

The OCoN Approach to Workflow Modeling in Object-Oriented Systems

G. Wirtz (guidow@math.uni-muenster.de)

*Westfälische Wilhelms-Universität Münster
Arbeitsgruppe Verteilte Systeme, Institut für Informatik
Einsteinstrasse 62, D-48149 Münster, GERMANY*

M. Weske (mathias.weske@acm.org)

*Eindhoven Technical University
Department of Information and Technology
PO Box 513, NL-5600 MB Eindhoven, THE NETHERLANDS*

H. Giese (hg@upb.de)

*Universität-Gesamthochschule-Paderborn
Arbeitsgruppe Software Engineering, Fachbereich 17 - Praktische Informatik,
Warburger Str. 100, D-33098 Paderborn, GERMANY*

Abstract. Workflow management aims at modeling and executing application processes in complex technical and organizational environments. Modern information systems are often based on object-oriented design techniques, for instance, the Unified Modeling Language (UML). These systems consist of application objects which collaborate to achieve a common goal. Although application objects collaborate in the context of business processes that can be supported by workflow technology, workflow modeling is typically done with proprietary workflow languages. Hence, two separate formalisms are present for modeling application objects and workflows. In this paper we try to remedy this situation by proposing the use of Object Coordination Nets (OCoN) for workflow modeling. OCoN nets provides a seamless integration with UML structure diagrams. The OCoN formalism also helps to deal with all relevant aspects of modeling complex workflow systems in a scalable and consistent manner.

Keywords: workflow modeling, workflow design strategies, object-orientation, Petri nets

1. Introduction

Workflow management is an important technology for modeling and controlling the execution of business processes in commercial applications (Leymann and Altenhuber, 1994, Georgakopoulos et al., 1995, Leymann and Roller, 2000). Workflow management systems provide the functionality to model the automated parts of business processes and the organizational and technical environment in which they are expected to be executed. While the construction of complex software systems nowadays uses object-oriented analysis and design (OOA and OOD) (Booch, 1993, Rumbaugh et al., 1991, Jacobson et al., 1992) and (Coleman et al., 1994) accompanied with suitable notations, workflow modeling is often performed separately from application object modeling, using different methods, techniques, and tools. This situation



© 2001 Kluwer Academic Publishers. Printed in the Netherlands.

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 does not use this fact as a means of structuring models that is close to the application domain and, hence, is highly useful. This makes a workflow modeling methodology which is compatible to software engineering techniques even more desirable.

To overcome this unsatisfactory situation, this paper proposes an integrated approach. As stated similarly in (Shrivastava, 1999), one of the basic assumptions of this work is that the problems arising in workflow modeling are not that far from those occurring with the analysis, design and implementation of complex sequential and distributed software systems. Hence a compatible methodology for both worlds is feasible. In quest of which languages to use, pragmatics require a well-accepted object-oriented notation. For the modeling of static aspects, this can be found in the UML (OMG, 1999). Additionally, so-called Object Coordination Nets (OCoNs) (Giese et al., 1998), a visual formalism for describing system dynamics are integrated into the UML context. This leads to a set of notations that are suitable for specifying not only application objects, but also workflows.

This paper is an extended version of (Wirtz et al., 2000). It is organized as follows: In Section 2 the rationale of using object modeling techniques in workflow specification and Petri nets in combination with the UML is discussed. The new approach using UML is presented in Section 3. A case study of a simple workflow and its context is used in Section 4 to illustrate the method. The evaluation of the approach and a discussion of related work in Section 5 and concluding remarks complete the paper.

2. Background and Rationale

In recent years, process orientation has become an important concept in business applications, since many companies provide services and products via complex business processes. With the explicit modeling and improvement of these processes, competitive advantages can be realized for particular companies. Workflow management aims at the technical side of business processes and their enactment. In particular, workflow management deals with modeling and controlling the execu-

tion of application processes in complex organizational and technical environments. To this end, a workflow application is an information systems application whose process structure is modeled explicitly using a workflow management system and which is executed under the control of a workflow management system. A workflow application may use external applications that are invoked by a workflow management system during workflow executions.

While workflow management has been discussed within a given organization, a new trend is interorganizational workflows, i.e., workflows spanning organizational units. Interorganizational workflows are very important in the context of electronic commerce, since modeling and efficiently enacting processes involving multiple companies is a key aspect in transactions involving multiple companies (also known as business-to-business or B2B electronic commerce).

The internal representation of a specific application process is called a workflow instance, and multiple workflow instances can be performed within a given workflow application. The structure of a set of similar workflow instances is specified in a workflow schema. For instance, the set of all order-processing workflow instances of a given organization can be described by a workflow schema “order-processing”. Workflow schemas are expressed in workflow languages (Weske and Vossen, 1998). Depending on the workflow language and on the workflow management system employed, several dimensions are covered in workflow schemas; these dimensions are also widely known as workflow perspectives (Jablonski and Bussler, 1996). While the set of perspectives depends on the particular requirements of the application, the following perspectives are present in almost all workflow applications: The *functional* perspective specifies what has to be done within a workflow. The *operational* perspective determines how it is done, i.e., which programs are used to perform particular workflow activities. The *behavioral* perspective defines when and under which conditions a workflow is executed. Start condition or transition conditions are typical language constructs to specify the behavioral perspective. The *informational* perspective specifies the data objects that are manipulated during workflow executions and the flow of data between workflow activities. Data flow connectors and parameters can be used to specify the information perspective. Finally, the *organizational* perspective describes the roles and personnel that are involved in workflow executions.

External application programs are often used as workflow activities; the workflow management system determines when a given workflow activity has to be started, it provides it with appropriate parameter values. After the application is complete, the result values can be

returned to the workflow management system, which uses the information to continue the workflow, as described in the workflow schema. Besides external applications, often legacy applications, a new form of embedded workflow arises in the context of enterprise resource systems (ERP systems), as marketed by companies such as SAP or PeopleSoft. In these settings, workflow activities are represented by particular functions provided by the ERP system. In the context of SAP R/3, for instance, particular SAP transactions are used to perform workflow activities. The advantage of the workflow approach in the context of ERP systems is that the process logic is made explicit in the process model rather than being hidden in the application. As will be described in some detail below, the approach presented in this paper is based on object-oriented design techniques, as developed in the context of business objects. However, similar developments emerge in ERP systems. In particular, the functionality of these ERP systems can be characterized by business objects, which allows applications to make use of the ERP system's functionality via specified and open interfaces.

2.1. INDEPENDENCE VS. CONSISTENCY

There are two conflicting goals regarding the different perspectives of a workflow schema discussed above, namely *independence* and *consistency*. At first glance, the *independence* of different perspectives has many benefits. It makes sure that when changes to one particular perspective of a workflow schema have to be performed, the other perspectives are not involved. However, in real-world workflow applications, workflow perspectives are normally not that independent of each other, for instance due to interrelationships between the operational perspective (external applications) and the informational perspective (data used by them). Moreover, complete independence of perspectives may not be desirable. Whereas the different perspectives of workflow applications help to concentrate on specific views 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 just the best guess for performing their specific subtask? Is the flow of data consistent with the security policy of the organization? Thus, the organizational perspective may give important hints as to how to specify the functional and behavioral perspectives or it may even place restrictions on the operational perspective in a given context. For example, if a task is performed in a specific part of an organization, a specific application may have to be used that 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 behavioral perspectives of the most important workflow schemas first, provides the detailed specification for *use cases* (OMG, 1999) needed to discuss the overall organizational design.

An important aspect of workflow languages is their modularity with respect to workflow perspectives. This means that the workflow perspectives discussed above are defined in a modular way, i.e., independent from each other. The approach described here employs object-oriented methods used for complex sequential and distributed systems to deliver, to some extent, a combination of both the above mentioned goals: focusing on the perspective currently at hand in a well-defined context of a structured specification of all relevant perspectives.

2.2. BENEFITS OF OBJECT-ORIENTED WORKFLOW MODELING

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 (Weske and Vossen, 1998, Jablonski and Bussler, 1996) or Petri Nets (cf. (van der Aalst, 1998)). Whereas the graphical languages are easy to use and of great benefit, they often respect the control perspective only and hence are hard to integrate into the global picture of an organization running multiple workflows. Especially, the combination of the graphs with the OO-structured technical environment of modern information system infrastructures is not well understood. Often, these systems are (and definitely will be in the future) developed using OOA and OOD methods. For the modeling of such systems of application objects, object-oriented design languages and, recently, the quasi-standard of the Unified Modeling Language (OMG, 1999) are widely used. The use of these notations when specifying workflow applications helps to close the gap between both worlds and eases the incorporation or adaptation of off-the-shelf software as well as the implementation of software. Describing, for example, the informational perspective 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 adaptation of formats via, for example, scripts. If no legacy code is available, the description of a specific task needed to implement a workflow schema using UML diagrams provides the 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 during software development. Workflow systems are as complex as application software systems and their development can make use of OOA and OOD to deal with this problem. Here, using structure modeling to capture the organizational perspective (Jablonski and Bussler, 1996) of a workflow system such as, for example, geographically distributed branches or divisions with distinguished roles and permissions to perform (parts of) the work to be done, can provide a basis for keeping an eye on the overall consistency from the very beginning. This helps prevent major design errors, which tend to become really costly to fix at a later stage. If all perspectives of a workflow application are specified in an integrated design language, independence of perspectives gets a new chance by explicitly specifying dependencies and the 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 model of a workflow system using OO methodology into subsystems with interfaces and specifying dependencies between subsystems explicitly. This deals much better with managing overall complexity and change management for workflow applications in-the-large than the above mentioned independence of perspectives which is provided by different diagrams when specifying workflow in-the-small.

2.3. PRAGMATICS

To be compatible with the standard for OO applications, workflow specifications should also be described with UML diagrams (OMG, 1999) wherever suitable. Although UML subsumes a variety of techniques to model different aspects of software systems, there remain some problems. First, there are simply too many diagrams to be used in a specific application and some of the proposed notations deal with the same aspects in different styles and ways. For example, overlapping aspects of behavior may be described by annotating class diagrams in so-called collaboration diagrams and sequence diagrams that are derived from message sequence charts (ITU, 1996), activity diagrams or statechart diagrams (Harel, 1987). Second, there is no clear semantics for most of these diagrams regarding the internal correspondence to other diagrams used in the same model. Unfortunately, this is a serious flaw in UML because the really essential aspects of system dynamics that are important for the functional and behavioral perspectives of a workflow system are not that well-covered. Using UML notations only for all perspectives is possible, but makes the level of consistency that we envision hard to achieve. The advantages and disadvantages of the

different UML notations for describing behaviour in the context of distributed software engineering have been evaluated elsewhere in great detail (Giese et al., 1999a) using various UML diagrams to describe aspects of the example discussed also in Section 4.

In the context of workflow modeling, a pragmatic approach helps to overcome the problems with UML. In order to introduce as few new notations as possible, only those notations of UML that are used to describe static structures like systems, subsystems, classes, associations, and so on, are utilized. These are close to the diagrams used in *entity relationship* modeling, have been in use in OO approaches for a decade, and, are thus well-understood and widely accepted. So-called Object Coordination Nets (OCoNs) (Giese et al., 1998) provide a high-level Petri nets (Brauer et al., 1987) formalism for specifying system dynamics. To avoid the problems with consistency, OCoNs are seamlessly integrated into the UML (Giese et al., 1999c) and have a formal semantics (Giese et al., 1999b). Most high-level net approaches such as (Genrich and Lautenbach, 1981, Jensen, 1992) work with complex textual and formal annotations that are hard to use for the non-expert. In contrast, the net dialect puts its focus on visual expressive power. The major aspects of a workflow system are expressed in different diagrams whereas related aspects are described in the same diagram but represented by explicitly distinguishable graphical entities in the nets.

2.4. APPLICATION OF THE APPROACH

This section sketches the development of workflow applications in general and positions the work presented in this paper with respect to the phases and persons involved in traditional workflow application development processes. In general, workflow applications are developed in complex processes in which numerous persons with different backgrounds participate. While workflow application development processes differ from one project to the next, the general procedure can be described as follows. The first phase deals with gathering information, relevant to the application process. Empirical studies based on interview techniques and analysis of available documentation are used. The activities in this phase are centered around the application domain, and technical issues are often not considered. The next phase involves business process modelling, in which the information gathered is used to specify business process models. The main purpose of business process modelling is to provide a general and easy-to-read notation that enables information system experts and domain experts to validate and optimize business process models. The result of this phase is specified in

a business process model, which is used as a basis for the next phase, the workflow modelling phase. The aim of this phase is to enhance the business process model with information needed for the controlled execution of workflows by a workflow management system. This involves adding technical information and purging application specific information that is irrelevant for workflow management. Finally, the workflow application is deployed in the target environment, and the operational phase starts.

The work presented in this paper deals with workflow modeling and enactment from a technical point of view. In the traditional workflow application development process as discussed above, information systems experts do workflow modeling. However, they make use of information provided by business experts on the business process. We do not expect workflow participants like clerks to model or modify workflows. In contrast, we expect workflow designers to have an understanding of both object-oriented modeling techniques and workflow modeling techniques. In addition, a good view for structure and process modeling is essential to obtain adequate overall structural and dynamic models. This is work for experts in process and workflow modeling. So, the approach is intended for companies and organizations that plan to structure and adopt their workflow facilities according to the needs of their evolved IT infrastructure.

Although the different departments of a huge organization may see a need to have clean interfaces between the processes going across department boundaries, e.g., for internal accounting, planning or responsibility reasons, the approach is of special benefit in situations where more or less independent partners are involved. If a company plans to provide parts of their internal workflow functionality to outside customers, e.g., because they are involved in business-to-business workflow projects, defining secure borders of information and control flow are mandatory. Even legal contracts are easier to oversee if the interaction across borders is done by providing and using interfaces with well-defined roles, responsibilities and obligations. Moreover, such an infrastructure pays off the initial costs of development especially when re-use is one of the main goals.

3. The Object Coordination Net Approach

We assume the reader to be familiar with standard place-transition nets (Brauer et al., 1987) and the basic concepts of object oriented analysis and design (cf. (Booch, 1993) and many others). Furthermore, we do not go into the details of the various UML diagrams (OMG, 1999).

We need only variants of structure diagrams, which are especially well-known for decades from entity-relationship diagrams.

In order to support a method for workflow modeling as described in the previous section, it is necessary to structure a system into encapsulated subsystems and to employ a well-defined and expressive formalism to describe interfaces and interface-based interaction. In the context of workflow modeling, the organizational perspective is used to obtain a coarse-grained structure of the environment for which workflow schemas are to be modeled. This structure is described by UML structure diagrams using subsystems, provided and imported interfaces, classes with operations and attributes, inheritance, associations and so on. For decoupling subsystems properly, we distinguish clearly between the external information provided by an interface and its internal implementation. Moreover, we rigorously permit the usage of functionality from a particular subsystem through its provided interface only. The internals of a subsystem itself may obtain a complex structure as well as complicated rules for handling control and data. This leads in practice to a hierarchical design of such systems. The result is a set of subsystems that interact via services specified in interfaces to the outside and possibly using services from other subsystems via their interfaces to implement the provided functionality. A typical example of this hierarchical view of a system's structure that ranges from subsystems representing entire companies down to the implementation of a single service, consisting of a few classes and methods only, is depicted in the Figures 6, 7 and 12. Here, each diagram refines a specific part of the previous one. Interfaces are visualized by small circles. Undirected lines link interfaces to their provider whereas directed, dashed lines specify the relationship between the user and the provider where the line points to the used interface.

Besides strict support for encapsulation, our method takes the behavioural aspects of systems much more seriously than standard UML modeling. This is the case for interfaces as well as their implementation in classes and operations. Whereas the former provides a secure basis for interface usage across subsystem boundaries, makes the latter the overall method more adequate for describing the process perspective of workflows. To achieve this goal, we use variants of Petri-Nets that are well-suited to describe the different aspects of systems and interfaces in an object-oriented style.

For the sake of a better understanding of the different usages of Petri-Nets in our approach, we give an overview over the basic elements and their meaning. Figure 1 gives examples for those elements that are used in this paper. Transitions (square boxes, e.g., $t_1 \dots t_6$) are interpreted to be *actions* that occur through calls to services provided

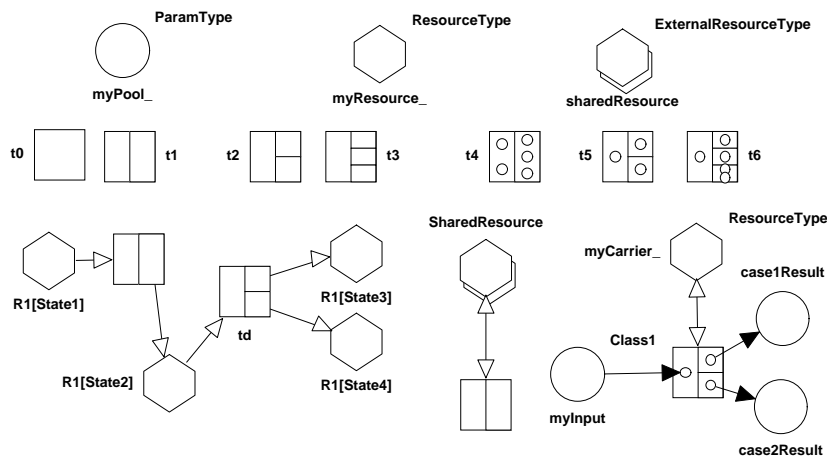


Figure 1. Elements of Object-Coordination-Nets

by objects. Depending on the usage context, actions may have a signature (little dots) visualizing their input and output parameters (see t_4 , t_5 , t_6) or not ($t_0 \dots t_3$). In both cases, the righthand-side of the box indicates whether there is a single output or whether there are alternative outcomes caused by internal decisions. In the latter case, the box is split as, e.g., in t_2 , t_3 , t_5 , and t_6 . For places, which are called *pools*, we distinguish between two principle entities: Simple data and control flow using objects typed according to the type system obtained by the informational perspective of workflows are stored in so-called *event pools*. These are visualized by circles. More permanent objects that, taking an object-oriented view, are the carrier of activity or provider of services, are interpreted as *resources*, the system uses to perform its work. Places for resources are represented by so-called *resource pools*, which are visualized by hexagons.

All elements may be typed, e.g., the resource `myResource` is of type `ResourceType`. The underlying type system is defined by the context of the nets, i.e., the UML structure diagrams. Additionally, there is a notion of *object state* integrated into the type system that allows for describing state changes for resources. As an example, the resource `R1` in Figure 1 undergoes a state change from `[State1]` to `[State2]`. Afterwards, it is changed to `[State3]` or `[State4]` depending on the internal decision in `td`. Dependent on their usage, resources may be under exclusive control of a specific object or have to be shared with others. The latter case is visualized by using a shadowed hexagon.

As usual in Petri-Nets, actions and pools are combined by a so-called *flow relation* that makes pools pre- or post-conditions of actions. We

distinguish the specific resource that executes an action, i.e., the *carrier of activity*, from all other resources and parameters by using a specific edge type (white arrow head). This resembles the role of the *this* or *self* object for a method-call in object-oriented languages. All other entities flow along so-called *parameter edges* (black arrow head).

The object-oriented view of method invocation for actions is also supported by the intuitive call semantics, which is well-known from procedural programming languages and remote procedure calls. Figure 2 illustrates the concept: As a precondition for firing, the carrier of activity, i.e., the object that processes the call (here `obj1` of type `Resource1`), is required as well as the parameters (here an object of type `InType`) and a simple `Event` (token). Moreover, the carrier is required to be an object of type `Resource1` 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 post conditions afterwards in a second step. Calls to other actions during firing inside a transition are allowed. Due to the synchronous nature of call and return, this works across hierarchies and can also be used as a wrapper call to legacy code. In Figure 2, the effect of `resu = obj1.fun(in)` is an object of type `ResultType` in place `resu` and a state change for the carrier which is afterwards in state `[State2]`.

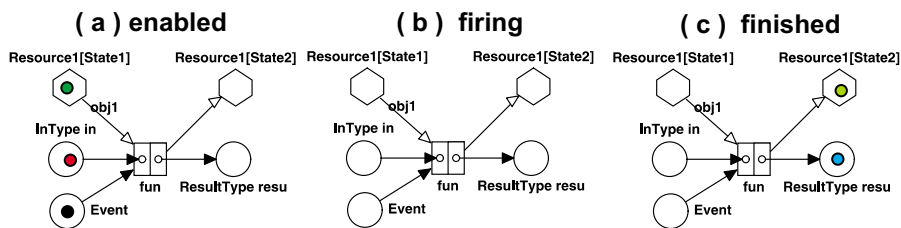


Figure 2. Steps during execution of `resu = obj1.fun(in)`

After introducing the basics of the nets, we discuss their usage within the UML context. The UML structure diagrams for `interfaces` and their implementing `classes` are extended to allow for a Petri-Net compartment. Moreover, `methods` (or `operations`) are extended to `services` that permit the description of detailed resource, control and data flow via Petri-Nets. Technically speaking, we use so-called `stereotypes`, a specific built-in extension concept of the UML to combine UML diagrams with our nets (cf. (Giese et al., 1999a)). Figure 3 depicts the three different usages of Petri-Nets in the context of a typical UML structure diagram that specifies a contract called `Contract1` and its implementation `ContractImpl`.

The three different kinds of nets allow for a clear separation between their different purposes, i.e., describing the external behaviour,

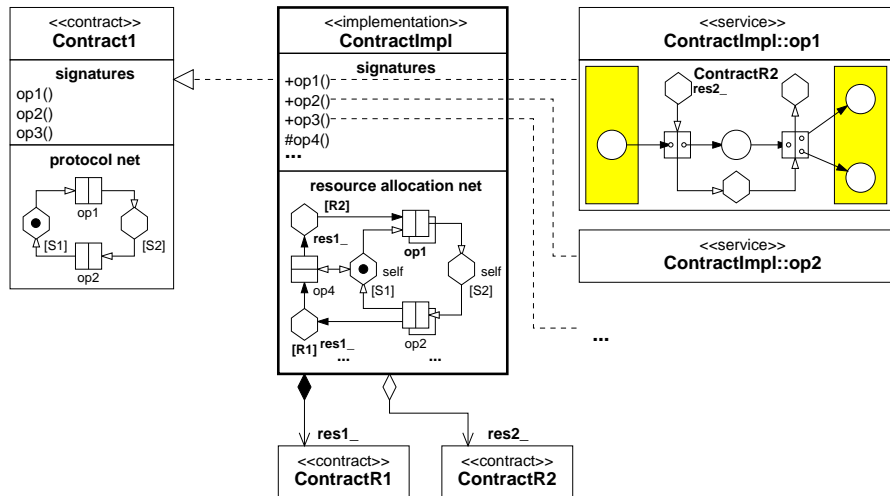


Figure 3. Embedding of Object-Coordination-Nets into the UML

the overall resource handling of a class and the details of a single service. It should be noted, however, that the different kinds of nets are derived from a common basis, which has already been introduced. All that is needed is more or less strict rules regarding the usage of the constructs in order to keep each type of description as simple as possible.

The stereotype `<<contract>>` replaces standard UML interfaces. Because the entire approach is centered around the *contract principle* (Meyer, 1997), this extension of interfaces is of special importance. The textual part of a contract is a traditional *abstract data type*-like signature (Ehrig and Mahr, 1985) that specifies the names and required parameters of all provided (public) operations, which are now called services. This standard interface is extended by a state-machine-like *protocol net* (PN) that specifies externally visible behavior. This kind of specification enhances the benefits of interfaces in the case of services that exhibit non-uniform availability. This is often the case in systems acting according to complex organizational rules. Protocol nets only use resource pools and abstract actions without signatures. In order to explain this kind of behaviour description, we use a contract that describes the well-known handling of a `file` (see Figure 4).

An appropriate protocol for a file resource with operations `open`, `read` and `close` starts with an initial state `[closed]` (identified by the token dot) that only permits an `open` request. Moreover, the protocol states that an `open` request may not always be successful: In the positive case, the resulting state is `[opened]`, otherwise the protocol is in state `[closed]` again. Reading data is only permitted for a file in state `[opened]` using `read`. This may work or end up in a state `[eof]` signalling the end-of-file.

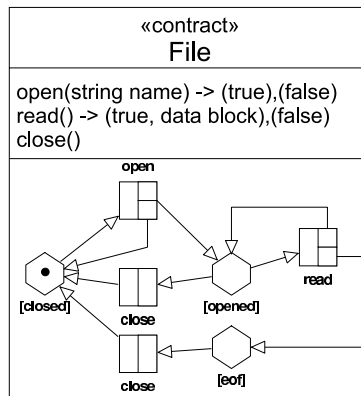


Figure 4. The file contract

In both, state [opened] and [eof], the close request can be used to close the file and reach the initial state again. Note, that the precondition of an operation (state with outgoing arc to operation) specifies that such is only offered if the object providing the interface is in exactly that state. The description adds essential information to an interface in specifying the usage conditions that allow for specific operations under specific circumstances. Whereas the states abstract from the internal details that prevent an operation from being available, they are detailed enough to give hints on how to use the interface in a secure way and on what kind of problems are to be expected.

The second extension, i.e., the stereotype <<implementation>>, is used to specify the request processing and autonomous behavior of classes by means of a so-called *resource allocation net* (RAN). It covers the internal details for implementing the provided services and handles the overall *management of resources*. Such implementation types delegate request processing to service nets or textual services where the former can be integrated into the model using the <<service>> stereotype. The detailed sub-workflows implementing a service provided by a contract are specified in a so-called *service net* (SN). Because the details of using these nets are explained in the context of the example in the next section, we only illustrate the abstract overall interaction between the three net types here (see Figure 5). An instance of a class provides a contract to the outside that is implemented by its public services and an instance-local description of the coordination of the required resources. 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.

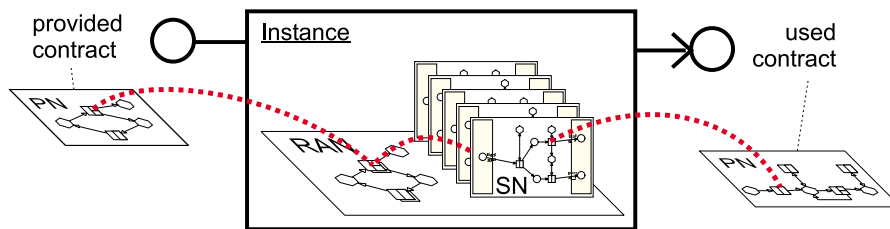


Figure 5. OCoN architecture and usage of Net types

4. Example Workflow

This section illustrates the use of the OCoN approach to modeling interorganizational workflows with an example from electronic commerce. In particular, a media store scenario is introduced, which involves three organizations: a media store, a production enterprise, and a transport company. The production enterprise produces media such as books and CD's and offers them to the online media store. While the details of the media store are described below, its responsibilities include customer management, reservation management, and accounting. However, rather than shipping the media to its customers, the media store makes use of the services provided by a transport company.

An important observation in this scenario is the fact that work is organized by intra-business as well as business-to-business processes. For example, adding a book to the media store, updating the inventory, registering a new customer, reserving a book for a customer, renting the book and shipping it to the customer is a textual description of such a processes. This business process involves multiple organizations. The media store buys the book from the production enterprise, the customer reserves the book, the media store checks the book for availability and, in the positive case, ships it to the customer, making use of the transport company. Since the business process spans multiple organizations, it must be realized by an interorganizational workflow.

Customers access the services provided by the media store using web browsers. At the front end, the media store looks like a traditional web site, with customer registration, online access to catalogs, reading book reviews, listen to music excerpts, and so on. However, the front end is connected to an information system, which models and implements an interorganizational workflow.

To give a flavour of the application of the different concepts introduced in the previous section, we describe parts of the system from different levels of granularity ranging from the internal handling of single media inside the media stock to complex processes involving most parts of the store but also external services.

4.1. TOP-LEVEL STRUCTURE DIAGRAM

We assume a successful system analysis and start with the structure of the overall system by describing the subsystems and their relationships. In terms of workflow terminology, each organization is represented by exactly one subsystem. Hence, the top level structure diagram can be regarded as part of the organization perspective in workflow modeling. (We remark that additional aspects need to be covered in the organization perspective, for instance roles, persons and resources. These aspects will be discussed in more detail below.)

In the scenario sketched in Figure 6, there are three principle organizations (or business partners) that are represented by subsystems in the structural model. The structural model is shown in Figure 6 as a UML structure diagram. The `EnterpriseProd` subsystem represents the production enterprise; the services it can offer are specified by contracts. In particular, there is a `MediaOffers` contract, which specifies how the enterprise production company offers media to its clients. The example abstracts from other contracts this organization might also have, for instance contracts involving the cooperation with artists, banks, and marketing companies.

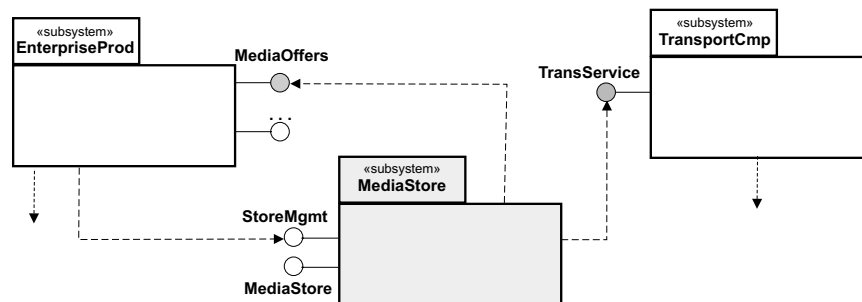


Figure 6. Media Store Example: Subsystems, Contracts and Usage Relationship

The media store is represented by the `MediaStore` subsystem, which provides two contracts to its clients: the `StoreMgmt` contract is used for management issues, e.g., ordering new media or filling the stock. Customers are served by the `MediaStore` contract. Both contracts are implemented within the subsystem itself using internal subsystems, discussed below. The transport company is represented by the `TransportCmp` subsystem, which provides a transport service (`TransService`) to its clients. The business scenario suggests that the business partners make use of services provided by their partners to fulfill their goals.

In the proposed approach, the functionality of other subsystems can only be used via contracts. To this end, the subsystems are interconnected by exporting and importing contracts from each other. In

structure diagrams, usage of contracts is represented by dotted arrows. The media store subsystem imports two external contracts, namely the `MediaOffers` contract for ordering new media from the production enterprise and the `TransService` contract from the transport company for shipping rented media. Besides the dependencies which are made explicit via the usage of local or imported contracts, no other interaction between subsystems is allowed.

From a workflow perspective this means that a top level structure diagram provides a coarse-grained description of the organizational perspective of all possible workflows which may be enacted in this system. More importantly, the contracts can be interpreted as a description of the different roles a subsystem may play during workflow execution. The top-level structure diagram specifies all sub-workflow schemas supported by a specific subsystem which are designed for external use. To this end, the contracts an organization offers can be regarded as the API (Application Programming Interface) of that company. Hence, services provided are encapsulated and can be re-used by other companies.

This makes it feasible for other business partners to import services in a reliable manner. In particular, if the organization fails to fulfill the contract then it can face legal consequences. While the legal aspects are not investigated at this point, the OCoN approach provides the technological infrastructure to tackle legal issues, which are of crucial importance in business-to-business electronic commerce and interorganizational workflows.

The description techniques provided by the UML for static structure modeling can be used to specify the details of the complex application objects, which represent entities such as users, media, offers, orders, reservations. Specifying the types of such data including their basic functionality in the familiar object-oriented way using attributes, methods, class diagrams and relations among them is quite standard.

Such diagrams provide the basis for all remaining aspects because classes are used to type event and resource pools, the parameters of services and the objects flowing through edges in a manner that allows for type matching and, hence, makes consistency checks much easier. This information, however, is not required at the very beginning of a workflow development project, it is provided in a step-by-step manner. We remark that the specification of the class diagrams should be done to maintain coherence with the subsystems. This means that classes should be defined as local as possible, and visibility rules should be utilized to avoid global information. Avoiding global information strengthens the power of encapsulation, as the following example shows. Assume a class that is used to wrap calls to a legacy application which is only used in a single subsystem. If this class is put into the respective

subsystem, a change due to the installation of a different application can be performed with a new wrapper class. Other subsystems will not notice that modification at all.

4.2. LEVEL 2: INTERNAL STRUCTURE OF MEDIASTORE SUBSYSTEM

The high-level representation of the subsystems is well suited to present an overview on the partners involved and their relationship as far as providing and using contracts is concerned. However, for modeling the process perspective and for system design, the internal structures of the subsystems also need investigation. In the OCoN approach, nested structure diagrams are used for that purpose. The internal structure of the media store subsystem is shown in Figure 7. This level uses the same techniques and notations as the higher level. The `MediaStore` subsystem consists of a number of subsystems that can provide contracts and which make use of contracts provided by other subsystems of the media store.

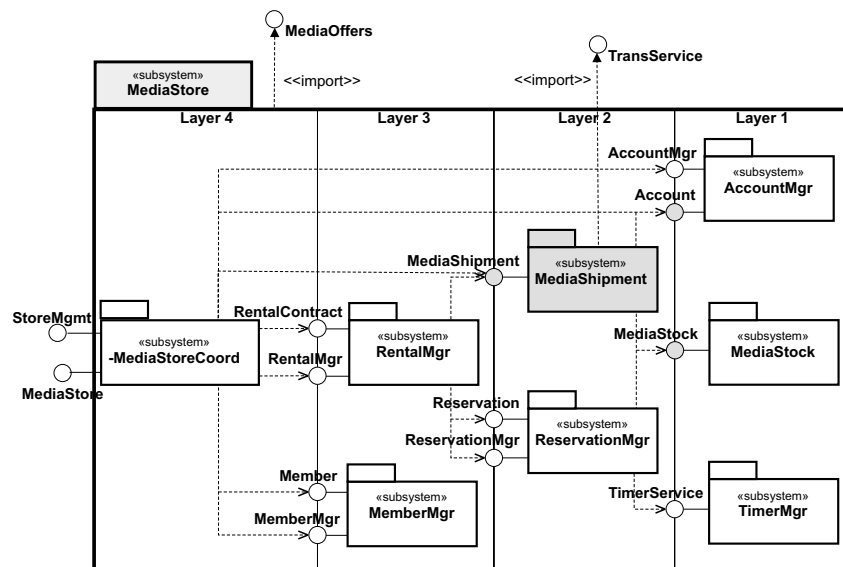


Figure 7. Details of the `MediaStore`

The subsystems include a media store coordinator (`MediaStoreCoord` subsystem) that is responsible for coordinating the activities inside the media store. In addition, it provides the `StoreMgmt` and `MediaStore` contracts, which are the contracts available at the higher level of abstraction, i.e., the overall structure diagram shown in Figure 6. Other subsystems of the media store include a rental manager (`RentalMgr`), which is responsible for renting media, involving, among other things,

recording the state of media. In order to do that, it makes use of contracts provided by the reservation manager (`ReservationMgr`) and of the media shipment (`MediaShipment`) subsystems. The other subsystems of the media store are a member manager (`MemberMgr`), an account manager (`AccountMgr`), a media stock manager (`MediaStock`), and a time manager, denoted by `TimerMgr` in Figure 7. Notice that the contracts offered and used by these subsystems are also specified in the figure.

4.3. LEVEL 3: A SINGLE CONTRACT AND ITS IMPLEMENTATION

Until now, the process perspective was hidden in contracts. The handling a single media inside the `MediaStock` is used to show the usage as well as implementation side of a single contract. This leads to the basic steps which are used later on in the process perspective to build up complex workflow schemata.

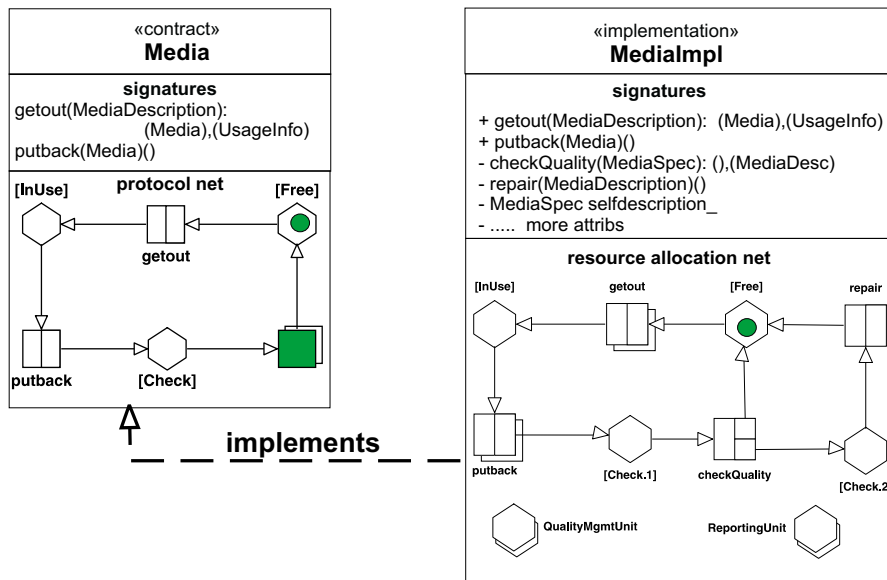


Figure 8. Media contract and its implementation MediaImpl

Figure 8 (left) specifies a simple contract to deal with an object of type `Media` such as, for example, a book or CD using the services `getout` and `putback`. The protocol net (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`. 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 (shadowed

transition). However, the protocol guarantees that eventually the object will be in state `[Free]` again.

The implementation `MediaImpl` in Figure 8 (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` as to whether they are ok to be used again or not. Moreover, the resource allocation net (RAN) implements the protocol provided by the PN. First of all, the RAN states that two external resources imported from other subsystems (shadowed hexagons), namely `QualityMgmtUnit` and the `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 split into two parts `checkQuality` and `repair`, in the case of incoming objects that are in bad shape. However, the only thing regarding these details that 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]`. Because this part of the behavior depends on the object at hand, the RAN uses a transition `checkQuality` with two possible alternative outcomes. The use of shadowed transitions indicates that the net (PN or RAN) at hand does not have complete control over the operation or how it is called. Just as the implementation details are out of control of the anonymous transition in the PN, the number and time of calls to `getout` and `putback` from the outside are not under control of the implementation. Hence, in contrast to the private operations, both public operations are represented by shadowed transitions in the RAN in the right of Figure 8.

The level of most detail is reached when the details of a single service implementation are described using a service net. The interaction between the RAN and services in the context of our example is shown by means of the internal `MediaImpl` service `checkQuality` (see Figure 9). 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 Figure 9, left). The transition here requires a single data parameter and produces an alternative output to abstract from an internal decision.

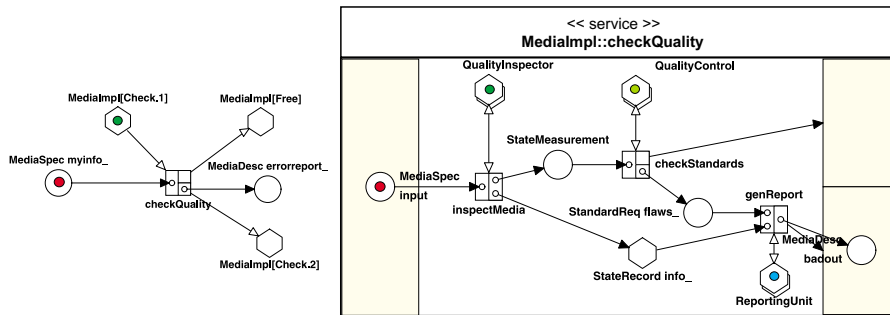


Figure 9. Call and implementation of a service

In conformance with the RAN in Figure 8, the operation can only be carried out by instances of the resource `MediaImpl` in state `[Check.1]` which is changed either to a `MediaImpl` of state `[Check.2]` or `[Free]`. Because this is specified in the RAN, it is not explicitly stated in the textual signature. However, 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. Similarly, in object-oriented programming, the resource that is the carrier of activity is not handled as an explicit parameter. Because both preconditions are fulfilled, the action is enabled to fire. The unfolded version of the calling transition (see Figure 9, 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 `QualityMgmtUnit` and the `ReportingUnit` (cf. the RAN in Figure 8) in order to carry out the delegated work. For example, the object is inspected, a status record is generated and the `QualityControl` decides whether the object is ok (event to upper output alternative) or a kind of error report has to be produced. Depending on the decision taken, the final state of the object will be `[Free]` or `[Check.2]`. In the latter case, the error report is also produced as a post condition.

Note, however, that this level of detail is rarely used if powerful application programs are available or simple processes are to be described. A service net is only used if it is necessary to specify the detailed scheduling of resources, workflows with explicit parallelism, or a specific, tricky use of fine-grained application programs. Most of the time, the functionality of the contracts abstracts from these details and processes are described on a much higher level of abstraction. In these cases, the signature of a called service describes the parameter and return types of the called legacy code.

4.4. PROCESS PERSPECTIVE

Having described the structure of our system (Section 4.2) and the needed detailed functionality (Section 4.3), we are prepared to explicitly model the process perspective, for an important business process which involves most subsystems of the media store. The workflow is made available to customers as part of the contract `MediaStore` that the media store offers (see Figure 6).

The business process is explained as follows. Assume a customer enters the media store (i.e., accesses the media store's web site) and wants to rent some media, let's say a CD and a book. After the customer logs on to the system it checks whether she is a registered member or not. If not then she is requested to fill a number of forms, registering her as a new user. She may browse catalogs, inspect books and book reviews, listen to music excerpts, etc., in order to choose media for rental. The media is then retrieved, packed, and shipped, and the customer receives information about the rental cost.

The activities and their relationships in this business process are shown in Figure 10. Notice that the Petri net shown there is rather incomplete and informal. However, this representation will serve as a basis for the next step; its refinement to represent an interorganizational workflow together with the other workflow perspectives.

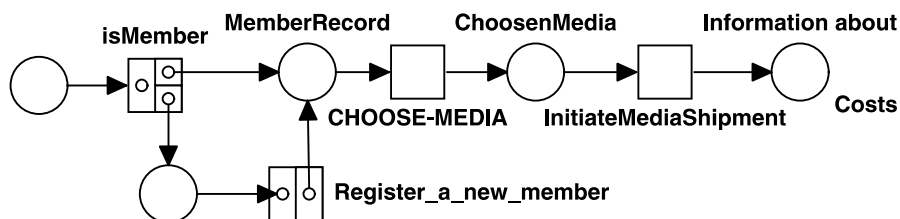


Figure 10. Business Process, Customer rents Media (simplified).

In the next step, organizational units and resources are added to the process description. The informational perspective can be made more clear when using the types derived from class diagrams to type some of the pools and hence restrict the type of items flowing through the net. More important is the decision as to 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. which are the carriers of activity for actions. Resources may be other subsystems but also specific roles in a company's organizational structure. Particularly important steps, for example, may be only performed by authorized personell, e.g., from management. On the other hand, information about the detailed

imports the `TransService` from the transport company, as shown. In addition to importing contracts external to the media store subsystem, it also makes use of contracts provided by the other subsystems of the media store, i.e., the contracts `Account` or `MediaStock`. The details of shipment orders and how they are handled are implemented locally in the `ShipmentOrderImpl`.

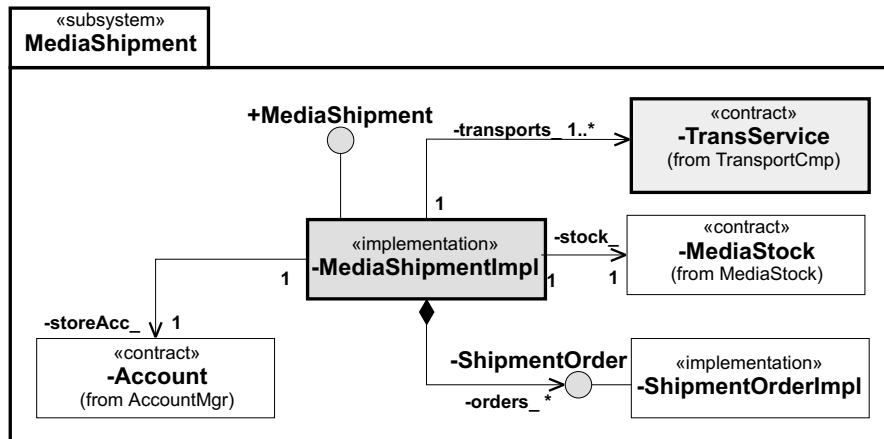


Figure 12. The `MediaShipment` subsystem

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 from users of the service, the contract may well be re-used in a completely different context, e.g., when the overall coordination `MediaStoreCoord` decides finally to sell media that have been rarely used over a specific time. The service itself is specified using the service net shown in Figure 13.

Only confirmed shipment orders in state `[Accepted]` are processed. The media stock is used to check out the ordered media which are delivered by the transport service. In addition to the real delivering, the state of this order (represented by a `ShipmentOrder`) is observed in parallel in the subsystem. The account contract is used to pay for the transport as well as to compute the shipping costs. These are represented by a payment record (which is assumed to be specified as a class in the informational perspective) as the final output of this service.

Of specific interest in the specification of the behavioral properties is how the external sub-workflow from the transport service contract is used to handle the transport of the media. 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 us-

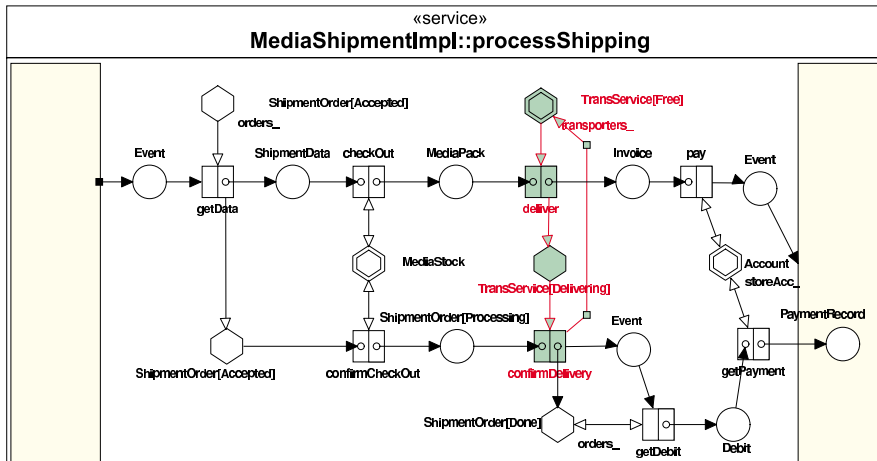


Figure 13. Service ProcessShipping uses external contract TransService

ages. 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], the resource is no longer needed and, hence, is available for new transports or other users. The imported contract is represented by the shaded area in Figure 13.

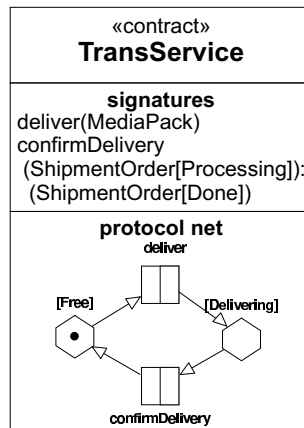


Figure 14. External contract TransService from TransportCmp

Importing the transport service contract is easy, because the protocol net (shown in Figure 14) of that contract provides the needed information, i.e., it specifies exactly how to use it. This exact protocol net is visually embedded into the using service net which makes consistent usage of subworkflows, even from different system parts, an easily manageable task. The formal background of visual embedding of contracts is introduced in (Giese et al., 1999c). This additional type of

information provided by contracts as compared to purely syntactical interfaces, eases the structured specification of behavioral workflow properties to a great extent.

5. Discussion

The approach presented in this paper does not invalidate previous approaches to workflow modeling and execution. In contrast, it makes use of workflow techniques in object-oriented applications. Its main strengths are in settings, where structure and functionality are modeled and implemented by objects. Given current developments in the areas of enterprise frameworks and business objects, we envision more and more projects to rely on object technology for system analysis, design, and implementation. In these modern object-oriented information system environments, the approach presented in this paper is well suited for seamless integration of workflow functionality.

5.1. RELATED WORK

Although there are many approaches involving a diversity of graphical descriptions as well as mechanisms such as statecharts (Harel, 1987), the use of Petri net variants in workflow languages is widespread, e.g. FUNSOFT Nets (Deiters and Gruhn, 1994), the HOON approach (Han and Weber, 1998) and especially the work of van der Aalst, e.g., (van der Aalst, 1998). However, most of the work, and even recent approaches dealing with interorganizational workflows, like that of van der Aalst (1999) focus on the process perspective and fail to utilize structure to overcome scalability and encapsulation problems. The approach described in (van der Aalst et al., 1999) combines (by means of a so-called system net) all perspectives (described in so-called object nets). Mechanisms solely based on Petri nets are very useful when the dynamic and static aspects of workflows are modelled from scratch. The OCoN approach, however, may reuse existing object-oriented models describing static aspects, typically expressed by UML structure diagrams.

In the Mentor project (Wodtke et al., 1996), 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 the Mentor project.

It is worth noting that contemporary workflow management systems (such as IBM's MQSeries Workflow (IBM, 1999)) consider software

programs only as external applications. Based on defined interfaces, a workflow management system invokes these applications with parameter values and transfers their result values into the workflow. However, the integration of software development and workflow management, as proposed in this paper, can provide workflow support for object-oriented applications without the need for a dedicated workflow management system.

The Object Management Group (OMG) has specified a Workflow Management Facility (OMG, 1998), which is supported by the Workflow Management Coalition. This endeavor has been initiated by the understanding that to be successful for complex business applications, the CORBA infrastructure requires standardized workflow support. The Workflow Management Facility aims at two issues: to provide runtime interoperability of existing workflow management systems and the ability to use workflow monitoring and auditing tools from different vendors in a given application (Schmidt, 1996). Since the proposal is open to a variety of workflow languages and systems, it does not introduce a workflow language. Rather than specifying interoperability between existing workflow management systems, our approach aims at an integrated framework in which the structure and process behavior of object-oriented application systems can be specified. Hence, workflows provided by different organizations can be assembled to a more complex workflow, a functionality which is not in the scope of the OMG Workflow Management Facility.

5.2. EVALUATION

Given this discussion of related work, the key properties of the proposed approach are now discussed. UML structure diagrams are widely accepted nowadays for analysis and design of object-oriented application systems. By adding behavior specification as proposed in this paper, the expressive power to standard object oriented design techniques is enhanced. Since information system designers are often experienced UML users, the basis for ease of communication among all people involved in workflow planning, specification, implementation, and monitoring is established. As a result, the gap in languages and notations between workflow and OO application development is closed to some extent. The example presented in this paper illustrates that the Petri net formalism used permits the gradual change from informal descriptions to correct nets including typing as a means of integrating workflow perspectives consistently. The key features of the proposed approach are as follows:

- *Integration*: The workflow perspectives are represented in a uniform framework. Namely the integration of the best-accepted nota-

tions from OOA and OOD taken from the UML for static aspects with a variant of high-level Petri nets for behavior. In particular, the object-oriented design of applications, the organizational aspects, and the operational aspects can be specified in an integrated way. The main benefit here is a suitable notation for consistency between the different perspectives.

- *Contracts*: Each subsystem publishes its behavior by contracts. Since contracts represent the workflows an organization can execute, the behavior of workflows is specified formally, and other subsystems can make use of workflows to implement their own functionality. This feature can be used to structure complex organizational as well as interorganizational workflows.
- *Legacy Applications*: Using wrapper techniques, legacy applications can be incorporated in workflow applications. Their integration into a design is supported on the subsystem level as well as the service-call level using the same contract techniques as a vehicle of encapsulation that are used to capture the organizational properties of workflows.
- *Simulation*: During workflow modeling, workflows can be simulated in the proposed approach, which can help to detect inconsistencies and potential bottlenecks in workflow schemas. In addition, simulation techniques can be used during the execution of workflows, for instance, to analyze the performance of different potential continuations of a given workflow instance. Because our method is based on a consistent language with a clear formal semantics, we were able to develop a prototypical implementation of a simulator.
- *Resource Planning and Scheduling*: Since resources required for the execution of workflows are represented explicitly in the proposed approach, resource planning and scheduling of workflow activities can be performed, which is also an important aspect of successful workflow simulations. Resource planning and scheduling may lead to more efficient resource utilization and, eventually, to faster and more effective workflow executions.

Besides these benefits, however, there are some disadvantages of our approach. First of all, the intended methodology is very formal and requires insight into structure and dynamic modelling to obtain its true benefits. Moreover, as with any new approach using new description languages and techniques, there is a learning curve which only pays off if the method is used in a structured way with mid- to long-term

goals. This holds especially for the non-trivial aspects of object-oriented modelling. Moreover, the tool is only of more use than other process description languages iff it is combined with a model of the organizational structure that requires a non-trivial amount of work before processes can be embedded into a proper structural setting.

In the long term, there are strategic benefits regarding the availability of important management information. Having specified a complex workflow system as proposed here, virtually 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 and optimization of organizational structures and business processes. Moreover, the possible effects of changes can be found in explicit specifications, that provide information crucial for local change management as well as more advanced situations such as, for example, finding subsystem candidates for outsourcing. Changes of this kind may even be evaluated by simulating instances of the already specified workflow schemas in changed environments in order to reduce risks due to unexpected effects of system changes.

5.3. EXPERIENCE

The method as well as the OCoN editor and a simulator prototype have been used for more than two years now in a university context for modeling distributed software systems. Recently, these techniques have also been used to describe workflow systems such as an agent-based storage system, a rent-a-car service and an international UPS-like transport service. At the moment, we are using case studies of business processes and workflow performed in cooperation with companies to evaluate the approach in real-life business settings.

6. Conclusions

This paper describes a method for specifying workflow schemas in a style that integrates all perspectives acknowledged to be useful in the literature. The workflow language is an enhanced object-oriented modeling language that combines UML with a specific form of adopted Petri-Nets. The approach is particularly useful for describing processes in large companies and for business-to-business szenarios. The next steps for future work will be the tight integration of the OCoN editor with a state-of-the-art UML development tool and a rigorous evaluation

of the case studies currently in progress. In the long term, extending the interface notion of a contract to additional aspects such as performance requirements for enactment or information about legal matters seems to be promising.

References

- Booch, G.: 1993, *Object-Oriented Analysis and Design with Applications*. Addison-Wesley, Menlo Park CA. (Second Edition).
- Brauer, W., W. Reisig, and G. Rozenberg [eds]: 1987, *Petri Nets: Central Models (part I)/Applications (part II)*, Vol. 254/255 of *Lecture Notes in Computer Science*. Berlin: Springer Verlag.
- Coleman, D., P. Arnold, S. Bodoff, C. Dollin, H. Gilchrist, F. Hayes, and P. Jeremaes: 1994, *Object-Oriented Development: The Fusion Method*. Prentice-Hall.
- Deiters, W. and V. Gruhn: 1994, 'The FUNSOFT Net Approach to Software Process Management'. *Int. Journal on Software Engineering and Knowledge Engineering* **2**(4).
- Ehrig, H. and B. Mahr: 1985, *Fundamentals of Algebraic Specifications 1*, Vol. 6 of *EATCS Monographs on Theoretical Computer Science*. Springer Verlag.
- Genrich, H. J. and K. Lautenbach: 1981, 'System Modelling with High-Level Petri Nets'. *Theor. Comp. Science* **13**, 109 – 136.
- Georgakopoulos, D., M. Hornick, and A. Sheth: 1995, 'An Overview of Workflow Management: from Process Modeling to Workflow Automation Infrastructure'. *Distributed and Parallel Databases* **3**, 119–153.
- Giese, H., J. Graf, and G. Wirtz: 1998, 'Modeling Distributed Software Systems with Object Coordination Nets'. pp. 107–116. Proc. Int. Symposium on Software Engineering for Parallel and Distributed Systems (PDSE'98), Kyoto, Japan.
- Giese, H., J. Graf, and G. Wirtz: 1999a, 'Closing the Gap Between Object-Oriented Modeling of Structure and Behavior'. In: R. France and B. Rumpe (eds.): *UML'99 - The 2nd Int. Conf. on The UML, Fort Collins, Colorado, USA*, Vol. 1723 of *LNCIS*. pp. 534–549.
- Giese, H., J. Graf, and G. Wirtz: 1999b, 'Contract-based Coordination of Distributed Object Systems'. In: H. R. Arabnia (ed.): *Proc. Int. Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA'99), Las Vegas, Nevada*.
- Giese, H., J. Graf, and G. Wirtz: 1999c, 'Seamless Visual Object-Oriented Behavior Modeling for Distributed Software Systems'. In: *IEEE Symposium On Visual Languages, Tokyo, Japan*.
- Han, Y. and H. Weber: 1998, 'Adaptive Workflow and Software Architecture Thereof'. *Journal of Integrated Design and Process Science* **2**(2), 1–21.
- Harel, D.: 1987, 'STATECHARTS: A Visual Formalism for complex systems'. *Science of Computer Programming* **3**(8), 231–274.
- IBM: 1999, *IBM MQSeries Workflow: Concepts and Architecture*, Vol. Version 3.2. IBM Publ. No GH12-6285-01.
- ITU: 1996, 'Message Sequence Chart (MSC) Recommendation Z.120'. International Telecommunication Union (ITU).
- Jablonski, S. and C. Bussler: 1996, *Workflow Management: Modeling Concepts, Architecture and Implementation*. Int. Thomson Computer Press.

- Jacobson, I., M. Christerson, P. Jonsson, and G. Övergaard: 1992, *Object-Oriented Software Engineering: a use case driven approach*. Addison-Wesley.
- Jensen, K.: 1992, *Coloured Petri Nets Basic Concepts Analysis Methods and Practical Use Volume 1*, EATCS Monographs on Theoretical Computer Science. Springer Verlag.
- Leymann, F. and W. Altenhuber: 1994, 'Managing Business Processes as an Information Resource'. *IBM Systems Journal* **33**, 326–347.
- Leymann, F. and D. Roller: 2000, *Production Workflow: Concepts and Techniques*. New Jersey: Prentice Hall.
- Meyer, B.: 1997, *Object-Oriented Software Construction*. Prentice Hall. 2nd edition.
- OMG: 1998, 'BODTF-RFP 2 Submission Workflow Management Facility (joint-Flow)'. CoCreate Software, Concentus, CSE Systems, Data Access Technologies, Digital Equipment Corp., DSTC, EDS, FileNet Corp., Fujitsu Ltd., Hitachi Ltd., Genesis Development Corp., IBM Corp., ICL Enterprises, NIIP Consortium, Oracle Corp., Plexus - Division of BankTec, Siemens Nixdorf Informationssysteme, SSA, Xerox. OMG doc bom/98-06-07.
- OMG: 1999, 'OMG Unified Modelling Language 1.3'. Object Management Group. OMG doc ad/99-06-08.
- Rumbaugh, J., M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen: 1991, *Object-Oriented Modeling and Design*. Prentice Hall.
- Schmidt, M.-T.: 1996, 'Building Workflow Business Objects'. In: *Proceedings of OOPSLA '98 Workshop 8: Business Object Design and Implementation IV: From Business Objects to Complex Adaptive Systems*.
- Shrivastava, S.: 1999, 'Workflow Management Systems'. *IEEE Concurrency* **7**(3), 16–17.
- van der Aalst, W.: 1998, 'The Application of Petri Nets to Workflow Management'. *Journal of Circuits, Systems and Computers* **8**(1), 21–66.
- van der Aalst, W.: 1999, 'Interorganizational Workflows: An Approach based on Message Sequence Charts and Petri Nets'. *System Analysis - Modelling - Simulation* **34**(3), 335–367.
- van der Aalst, W., D. Moldt, R. Valk, and F. Wienberg: 1999, 'Enacting Interorganizational Workflows Using Nets in Nets'. In: *Proceedings of the 1999 Workflow Management Conference*. pp. 117–136.
- Weske, M. and G. Vossen: 1998, *Workflow Languages*, pp. 359–379, International Handbooks on Information Systems. Berlin: Springer.
- Wirtz, G., M. Weske, and H. Giese: 2000, 'Extending UML with Workflow Modeling Capabilities'. In: *Fifth International Conference on Cooperative Information Systems (CoopIS-2000), 6-8 September, Eilat, Israel*.
- Wodtke, D., J. Weissenfels, G. Weikum, and K. Dittrich: 1996, 'The Mentor Project: Steps towards Enterprise-Wide Workflow Management'. In: *Proceedings of the 12th International Conference on Data Engineering*. pp. 556–565.