has brought forth the new process language - Higher-Order Object Nets - is to extend Petri nets so that the above-stated problems can be solved. The central idea of HOON is to arrange net models and their surrounding environments in a client/server manner and to model the client/server interfaces explicitly. The interface between a process model and its external environment is explicitly modeled with a set of special places called resource places, through which information can be dynamically exchanged. As is common in client/server software systems where multiple levels of proxies of real resources exist, the resource entities in HOON may also be proxies which refer to other proxies or the real entities. In HOON, places may contain 1) data that are private to their process models and comparable to the transient program variables; 2) resource entities that are external to process models and encapsulated as distributed objects (the encapsulated objects are persistent and managed systematically by workflow software, whereas the internal data of processes are transient); or 3) a combination of private data and references to certain external distributed objects. Transitions of HOON represent only skeletons of process activities. The active resources that are fixedly bound to transitions in many executable variants of Petri nets can now be dynamically bound to transitions during transition firing. The breaking-up of the fixed association is essential for enhanced flexibility. In HOON, provision of resources in nets will cause rearrangements or actions in the real world, and rearrangements or actions in the real world may result in a new assignment or modification of resource assignment in nets.

Except for the introduction of a new graphical symbol - the resource place - the graphic notations of HOON are similar to the standard ones of Petri nets. However, in order to incorporate mechanisms supporting late-binding of resources, handling of exceptional events, dynamic model configuration, copy-read, and selective input and output, etc, the

---

1. Resource provision generally refers to the procedures in which request, assignment, modification, invocation, release and reallocation of resources are performed.

semantics of the standard graphical notations have been extended and some extra labels are introduced. For instance, copy edges are introduced to ensure that tokens on an input place have influence on the transition firing, the token values can be read, but the tokens will not be really removed. The copy edges are graphically represented as edges without arrows (see Figure 4 and 5). The important features of HOON, together with some incorporated mechanisms, are discussed in the following subsections. The graphical notions are explained with the example later (for a detailed description see [LöWi95]).

2.3 Open Interfaces between Models and Their Environments

In [LöHa94], we proposed Higher-Order Nets (HON) which generalize the data types that can be used in nets to include function spaces and power sets. As a result, functions can also be modeled as tokens. Figure 1 shows a standard Higher-Order Net. The shaded place indicates a higher-order place on which tokens representing functions can be laid. Figure 2 is a concrete example of HON. In this example, a text is to be edited with a software tool - FrameMaker text editor. The available licences for the FrameMaker tool are modeled as tokens on the higher-order place. If no licence is available, the transition cannot fire. In normal executable variants of Petri nets, in contrast, a special program must be written to solve this kind of problem or deadlock may occur[1].

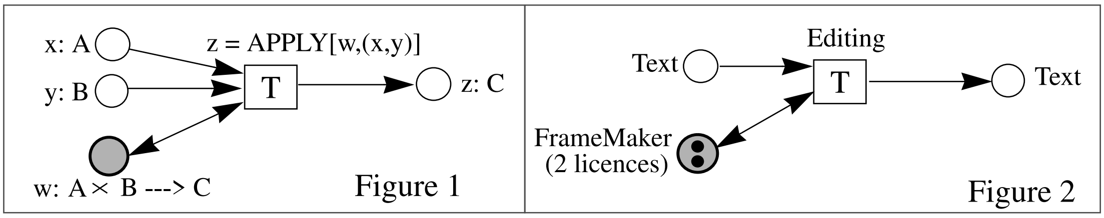


Figure 1
Figure 2

Higher-Order Object Nets are a further development of HON. We have recognized that the part of nets which deals with the management of the external resources (see the diagram surrounded by the rope ring in Figure 3, where the dark transitions correspond to the activities manipulating resources) is irrelevant to real-world businesses. Therefore, a HON model is separated into two parts: workflow nets and resource control nets. The workflow nets are responsible for describing information flows between business process activities. The information flows are expressed by the changes on the prestates and the poststates as well as the temporal sequences of the process activities. The user interaction and the management of workflow resources are modeled and controlled within resource control nets. For some very complicated problems, it is imaginable that we have several such resource control nets at different levels and the $n-1$ -th level resource control nets are the workflow nets for the n -th level resource control nets. The resource control nets can of course also shrink to a single software component managing all resources. This is exactly the case in this paper.

Along the lines of the above discussion, we further suggest that all relevant higher-order places for a transition are assembled together to form a meta place[2] - the resource place.

1. Assume that we have 10 FrameMaker licences and 12 transitions are equipped with the software tool. If all 12 transitions are enabled and fire, the following deadlock situation may occur, i.e. 2 transitions have consumed their input tokens but can do nothing, and no output tokens can be produced.
2. A meta place is an accumulation of places. Since a member place can also be a meta place, a meta place may be hierarchically structured. Tokens on a meta place can naturally be of different data types.

A further unharmful restriction is made to facilitate systematic model interpretation, namely, each transition owns a resource place exclusively. This means that shared higher-order places are split and some conflict situations are hidden from the workflow nets. These conflict situations can be easily reflected in the resource control nets where the split places can be merged again. It is the resource control nets or the software tool that is responsible for the solution of the conflicts caused by any resource contention. Resource places are shared by the two sorts of nets. Graphically, they are represented as ovals (see the oval element in Figure 4). From the viewpoint of business processes, the net shown in Figure 3 is equivalent to that shown in Figure 4.

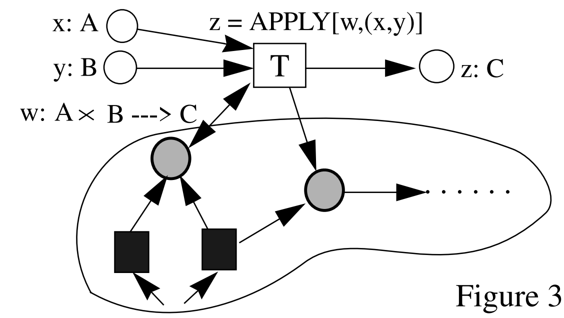


Figure 3

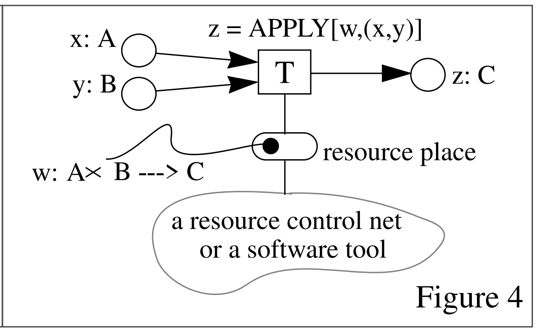


Figure 4

Tokens on a resource place, called resource tokens, are of various types. Resource tokens represent resources at two different angles of observation, leading to declarative and operational resource tokens. Through the exchange of operational resource tokens, the dynamic interaction between a working net and its "external world" can be modeled and interpreted. At the declarative level, resource tokens represent the subset of resources visible to their transition. Through assignment or withdraw of declarative resource tokens, the configuration of a net model can be changed (see also section 2.5). A script language is developed for the specification of declarative resource tokens. The content of a declarative token is a script. The script determines the visibility of the associated transition to certain classes of resources (i.e. various possible choices at firing time). Moreover, a script defines the conditions under which a certain resource or class of resources may be requested, the marshaling and demarshaling of input/output arguments, as well as the operations to be performed.

2.4 Object-Oriented Resource Representation in Nets

Much effort has been made in the Petri net research community to make Petri nets "object-oriented", see, for example, [BaBr88] [BuGu91] [HeVe91] [ScRe93] [Sibe93] [Lako94]. We still doubt whether a pure object-oriented version of Petri nets is feasible because of some intrinsic distinguishing characteristics of Petri nets such as the intentional and explicit separation of states and state changes (passive and active elements), the emphasis on the flow relation, and the special ways to model concurrency and nondeterminism. Furthermore, it is difficult to naturally define and apply class hierarchy and object inheritance at the net level. In our opinion, a coexistence of Petri net formalism and object-oriented technology is more meaningful. Petri nets can be used to express inter-object interactions and objects are treated as tokens in the context of Petri nets. If we again treat a Petri net as an object, the two paradigms can be applied in an integrated but unmixed way. However, many problems still have to be solved to realize this. For example, mechanisms are needed to deal with the following problems in nets: how to facilitate object interactions and message passing, maybe including even message broadcasting in nets; how to derive and represent objects which combine both active resources and passive resources; how to deal with object persistency which indicates that objects have their own life cycles, identities, and local memories (though, in the Petri net world, tokens are not at all persistent). According to the standard Petri net definition, transitions do not hold any data and tokens are dynamically created or destroyed by 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, in [Wika90] for instance, to keep potential conflicts within the objects which correspond to a special kind of subnets. This idea is also incorporated in HOON.

In our approach, tokens on resource places may represent persistent objects. These coarse-grained objects are encapsulated resources managed in the underlying software system (the object structures and relationships are described in an accompanying object model which is not included in this paper). Although in nets all object tokens are of the single token type "Distributed-Object", we operationally distinguish objects at different levels of granularity, i.e. net objects, component objects, data objects, and a special kind of object tokens representing persons.

Net templates are structural abstractions of subnets, i.e. they are skeletons with topological information of the corresponding subnets. Mature and reusable net templates are stored in a special database called HOON process repository. 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 are instances of the net templates and can be represented by tokens. In this way, a subnet can be associated with a transition dynamically by assigning the corresponding token to the resource place of the transition.

Component objects are encapsulated software components which have special functionality and provide services through defined interfaces. During a transition firing, a component object can be activated if its activation predicate defined in the script is evaluated to be true. 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. A component object is encapsulated with the CORBA interface specification language IDL, it interacts with its "outside world" only through the defined interfaces.

Data objects are the data entities constructed and encapsulated in an object-oriented way. They are normally grouped into classes and the full description of data object classes is stored in the HOON process repository. During net interpretation, data objects are bound to a net element by referencing the HOON process repository.

Roles are a special kind of data object classes. Objects of such a class represent human beings who play the defined role in a certain context. Roles and persons have an n-to-n relationship.

2.5 Dynamic Configuration of Models and Handling of Abnormality

To explain the firing behavior of HOON, the semantic refinement of a single HOON-transition is diagrammatically drawn in Figure 5. The bigger place P_1 in the enlarged resource place, still drawn as a meta place, indicates various declarative resource tokens showing which resources or classes of resources are visible to the transition. Each declarative token corresponds to a script which specifies the input/output predicates as well as the concrete behavior. The smaller places in the enlarged resource place contain operational resource tokens modeling runtime resource provision. If the input places are marked and the precondition of a declarative token is satisfied, the transition T_1 will fire[1], sending a request for services to the outside. According to this standard refinement, input tokens will not be destroyed until the needed resource objects are provided, that is, the input tokens of the refined HOON-transition will be consumed when the transition T_3 fires. Note that the edges without arrows indicate copy edges. If the request fails, the built-in exception handling procedure will be called. In response to a certain non-routine event, the built-in exception handling mechanism (T_5) may interact with the underlying resource control nets or a software tool in order to request a new resource token. New resource tokens can be dynamically assigned to the place P_1 so that T_1 may fire once more. More complicated exception handling procedures can be statically specified in a script, or alternatively, what happens in an exceptional situation can be decided by the responsible person.

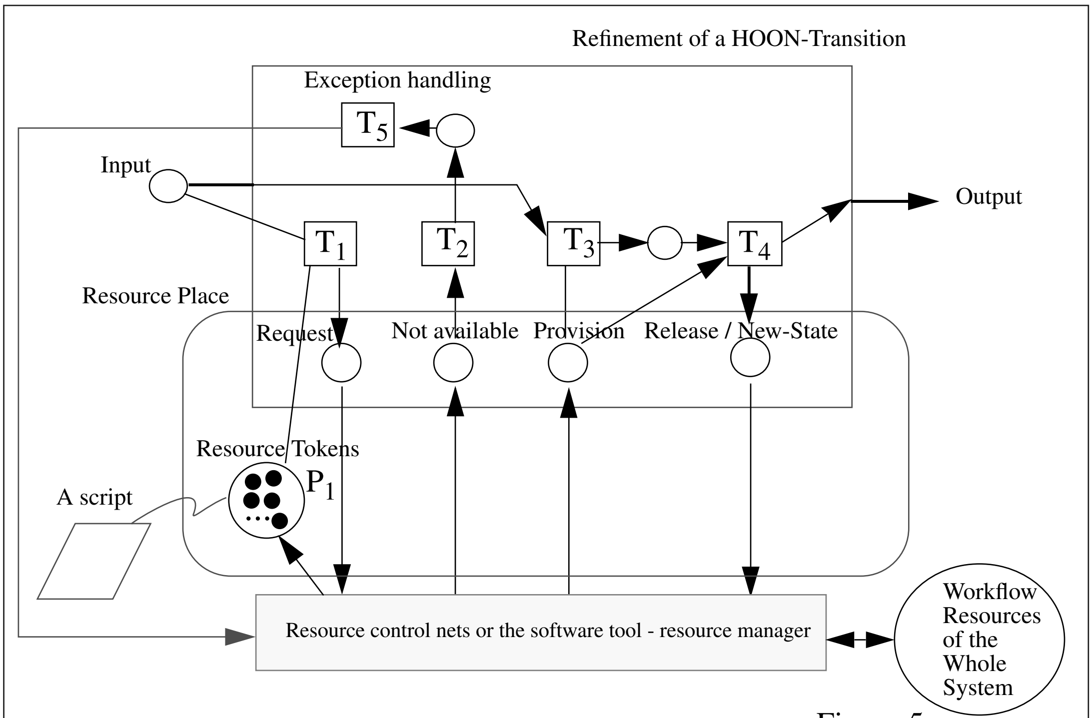


Figure 5

In HOON, within a single transition firing, several parallel activities can be instantiated. Here, a single transition firing means that the input token will be read only once. This feature is modeled with Transition T_3 and T_4. The transition T_4 fires repeatedly as long as there are any fresh operational resource tokens on its preplace.

When a net object token on a resource place is „unpacked“, a net at a lower level will be instantiated. Note that resource places of the instantiated net may again contain net object tokens. A HOON model can practically be a dynamic and hierarchical combination

1. For control purposes, transition firing normally conforms to the must firing rule, i.e. a transition fires immediately when it is enabled.

of various net objects instantiated from net templates. A net template can be modified even when its former instances are still running. As resource tokens at the model level, net objects can be assigned or withdrawn by hand (by a double-click on the corresponding resource place, see also the example in the next subsection), or automatically by the resource control nets or an appropriate software tool. As a result, we can obtain a very flexible, reconfigurable net topology. This new method of hierarchical model construction is intentionally shown in the example in Figure 8. Traditionally, the whole business process is designed by a team of designers as a predefined plan. During the process execution, tasks appearing in a to-do list are delegated to persons who only function passively and have limited scope for their own decision-making. However, a real business process can seldom be 100% defined in advance because non-routine events at various management levels frequently take place. Also from Figure 8, one can see that more than one choice may be provided to a HOON-transition, allowing a certain amount of local decision-making.

3. Technological Support for Systems Development and Evolution

3.1 Model-Driven Systems Development and Evolution

With suitable abstraction and composition mechanisms, software components can be used, instead of programming language elements, as basic building blocks in constructing software systems. The overall system behavior can be determined by the internal behavior of the components and the inter-component interactions. If the interaction schemes which link the components together are reflected within the process model, the system can then be (re)configurated by constructing and manipulating the process model. However, model changes should be performed on the basis of available workflow resources.

Based on the above considerations, we propose a consistent process-model-driven approach for designing workflow systems. The meaning of model-driven is twofold here: If the workflow system is driven by a process model during its execution, the system behavior will be directly controlled by the model interpretation. In this case, the model can be imagined as a very-high-level program. If the model-driven principle is applied to the development process, the models can also be used to guide all development life cycle activities.

Figure 6 is the referential model of our approach. A workflow system consists of three inter-related parts. An explicit boundary is highlighted between two abstraction levels, namely conceptual level and system level. Resources are encapsulated and registered in the resource pool as CORBA-based distributed objects. The distributed objects are derived in such a way that they can be added, dropped, distributed, substituted and modified separately and independently of each other. On the conceptual level, there is no fixed central control, or rather, each reconfigurable process model represents a thread of control. Distributed objects related to a real process are attached to the corresponding process model either statically at modeling time or dynamically at runtime. The execution of process models results in real processes which coordinate underlying distributed objects and other kinds of resources. The execution of such a process model is equivalent to the execution of a traditional system with fixed structures. In other words, each process model together with its model interpreter can be viewed as a workflow system (or a subsystem in a larger context). As a result, we can derive different configurations of a workflow system simply by constructing suitable process models. Furthermore, different process models can be executed concurrently and may be coordinated again by a more abstract process model (compare the "higher-level process models" in Figure 6). Processes at the same level or in different hierarchical branches may run concurrently in a distributed environment. Higher-level reuse is promoted by treating instances of process model templates as persistent net objects. The net objects can be used as white boxes because other net objects can be assigned to their resource places. As a result, a flexible hierarchical system control structure can be realized.

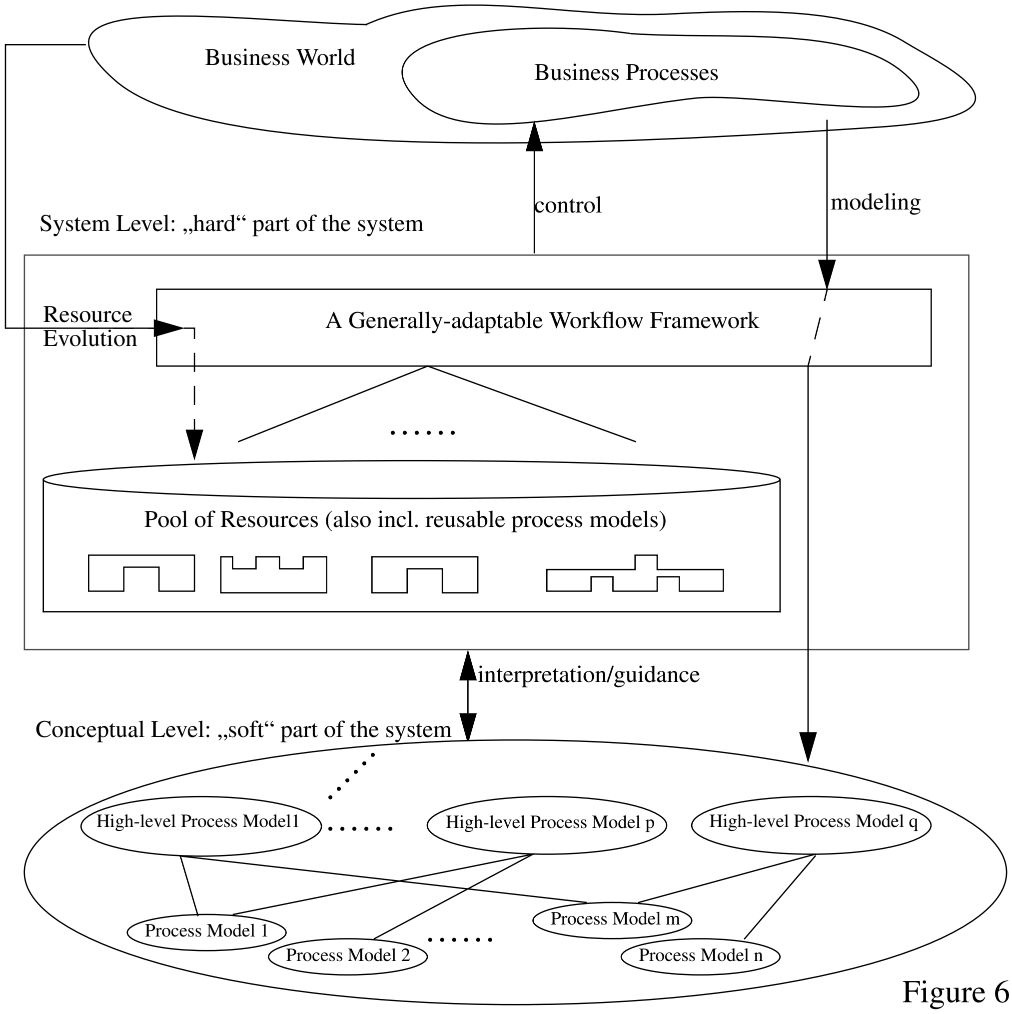


Figure 6

3.2 The Model-Driven Framework

This subsection presents a process-model-driven framework which provides technological support for the development and evolution of configurable workflow systems. Figure 7 shows the architecture of our framework whose functional view has already been shown as a constituent part of the referential model given in Figure 6.

In designing workflow systems, we normally have to deal with large-scale cooperation and coordination across different operating systems, platforms, and even networks. The CORBA-based infrastructure provides a basic mechanism which alleviates problems concerning distribution, location, heterogeneity, and interfacing. In our method, software components and other resources like documents may be encapsulated as distributed objects on the basis of CORBA. These encapsulated distributed objects provide the necessary services covering basic functions of an application problem. However, there is a conceptual gap between the low-level CORBA-based resource management and the higher-level process model. As shown in Figures 6 and 7, a fundamental task of the proposed framework is to build the bridge that fills this gap. With the framework, the connection and accordance between the conceptual level and technical level are properly reflected to enable us to configure real processes directly by manipulating the model and then to interpret the executable model on the basis of CORBA.

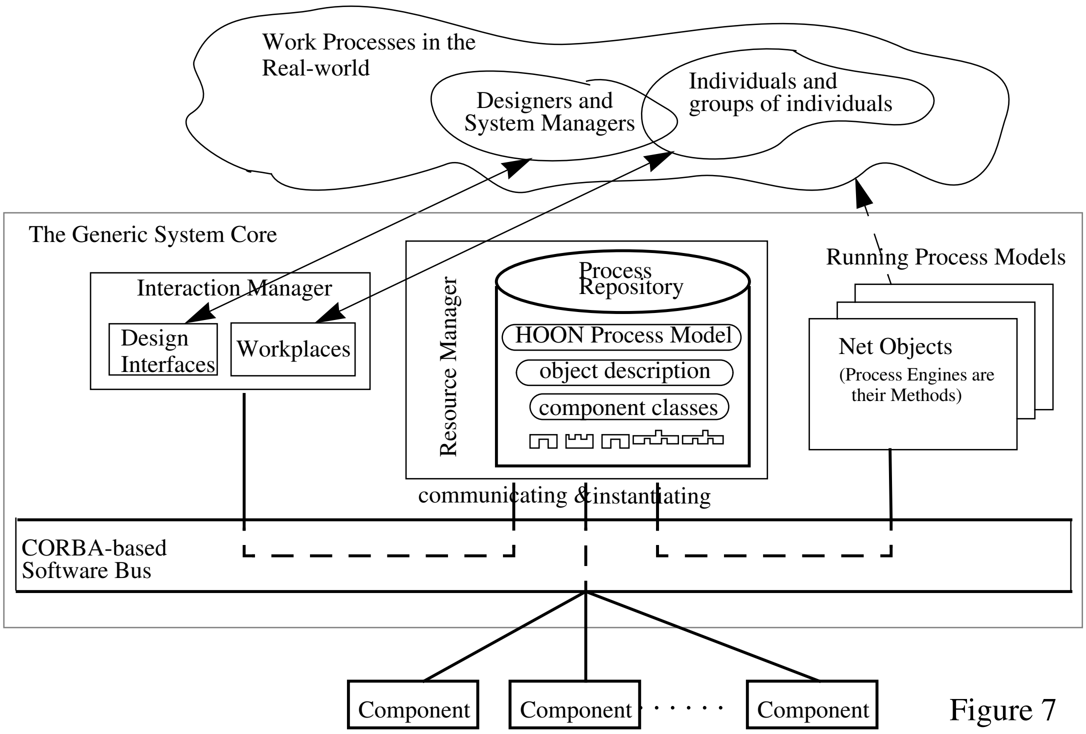


Figure 7

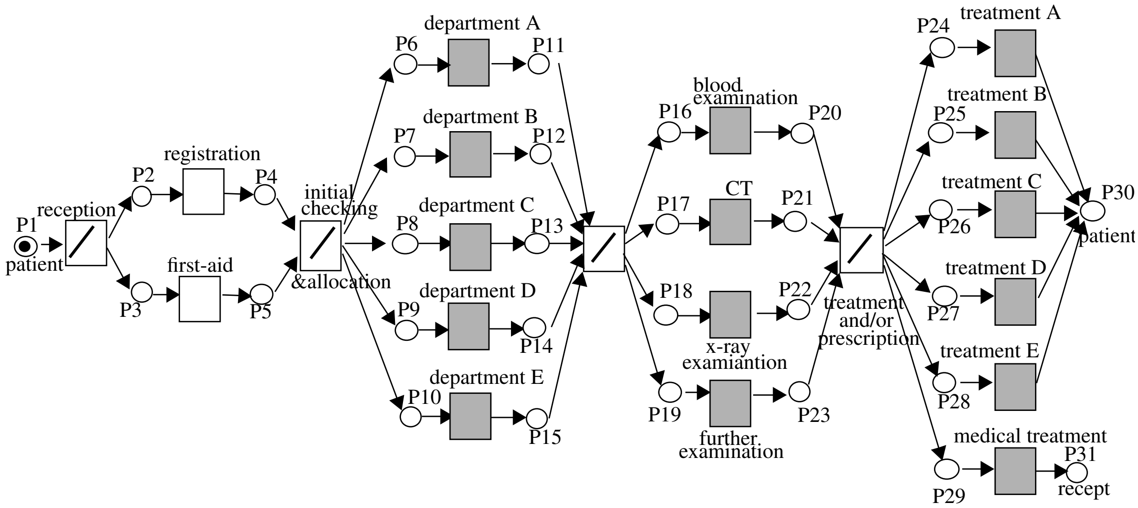
Unlike traditional CASE tools, the framework is always a constituent part of the workflow systems to be built. The framework consists of two basic components: interaction manager and resource manager. The interaction manager supports different kinds of user interactions with the workflow software. The user interactions serve for various purposes, including workplace support for normal users involved in the workflows, modifying workflow resources statically or dynamically, and modifying running models. The resource manager is based on CORBA and the HOON process repository (see the corresponding boxes in Figure 7). CORBA and the repository provide the integration basis between models and software systems. All implicit communication is done through the repository in which various meta-information is stored. Here, the meta-information describes both static information (like relationships, structures and interfaces of distributed objects and structures of net templates) and dynamic information (like tokens and their dynamic distribution).

For controlling real processes with the aid of computers, the process model has to be interpreted by computer software. The framework component which realizes this task is the process model interpreter. An interpreter can be instantiated many times and each instance is called a process engine. In the proposed approach, the process engines are treated as "methods" of net objects which can run concurrently in a distributed environment. In fact, as shown in Figure 7, the major elements of the framework are also encapsulated as distributed objects and communicate through the CORBA object request broker (drawn as software bus in the diagram).

4 Modeling Example

In this subsection, we model the clinical example introduced at the beginning with Petri nets and Higher-Order Object Nets comparatively. Figure 8 shows the concise Petri net model in which transition refinements and selective output structures are represented with single compact symbols. The simplified virtual clinical treatment process is described as follows: firstly, a patient is registered, and a patient record is retrieved if the patient is a former client or created if not. If the patient had an accident, first-aid may be performed during the registration. Then, according to an initial examination, the patient is allocated to one of the departments responsible for special diseases. Thereafter, a disease specialist may instantiate different laboratory procedures for x-ray, CT, or blood checking, etc. Finally, the specialist prescribes medication and/or certain technical treatments such as radiotherapy or massage. The transitions with a bar are a compact representation which models selective output, i.e. the postplaces can be marked individually or in certain combinations. The dark transitions correspond to subnets abstracted as single elements. The subnets will not be shown here to save space.

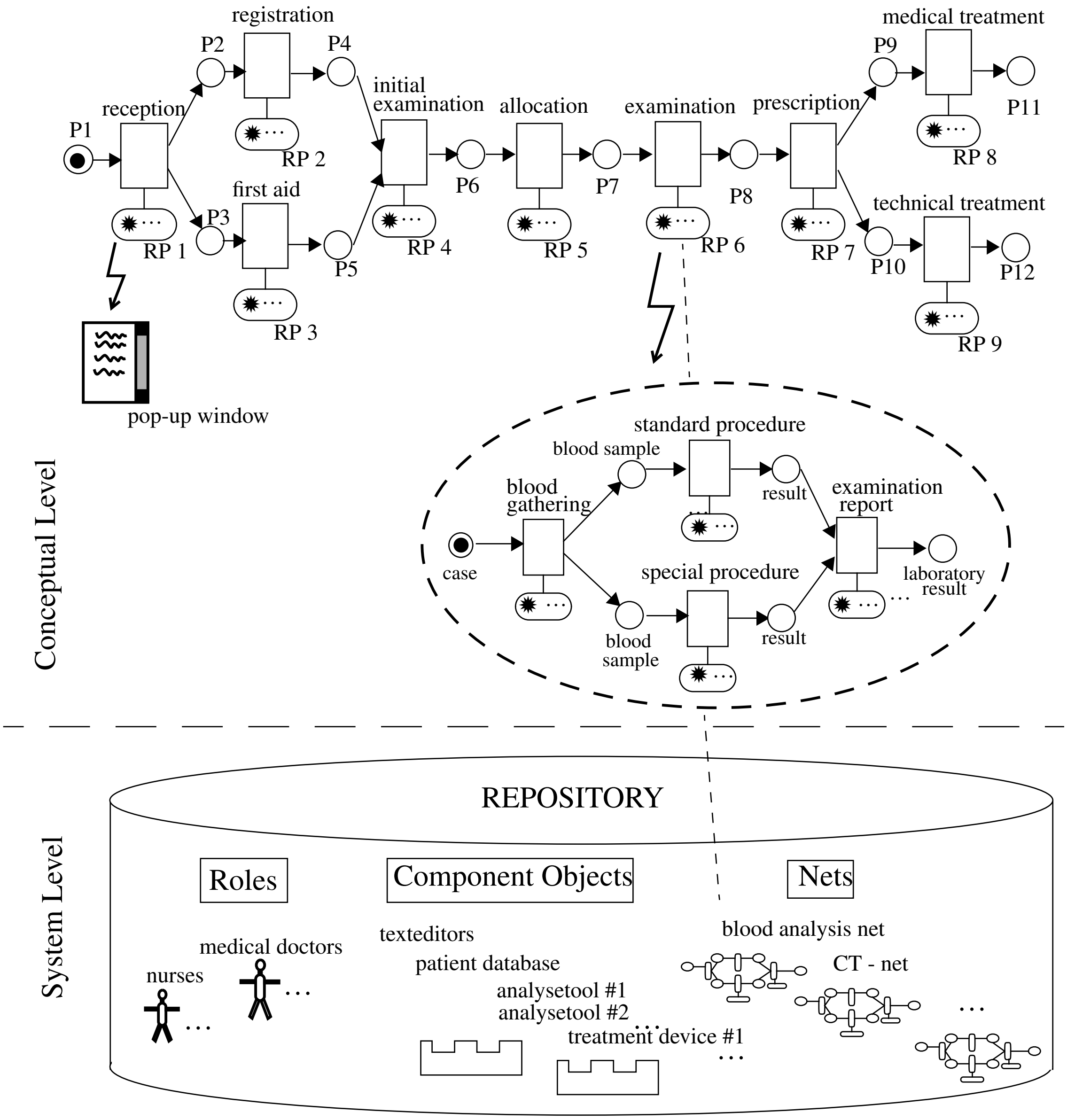
While the ordinary Petri nets are suitable to describe the fixed part of a business process, they are incapable of modeling unpredictable events which can only be detected at runtime. As active workflow resources are usually bound to transitions fixedly, modifications to the model structure are often unavoidable even when only a minor modification to a workflow resource is necessary. Even worse, the unavoidable structural modification is often required during the execution of the process models. For example, at runtime, a technical device may break down or a radiotherapist on duty may become ill. Consequently, the corresponding subprocesses have to be dynamically substituted by other predefined variants or reconstructed.



1) In this example, the transitions with a bar are compact representations of selective outputs, i.e. the output places may have "and","or " or " xor" relationships. They can all be replaced with a refinement in which only the canonical Petri net symbols are used.
2) The dark transitions represent transition refinements which correspond to fixed subprocesses.

Figure 8

Figure 9 is the HOON model describing the same scenario. In our approach, as indicated in the diagram, process modeling and system support to the modeling are not treated separately and both the structured part and the ad-hoc part are taken into consi-



Place Types:

P1 patient case
P2 patient data
P3 patient
P4 patient record
P5 patient
P6 <patient, patient record>
P7 <patient, patienr record>
P8 <patient, patient record, laboratory results>
P9 same as P8 plus advises for medical treatment
P10 same as P8 plus advises for technical treatment
P11 same as P9 plus advises for further medical treatments
P12 same as P10 plus advises for further techn. treatments

Contents of Resource Places (incomplete):

RP1 Role = nurse;
RP2 Role = nurse; Tools =Texteditor, Patient Database
RP3 Role = Doc; Tools = First-Aid-Treatment-Devices
RP4 Role = nurse, Doc; Tools = Texteditor, Patient Database, Initial-Examination-Devices
RP5 Role = nurse; Tools: Patient Database, Disease-Expert-System
RP6 Net = Blood Examination
RP7 Role = Doc; Tools = Texteditor, Patient Database
RP8 Role = Doc: Tools = Medicine Database
RP9 Role = Doc; Tools = Technical Treatment Devices

Legend: Doc = Disease Specialists; Tools = Component Objects

Figure 9

deration at the conceptual level. According to our approach, the tool support at modeling is considered to be indispensable and resources appearing in models must have concrete association to the resources at the system level to ensure consistency. As indicated by the lower part of the diagram, reusable models and other resources are systematically described, stored and managed. With the support of suitable tools, resources can be assigned to resource places of a HOON model statically or dynamically.

As shown in the diagram, each transition is equipped with a resource place on which all accessible (classes of) procedures or other resources are modeled as resource tokens indicated by the black stars. Depending on the specification defined by the script associated with the transition, various scripts (one of which is usually default) can be assigned to the transition. One of the scripts will be called each time the transition fires. If a required resource is not available, the built-in exception handling procedure will be called. Default automatic activities can be defined in the script (for example, to select the next available resource or to activate a predefined procedure if the binding of the default one fails). But normally, the person, if any, has the responsibility for handling exceptions. S/he can select other accessible resources manually or apply for new resources with the technological support provided by the workflow framework. The person can even work out a new subprocess model and then assign instances of the new model as resource tokens. The small window shown in the diagram is intended to show this kind of dynamic interaction between a workflow system and its users who play the role of designers or system managers. By double-clicking on a resource place, a tool can be called with which the assigned resources can be browsed and edited, and new resources can be assembled and assigned. As a special kind of classes, a role and several members (with one of them as default) of this role class can be associated with a transition. If the default person is not available, others can be deployed.

5 Concluding Remarks

Modeling formalisms and infrastructure support are most essential to workflow technology. A novel process modeling formalism - HOON - that supports the integration of business process engineering and software engineering has been introduced. Though Petri-net-based, the graphical notions in HOON alone no longer give a complete semantics to the whole model. Structured texts are used as a complement because we believe that in order to make Petri-net-based models more practicable, a proper trade-off between graphical intuition and structural compactness has to be made. Most importantly, if the textual part covers the volatile aspects, the graphical part can be left untouched in the course of model modifications. Although Petri-net-based models are a nice formalism for process modeling, we should not restrict ourselves to using them as a panacea. We believe that our experimental research could eventually lead to a Petri-net-based formalism which is open to other modeling formalisms and modern software technologies.

Concerning the infrastructure support, our current work is focused on the implementation of the presented workflow framework. In the next step, attention will be paid to the mechanisms supporting dynamic model construction, better security and exception handling, as well as suitable user interfaces both for the application modeling in a convergent way as discussed and for the normal workplaces.

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, pp. 1-19

[BaFu93] Bandinelli S.C., Fuggetta A., Ghezzi C., Software Process Model Evolution in the SPADE Environment, IEEE Trans. on Software Engineering, Vol.19, No.12, December 1993

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

[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

[FaWi96] Faustmann, G., Wikarski, D., Exception Handling in Petri-Net-based Workflow Management, to appear in: Proc. Workshop on Adaptive Workflow, October 1996, Basel, Switzerland

[Godb83] Godbersen H. P., Funktionsnetze, Ladewig-Verlag, 1983

[Gruh91] Gruhn V., Validation and Verification of Software Process Models, Ph.D. Thesis, University of Dortmund, 1991

[HeVe91] Hee K.M.V. and Verkoulen P.A.C., Integration of a Data Model and High-Level Petri Nets, Proc. of 12th International Conference on Application and Theory of Petri Nets, 1991

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

[LöHa94] Löwe M. and Han Y., Process Modeling and Control with Higher-order Nets, Tech. Report 94-34, Technical University of Berlin, 1994

[LöWi95] Löwe M., Wikarski D. and Han Y., Higher-Order Object Nets and Their Application to Workflow Modeling, Tech. Report 95-34, Technical University of Berlin, 1995

[MiGr93] de Michelis G.D. and Grasso M., Routines and Conversations Structured Programming, (93)14, pp. 110-118

[Mura89] Murata T., Petri Nets: Properties, Analysis and Applications Proc. of IEEE 77(1989), pp. 541-580

[OMG91] Object Management Group Common Object Request Broker: Architecture and Specification, Document No. 91.12.1, OMG 1991

[PeSc92] Peuschel B. and Schäfer W., Concepts and Implementation of a Rule-based Process Engine, Proc. of 14th International Conference on Software Engineering, 1992

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

[ScRe93] Scheschonk G. and Reisig W., Petri-Netze im Einsatz für Entwurf und Entwicklung von Informationssystemen, Springer-Verlag, 1993

[Sibe93] Sibertin-Blanc C., Cooperative Nets, Application and Theory of Petri Nets 1993, LNCS691, Springer-Verlag, pp. 377-396

[Tayl95] Taylor D. A., Business Engineering with Object Technology, Wiley, 1995

[Wika90] Wikarski D., Object Nets - A Canonical Class of Models for Behavior Simulation and Structure Synthesis of distributed Systems?, Informatik - Informationen, Reporte 6, 12, Berlin, 1990, pp. 91-100

[Wika96] Wikarski D., An Introduction to Modular Process Nets, Technical Report 96-019, International Computer Science Institute, Berkeley, CA, April 1996

[Wino89] Winograd T., A Language/Action Perspective on the Design of Cooperative Work, Human Computer Interaction, (89)3-1, pp. 3-30.