

Extending UML with Workflow Modeling Capabilities

Guido Wirtz¹, Mathias Weske², and Holger Giese¹

¹ Institute for Computer Science, Westfälische Wilhelms-Universität
Einsteinstrasse 62, 48149 Münster, GERMANY
{guidow,giese,h}@cs.uni-muenster.de

² Department of Information Systems, Westfälische Wilhelms-Universität
Steinfurter Strasse 109, 48149 Münster, GERMANY
weske@helios.uni-muenster.de

Abstract. Nowadays application systems are modeled in object-oriented design languages like the Unified Modeling Language (UML). Although workflow management has to deal with such environments, typically proprietary modeling languages are used. We try to remedy the use of separate languages by proposing an extension of the UML to allow workflow modeling, thereby providing a seamless integration of workflow functionality in object-oriented application systems.

1 Introduction

Workflow management is an important technology for modeling and controlling the execution of business processes in commercial applications [4, 9]. While the construction of complex software systems nowadays uses object-oriented analysis and design [1], workflow modeling is often performed separately, using different methods, techniques, and tools. This situation is especially unfortunate because automating workflows means implementing the needed processes in the environment at hand where the gap between application object modeling and workflow modeling may lead to inadequate workflow support for object-oriented applications, both with respect to maintainability of the software and the reuse of workflow schemas. Although organizations tend to be complex but strictly structured, the process-oriented view found in most workflow approaches declines to use this fact as a means of structuring models which is close to the application domain and hence is highly useful, especially for managing overall complexity and scalability of workflow specifications. This makes a workflow modeling methodology which is compatible to the used software engineering techniques even more desirable. To overcome this unsatisfying situation, this paper proposes such an integrated approach. As stated similarly in [12], one of the basic assumptions of this work is that the problems arising in workflow modeling are similar to those occurring with the analysis, design and implementation of complex sequential and distributed software systems. Hence a compatible methodology for both worlds is feasible. Our work is based on object-oriented notation. For modeling static aspects, we use the UML [11]. Additionally, so-called Object Coordination

Nets (OCoNs) [5], a visual formalism for describing system dynamics are integrated into the UML context. This leads to a set of notations which are suitable to specify not only application objects, but also workflows.

2 Background and Rationale

Depending on the workflow language and on the workflow management system employed, several dimensions are covered in workflow schemas; these dimensions are also known as workflow aspects [7]: the *functional* aspect specifies what has to be done within a workflow. The *operational* aspect determines how it is done, i.e., which techniques and tools are used to perform the workflow. The *behavioral* aspect defines when and under which conditions a workflow is executed. The *informational* aspect specifies the data objects which are being manipulated during workflow executions and the flow of data between workflow activities. The *organizational* aspect describes the roles and personnel which are involved in workflow executions. There are two conflicting goals regarding these aspects, namely *independence* and *consistency*. At the first glance, the *independence* of different aspects of a workflow schema has a lot of benefits. It makes sure that changes w.r.t. one particular aspect do not involve the others. However, in real-world workflow applications, aspects are normally not that independent from each other, for instance due to interrelationships between the operational aspect (external applications) and the informational aspect (data used by them). Moreover, complete independence of aspects may not be desirable. Whereas the different aspects of workflow applications are helpful to concentrate on specific views separately when modeling workflows, afterwards questions of *overall consistency* of the specification become important: which parts of an organization are involved in a specific workflow, what are the roles of the organizational units, are they capable of or really the best guess for performing their specific subtask or is, for example, the flow of data consistent with the security policy of an organizational structure. Thus, the organizational aspect may give important hints how to specify the functional and behavioural aspects or even place restrictions on the operational aspect in a given context, e.g., if a task is performed in a specific part of an organisation, a specific external application has to be used which differs from those of other divisions due to a non-uniform software (version) environment. On the other hand, during the restructuring of a complex workflow environment, modeling the functional and behavioural aspects of the most important workflow schemas first, provides the detailed specification for *use cases* [11] needed to discuss the overall organizational design. Workflow languages typically allow the specification of workflow schemas in a manner which puts its focus on the fact that the aspects are defined independently. The approach described in this paper employs object-oriented methods to deliver to some extent a combination of both goals: focussing on the aspect currently at hand in the well-defined context of a structured specification of all relevant aspects.

The use of object-oriented methods implies two additional benefits in the workflow context. Traditionally, workflow schemas are described using specific and often proprietary languages. These workflow languages often use formalisms based on directed graphs [16] or Petri Nets [13] in a manner which does not fit very well into the technical environment of modern information system infrastructures. Often, these systems are (and definitely will be in the future) developed using object-oriented methods. For the modeling of such systems, object-oriented design languages and recently the quasi-standard of the Unified Modeling Language [11] are widely used. The usage of these notations when specifying workflow applications helps to close the gap between both worlds and eases the incorporation or adaption of off-the-shelf software as well as the implementation of software. Describing, e.g., the informational aspect of a workflow in the same language as the data interfaces of the applications permits the re-use of the data format descriptions or at least provides a clear basis for the adaption of formats. If no legacy code is available, the description of a specific task of a workflow schema using UML diagrams provides an ideal starting point for implementing the needed piece of software with state-of-the-art methods. The second benefit comes for free and is due to the capabilities of OO methods for managing complexity. Workflow systems are as complex as application software systems and their development can make use of OOA and OOD to deal with the complexity. Here, using structure modeling to capture the organizational aspect [7] of a workflow system like, e.g., geographically distributed branches of an organization with distinguished roles and permissions to perform (parts of) the work to be done, can provide a basis for keeping an eye on the consistency w.r.t. all other aspects from the very beginning. If all aspects of a workflow application are specified in an integrated design language, independence of aspects gets a new chance by explicitly specifying dependencies and assumed independence may even be checked in the model. More important, a different dimension of independence is introduced which is of much more use: the means of structuring the design of a workflow system using OO methodology into subsystems with interfaces and specifying dependencies between subsystems explicitly deals much better with managing overall complexity and change management for workflow applications in-the-large than the abovementioned independence of aspects which is provided to some extent by different diagrams when specifying workflow in-the-small.

In order to be close to the standard for OO applications, workflow specifications should also be described with the diagrams provided by the UML wherever suitable. Although the UML subsumes a variety of techniques to model different aspects of software systems, there remain some problems. First, there are simply too many kinds of diagrams to be used in a specific application. Overlapping aspects of behaviour may be described by annotating class diagrams in so-called collaboration diagrams, sequence diagrams, activity diagrams or statechart diagrams. Second, there is no clear semantics for most of these diagrams regarding the internal correspondence to other diagrams used in the same model. This is a serious flaw in the UML because the really essential aspects of system dynamics which are important especially for the functional and behavioural aspects of a

workflow system are not that well-covered. Using such notations would make the intended notation of consistency hard to implement (for a detailed discussion see [5]). In the context of workflow modeling, a pragmatic approach helps to overcome the problems with the UML. In order to introduce as few new notation as possible, only those notations of the UML which are used to describe static structures like systems, subsystems, classes, associations and so on are utilized. So-called Object Coordination Nets (OCoNs) [17] provide a high-level Petri-Net [2] formalism for specifying system dynamics. To avoid the problems with consistency, OCoNs are seamlessly integrated into the UML and have a formal semantics. In contrast to most high-level net approaches working with complex textual and formal annotations which are hard to use for the non-expert in formal methods, the net dialect puts its focus on the visual expressiveness. Principal different aspects of a workflow system are expressed in different diagrams whereas related aspects are described in the same diagram but represented by distinguishable graphical entities.

3 The Object Coordination Net Approach

We assume the reader to be familiar with place-transition nets [2] and the basic concepts of object orientation. Transitions in our context are interpreted to be *actions* which occur through calls to services provided by objects. For places which are called *pools*, we distinguish between two principle entities: simple data and control flow using untyped tokens and objects typed according to the type system obtained by the informational aspects of workflows are stored in so-called *event pools* (circles). More permanent objects which are in the object-oriented view the carrier of activity or provider of services are interpreted as *resources* the system uses to perform its work. Places which hold resources are represented by *resource pools* (hexagons).

In order to support a method for workflow modeling as described in the previous section, structuring a system into encapsulated subsystems and a well-defined and expressive formalism to describe interfaces and interface-based interaction are crucial. The OCoN approach introduced in [17] fulfills these requirements by putting its focus on a system design which is ruled by the *contract principle* [10]. Organisational aspects are used to obtain a coarse-grained structure of the context in which workflows are to be modeled. Using UML structure diagrams this results in a set of (nested) subsystems providing services specified in contracts to the outside and possibly using services from other subsystems via their contracts to implement the provided functionality (see Fig. 5). A single contract is a traditional signature specifying the names and required parameters of all provided services which is extended by a state-machine-like *protocol net* (PN) to specify externally visible behaviour if not all services are always available. This kind of specification enhances the benefits of interfaces in case of services which obtain nonuniform service availability as it is often found in systems acting according to complex organizational rules. The example shown in Fig. 1 specifies a simple contract to deal with an object of type Media like,

e.g., a book or CD using the services `getout` and `putback`. The PN adds important usage information to the signature. Only if the required entity is in state `[Free]`, is the service `getout` available. Moreover, the contract requires that a `getout` which brings the object to state `[InUse]` has to be followed definitely by a `putback`. Hence, the contract specifies restrictions for its users. Finally, a hint that the object may not be available for some (finite) time due to internal regulations of the implementation is specified by introducing a third state `[Check]` and an anonymous step (shaded transition). However, the protocol guarantees that eventually the object will be in state `[Free]` again. Using states as pre- and post-conditions for service availability supports the abstract description of the behavioural workflow aspect.

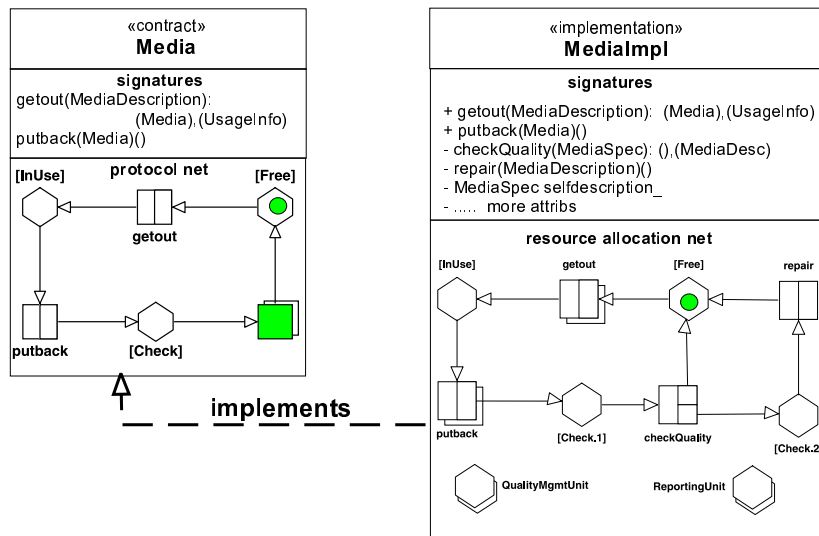


Fig. 1. Media contract and its implementation **MediaImpl**

For decoupling subsystems properly, we distinguish clearly between the external information provided by a contract and its internal implementation. Moreover, we rigorously permit the usage of services by means of provided contracts only. The internals of a subsystem itself may have a complex structure as well as complicated rules of handling control and data. This leads in practice to a hierarchical design of such systems. Besides the internal subsystem structure, there are two aspects of special importance inside a single system: the internal details of the *implementation* of the provided services and the overall *management of resources* to accomplish that goal. The concepts needed to model these aspects include two additional variants of Petri-nets, namely *service nets* (SN) for describing the detailed workflow when performing a single service and *resource allocation nets* (RAN) for the resource management.

The interaction mechanism of the nets is visualized in Fig. 2: calling a provided service as described in the corresponding PN activates the RAN of the instance which provides the call with the needed resources and delegates it to

the requested service (SN). If the service uses external services from other contracts, parts of the work are delegated to instances of the used type. For the Media example, the implementation `MediaImpl` in Fig. 1 (right) defines the local attributes needed for the implementation, publishes the services to implement `getout` and `putback`, but uses additional private services, e.g., `checkQuality` to control incoming Media whether they are ok to be used again or not. Moreover, the RAN implements the protocol provided by the PN. It states that two external resources imported from other subsystems (shaded hexagons), namely `QualityMgmtUnit` and `ReportingUnit`, are needed for the implementation of the contract. The initial state of a new instance of `MediaImpl` is defined to be `[Free]` by putting a token in the corresponding resource place. Compared to `Media`, the possible states of `MediaImpl` are refined. Where `[Free]` and `[InUse]` are used to represent the corresponding states of the PN, `[Check]` is partitioned into `[Check.1]` and `[Check.2]`. This reflects the fact that the work which has to be done inside the implementation is splitted into two parts `checkQuality` and `repair` in the case of incoming objects which are in bad shape. However, the only thing w.r.t. these details which is of any interest for the contract, is the existence of internal work which prevents `Media` in state `[InUse]` from becoming `[Free]` immediately after a `putback`. `MediaImpl` fulfills the protocol because either `checkQuality` decides that everything is ok and the object is put back in state `[Free]` or the object is put into state `[Check.2]` and has to undergo a `repair` after which it is put back in state `[Free]`, too. Because this part of the behaviour depends on the object at hand, the RAN uses a transition `checkQuality` with two possible alternative outcomes.

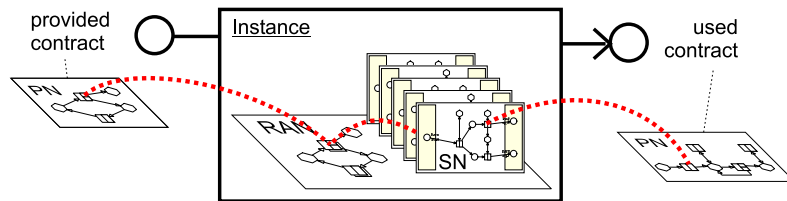


Fig. 2. OCoN architecture and usage of Net types

The usage of shaded transitions indicates that the net (PN or RAN) at hand does not have complete control over the operation or how it is called. As well as the implementation details are out of control of the anonymous transition in the PN, are the number and time of calls to `getout` and `putback` from the outside not under control of the implementation.

The detailed subworkflows implementing a service are specified in a *service net* (SN). It adopts the object-oriented view of method invocation by supporting an intuitive call semantics which is well-known from procedural programming languages and remote procedure calls. Fig. 3 visualizes the concept: a transition is interpreted to represent a call to a method-like service. As a precondition for firing, a *carrier of activity* (here `obj1` of type `Resource1`), i.e., the object which processes the call, is required as well as the parameters (here an object of type `InType`) and a simple Event. Moreover, the carrier has to be not only of type

Resource1 but especially a resource in state [State1]. If the call to fun fires, it consumes all preconditions in the first step, performs some durable activity inside the service fun and produces all postconditions afterwards in a second step. The effect is an object of type ResultType in place resu and a state change for the carrier which is afterwards in state [State2].

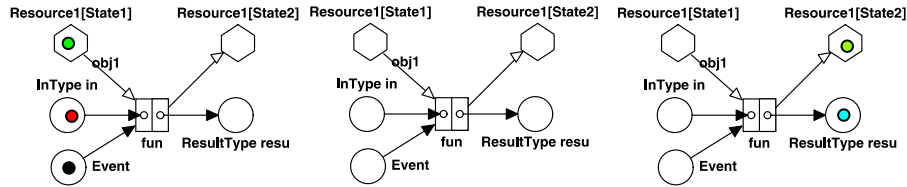


Fig. 3. Steps during execution of $resu = obj1.fun(in)$

The hierarchical design of the service nets allows for calls to other services during firing inside a transition. Due to the synchronous nature of call and return, this works across hierarchies and can also be used as a wrapper call to services implemented by legacy code. The concept is explained by means of the internal `MediaImpl` service `checkQuality` (see Fig. 4). In order to support hierarchical abstraction but keep the essentials also on the abstract call level, calling transitions visualize the signature of the service they call (see Fig. 4, left). The transition here requires a single data parameter and produces an alternative output to abstract from an internal decision.

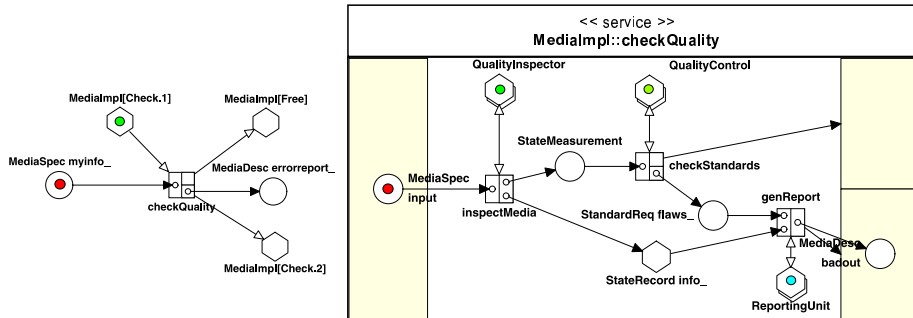


Fig. 4. Call semantics of OCoNs

In conformance with the RAN in Fig. 1, the operation can only be carried out by instances of the resource `MediaImpl` in state [Check.1]. Because this is specified in the RAN, it is not explicitly stated in the textual signature. The signature requires an additional parameter of type `MediaSpec` which is represented by the precondition pool `myInfo` in the calling net. The result `MediaDesc` which is expected only in one of the possible decision outcomes is handled in a similar manner. Because both preconditions are fulfilled, the action is enabled to fire. The unfolded version of the calling transition (see Fig. 4, right) uses two bars to represent the input and output parameters. The figure shows the initial situation after consuming the preconditions. Additional resources are present from the

no other interaction between subsystems is allowed. This means, the structure provides a coarse-grained description of the organizational aspect of all possible workflows which may be performed in this system. More important, the different contracts can be interpreted as a description of the different roles a subsystem may play during workflow execution. It specifies all subworkflow schemas supported by a specific subsystem which are designed for external use and hence supports encapsulation and re-use.

The description techniques for the static structure are also used to specify the details of the complex data objects which represent entities like users, media, and so on. Such type descriptions using attributes, methods, class diagrams etc. provide the second part of the basis for all remaining aspects because these types are used to type event and resource pools as well as the objects flowing through edges and being consumed as matching typed actual parameters in actions. The specification of this part of the informational aspect should be done in a manner which is consistent with the usage of subsystems, i.e., defines classes as local as possible and utilizes the visibility rules to avoid too much global information in the system.

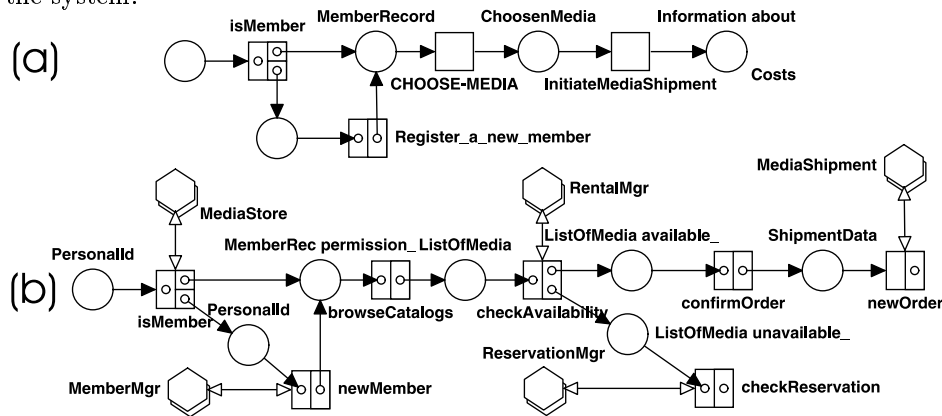


Fig. 6. Customer wants to rent some media

The remaining workflow aspects become more relevant when discussing a specific workflow schema. This is done now by considering the situation that a customer enters the `MediaStore` and wants to rent some CDs and a book. First of all, it should be checked whether the person is a registered member; if so, the member is able to browse catalogs, look into books and so on in order to choose a list of media which he wants to rent. The sketch shown in Fig. 6 (a) is rather incomplete and describes the needed functionality in an informal way as well as the accompanied flow of control and some data aspects. The next step in Fig. 6 (b) makes things more concrete. The informational aspect can be made more clear when using the types derived from class diagrams to restrict the type of items flowing through the net. More important is the decision which parts of the system are used to perform the different steps. This is done by explicitly stating which resources are needed to execute actions, i.e., who are the carriers of activity for actions. The structure of the system (cf. Fig. 5) has

been designed in a manner which provides contracts implemented by subsystems for the different kinds of functionality. Using these resources when specifying a workflow schema introduces information about the operational aspect and combines it with behavioural aspects. Additionally, the schema now deals with a more complex situation: some of the wanted media may be not available because they are in use by other customers and the member may require to be notified when they become available. Because this is a special kind of service, it is not a good idea to implement that in the RentalMgr. Instead, it is done in parallel by an additional subsystem, namely the ReservationMgr, implementing a fair reservation policy.

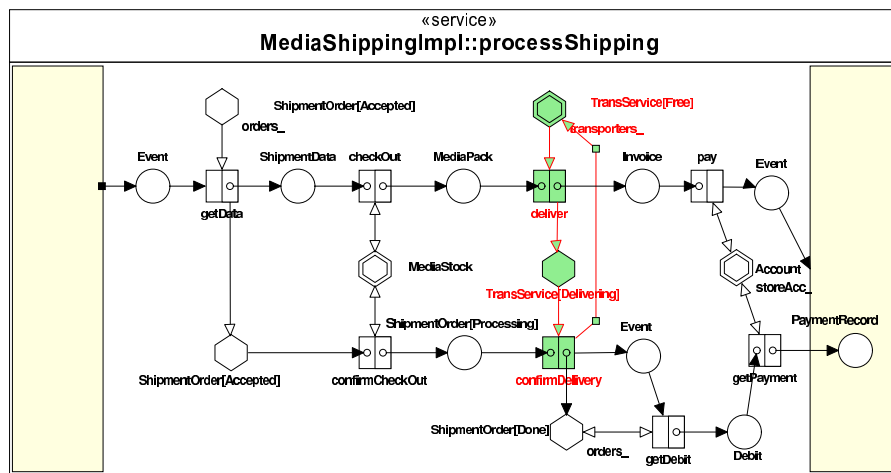


Fig. 7. Service ProcessShipping uses external contract TransService

In order to keep the example manageable, we only go into some of the details of the shipment of rented media. The MediaShipment subsystem in Fig. 5 (right) offers this functionality by means of a simple contract using the same name. Although the subsystem needs an external subworkflow, this is handled much like using the local resources by importing the contract TransService which is from a completely different subsystem in contrast to, e.g., the contracts Account or MediaStock which are in the same subsystem as MediaShipment itself. The details of shipment orders and how they are handled are implemented locally in the ShipmentOrderImpl. The partial workflow using all these entities is encapsulated within the service processShipping which performs the real work. Because all details of the subsystem are hidden, the contract may well be re-used in a completely different context. The service itself is specified using the service net shown in Fig. 7. Only confirmed ShipmentOrders in state [Accepted] are processed, the MediaStock is used to check out the ordered media which are delivered using the TransService. Besides the real delivering, the state of this job (represented by a ShipmentOrder) is observed in parallel in the subsystem. The contract Account is used to pay for the transport as well as to compute the shipping costs. These are represented by a PaymentRecord (which is assumed to be specified as

a class in the informational aspect) and built the final output of this service. The only thing of specific interest here is how the external subworkflow from the contract `TransService` is used for handling the transport of media itself (see highlighted area in Fig. 7). The contract is used as a shared resource, because other systems may also use this contract. However, for performing the action, only a `[Free]` resource can be used, and it has to be used exclusively, i.e. not in parallel with other usages. This is because the action changes the state of the resource to `[Delivering]` until the `confirmDelivery` has been executed, too. Afterwards, the resource state is changed again into `[Free]`. All this is easily done in complete conformance with the contract `TransService` because its protocol net (see Fig. 5, right) is visually embedded into the using service net which makes the consistent usage of subworkflows even from different system parts an easy manageable task.

5 Evaluation and Conclusions

UML diagrams are widely accepted nowadays and their use for the workflow aspects they are suited for eases the communication among all people involved in workflow modeling, implementation, and monitoring. Moreover, the gap in languages and notations between workflow and application development is closed to some extent. The presented example should illustrate that the Petri-Net formalism used permits the gradual change from informal descriptions to correct nets including typing as a means of integrating workflow aspects consistently. In the long term, there are strategic benefits w.r.t. the availability of important management information: having specified a complex workflow system in the proposed way, almost all important information about the organizational structure, responsibilities, dependencies between subsystems and workflow schemas as well as the resource usage and requirements present in the system are available in a structured set of documents in a nearly formal language. Hence, there is a proper basis for the overall analysis, consistency checking, and optimization of structures and processes. Moreover, possible effects of changes can be found in explicit specifications which provides information crucial for local change management as well as more advanced situations like, e.g., finding (subsystem) candidates for outsourcing.

Although there are many approaches dealing with a diversity of graphical descriptions, the usage of Petri-Net variants in workflow management is widespread, e.g. FUNSOFT Nets [3], HOON [6] and especially the work of van der Aalst, e.g., [13] and colleagues. However, most of the work and even recent approaches dealing with interorganizational workflows (e.g., [14]) put their focus on the process aspect and fail to utilize structure to overcome scalability and encapsulation problems. The approach described in [15] tries to combine all aspects, but its puristic use of mechanisms based solely on nets and their interaction makes the resulting models too complex for real-life applications. Moreover, the underlying so-called *reference nets* [8] use the metaphor of nets flowing through nets which seems to be too complicated for a non-expert in high-

level Petri-Nets. In the Mentor project [18], state-charts and activity-charts are used to model workflows. The main focus of that project is distributed workflow execution control based on persistent message queues. Whereas these description techniques are close to some of the UML notations, workflow modeling is not embedded in an object-oriented design in that project.

The method sketched in this paper has been used for two years now by students in classes modeling distributed software systems but recently also to describe workflow systems. At the moment, case studies of business processes and workflows in companies are performed in order to evaluate the approach in real-life business settings.

References

1. Grady Booch. *Object-Oriented Analysis and Design with Applications*. Addison-Wesley, Menlo Park CA, 1993. (Second Edition).
2. W. Brauer, W. Reisig, and G. Rozenberg [eds]. *Petri Nets: Central Models (part I)/Applications (part II)*, volume 254/255 of *LNCS*. Springer Verlag, Berlin, 1987.
3. W. Deiters and V. Gruhn. The FUNSOFT Net Approach to Software Process Management. *Int. Journal on SWE and Knowledge Engineering*, 2(4), 1994.
4. D. Georgakopoulos, M. Hornick, and A. Sheth. An Overview of Workflow Management. *Distributed and Parallel Databases*, 3:119–153, 1995.
5. Holger Giese, Jörg Graf, and Guido Wirtz. Closing the Gap Between Object-Oriented Modeling of Structure and Behavior. In Robert France and Bernhard Rumpe, editors, *UML'99*, volume 1723 of *LNCS*, pages 534–549, October 1999.
6. Y. Han and H. Weber. Adaptive Workflow and Software Architecture Thereof. *Journal of Integrated Design and Process Science*, 2(2):1–21, 1998.
7. S. Jablonski and C. Bussler. *Workflow Management: Modeling Concepts, Architecture and Implementation*. Int. Thomson Computer Press, 1996.
8. O. Kummer and F. Wieberg. Renew homepage. Dept. of CS, University of Hamburg, Germany, URL: <http://www.renew.de>, 1999.
9. F. Leymann and D. Roller. *Production Workflow: Concepts and Techniques*. Prentice Hall, New Jersey, 2000.
10. Bertrand Meyer. *Object-Oriented Software Construction*. Prentice Hall, 1997. 2nd.
11. Object Management Group. *OMG UML 1.3*, June 1999. OMG doc ad/99-06-08.
12. S.K. Shrivastava. Workflow Management Systems. *IEEE Concurrency*, 7(3), 1999.
13. W.M.P. van der Aalst. The Application of Petri Nets to Workflow Management. *Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.
14. W.M.P. van der Aalst. Interorganizational Workflows: An Approach based on MSCs & Petri Nets. *System Analysis-Modelling-Simulation*, 34(3):335–367, 1999.
15. W.M.P. van der Aalst, D. Moldt, R. Valk, and F. Wienberg. Enacting Interorganizational Workflows Using Nets in Nets. In *Proc. of the 1999 Workflow Management Conference*, pages 117–136. University of Muenster, Muenster, Germany, 1999.
16. M. Weske and G. Vossen. *Workflow Languages*, pages 359–379. International Handbooks on Information Systems. Springer, Berlin, 1998.
17. Guido Wirtz, Jörg Graf, and Holger Giese. Ruling the Behavior of Distributed Software Components. In H. R. Arabnia, editor, *PDPTA*, July 1997.
18. D. Wodtke, J. Weissenfels, G. Weikum, and K. Dittrich. The Mentor Project: Steps towards Enterprise-Wide Workflow Management. In *Proceedings of the 12th International Conference on Data Engineering*, pages 556–565. IEEE CS, 1996.