Advanced Topics in Workflow Management: Issues, Requirements, and Solutions

Wil M.P. van der Aalst
Department of Information and Technology
Eindhoven University of Technology
PO Box 513, NL-5600 MB Eindhoven
w.m.p.v.d.aalst@tm.tue.nl

Mathias Weske
Hasso Plattner Institute for Software Systems Engineering
Prof. Dr.-Helmert-Straße 2-3, D-14482 Potsdam, Germany
weske@hpi.uni-potsdam.de

Guido Wirtz
Distributed Systems Group
Department of Computer Science
Westfälische Wilhelms-Universität Münster
Einsteinstrasse 62, D-48149 Münster
guidow@math.uni-muenster.de

This paper surveys and investigates the strengths and weaknesses of a number of recent approaches to advanced workflow modelling. Rather than inventing just another workflow language, we briefly describe recent workflow languages, and we analyse them with respect to their support for advanced workflow topics. Object Coordination Nets, Workflow Graphs, WorkFlow Nets, and an approach based on Workflow Evolution are described as dedicated workflow modelling approaches. In addition, the Unified Modelling Language as the *de facto* standard in object-oriented modelling is also investigated. These approaches are discussed with respect to coverage of workflow perspectives and support for flexibility and analysis issues in workflow management, which are today seen as two major areas for advanced workflow support. Given the different goals and backgrounds of the approaches mentioned, it is not surprising that each approach has its specific strengths and weaknesses. We clearly identify these strengths and weaknesses, and we conclude with ideas for combining their best features.

Keywords: Applied workflow systems, flexible workflow management, workflow analysis

1. Introduction

In recent years, a variety of approaches to modelling and executing workflows have been proposed, based on different workflow languages, methods, and tools, and aiming at supporting different aspects in workflow management [Georgakopoulos et al. (1995), Leymann et al. (1994), Jablonski et al. (1996), Weske et al. (1998)]. Rather than inventing just another workflow language, this paper briefly describes and analyses existing workflow languages with respect to their support for advanced workflow topics. In addition to dedicated workflow modelling approaches, the *de facto* standard in object-oriented modelling and design, the Unified Modelling Language, is investigated, specifically the UML extensions for workflow process modelling [OMG (2000c)]. The dedicated workflow modelling approaches include Object Coordination Nets [Giese (2001), Wirtz et al. (2000)], Workflow Graphs [Weske (2000)], WorkFlow Nets [Aalst (1998)], and an approach based on Workflow Evolution [Casati et al. (1998a)]. These approaches are discussed with respect to coverage of workflow perspectives and their support for flexibility and analysis issues in workflow management. Given different goals and backgrounds, it is not surprising that each approach has its specific strengths

and weaknesses. Based on a common sample workflow process, we clearly identify these strengths and weaknesses, and we conclude with ideas for combining the best features of the approaches.

This paper is organized as follows. Section 2 introduces a set of requirements for advanced workflow modelling. It introduces necessary capabilities for modelling dynamic behaviour in general and for modelling workflow processes in particular. Workflow perspectives are used to characterize the fundamental requirements; flexibility issues in workflow management and analysis features are two advanced topics in workflow management, which are discussed next. Section 3 introduces the approaches to workflow modelling and execution, namely the Unified Modelling Language, Object Coordination Nets, Workflow Graphs, and WorkFlow Nets. Section 4 compares the four approaches based on the required capabilities identified in Section 2. Concluding remarks complete this paper.

2. Requirements for Modelling Dynamic Behaviour

This section discusses requirements for modelling and executing workflows. We start with a set of fundamental requirements, based on dimensions, which have to be considered in workflow modelling; these dimensions are called workflow perspectives. To proceed to advanced topics, flexible workflow management and workflow analysis are discussed. While these topics do not cover the whole workflow area, we believe that some of the most important requirements for advanced workflow technology are addressed.

2.1. Workflow Perspectives

Workflow management deals with modelling and controlling the execution of application processes in heterogeneous organizational and technical environments [Georgakopoulos et al. (1995)]. Workflow management systems are proactive software systems which support modelling and controlling the execution of the automated parts of business processes, i.e., workflows [Leymann, Altenhuber (1994)]. To achieve this goal, a workflow management system needs information about the business process and the organizational and technical environment in which the process should be performed. This information is organized into a set of workflow perspectives [Jablonski, Bussler (1996)]. For the purpose of this paper, we consider functional perspective, process perspective, organizational perspective, informational perspective, and operational perspective. Workflows are expressed in workflow languages [Vossen, Weske (1998)], which provide dedicated language constructs to cover these perspectives.

The *functional perspective* characterizes the activities that have to be performed during a workflow execution. In addition it specifies how these activities are decomposed into smaller units, i.e., it specifies the functional decomposition of a workflow, often represented by a hierarchical structure. The leaves of that structure are logical units of work, called tasks (or atomic workflows). The internal nodes represent complex workflows. The root is referred to as the top-level workflow; business processes are typically represented by top-level workflows. For instance, in a credit request top-level workflow, filling in forms, assessing risks of granting the credit request and preparing and mailing documents can be represented by individual tasks. Note that the functional perspective prescribes what has to be done. However, it does not specify when and under which conditions the tasks are carried out, nor does it specify who performs a given task and which data and applications are used. The other workflow perspectives cover these aspects.

In the *process perspective*, execution conditions are specified, namely start conditions and execution order conditions. These concepts are used to specify if (start condition) and when (execution order) a given workflow should be executed, respectively. The process perspective and the functional perspective are commonly represented by workflow process definitions. An example of a workflow process definition in a banking environment is CreditRequest, which models the activities to process a credit request by a customer. This workflow may be composed of tasks PutCreditRequest, AssessRisk, GrantCreditRequest, and RejectCreditRequest. The ordering of these tasks and their causal interrelationships are specified by the workflow process definition. For example, the top level workflow starts with PutCreditRequest, which is the first task to be executed for that workflow, followed by AssessRisk and—depending on the outcome of the risk assessment task—either a

GrantCreditRequest or RejectCreditRequest task, which shows the use of both execution order constraints and start conditions. Note that the workflow process definition is instantiated for specific workflow cases, i.e., individual workflow instances are handled according to routing specified in the process perspective.

To control the execution of workflow instances, the workflow management system needs information on the organizational structure and the population in which the workflow is executed, covered by the *organization perspective*. Typically the structure of an organization is defined by roles, groups and other artefacts clarifying organizational issues, for example responsibility and availability of persons. In the banking example, for instance, CreditClerk and Secretary are roles, while specific persons may be selected at runtime to play these roles based on availability and skills. The functionality of a workflow management system to determine persons available and competent to perform certain tasks during a workflow instance is known as role resolution [Leymann, Altenhuber (1994)].

The *information perspective* covers data, partitioned in control data and production data. Control data are introduced solely for workflow management purposes, e.g., variables introduced for routing. On the other hand production data is represented by information objects (e.g., documents, forms, and tables) whose existence does not depend on workflow management. The information perspective assigns input and output parameters to single tasks and, hence, covers data dependencies between them. For example, data generated by the PutCreditRequest task is used as input data by the AssessRisk task. This form of data dependencies is called data flow, and it is an important functionality of a workflow management system to guide and control data transfer between related workflow tasks.

The *operation perspective* describes the elementary operations performed by resources and applications. Typically, these operations are used to create, read, or modify control and production data. Depending on the technical environment of the workflow application, operations are implemented by legacy applications or by business objects. Business objects are information system representations of real-world entities, which have a meaning in the business domain. Examples of business objects include business partner, order, and invoice. Depending on the workflow management system, each task, i.e., a leaf in the functional perspective may be implemented by a single application or by multiple applications, whose ordering may be controlled by a script language (cf. traditional workflow management systems such as Staffware and COSA).

A workflow schema is the specification of a workflow covering all perspectives. Typically, workflow management systems have two parts: The build-time part allows for the specification of workflow schemas; the run-time part takes care of the actual enactment of the workflow in the given technical and organizational environment.

2.2. Flexibility

Enhancing the flexibility of business applications has been one of the main motivations for workflow management from the beginning. In traditional workflow management, the main mechanism to achieve this goal is to extract process information from applications with the aim of representing it explicitly to be able to improve the structure of the business process with little effort [Hammer, Champy (1993), Leymann, Altenhuber (1994)]. While this aspect is important for workflow applications, it falls short of supporting highly dynamic business processes, which are typically occurring in the networked global economy [Sheth et al. (1999)]. One of the main obstacles for the use of traditional workflow technology in this context is the fixed structure of workflows, meaning that once a workflow has started, changes to the process structure are no longer feasible. In dynamically changing settings (for example in highly competitive markets or long-running workflows), this limitation of contemporary workflow management systems makes the use of workflow technology in these settings very hard or prevents it altogether. This observation has triggered considerable work on flexible workflow management, for example [Ellis et al. (1995), Casati et al. (1998), Reichert et al. (1998), Aalst et al. (1999), Weske (2001)].

Flexibility in general and the need for workflow change in particular can be classified according to different aspects [Aalst, Jablonski (2000)]. First of all, there may be different reasons for workflow change, for instance changes due to the business context, the legal, or the technical context of the workflow. Next, workflow changes can be classified according to the workflow perspectives involved.

While in general all of the abovementioned perspectives can be subject to change, modifications to the process structure are the most important ones, since the dynamic aspect of workflow execution control is concerned. In this context, the scope of a dynamic change is important. In general, a modification can apply to a single workflow instance, or a modification of a workflow schema can apply to all future workflow instances. An interesting question is raised if active workflow instances should also be changed to the new workflow schema, known as dynamic adaptation (or migration) of workflow instances to new workflow schemas.

The need for flexibility raises many challenging scientific and technical questions. In this paper, we focus on three issues related to flexibility in workflow management: *constrained flexibility*, *instance change*, and *instance migration*.

- *Constrained flexibility.* For many applications, flexibility is required for a smooth workflow. However, unlimited flexibility results in chaos. Therefore, workflow management systems should offer support for constrained change. Given a former version of the workflow process definition, the new or modified workflow process definition should preserve certain properties, e.g., certain tasks in the original workflow should not be deleted. Constrained flexibility is particularly relevant for mission-critical workflows: Only by restricting change it is possible to guarantee certain execution properties of workflows like, e.g., overall consistency.

- *Instance change.* The vast majority of workflow management systems does not support change at an instance level. InConcert is one of the few systems, which supports instance change, i.e., the workflow of a specific instance can be changed on the fly. To support instance change, there should be a private process definition for each instance.

- *Instance migration.* Most workflow management systems only provide a process definition at the schema level. If process definitions are at a type level, instance migration becomes relevant. Instance migration is concerned with transferring a workflow instance from one workflow process definition to another. Instance migration is far from trivial because the state of the instance in the old workflow may not correspond to any of the states of the new workflow. As shown in [Ellis et al. (1995)], the so-called "dynamic change bug" can occur. The dynamic-change bug refers to errors introduced by migrating a workflow instance from an old process definition to a new one, which can lead to undesired situations like duplication of work, skipping of tasks, deadlocks, and livelocks.

2.3. Analysis

Workflow management systems are typically used to improve mission critical business processes of an organization. Hence, the correctness, effectiveness, and efficiency of business processes supported by the workflow management system are vital to the organization. A workflow process definition, which contains errors, may lead to angry customers, backlog, damage claims, and loss of goodwill. Flaws in the design of a workflow may also lead to high throughput times, low service levels, and a need for excess capacity. This is why it is important to analyse workflow process definitions before putting them into production. Basically, there are three types of analysis: validation, i.e., testing whether the workflow behaves as expected, verification, i.e., establishing the correctness of a workflow, and performance analysis, i.e., evaluating the ability to meet requirements with respect to throughput times, service levels, and resource utilization [Aalst (1998), Sheth et al. (1999)].

- *Validation* is mainly concerned with the gap between the specified workflow and the intended workflow. Validation needs to be done by domain experts and analysis is context dependent. Workflow management systems can provide simulation and animation tools, which allow the easy tracing of processes with the aim of detecting errors.

- *Verification* is concerned with the logical correctness of workflow process definitions. Depending on the workflow language used, there may be different properties, which have to be satisfied. Today's workflow management systems only support some syntactical checks, i.e., workflow processes with potential deadlocks and never-ending loops can be put into production without any warnings at design time. Techniques such as model checking and structural analysis based on the graph structure can be used to detect inconsistencies.

- Well-established techniques for *performance analysis* are simulation and queuing theory. Both types of techniques can be used to detect potential bottlenecks. A prerequisite for this activity is

that durations for the execution of tasks and the arrival pattern of new cases are known. Together with the role information and information on the number of persons able to fill these roles in specified time intervals, which can be viewed as the resources available, the system can detect bottlenecks. For business reasons it is of vital importance to detect potential bottlenecks before the system goes operational.

3. Recent Workflow Modelling Approaches

This section introduces recent approaches to modelling dynamic behaviour of application systems in general and workflows in particular. Starting with the Unified Modelling Language (UML) [Rumbaugh et al. (1999), Rational (2000)], a general-purpose object modelling language is investigated with respect to the capabilities it provides for process modelling. The Object Coordination Net approach uses a Petri net formalism to add process-modelling capabilities to UML [Wirtz et al. (2000)]. Workflow graphs can be regarded as the traditional approach to specify workflows; different flavours of workflow graphs are supported by commercial workflow management systems (e.g., IBM MQSeries Workflow [IBM (2000)] as well as university prototypes, e.g. WASA [Weske (2000), Weske (2001)]). WorkFlow Nets are Petri nets tailored towards modelling workflows. Based on a strong theoretical foundation, they focus on properties of workflows and the application of inheritance concepts.

3.1. Unified Modelling Language

In the last few years, the Unified Modelling Language (UML) has gained enormous attention in the software engineering area as the *de facto* standard for modelling during object-oriented analysis and design. There are two main reasons for this success. Firstly, the extensive usage of a rich set of visual formalisms, which are generally assumed to be easy to use and, more importantly, to communicate among customers, designers and developers. Secondly, the standardization efforts that resulted in the UML notations were an answer to the highly pressing needs of the software industry, especially for tool builders. Although the first UML versions had a number of deficiencies, the insight that the UML notations are useful for business process modelling and, hence, could also be adopted to the workflow area were present right from the beginnings of the UML; see [Hruby (1998), Wiegert (1998)] for an in-depth discussion.

The importance of the UML for workflow modelling is due to the fact that the early analysis and design phases of modern software engineering, e.g., obtaining relevant use cases, are in fact closer to business process modelling than to programming. Moreover, the operational perspective of workflows is situated in a context, which is governed by legacy applications that have been developed using OOA (Object-Oriented Analysis)/OOD (Object-Oriented Design), maybe even tools based on the UML. Using similar techniques simplifies the integration of applications into the workflow context. Details from the information perspective like, e.g., basic data types and classes used to model production data, may be reused without change of input and output parameters of applications.

Most recent approaches to model workflows with the UML are grouped around only a few kinds of UML diagrams. Use-case diagrams are well suited to describe roles (like customer or clerk), involved in a business process, their interaction via top-level processes as well as the relationships between different use-cases. A use-case may involve other use-cases (<<include>>), build upon others (<<extend>>) and so on. In this manner, a coarse-grained model of all relevant top-level processes—called the use-case model—is obtained that can be refined afterwards by describing the different processes in more detail. This is done using four types of UML diagrams: sequence diagrams, collaboration diagrams, statechart diagrams, and activity diagrams. UML statecharts are based in the process modelling technique introduced in [Harel (1987)]. Statecharts are useful to describe the life-cycle of a specific system or subsystem in a reactive way. UML sequence diagrams, also referred to as message sequence charts, are used to visualize specific instantiations of a use-case. With the UML Version 1.3, activity diagrams, which are a combination of statecharts and Petri nets, have been made more expressive to match most of the needs of modelling the process perspective of workflows: Start and end states for complex processes including sequential, alternative and parallel routing can be

extended by object flow using different kinds of arcs for routing control flow and data flow. The different roles involved are visualized by so-called swim lines, which partition the diagrams from top to bottom into areas where specific system parts take the responsibility for all actions occurring in their partition. Pre- and post-conditions from the process perspective can be specified by adding Object Constraint Language (OCL) expressions, which allow for time constraints and so on. Extensive use of OCL expressions, however, puts important information into text and reduces the benefits of the UML as a visual notation. Besides activity diagrams, a more structure-oriented view is supported by collaboration diagrams, which visualize the flow of control through a numbering scheme in the context of classes and their static relations in structure diagrams. The numbering scheme as well as the swim lines in activity diagrams have the major drawback that they do not scale for real-life complex processes involving more than up to, e.g., 5-6 roles or nested parallelism. The description of a method for business process modelling along the lines sketched above that is supported by a powerful toolset can be found in [Rational (2000)]. The method supports a set of predefined UML stereotypes that is part of a specific UML profile for business modelling.

The UML has much more potential regarding the organizational perspective but almost all approaches developed so far ignore the chances of OOA and OOD for structural modelling using packages, subsystems and strict interface-based interaction in the context of workflow modelling with the UML (as detailed in the next section). Closely related to this observation is the lack of an adequate resource concept. Most of these deficiencies have been recognized by the UML community, and recent OMG Request for Proposals [OMG (2000a)] or submissions to the committee [OMG (2000b)] discuss the abovementioned aspects like resource assignment and organizational structure. Moreover, a new set of consistent UML extensions for workflow process definition in general are under development [OMG (2000c)].

While the UML provides some mechanisms to model workflows, flexibility is hard to achieve in the current situation, given the UML semantics. In general, workflow instance migration may be possible through advanced typing and type changes at runtime (facilitated by casts) but instance change does not fit well into the class-instantiation context of object-oriented modelling. Even migration or other forms of constrained flexibility, which should be based on inheritance notions respecting behaviour, are hard to implement in a context where diagrams essential for the modelling process have no clear formal semantics in isolation or in overall combination, as shown in [Giese et al. (1999)]. For the same reason, analysis in the sense of formal verification is hopeless but validation through test cases and simulation, e.g., generating message sequence charts for typical system runs that is quite standard in software development can be used in the workflow context, too.

To illustrate the use of UML for workflow modelling, we use the process of ordering a book from an electronic bookstore. We will use this process to illustrate the four types of UML diagrams mentioned before (sequence diagrams, collaboration diagrams, statechart diagrams, and activity diagrams). We will also use this example to illustrate the other approaches.
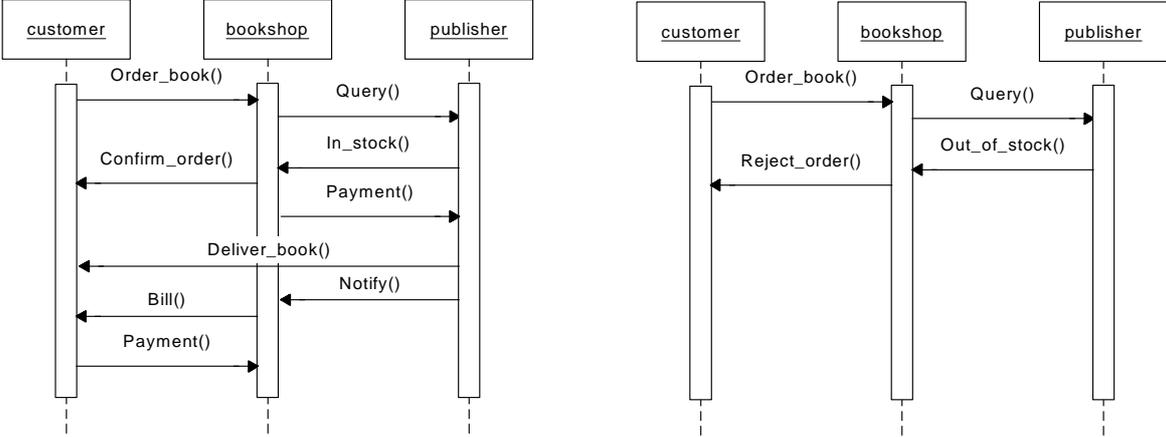

Figure 1: Two UML sequence diagrams.

Figure 1 shows two sequence diagrams. The diagram on the left-hand-side models a scenario which corresponds to a customer successfully ordering a book. The right-hand-side diagram models the scenario where a customer order is rejected because the ordered book is not in stock. A sequence diagram shows for each object or actor a so-called lifeline. In both diagrams shown in Figure 1 there are three lifelines: the *customer* lifeline, the *bookshop* lifeline, and the *publisher* lifeline. Time is increasing along each lifeline from top to bottom. A sequence diagram also shows the messages exchanged. Consider for example the left-hand-side diagram. First, the customer orders a book by sending the message *Order_book*. Then, the (on-line) bookshop sends a query to the publisher to see whether the book is available (message *Query*). The publisher responds by sending the message *In_stock* indicating that the book is available. The bookshop confirms the order (message *Confirm_order*) and pays for the book (message *Payment*). After receiving the payment, the publisher sends the book to the customer (message *Deliver_book*) and notifies the bookshop (message *Notify*). Triggered by this notification, the bookshop sends a bill (message *Bill*) and the customer pays for the book (message *Payment*).

Note that the left-hand-side diagram does not specify a process but merely one scenario. This scenario corresponds to handling a customer order successfully. If the book is not in stock, the diagram on the right-hand-side applies. In the second scenario, the book is not available (message *Out_of_stock*) and the customer order is rejected (message *Reject_order*). Figure 1 illustrates that sequence diagrams can only be used to model scenarios and are not suitable for making full-fledged process models. The basic sequence diagram has no provision for routing constructs such as choice, synchronization, iteration, etc. Sequence diagrams have been extended with features to handle these routing constructs. However, these extended diagrams become difficult to read and difficult to interpret.
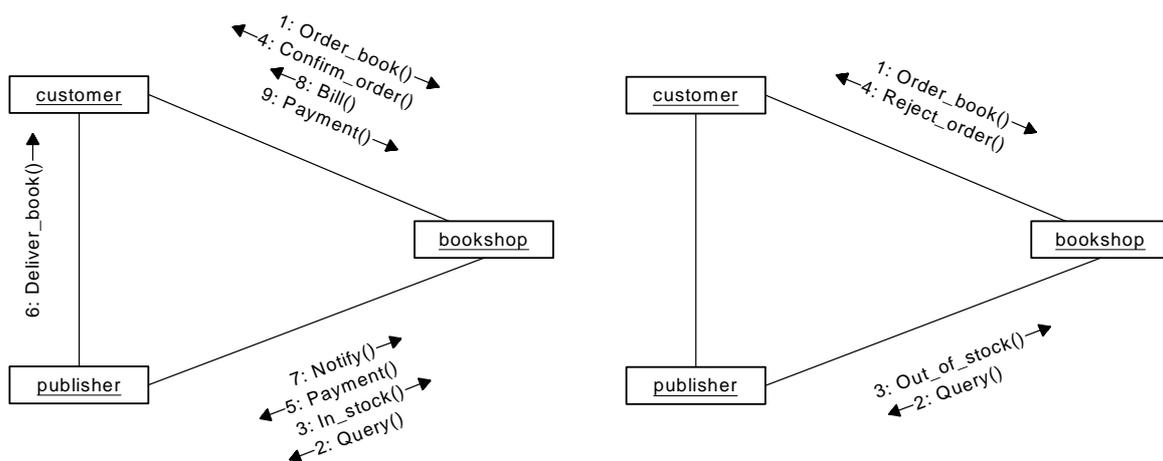
Figure 2: Two collaboration diagrams.

A collaboration diagram highlights the organization of objects that participate in an interaction. Compared to sequence diagrams the emphasis is shifted from temporal relations to organizational relations. From a semantic point of view collaboration diagrams and sequence diagrams are interchangeable, i.e., semantically equivalent. The lifelines are replaced by numbered sequences. Consider Figure 2. The two collaboration diagrams correspond to the two sequence diagrams shown in Figure 1. One can translate a sequence diagram and translate it to a collaboration diagram without any loss of information (and vice-versa). The order of the messages exchanged is captured by a numbering scheme. The numbers in Figure 2 indicate the order in which messages are exchanged among the customer, bookshop and publisher. Collaboration diagrams can be extended with more complex constructs such as nesting, iteration, and branching. However, just like sequence diagrams, collaboration diagrams are particularly suited for modelling scenarios, i.e., examples of straight sequential flows of control. For true process modelling one should use statecharts diagrams or activity diagrams.

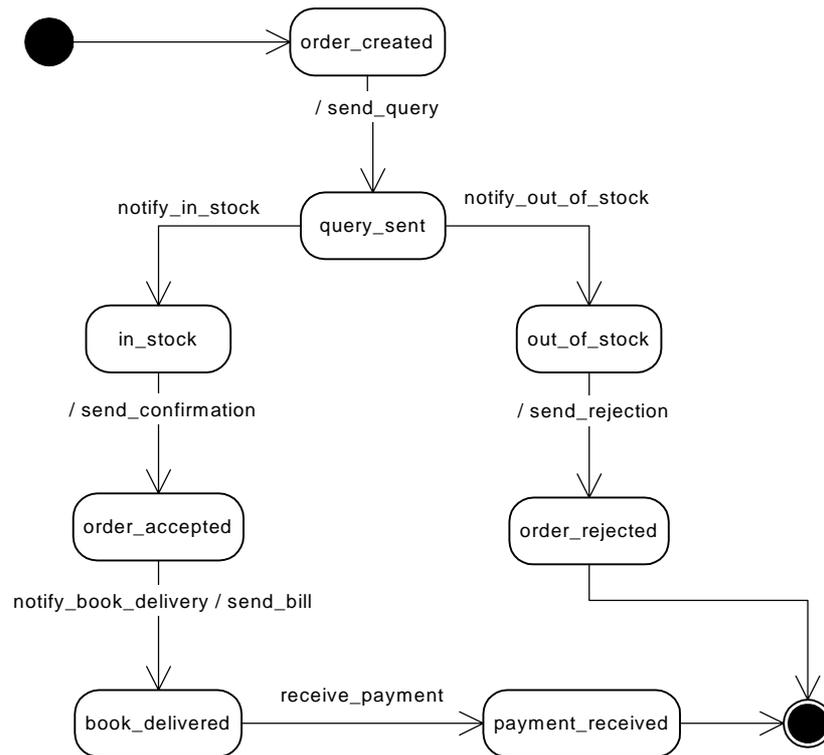**Figure 3: A statechart diagram describing the life-cycle of one order.**

Statecharts are an extension of basic state machines. A basic state machine consists of states and transitions. At any point in time, the system (or object) resides in one of these states. A transition moves the system from one state to another. The basic state machine corresponds to the class of Petri nets where each transition has one input and one output place. In a statechart diagram one can have composite states, orthogonal regions, variables, events, conditions, and actions. Composite states can be used for nesting. Orthogonal regions can be used to model parallelism. Transitions can be augmented with so-called ECA (Event-Condition-Action) rules. This means that a transition only takes place when a specified event occurs and a condition is satisfied. Both the event and condition are optional. It is also possible to add an action to a transition. This means that the action is executed the moment the transition takes place. The standard notation for these ECA rules is "*event [condition] / action*".

Figure 3 shows a very simple statechart diagram. This statechart models the lifecycle of an order. The initial state is modelled by a black dot. The final state is modelled by a black dot within a circle. A state is modelled by a rounded rectangle. Transitions are modelled by arcs. The transition connected to the states *order_created* and *query_sent* generates the action *send_query*. In state *query_sent* two potential transitions are enabled. One of them is triggered by the event *notify_in_stock* and leads to state *in_stock*. The other one is triggered by the event *notify_out_of_stock* and leads to state *out_of_stock*.
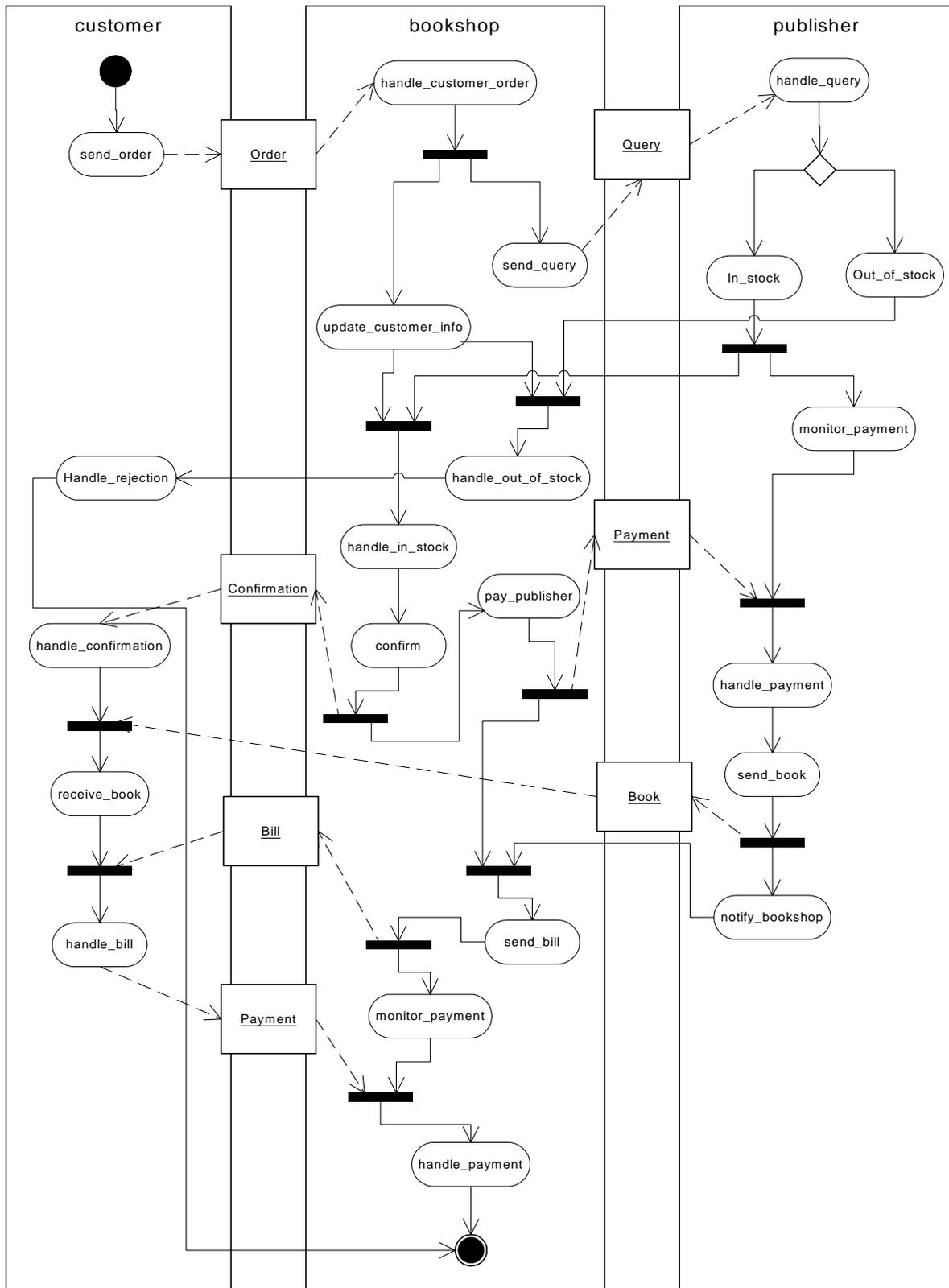
Figure 4: An activity diagram describing the whole process.

Statecharts are well-suited for modelling the lifecycle of one object. Unfortunately, statecharts are less suitable to model the control flow among objects. For this purpose UML offers activity diagrams. Activity diagrams are close to the workflow languages discussed in this paper. Therefore, it is no surprise to see that activity diagrams are used for enterprise modelling, workflow modelling, and business process reengineering. Consider Figure 4. This activity diagram models the process

illustrated by the two sequence/collaboration diagrams. The diagram is divided into three main parts: customer, bookshop, and publisher. These parts are called *swimlanes*. A swimlane specifies a locus of activities and is particularly useful for business modelling. Using swimlanes it is possible to partition the process into roles or organizational units. Please note that most of the other techniques discussed in this paper can be extended with swimlanes. Just like in a statechart diagram the initial and final state are indicated using black dots. Activities (also called activity states) are denoted by rounded rectangles. Solid lines correspond to control flow. Dashed lines correspond to object flow. The objects passed are modelled by rectangles. Consider for example the upper left corner of the activity diagram. Starting in the initial state the activity *send_order* is executed. After execution of *send_order* an object *order* is passed on to the bookshop which executes *handle_customer_order*. The thick horizontal lines in Figure 4 correspond to *synchronization bars*. A synchronization bar is either a *fork* or a *join*. Forks correspond to AND-splits. Joins correspond to AND-joins. An exclusive OR-split is modelled by a so-called *branch* and is depicted by a diamond. The diamond symbol can also be used to model OR-joins. The activity diagram shown in Figure 4 has one branch. This branch makes the process dependent upon the availability of the book ordered by the customer. The remainder of the process is self-explanatory.

3.2. Object Coordination Nets

The Object Coordination Net (OCoN) approach has been developed originally for the area of object-oriented distributed software systems design [Wirtz et al. 1997]. Many aspects of the approach are inspired by the idea to provide an OOA/OOD-based method and language that fits well into the context of distributed and parallel software development. The approach uses structural modelling techniques (subsystems, interfaces, classes, relations among structural entities) from the UML, but puts much more emphasis on separation of concerns through a strict abstraction discipline. Abstraction is supported by extending UML interfaces to become contracts [Meyer (1996)], which describe the allowed behaviour and the intended usage restrictions of interface operations in a visual manner. The consistent combination of structural and behavioural modelling is used to introduce a notion of resources that is needed in the distributed software area as well as in workflow management. Moreover, severe technical deficiencies, especially the lack of clear guidance regarding the integrated meaning of different diagrams in a UML model and the absence of an overall consistent semantic model [Giese et al. (1999)] have led to the usage of a specific kind of high-level object-oriented Petri nets instead of the various UML interaction diagrams in combination with UML structure diagrams. These nets are used for specifying the behaviour of interface contracts, the overall handling of resources, and the detailed flow of control and data for processes on all system levels. OCoNs are capable of describing all typical control flow situations occurring in workflows [WfMC (1997), Aalst, Hofstede et al. (2000)].

Object Coordination Nets represent activities by net transitions, which are called actions. An action on the system level may represent a complete top-level workflow, on subsystem level sub-workflows performed by a specific part of the overall system or (atomic) calls to an application. We interpret actions of all levels as services that are provided by entities of a specific level. Hence, the different levels of functionality available in a system are described with the same mechanism. Actions may only fire if all pre-conditions specified in the net are fulfilled. These pre-conditions are either typed input parameters and events representing the flow of control and data through the net or an obligatory unique carrier of activity, i.e., the resource responsible for executing that action. Such a resource corresponds at the most detailed level to the instance of a class that provides the service *s* as a method *self.s(...)*. At higher levels of abstraction, a resource represents an instance of the (implementation of a) subsystem that offers the corresponding service in its contract. Preconditions are visualized by arcs pointing from a place or resource to an action. In contrast to many other net models, e.g., [Brauer et al. (1987)], firing an OCoN transition consumes time and is more like a call to a (remote) procedure in taking three steps: synchronous consumption of input parameters, internal processing and synchronous producing of the resulting output parameters for the post-condition resource and output places. The output places or post-conditions are visualized by arcs pointing from the action to the corresponding places. The internal processing may include further hierarchical calls to other services using more resources and so on. Resources may be used in an exclusive or shared manner, which allows for a

detailed resource usage and dependency model of the entire system even in the case of parallel service requests. The usage of services and, hence, resources has to be compatible with the export and import of interfaces by subsystems in the structural model, i.e. all possible dependencies are already visualized in the organization perspective. The external and internal resources of a subsystem are not managed by the single service calls, i.e., actions but by a subsystem-global resource allocation net that receives all calls to the subsystems interfaces, provides the needed resources for each service and delegates the call. Working in this way, it schedules the combined resource usage of all it's services and provides a well-defined point-of-control for analysing the resource situation of a subsystem or even the entire system.

A typical modelling process starts with a use-case driven analysis to get an insight into the needed functionality, involved partners (e.g., companies, departments, and specific roles). Use-cases of this type can be described by UML use-case diagrams, which are refined through Object Coordination Nets instead of activity diagrams. If already known and intended, responsibilities are assigned to actions by stating which system resources (represented by resource pools that correspond to subsystem or class interfaces) should perform a specific action. Combined with knowledge about the problem domain and the organizational structure of the involved organizations, this provides the knowledge for a first coarse-grained specification of the organizational perspective. This structure is described by UML structure diagrams using subsystems, which are related by providing and importing interfaces, classes with operations and attributes, inheritance, associations and so on. Typically, analysis classes are obtained for the most important entities flowing through the system during this process, too. These provide the basic knowledge for describing the informational perspective. The more fine-grained functional perspective as well as the process perspective are modelled afterwards in the context of the structural environment specified so far. This can be done top-down, bottom-up or in a mixed style. A top-down approach refines the already obtained informal nets describing top-level use-cases step-by-step until detailed workflow schemata depending only on available functionality (leaves) are reached. In some situations, working bottom-up is much more appropriate: powerful application software or already available sub-workflows from existing subsystems may provide a rather high-level functionality and may rule the design decisions. These system parts are encapsulated into subsystems with interfaces that export all intended functionality. If parts of the provided functionality require specific execution orders or may not always be available, the syntactical operation and parameter information is not sufficient to provide enough information for a secure use. In this case, state-machine-like OCoNs are used to describe the needed application rules, which may be as detailed as prescribing the permitted operation orders as set of legal sub-workflows. In this manner, the integration of already working parts of an organization during a re-structuring process, new or already available legacy code as well as external functionality, i.e., external sub-workflows provided by other companies in B2B workflow systems, can be integrated as subsystems that are used according to the rules stated in their interface contracts. Usually, the described procedure will be an iterative process obtaining more and more detailed information about workflows and structural information.
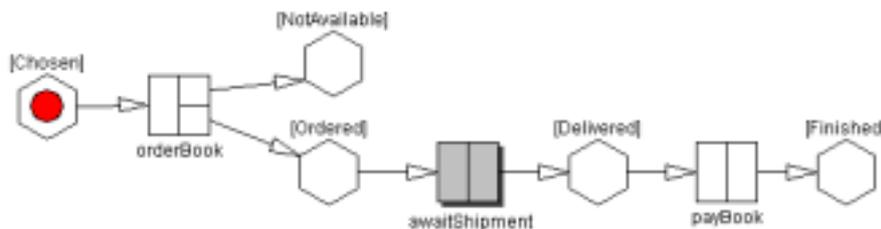
Figure 5: Ordering a book from an abstract Users view

Figures 5-7 present the usage of OcoNs in the bookstore example at different abstraction levels. Fig. 5 describes the possible states (hexagons), permitted operations (boxes) as well as the required execution order for ordering an already chosen book. The *orderBook* call may be successful or not, which is described using a transition with alternative outputs, and, hence ends in one of the states [*NotAvailable*] or [*Ordered*]; in the latter case, the customer is assumed to await the shipment and is obliged to pay for the book afterwards. Only after finishing this step, the entire process is finished, too. The chosen view abstracts completely from explicit notifications and the details of shipment.

The details of *orderBook* are somewhat more complex from the *BookShops* view. Figure 6 models the handling on a level of detail that is close to an object-oriented programming language. A real call of *orderBook* provides information about the order as well as the calling customer by means of parameters (circles in the bar, left-hand-side), uses this information and the *BookShop*'s customer management resource to update the customer profile, and calls *Publisher.query* (yellow transition) for one of its external publisher resources to obtain the book in parallel. As shown in the details of the *Publisher::query* service in Figure 7, this may end without success because the book requested is out of stock, the entire orderBook sub-process is terminated and the order ends up in state [*NotAvailable*] (Fig. 5). Otherwise, a lot of work remains to be done in the *BookShop* subsystem.
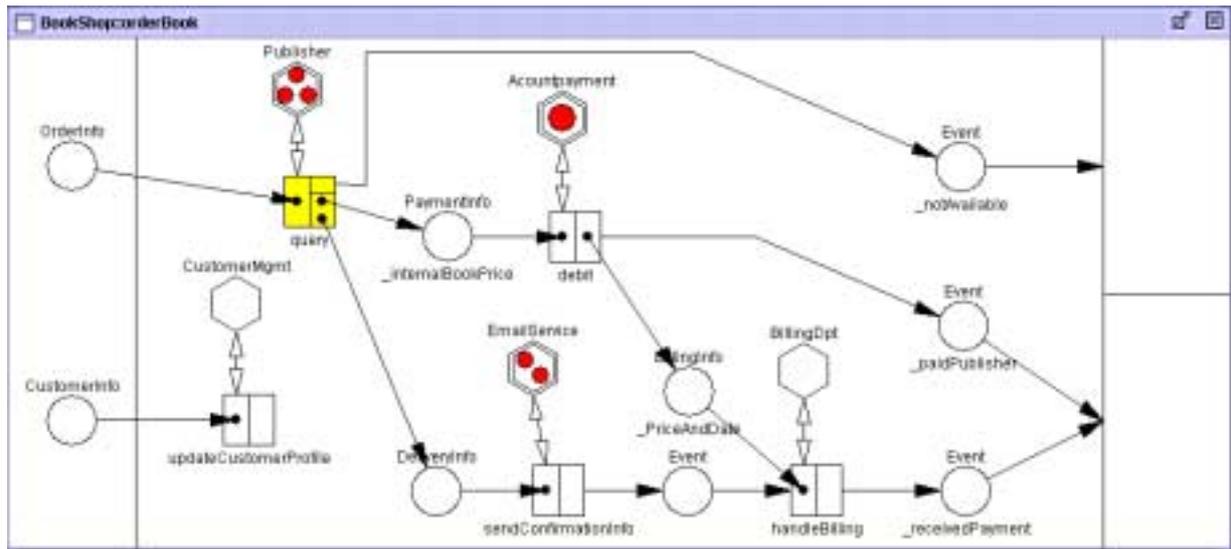
Figure 6: The detailed handling of an order in the BookShop

An external *Accountpayment* resource is used to pay the publisher, a confirmation is sent to the customer using a public email service and the internal billing department is used to delegate the details of invoicing and payment checking. Only after all these steps have been performed, both precondition event pools for finishing *orderBook* are filled and the process is able to terminate with it's lower alternative (bars in the right-hand-side). This kind of delegation to other subsystems acting in specific roles can be used for simple stateless operation calls as well as for more advanced usage protocols. A more complicated handling, for example, is required when using the *ShipmentDpt* of the publisher (Fig. 7) because due to a lack of trust between the publisher and the bookshop, the book is only sent to the customer iff the bookshop has already paid for the book. Hence, the interface contract for using the shipment department describes a state-based behaviour that allows only for the sequence of steps described in its so-called protocol-net. The first part of this work, i.e., *prepareShipment* is done in the query operation itself, which works only in the initial state [*Ready*] and changes the protocol state to [*Prepared*]. The remaining parts of handling the shipment can be performed as follows: the *BillingDpt* confirms the shipment after payment is received, which brings the state to [*Confirmed*]. Afterwards, the real shipment, e.g. using an express service or postage can be performed.

Although only a small part of the entire system model has been presented here, the principle of state-based interaction and the abstraction obtained by breaking a complex system into subsystems offering and using contracts first in the structural description and using these interfaces as resources afterwards when describing the behaviour should become clear. Note, that the level of detail in describing the different sub-processes may vary in a wide range. Entire subsystems modelling, for example, an external business partner may be reduced to state-based or even stateless contracts on the one hand, whereas the details of gluing fine-grained application programs together in a visual script-like but type-safe manner can be described for other parts of the same system. Hence, OCoNs provide a means for describing the process and functional perspective of workflows embedded in their organisational and informational context in a seamlessly integrated way.
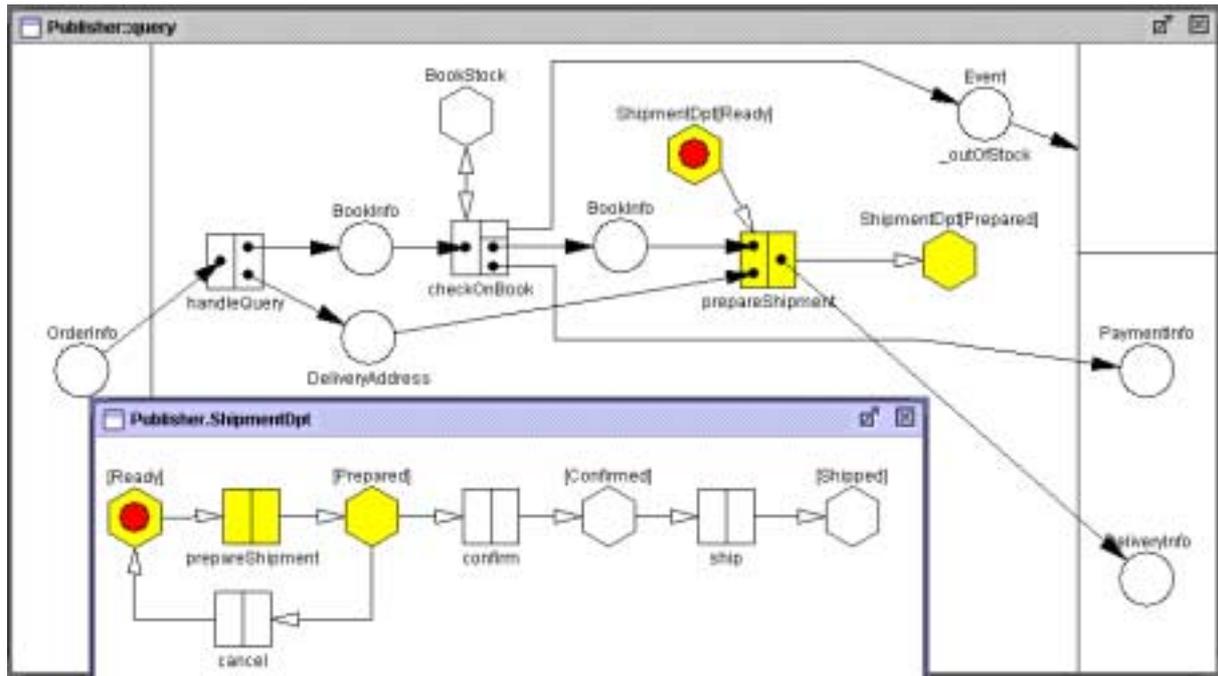
Figure 7: Processing the query at the Publisher site using the ShipmentDpt protocol

Flexibility of workflow systems is provided at different levels. Regarding the structural context and the organizational, informational and operational perspectives, strict encapsulation of subsystems, which permits dependencies through explicitly defined interface-usage only, makes the incorporation of new subsystems or the exchange of subsystems or applications a manageable task that even allows for cost estimations. Changing active workflow processes via instance change is not possible but a controlled change on the level of a subsystems contract is supported by an inheritance notion based on behavioural properties for contracts [Giese (2001)].

The process is supported by an editor for OCoNs and a simulator that allows for an early evaluation of a design obtained at a specific point during development [(Giese et al. 2000)]. OCoNs have a clear formal semantics [Giese (2001)] and can be mapped to the proposed standard for high-level Petri-Nets [Jensen (1992)]. Hence, some of the tools available for analysing Petri nets can be made available for this approach. Whereas this is true for interfaces, their usage through embedding and their direct implementation, i.e., for protocol nets and resource allocation nets, an efficient and complete analysis of complicated hierarchical high-level nets is not feasible.

Due to their strong interface concept including behaviour and the explicit resource model, the OCoN workflow specification approach is especially well suited for workflows that are implemented in complex distributed environments and B2B systems involving more than a single organization. For the latter application, the contract-centred approach provides a well-defined specification for an analysis of the complex interrelationships caused by the B2B interaction.

3.3. Workflow Graphs

This section introduces workflow graphs developed in the context of the WASA project [Weske (2000)]. In this approach, workflow process definitions are based on directed graphs, whose nodes represent workflows and whose edges represent constraints between workflows. Workflows can be atomic or complex; while atomic workflows (or tasks) do not have an internal structure, complex workflows consist of a set of workflows, each of which can be atomic or complex, resulting in a hierarchical structure. The terms sub-workflow and super-workflow are used to refer to the relative position of workflows in the hierarchy. Each workflow can have relationships with their respective sibling workflows, for instance execution order constraints and data dependencies, which are covered by the process perspective and the information perspective, respectively. In workflow graphs, there are

two language constructs to define the process perspective: Control flow constraints and start conditions. While control flow specifies execution order, the conditions under which a given workflow instance is executed are defined by its start condition. By evaluating the start condition, the system can decide whether the workflow has to be executed in a particular case. Workflow control data as well as application data can be used In start conditions. For instance, a CreditRequest workflow may have two sub-workflows for different credit amounts requested. If the amount is less then a defined threshold then workflow A will be performed, otherwise B will be executed. In this example the start condition of A will check whether the amount requested is less than the threshold, and the start condition of workflow B will check if the amount requested exceeds the threshold.

Loops in workflow graphs are not permitted. This is due to the fact that workflow execution control is based on a technique called dead path elimination [Leymann, Altenhuber (1994)], which allows the compact representation of processes. By that we mean that only at runtime it is decided whether a branching is an alternative or a parallel execution, depending on the start conditions of the workflows involved. To compensate for the lack of loops, WASA supports recursive workflow schemas, i.e., a workflow schema can be a sub-workflow of its own. Recursion has the same expressive power as loops, and hence they can be used to simulate cyclic structures in workflow schemas. However, it is often more natural to specify loops rather than to specify recursive workflows. This limitation of workflow graphs can be overcome by a suitable front-end, which allows the specification of loops. In a next step, the cyclic structure can be translated to the recursive structure, which is understood by the workflow system.

The organizational perspective is covered by roles and agents, which are selected during role resolution to perform tasks. The role concept is a flexible mechanism to represent a variety of organizational structures as well as advanced concepts, for instance delegation of work between persons and case-dependent role resolution, meaning that application data can be used in role resolution. A popular example in this context is granting request for vacation: A department chair is allowed to grant the requests of all members of the department, except her own.

The information perspective is based on class definitions, objects, and parameters. In particular, each workflow is assigned a set of typed input parameters and a set of typed output parameters. Parameters are typically objects, for instance a credit request form. When starting a workflow, the input parameters of the workflow are read; on its termination, the results are written into the workflow's output parameters. Data dependencies between workflows are specified by data flow, i.e., mappings between input and output parameters of workflows. Data flow can be defined between sibling workflows and between a complex workflow and its immediate sub-workflows. Due to the direction of the data transfer, the kinds of data flow are called horizontal and vertical, respectively.

Flexibility issues in general and dynamic adaptation of running workflow instances in particular have been among the key issues in the WASA project. We now sketch the formal foundation and conceptual design of dynamic adaptations; details can be found in [Weske (2000), Weske (2001)]. In the formal model, workflow schemas and workflow instances are represented by graphs. For each workflow schema graph, there may be an arbitrary number of workflow instances graph, which are associated to the workflow schema graph. For example, the workflow instances CreditRequest("Smith", 50.000) and CreditRequest("Jones", 75.000) are two workflow instances, associated with the workflow schema CreditRequest. Notice that associations apply both to complex workflow schemas and to tasks. Now assume that the structure of the business process has to be changed, for instance to cope with a new market situation. In this case, the respective workflow schema is modified and stored as a new version, let this be CreditRequest1. One can think of the new workflow schema as an improvement of the original one, which aims at improving customer satisfaction or enhances throughput. In large banks, at each instance hundreds or thousands of CreditRequest workflow instances are running. Assume that from a business point of view it is desirable that all new workflow instances but also all currently active workflow instances should make use of the improved workflow schema. It is easy to see that all future workflow instances can use the new workflow schema. How about the active ones? It is obvious that not all of these workflow instances can use the new workflow schema, since the modification may affect parts of the workflow, which were already executed. Hence, the system has to decide for which of the active workflows the adaptation to the improved workflow schema is possible and should be performed. In WASA, this decision is based on the existence of a valid mapping. A valid mapping relates sub-workflow instances

of the active workflow with sub-workflow schemas of the new workflow schema. Loosely speaking, a valid mapping exists if the active workflow can be continued in a way that complies to the improved workflow schema. We remark that the notion of valid mappings and the meaning of "complies" are formalized in [Weske (2000)].

The concepts are implemented in a prototypical system, which is based on object technology as far as design and implementation is concerned. The most prominent feature of the design in the context of dynamic adaptations is the representation of both workflow schemas and workflow instances by objects, as opposed to representing workflow schemas by classes and workflow instances by objects of these classes, which results in complex class evolution and class migration issues on the technical level. Using this design decision, dynamic adaptations can be performed by changing the association of a workflow instance object, once a valid mapping was found. In particular, the association of that object with the original workflow schema object is purged, and an association with the improved workflow schema object is created. It is obvious that additional steps have to be carried out on the implementation level, for instance purging no longer required workflow instance objects and creating new ones. For details of the implementation, the reader is referred to [Weske (2000)].

Syntactic correctness of workflow schemas is defined by a notion of consistency. In particular, workflow schemas have to satisfy completeness of data flow (it must be possible to fill input parameters), type compatibility (data connectors must link type compatible parameters), data availability and acyclic control structures, as discussed above. Verification and validation in this approach is in the responsibility of the workflow modeller. However, workflow management systems should provide powerful tools to animate and simulate workflows.

Due to the object-oriented approach, the WASA system is well equipped to support modern business applications, which are based on business objects. In particular, business objects can be transferred by data flow, and tasks can make use of business objects methods during task execution. This means that depending on the particular business object transferred to a task at runtime, a specific implementation of a method (carried by the business object) is invoked. We mention that rather than transferring large business objects it suffices to transfer object identifiers, which can then be accessed via the object middleware.

Figure 8: A Workflow Graph Describing the Sample Process.

15

Figure 8 illustrates workflow graphs by specifying the above mentioned sample process involving customers, a bookshop and a publisher. Each organizational entity involved is characterized by a rectangle in which the activities of that particular entity appear. Activities are ordered by control flow; not to overload the example, data flow is eliminated from the workflow schema. Transition conditions which mark control flow edges are an important aspect in workflow graphs. Transition conditions are evaluated if and when the source node of the corresponding control flow edge completes. Typically the result value of that activity is used to evaluate the transition condition. If no such condition is provided for an edge then *true* is assumed. This means that multiple outgoing edges correspond to a AND split, i.e., the follow-up activities will be executed concurrently. To this end, transition conditions can be used to describe AND-split (Boolean constant *true* on all outgoing edges), OR-split (C and not C as transition conditions) and other forms in a flexible way.

The customer starts the interorganizational process by sending an order (*send order*) to the bookshop, nowadays typically implemented by filling a web form and submitting the order. In the side of the bookshop, receiving an order triggers the process as specified. Customer information is updated and concurrently a query to the respective publisher is sent out to check for availability of the ordered books. When the publisher responds, the information is passed to the customer in the *send status* activity of the bookshop workflow graph. Depending on the availability—indicated by the transition condition *avail*, which evaluates to *true* if and only if the book is available—, the process at the publisher, bookshop, and customer are terminated. Of course, this procedure represents a simplification. However, more sophisticated mechanisms can also be specified using the workflow graph formalism. The continuation of the process is immediate from the figure.

3.4. WorkFlow Nets

Petri nets have been proposed for modelling workflow process definitions long before the term "workflow management"' was coined and workflow management systems became readily available. Consider for example the work on Information Control Nets, a variant of the classical Petri nets, in the late seventies [Ellis (1979)]. Petri nets constitute a good starting point for a solid theoretical foundation of workflow management. Clearly, a Petri net can be used to specify the routing of cases (workflow instances). *Tasks* are modelled by *transitions* and *causal dependencies* are modelled by *places* and *arcs*. In fact, a place corresponds to a condition which can be used as pre- and/or post-condition for tasks. An AND-split corresponds to a transition with two or more output places, and an AND-join corresponds to a transition with two or more input places. OR-splits/OR-joins correspond to places with multiple outgoing/ingoing arcs. A Petri net which models the control-flow dimension of a workflow, is called a *WorkFlow net* (WF-net) [Aalst (1998)]. A WF-net has one source place and one sink place because any case (workflow instance) handled by the procedure represented by the WF-net is created when it enters the workflow management system and is deleted once it is completely handled, i.e., the WF-net specifies the life-cycle of a case. An additional requirement is that there should be no "dangling tasks and/or conditions", i.e., tasks and conditions which do not contribute to the processing of cases. Therefore, all the nodes of the workflow should be on some path from source to sink.

Figure 9 shows a WF-net using some graphical "sugaring", i.e., the meaning of the symbols is explained in the right-bottom corner. The WF-net corresponds to the process described in the UML activity diagram shown in Figure 4 and specifies the handling of orders for an electronic bookstore. Cases start in the place labelled *start*. Each case corresponds to one or more tokens. Initially there is one token in *start*. After executing task *send_order*, place *order* is marked and *handle_customer_order* becomes enabled. Note that send_*order* consumes one token (from place *start*) and produces one token (for place *order*). Task *handle_customer_order* is an AND-split and its execution enables two parallel tasks: *update_customer_profile* and *send_query*. Using the simple rule of task enabling (a task is enabled if each/any of its input places is marked) and task execution (firing a task results in the consumption of the tokens needed to become enabled and the production of tokens for the output places), Figure 9 unambiguously specifies the workflow process. Therefore, the rest of the diagram is self-explanatory. Compared to the activity diagram shown in Figure 4, two major

differences can be noted. First, states are made explicit through the representation of tokens in places. Second, the formalism does not leave room for multiple interpretations.
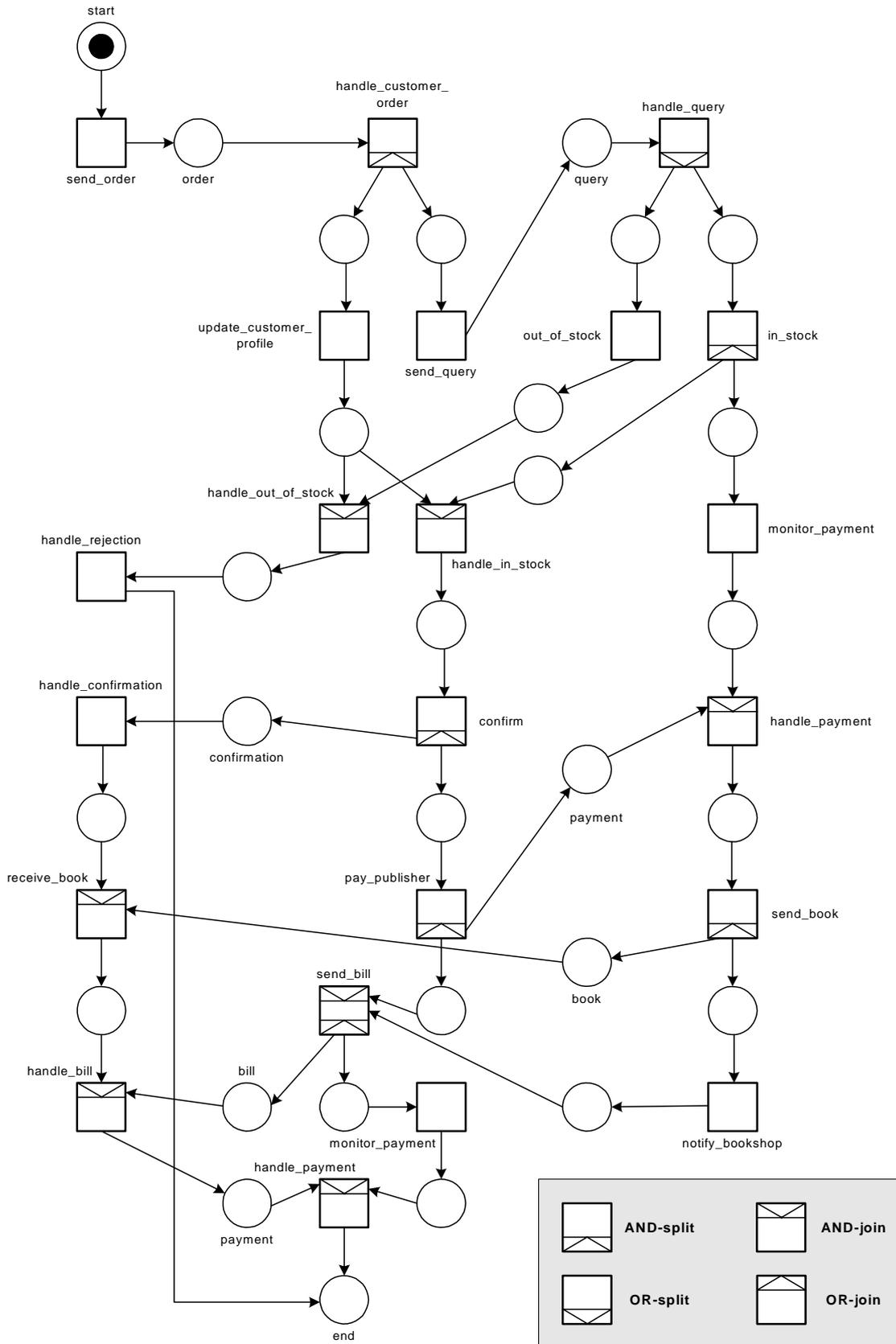

Figure 9: A WF-net describing the processing of orders for an electronic bookstore.

The WF-net focuses on the process perspective and abstracts from the functional, organization, information and operation perspectives. These perspectives can be added using for example high-level Petri nets, i.e., nets extended with colour (data) and hierarchy. Although WF-nets are very simple, their expressive power is impressive. WF-nets can be used to model the basic constructs identified by the Workflow Management Coalition [WfMC (1997)] and used in contemporary workflow management systems. Moreover, WF-nets support constructs which are often needed but seldom supported by commercial systems. In [Aalst et al. (2000)] we describe several workflow patterns (e.g., the deferred choice and milestone) supported by WF-nets but not by leading workflow management systems such as Staffware and IBM's MQ Series. One of the features of WF-nets enabling more advanced routing constructs is the explicit representation of states through tokens in places.

Although WF-nets by themselves do not offer specific mechanisms for supporting workflow flexibility, relevant results have been obtained using inheritance of WF-nets [Aalst, Basten (2002)]. Inheritance is one of the key concepts of object-orientation. Classes and objects in object-oriented design correspond to workflow process definitions and cases in a workflow management context. In object-oriented design, inheritance is typically restricted to the static aspects (e.g., data and methods) of an object class. For workflow management, the dynamic behaviour of classes is of prime importance. Therefore, we developed four notions of workflow inheritance [Aalst, Basten (2002)]. The four inheritance relations use branching bisimilarity (to compare processes) in combination with the notions of encapsulation and abstraction. Encapsulation corresponds to blocking tasks, whereas abstraction corresponds to hiding tasks. The inheritance mechanism allows for the definition of a subclass which inherits the features of a specific superclass. When adapting a workflow process definition to specific needs (ad-hoc change) or changing the structure of the workflow process as a result of reengineering efforts (evolutionary change), inheritance concepts are useful to check whether the new workflow process inherits some desirable properties of the old workflow process. This way it is possible to constrain flexibility when desired. Based on the four notions of inheritance, we have developed inheritance preserving transformation rules for workflow processes [Aalst, Basten (2002)]. These rules correspond to design constructs that are often used in practice, namely choice, iteration, sequential composition, and parallel composition. If a workflow designer sticks to these rules, inheritance is guaranteed. The transformation rules can be used to avoid problems such as the "dynamic-change bug" [Ellis et al. (1995)], as mentioned above. Restricting change to the inheritance-preserving transformation rules guarantees transfers without any of these problems. This way it is possible to migrate instances when desired. Moreover, the transformation rules can also be used to extract aggregate management information in case multiple versions of a workflow process are active. The inheritance notions allow for the definition of concepts such as a Greatest Common Divisor (GCD) and Least Common Multiple (LCM) of a set of variants/versions [Aalst, Basten (2002)]. These concepts can be used to create a condensed overview of the work-in-progress. Clearly, the dynamic-change problem and the management-information problem are related. By solving the dynamic-change problem (i.e., instantly migrating all cases to a single version of the process), there is no need to construct aggregate management information because there is just one active version. However, ad-hoc changes inevitably lead to multiple variants and, multiple active versions of a workflow process are typically unavoidable.

The strong theoretical basis of WF-nets allows for powerful analysis techniques. Petri nets have been studied for four decades and there are dozens of Petri-net-based tools supporting the analysis and design of processes and systems. Extensions of WF-nets can be used for validation and performance analysis. For example, the first author has been involved in the development of ExSpect, a Petri-net-based simulation tool which can be used for validation and performance analysis of workflows. ExSpect can be used for modelling and analysing workflow processes and it can interface with workflow management systems such as COSA and BPR-tools such as Protos. Note that validation and performance analysis through simulation is not unique for WF-nets. The real added value of the results on WF-nets is in *verification*. We provide techniques to verify the so-called *soundness property* introduced in [Aalst (1998)]. A workflow is sound if and only if, for any case, the process terminates properly, i.e., termination is guaranteed, there are no dangling references, and deadlock and livelock are absent. Soundness corresponds to well-known properties such as liveness and boundedness (of the short-circuited WF-net). Therefore, standard Petri-net-based analysis techniques and tools can be used to verify soundness. Nevertheless, for a complex WF-net it may be intractable to decide soundness.

For arbitrary WF-nets liveness and boundedness are decidable but also EXPSPACE-hard [Aalst (2000)]. Fortunately, for most of the contemporary workflow management systems, the soundness property can be checked in polynomial time using state-of-the-art analysis techniques. These techniques exploit the structure of the WF-net without exploring the state space. These so-called structural techniques also help identifying suspicious constructs which may endanger the correctness of a workflow. The techniques presented in [Aalst (2000)] also allow for the compositional verification of workflows, i.e., the correctness of a process can be decided by partitioning it into sound sub-processes.

To support the application of the results mentioned, we have developed a Petri-net-based workflow analyser called *Woflan* [Verbeek, Aalst (2000)]. Woflan is a workflow management system independent analysis tool which interfaces with Staffware, COSA, Protos, and Meteor. Woflan can be used for verification: It checks whether the WF-net is sound and generates detailed diagnostics if the WF-net is not sound. Moreover, Woflan supports the four inheritance notions mentioned: It can check whether one WF-net is a subclass of another WF-net. This enables intriguing possibilities. It is possible to check whether a workflow implemented using the workflow management system COSA realizes (i.e., is a subclass of) a workflow specified using the BPR-tool Protos.

3.5 Workflow Evolution

The workflow evolution approach was developed by Casati and colleagues [Casati (1998), Casati et al. (1998a)]. It uses the workflow language developed in the context of the WIDE project [Grefen and Pernici (1999)]. Tasks are atomic units of work that are represented by a name, a textual specification, a set of information items associated with the task, and a set of roles. Nesting of tasks is supported, i.e., each workflow schema can consist of a set of elements which are either (atomic) tasks or sub-processes. Besides sub-processes, business transactions are another way to model complex processes. Business transactions define work that has to be executed in an atomic fashion. Within the WIDE project, transactional capabilities are developed to provide execution guarantees for business transactions. Tasks (and sub-processes) can be connected by fork and join connectors, which are marked with transition conditions.

Based on this workflow language, workflow schemas, workflow instances, and workflow enactment rules are defined in a formal manner. Flexibility is provided by *workflow evolution*: Given a workflow instance with a workflow schema, migrate that workflow instance to a modified workflow schema. The approach presented in [Casati et al. (1998a)] uses specific modification operations, which are applied to the original workflow schema in order to modify it. A notion of compliance is introduced that formally defines which workflow instances can be migrated to the new workflow schema version, created by a set of modification operations, called workflow evolution primitives.

Workflow evolution primitives are partitioned in declaration primitives and flow primitives. While declaration primitives modify the declaration of workflow variables, flow primitives modify the control flow structure of workflow schemas. Typical declaration primitives are AddVar and RemoveVar (representing adding and removing a workflow variable, respectively), while AppendTask and RemoveTask are typical flow primitives to append a task to a given workflow schema and removing a given task from a workflow schema. By applying these primitives to workflow schemas, the global variables, the task structure, and the control flow constraints of workflow schemas can be altered in an evolutionary way. Evolution means here that based on a given workflow schema by incrementally applying workflow evolution primitives the workflow schema is modified in an evolutionary way.

Different policies to handle workflow evolution are discussed. Besides the most obvious ones (aborting all active workflow instances and completing all active ones with the original workflow schema), the progressive policies "migration to final workflow" and "migration to ad-hoc workflow" are proposed. In migration to final workflow, workflow instances that are compliant to the new workflow schema are migrated to that schema, possibly after compensation activities, in case they are available. If compensating activities are required to migrate a workflow but no compensating activities are present then the migration is not possible. In the migrate to ad-hoc workflow policy, the workflow administrator can choose to perform ad-hoc changes to the workflow instance, so that from an

application-specific point of view, the workflow instance can now be migrated to the new workflow schema. These operations can generally not be performed automatically, since the expertise and semantic knowledge of the workflow administrator is required to decide on the modifications to make the migration feasible. This interesting approach presents a formal model to specify workflow evolution in the presence of workflow schemas and multiple workflow instances, and different policies to handle workflow evolution from an organizational point of view are discussed.

Figure 10: Electronic Bookshop Workflow Schema expressed in WIDE Workflow Language.

To illustrate the concepts, the sample workflow process is expressed in the WIDE workflow language in Figure 10. The workflow schema representation looks similar to the workflow graph notation as shown in Figure 8. However, there are a variety of differences: AND forks always require an additional circle, similar to a place in Petri net notation. OR forks associated with transition conditions are represented by diamonds, where each outgoing edge is marked with a predicate. Predicates use information variables, associated with previously executed tasks. Dedicated symbols mark start and end tasks.

4. Evaluation of Approaches

This section evaluates the approaches to modelling and executing workflows with respect to the requirements mentioned above by comparing them and stressing their strengths and weaknesses. Some general remarks are in order. Each of the approaches introduced has its own background and motivation. Hence, it is not surprising that each has its specific strengths and weaknesses. Clearly, there is no workflow modelling approach that copes best with the requirements of advanced workflow applications. However, by stressing the strengths and weaknesses of the approaches we will try to open the door for an integration of research results based on different approaches, as will be discussed below. While we do not aim at the best integrated workflow modelling approach, we show potentials for integrating the good parts of the proposed approaches, which could open new research directions in the near future.

The requirements of advanced workflow management are used to evaluate the approaches introduced above. In particular, we discuss how the workflow perspectives are supported, and how the issues in

flexible workflow management and workflow analysis are handled by the approaches. The result of the evaluation is summarized in Table 1, which for each approach states the support for workflow flexibility and analysis features. As introduced in Section 2.2, flexibility issues include constrained flexibility, instance change and instance migration, represented in Table 1 by ConsFlex, InstChge, and InstMig, respectively. Workflow analysis contains the features validation (Valid), verification (Veri), and performance analysis (Perf).

As detailed in Section 3.1, the *Unified Modelling Language* (UML) is tailored towards modelling structure and behaviour of object-oriented systems. The structural aspects of systems are supported by a rich set of state-of-the-art formalisms. Hence, the information perspective is covered very well. Because nowadays workflow systems act in a software context that is built using object-oriented techniques, the operation perspective is covered well, too. The UML object constraint language can be used similarly to other logic-based formalisms in describing the functional perspective of workflows. However, there is no direct support for organizational modelling, so that the generic concepts provided by the UML have to be used to model organizational aspects. A set of UML specializations as outlined in [Rational (2000)] or the ongoing OMG work on workflow process definition [OMG (2000c)] may help to overcome this problem. The UML diagrams intended to model behaviour, esp. activity diagrams, may be used to describe the process perspective but have their limitations, both in terms of expressive power and lack of a precise semantics. Whereas the latter may even be tolerable regarding the process definition phase, it does not allow for any kind of in-depth evaluation. Simple system traces may be visualized using MSCs and help the expert to find errors, but a simulation-based evaluation that relies on an accepted automatic simulation standard is hard to implement with a notation lacking precise semantics. At the present state, formal verification is not possible at all. For the same reasons, controlled and secure flexibility mechanisms are not feasible because there is no basis to evaluate whether the original and the changed schema or instance fulfil the required compatibility properties.

The *Object Coordination Net* approach reduces the set of UML diagrams to those that have a clear and well-understood semantics, adds object-oriented high-level Petri nets for describing the process perspective and provides a guideline how to use all these diagrams for modelling all workflow perspectives in an overall consistent manner. Hence, the benefits of the UML for the information, operation and functional perspectives apply to OCoNs, too. The contract-based method to model structures enhances the modelling of the organizational perspective and makes it especially useful for describing workflow systems that act across organizational boundaries, have to deal with outsourced parts and so on. Moreover, the usage of a Petri net formalism, which is seamlessly integrated into the UML diagram context, allows for an optimal integration of the process perspective into the more static workflow perspectives. Especially the explicit handling of resources in nets provides the link to the organizational perspective. This is one of the major benefits of the OcoN approach. The fact that the used Petri net formalism has a well-defined formal semantics [Giese (2001)] opens a number of possibilities regarding analysis and flexibility issues. Evaluation can be performed on the basis of simulating a specified organization for different resource instantiations, which even provides the basis for performance analysis. A more powerful simulator for OCoNs that supports the latter is currently under development. Verification is also an option, but not planned at the moment due to the high complexity of verifying high-level nets in the context of an object-oriented structural environment. Due to its overall design philosophy, flexibility in the sense of instance change is not possible with the OCoN approach. Constraint flexibility is supported on the level of contract-based abstraction and encapsulation by exchanging complete subsystems, i.e., organizational substructures as long as the new part supports the same interface regarding structure and behaviour as the replaced part. This strategy becomes even more flexible when utilizing inheritance for contracts as developed in [Giese (2001)]. Although this notion provides the basis for instance migration, too, the approach has not been developed into that direction so far.

Workflow graphs have been developed with two goals in mind: To provide a mechanism to model and enact workflows in a flexible way and as a basis for the development of a flexible workflow management system based on object-oriented technology. Since workflows are represented as nested

graphs, the functional decomposition of workflows is supported very well. The process perspective is represented by control flow constraints between workflows and start conditions, which are evaluated at run time in order to decide on the execution of a particular workflow. The information perspective is addressed by business objects, which represent entities of the real world that are relevant from an application point of view. The operational perspective is covered by methods provided by business objects. As discussed above, legacy systems can be wrapped, i.e., provided with an interface that makes them look and behave like a business object. The organizational structure can be represented using role information, attached to workflow schemas. In terms of flexibility, workflow graphs provide powerful means for constrained flexibility. In particular, dynamic adaptations of workflow instances to workflow schemas is constrained to the cases, where the new workflow schema fits nicely with the workflow instance, formalized by the notion of valid mapping. Along the lines of this discussion, instance change is also facilitated by workflow graphs and the formalisms and functionality provided by WASA. Specifically, each workflow instance can be changed. If the future parts of a given workflow instance is modified then it is guaranteed that a valid mapping exists and, consequently, the instance change is allowed. If schema modifications are involved and all workflow instances controlled by a given workflow schema are going to be changed, valid mappings are computed for all active workflow instances. For all instances for which such a mapping can be found, the instance can be migrated to the new workflow schema. As a result, workflow graphs provide good support for these flexibility issues. Analysis properties are rather weak in workflow graphs. There are structural properties of workflow schemas, which make sure syntactic properties are guaranteed, for instance the absence of loops and compliance of control flow and data flow. The complete consistency properties of workflow schemas can be found in [Weske (2000)]. However, verification techniques are not present, and performance analysis features are also not supported directly; they rather have to be developed on top of the existing workflow graph formalism.

WF-nets focus on control flow, i.e., the process perspective, and do not address the other perspectives. As a result, issues related to flexibility and analysis can be dealt with in a concise and rigorous manner. Clearly, WF-nets are well equipped to represent workflow processes. WF-nets are based on Petri nets and therefore build on a solid and highly expressive formalism. WF-nets allow for the basic routing constructs identified by the WfMC [WfMC (1997)], i.e., sequential, parallel, conditional, and iterative routing. In addition more advanced constructs involving states and mixtures of choice and synchronization are supported [Aalst, Hofstede et al. (2000)]. The basic Petri net model has been extended with time, data (colour), and hierarchy [Jensen (1992)]. Clearly WF-nets can be extended in this fashion to directly support the functional and information perspectives. These extensions can also be used to model the organization and operation perspectives.
The development of WF-nets was triggered by the lack of verification capabilities in contemporary workflow management systems [Aalst (1998)]. Clearly, WF-nets offer powerful verification techniques based on state-of-the-art Petri-net-based analysis routines. Safety and liveness properties are relevant for workflow processes and have been studied in the context of Petri nets for four decades. Tools such as Woflan show that these results can be applied while using commercial workflow management systems by translating workflow specifications into WF-nets [Verbeek and Aalst (2000)]. WF-nets do not offer direct support for validation and performance analysis. However, it is quite easy to extend WF-nets and the associated tools to support validation and performance analysis. For example, by adding stochastic delays it is possible to use the results for (Generalized) Stochastic Petri nets (GSPN) and enable Markovian performance analysis. It is also possible to use simulation. Tools such as ExSpect demonstrate that Petri-net-based simulators can be used to analyse the performance of workflow processes.
WF-nets support workflow flexibility by offering four inheritance notions [Aalst, Basten (2002)]. The inheritance notions have been equipped with inheritance-preserving transformation rules and migration rules. The inheritance-preserving transformation rules can be used to limit change such that certain dynamic properties are preserved. This way WF-nets offer direct support for constraining flexibility. The migration rules allow for instance migration as long as there is a subclass-superclass relation between the old and the new workflow process. WF-nets and the four inheritance notions do not directly support instance change. However, the theoretical results obtained for WF-nets provide a good basis for supporting and controlling on-the-fly changes of a workflow instance.

Table 1: Strength and Weaknesses of Modelling Approaches

	Wf perspectives					Flexibility			Analysis		
	Fct	Proc	Org	Inf	Op	ConsFlex	InstChge	InstMig	Valid	Veri	Perf
UML	+	+	0	++	+	0	-	0	0	-	0
OCoN	+	++	+	++	+	+	-	0	+	0	0
Wf graphs	+	+	0	+	+	+	+	++	+	-	0
WF-nets	0	++	0	0	0	++	0	+	0	++	0
Evol. Wf	+	++	+	+	0	++	+	++	+	0	0

Note: ++: designed for, + supported, 0 supported through known extensions, - no immediate support

The *Workflow Evolution* approach presented in Casati et al (1998a) is based on a workflow language that supports the functional perspective by tasks and multi-level nesting of complex workflow. The process perspective is covered well, since control flow constraints including join and fork connectors as well as transition conditions are provided. Information modelling is based on so called information variables that are accessed via forms. Forms present a nice way to restrict access to variables at certain states during the process execution. Documents are additional information elements that cannot directly be controlled by the workflow management system at hand; they can be regarded as external information that, however, is relevant for the workflow process. Documents and forms can be grouped to folders. While each task is associated with information elements, data flow is not immediately represented. If parallel strands of executions access the same information variable or the same document then race conditions may arise, which have to be handled by the WIDE transaction mechanism. However, data variables managed by a database may use the database's own transaction processing capabilities.

Since the approach focuses on a conceptual specification of workflow systems, the operational perspective is not adequately addressed. However, operational details could be supported by known extensions. To discuss the capabilities of that approach with respect to flexibility, we mention that constraint flexibility is supported very well. In fact, the approach offers formal correctness criteria which rule if and when a given workflow instance can be migrated to a new workflow schema, which evolved from the original workflow schema by applying a set of workflow evolution primitives. By a clear separation of workflow schemas and workflow instances and by the feasibility of ad-hoc modifications of workflow instances, the instance change property is supported well. The abovementioned formal rules concerning workflow instance migration, the instance migration property is also supported well. Validation is provided with respect to properties of workflow schemas. Verification is mainly defined in connection with evolutionary changes of workflows. For additional information on the flexibility of the proposed approach, the reader is referred to the original literature.

5. Conclusions

This paper describes and analyses contemporary approaches to workflow modelling as well as the workflow modelling support provided by the Unified Modelling Language with respect to the requirements of advanced workflow management. The evaluation is based on a set of widely acknowledged workflow perspectives, as well as advanced workflow modelling features in the areas of flexibility and analysis of workflows. As discussed above, each approach has its specific strengths and weaknesses. While UML has its main strength in structural object modelling, OCoNs add process modelling and validation features to UML structure diagrams. Workflow graphs were developed with flexibility in mind; hence, they provide good support for the flexibility aspects mentioned, while analysis features of Workflow graphs are rather weak. WF-nets make available to workflow management powerful analysis methods from Petri net theory, and they are well equipped to support

complex control flow structures and—through the use of inheritance of behaviour—support flexibility well, while data modelling and organizational modelling is not their main scope. Workflow evolution deals with incremental modifications to existing workflow schemas, and it investigates the formal properties of these changes.

To conclude, we like to point out some general principles of the object-orientation paradigm that can be regarded as the basics behind the approaches. In doing this, we have two goals in mind: Firstly to find commonalities and differences of the approaches and, secondly, to open the door for future work in advanced workflow management. Obviously, the UML is strongly based on object-oriented principles, such as inheritance. However, inheritance typically only applies to inheritance of structure, not of behaviour. In the workflow context where processes are in the centre of attention, however, inheritance of behaviour is a very important and helpful feature. This feature is formalized by WF-nets where a variety of application areas are created by inheritance and sub-typing of workflows, for instance migration of workflows to more specific workflow schemas. We envision that inheritance of behaviour can also be combined with other approaches. Probably the most obvious way of doing this is enhancing the notion of refinement of Petri nets in the OCoN approach with a notion of inheritance, which not only works on state-machine-like interface protocols but also on the more expressive nets used in this approach. Due to the strong connection of OCoNs and structure diagrams of the UML, such an approach would improve the development, maintenance, and usability of real-world workflow applications.

The clear separation of schema and instance information is another strong feature of the object-orientation paradigm. In particular, schema information describes the common properties of a set of similar real-world objects, typically represented by a class. When it comes to workflow flexibility, in particular the modification of workflow schemas during run-time, it is important that modifications of workflow schemas can be performed easily with little effort. The concept of meta-objects helps here. In particular, objects, which specify structure, can easily be modified. As discussed above, this concept is used in the Workflow graph formalism, where workflow schemas are represented by workflow schema objects, such that schema modifications can be performed by value changes (to workflow schema objects). In a second step, the set of workflow instances, which rely on the modified schema, are investigated to decide about correct adaptations. This approach allows to modify many workflows in consequence of the modification of a workflow schema. While this technique is used in Workflow graphs, it can also be used in the other formalisms discussed in this paper, to gain good support for workflow flexibility.

The next aspect of object-oriented modelling that we like to raise is overloading and late binding. Overloading represents the fact that a single method name is implemented many times, and late binding adds the run time aspect to overloading in the sense that only at run-time the system decides on the particular implementation. While these concepts are so far not used in the workflow modelling approaches mentioned, they can in fact be incorporated, as the following discussion shows: Assume the structure of a given application process depends on the client for which the process is being executed or, more generally, on a data object manipulated during the process. In this case, the implementation of a sub-workflow activity can be resolved late, i.e., the sub-workflow schema is bound only during run-time to the workflow instance, making use of late binding in the workflow context. For instance, assume the processing of an insurance claim, where a very good customer (e.g., a large company) claims the loss of a rather inexpensive part. In order not to lose this customer, you may want to process that claim very fast with little or no additional information. If a private customer submits the same claim, it would have to undergo complex checks before the loss is compensated for. Hence, different implementations of the checking activity can be found in this case. We believe that this form of late binding in the workflow context is valuable for a wide range of applications in business settings. Just like overloading and late binding in OOD improves development and maintenance of programs, it may also improve modelling and maintenance of workflow applications considerably.

Finally, we mention that there is work in workflow exception handling, e.g., [Casati et al (1998b)]. The issue of designing exceptions, which can be statically foreseen, is tackled. The set of foreseen exceptions, specified as patterns, are maintained in a pattern catalogue. Sample patterns are Remainder, Document Revision, and Termination. Patterns are specified using template structures as well as guidelines how to use the pattern. There is tool support for specifying exception patterns and

for instantiating them in the context of particular workflow instances. Exception patterns can be regarded as application-oriented and often application-specific partial workflows, which are used to handle foreseen situations during workflow executions. This work is rather generic and independent from the underlying workflow language; in particular, it can be expected that it can be used in the other workflow languages investigated in this paper as an exception handling facility.

References

Van der Aalst, W.M.P. (1998): The Application of Petri nets to Workflow Management. The Journal of Circuits, Systems and Computers, 8(1):21-66

Van der Aalst, W.M.P. (2000): Workflow Verification: Finding Control-Flow Errors using Petri-net-based Techniques. In Business Process Management: Models, Techniques, and Empirical Studies, volume 1806 of Lecture Notes in Computer Science, pages 161-183. Springer, Berlin

Van der Aalst, W.M.P., Basten, T. (2002): Inheritance of Workflows: An Approach to Tackling Problems Related to Change. Theoretical Computing Science, 270(1-2):125-203

Van der Aalst, W.M.P., Jablonski, S. (2000): Dealing with Workflow Change: Identification of Issues and Solutions. International Journal of Computer Systems, Science, and Engineering, 15(5):267-276

Van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., and Barros, A.P. (2000): Advanced Workflow Patterns. In O. Etzion and P. Scheuermann, editors, 7th International Conference on Cooperative Information Systems (CoopIS 2000), volume 1901 of Lecture Notes in Computer Science, pages 18-29. Springer, Berlin

Brauer, W., W. Reisig, Rozenberg, G., Editors (1987): Petri nets: Central Models (I) / Applications (II), Vol.254/255 of Lecture Notes in Computer Science. Springer, Berlin

Casati, F., Ceri, S., Pernici, B., Pozzi, G. (1998a): Workflow Evolution. Data and Knowledge Engineering, 24(3): 211-238.

Casati, F. (1998): Models, Semantics, and Formal Methods for the design of Workflows and their Exceptions. Ph.D. thesis Politecnico di Milano

Casati, F., Fugini, M.G., Mirbel, I. (1998b): An Environment for Designing Exceptions in Workflows. Proc CAiSE'98. pp 139-157. Springer Lecture Notes in Computer Science 1413. Berlin: Springer

Ellis, C.A. (1979): Information Control Nets: A Mathematical Model of Office Information Flow. In Proceedings of the Conference on Simulation, Measurement and Modelling of Computer Systems. Boulder, Colorado, pages 225-240. ACM Press

Ellis, C.A., Keddara, K. and Rozenberg G. (1995): Dynamic change within workflow systems. In N. Comstock and C.A. Ellis, editors, Conf. on Organizational Computing Systems, pages 10 - 21. ACM SIGOIS. Milpitas: ACM

Georgakopoulos, D., Hornick, M., Sheth, A. (1995): An Overview of Workflow Management: From Process Modelling to Workflow Automation Infrastructure. Distributed and Parallel Databases, 3:119-153

Giese, H. (2001): Object-Oriented Design and Architecture for Distributed Systems. Doctoral Dissertation. University of Münster

Giese H., Graf J., Wirtz, G. (1999): Closing the Gap Between Object-Oriented Modelling of Structure and Behaviour. In: Proc. of UML-99 The 2nd Intern. Conf. on The Unified Modelling Language

Giese, H., Wirtz, G. (2000): Early Evaluation of Design Options for Distributed Systems. In: Proceedings International Symposium on Software Engineering for Parallel and Distributed Systems (PDSE), June 2000, Limerick (Ireland), IEEE Press

Grefen, P., Pernici, B., Sanchez, G. (1999): Database Support for Workflow Management: the WIDE Project. Kluwer Academic Publishers, 1999.

Hammer, M., Champy, J. (1994): Business Reengineering, Frankfurt, New York (2nd)

Harel, D. (1987): Statecharts: A Visual Formalism for complex systems. Science of Computer Programming 3 (8): 231 – 274

Hruby, P. (1998): Specification of Workflow Management Systems with UML. In: Proceedings of the OOPSLA'96 Workshop on Business Object Design and Implementation, San Jose

IBM (1999): IBM MQSeries Workflow: Concepts and Architecture, Version 3.2.   Publication No GH12-6285-01

Jablonski, S., Bussler, C. (1996): Workflow-Management: Modelling Concepts, Architecture and   Implementation. International Thomson Computer Press

Jensen, K. (1992): Coloured Petri nets. Basic Concepts, Analysis Methods and Practical Use, EATCS monographs on Theoretical Computer Science, Springer, Berlin

Leymann, F., Altenhuber, W. (1994): Managing Business Processes as an Information Resource. IBM Systems Journal 33, pages 326-347

Meyer, B. (1997): Object-Oriented Software Construction. Prentice Hall, 1997 (2nd)

OMG (01/2000): Workflow Resource Assignment Interfaces (RAI) – Request for Proposal. OMG-document bom/2000-01-03, Framingham

OMG (11/2000): Organizational Structure Facility (Revised Submission by 2AB Inc., Gazebo Software Solutions Inc. Genesys Software Inc.). OMG Business Object Domain Task Force, OMG-document bom/2000-11-05, Needham

OMG (12/2000): UML Extensions for Workflow Process Definition – Request for Proposal. OMG-document bom/2000-12-11, Needham

Reichert, M., Dadam, P. (1998): Supporting Dynamic Changes of Workflows Without Loosing Control. Journal of Intelligent Information Systems, Special Issue on Workflow and Process Management, Vol. 10, No. 2

Rumbaugh, J., Jacobson, I., Booch, G. (1999): The Unified Modelling Language Reference Manual. Reading, MA: Addison-Wesley

Rational (2000): Business Modelling with the UML and Rational Suite AnalystStudio. A Rational Inc. Software White Paper

Sheth, A.P., van der Aalst, W.M.P., Arpinar, I.B. (1999): Processes Driving the Networked Economy. IEEE Concurrency 7(3): 18-31

Verbeek, H.W.M., Van der Aalst, W.M.P. (2000): Woflan 2.0: A Petri-net-based Workflow Diagnosis Tool. In M. Nielsen and D. Simpson, editors, Application and Theory of Petri nets 2000, volume 1825 of Lecture Notes in Computer Science, pages 475-484. Springer, Berlin

Weske, M., Vossen, G. (1998): Workflow Languages. In: P. Bernus, K. Mertins, G. Schmidt (editors): Handbook on Architectures of Information Systems. (International Handbooks on Information Systems), pages 359-379. Springer, Berlin

Weske, M. (2000): Workflow Management: Systems: Formal Foundation, Conceptual Design, Implementation Aspects. Habilitation Thesis. University of Münster

Weske, M. (2001): Formal Foundation and Conceptual Design of Dynamic Adaptations in a Workflow Management System. Accepted for: Minitrack Internet and Workflow Automation: Technical and Managerial Issues. 34th Hawaii International Conference on System Sciences (HICSS-34).

Wiegert, O. (1998): Business Process Modelling and Workflow Definition with UML – Deficiencies and Actions to Improve. Presentation at the OMG Meeting, Manchester, 1998-03-31

Wirtz, G., Graf, J., Giese, H. (1997): Ruling the Behaviour of Distributed Software Components. In: Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA), June 1997, Las Vegas (USA); CSREA Press

Wirtz, G., Weske, M., Giese, H. (2000): Extending UML with Workflow Modelling Capabilities. In Proc. of CoopIS-2000, 7th Intern. Conf. on Cooperative Information Systems (O. Etzion and P. Scheuermann, eds.), vol. 1901 of LNCS, Springer, pages 30-41

Wodtke, D., Weissenfels, J., Weikum, G., Kotz Dittrich, A. (1996): The Mentor Project: Steps Towards Enterprise-Wide Workflow Management. In Proc. 12th IEEE International Conference on Data Engineering (1996), pages 556-565

Workflow Management Coalition, WfMC (1997). Workflow Handbook. John Wiley in association with Workflow Management Coalition