Object-Oriented Design of a Flexible Workflow Management System

Mathias Weske

Lehrstuhl für Informatik, Universität Münster
Steinfurter Straße 107, D-48149 Münster, Germany
weske@helios.uni-muenster.de

Abstract. Workflow management systems aim at controlling the execution of complex application processes in distributed environments. Workflow management currently moves from modeling and executing mostly static structured workflows to supporting flexible workflows, which are typically executed in distributed and heterogeneous environments. This paper discusses the use of distributed object technology to built a flexible workflow management system. In particular, based on a detailed object-oriented object model, we discuss the dynamic behavior of workflow instances, and we show how flexibility requirements have influenced our design.

Keywords: workflow management system design, object modeling, flexible workflow management, distributed workflow executions

1 Introduction

Workflow management has gained increasing attention recently as a core technology to foster information system development in dynamically changing and distributed environments [6, 13, 22]. While the first generation of workflow management systems (WFMS) were developed to control the execution of business applications with fairly static structures to be executed in homogeneous environments, recently the need for enhanced flexibility and the integration of applications in heterogeneous environments emerged [24, 19]. In this paper we look into these issues. In a first step, an object model of a workflow management system is developed, which identifies workflow relevant objects and their relationships. In a second step, we show how the design decisions as specified in the object model support flexibility, namely by permitting dynamic modifications of running workflow instances.

The work presented in this paper is influenced by the OMG's (Object Management Group) quest for the specification of a Workflow Facility [17]. In recent years, the OMG has led the development of the CORBA (Common Object Request Broker Architecture) standard for distributed object computing; while the specification of basic CORBA Common Object Services (COS) [16, 23] can be considered completed, the focus recently has turned to the specification of higher-level and domain-specific concepts, known as CORBA Facilities. In this

context, the CORBA Workflow Facility currently is in the process of specification. A number of proposals by major workflow system vendors were submitted [1, 15, 9, 4], most of which expose considerable deficiencies in object modeling and in showing how existing CORBA COS can be used to implement the respective systems [21]. As opposed to these proposals, our approach to object oriented modeling and development of a flexible workflow management system is not restricted by existing products which we want to adapt to an upcoming standard; we are currently implementing the system whose design is presented in this paper using the Orbix [11] object request broker implementation.

The paper is organized as follows: Section 2 introduces preliminaries on workflow management and workflow modeling and execution. Section 3 proposes an object model of a flexible workflow management system. In Section 4 we describe the dynamic behavior of workflow instances; Section 5 discusses flexibility issues by showing how the emerging requirements on enhanced flexibility of workflow management systems has influenced the design decisions. A section on related work and a summary conclude this contribution.

2 Preliminaries

Workflow management deals with modeling and controlling the execution of application processes in given organizational and technical environments. To control the execution of a workflow instance, a workflow management system requires a computerized representation of the respective business process. These representations are known as workflow models, which are expressed in workflow languages [26]. To cope with the problem complexity in workflow applications, workflow languages usually support some form of hierarchical specification. In graph-based workflow languages [13], for instance, workflow models are specified as enhanced directed graphs (workflow model graphs), in which nodes represent activities and edges represent constraints between activities, e.g., control flow dependencies and data flow dependencies. A node of a workflow model graph can represent either an atomic workflow (in which case an application program is specified, which implements the workflow) or a complex workflow, which is refined by another workflow model graph. This specifies a tree-like structure of complex workflow models, where the root node (toplevel workflow) represents the application, the inner nodes represent other complex workflow models, and the leaf nodes represent atomic workflows, which are implemented by application programs.

Depending on the workflow language and on the workflow management system used, several dimensions are covered; these dimensions are also known as workflow aspects [12]. The functional aspect specifies what has to be done within a workflow; the operational aspects determines how it is done, i.e., which techniques and tools are used to perform activities. The behavioral aspect defines when and under which conditions a workflow is executed. The informational aspect specifies the data objects which are being manipulated during workflow executions and the flow of data between workflow activities. The organizational

aspect describes the roles and personnel which are involved in workflow executions. Workflow languages allow the specification of workflow models, such that the workflow aspects are defined independently. This approach allows to concentrate on specific aspects separately when modeling workflows.

3 Object Model

A concise object model is an important requirement for the development of complex software systems. This section discusses a logical object model of a workflow management system, which centers around workflow-related entities and their relationships from a logical point of view without prescribing implementation details. These aspects are covered by a design object model, which is described in [25].

The central class of the object model is the Workflow class, which appears in the center of the object model in Figure 1, showing the object model in UML notation [18]. The Workflow class contains workflow objects which are partitioned into workflow model objects (or workflow models) and workflow instance objects (or workflow instances). By representing by a single class workflow models and workflow instances, redundancy is omitted, and the border between a workflow's built time and its run time (as typically found in today's commercial WFMS [8]) is broken, which is important when it comes to supporting flexibility properties, which will be discussed below.

Each workflow instance is instantiated using a workflow model. The relationship between workflow instances and workflow models is represented by the instance-of relationship. Workflows can be either complex or atomic, and atomic workflows are either manual or automatic, leading to the generalization hierarchy shown in Figure 1. Workflows can be related to each other in hierarchical fashion, representing the functional decomposition of the application process, hence the functional aspect, as discussed in the previous section. Each application process is typically represented by a complex workflow, which consists of a number of sub-workflows. Sub-workflows of a given complex workflow can be related to each other by control flow and data flow. The WF-SubWF Relationship determines for each complex workflow its constituent (sub-)workflows. Notice that each workflow model can be re-used in multiple complex workflows, i.e., it can occur as a sub-workflow in different complex workflows. This concept to re-use workflow models is represented in the object model as follows. For each occurrence of a workflow in a complex workflow, one object in the WF-SubWF Relationship class is created. Since the control flow and data flow constraints can (and often will) vary in different occurrences of a given workflow, control flow and data flow constraints are defined based on WF-SubWF Relationship objects rather than on workflow objects.

To explain the object model in more detail, we use an example, which is shown in Figure 2. In part (a) of that figure, a set of workflow models are connected to each other by WF-SubWF Relationships (represented by dotted lines) and control flow constraints (represented by solid arcs). The hierarchical struc-

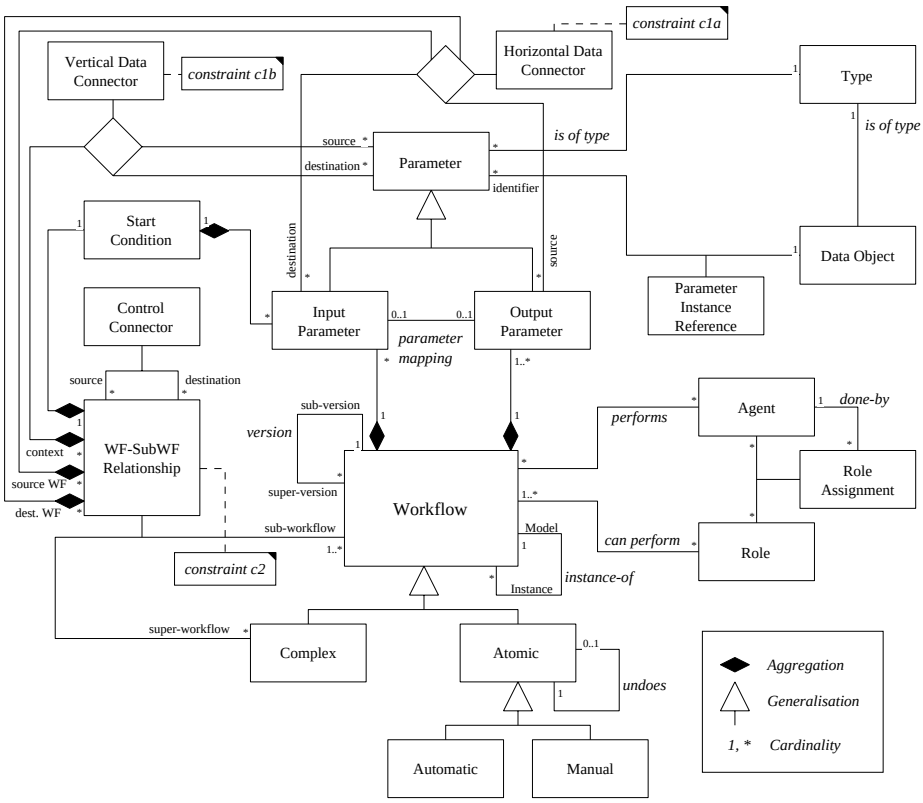

Fig. 1. Object Model

ture of the complex workflow is represented by distinct WF-SubWF Relationship objects $o_1 = (1,2), o_2 = (1,4), o_3 = (1,3), o_4 = (1,2), o_5 = (4,2), o_6 = (4,3)$. Notice that workflow model 2 appears twice in complex workflow 1 (represented by WF-SubWF Relationship objects o_1 and o_4) and once as a sub-workflow of complex workflow 4 (represented by o_5). The control connectors shown are represented by Control Connector objects $(o_1, o_2), (o_2, o_3), (o_3, o_4), (o_5, o_6)$, where (s, t) specifies a control connector with source s and target t. Notice that a control connector connects two objects of the WF-SubWF Relationship class rather than two objects of the Workflow class. As explained above, this allows to define different control connectors for different occurrences of a given sub-workflow. In fact, workflow 2 is involved in two control connectors: Control flow $2 \rightarrow 4$ is represented by Control Connector object (o_1, o_2), while $3 \rightarrow 2$ is represented by (o_3, o_4).

Each workflow has a set of input parameters and a set of output parameters, specified in the object model by the Input Parameter class and the Output Parameter class, resp., which are sub-classes of the Parameter class. Data flow

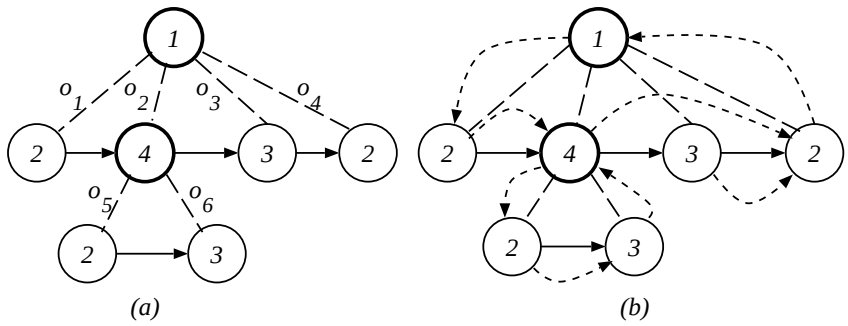

Fig. 2. Complex Workflow Model with Control Flow (a) and Data Flow (b).

can be vertical or horizontal. Vertical data flow corresponds to data flow from a complex workflow to one or more of its sub-workflows (or vice versa), while horizontal data flow connects (parameters of) sub-workflows of a given complex workflow. Hence, the informational aspect is covered by the parameter classes and the Data Connector class; application data is maintained in the Data Object class, and Type class maintains information on the data types of data objects.

In our object model, vertical data flow is represented by the Vertical Data Connector relation between two parameter objects (source parameter, destination parameter) and one WF-SubWF Relationship object. Notice that a single object of the WF-SubWF Relationship class suffices, since it defines the connection between a complex workflow and one of its sub-workflows, i.e., a single WF-SubWF Relationship object represents both workflow objects involved in a vertical data flow. Vertical data flow either connects two input parameters (vertical data flow direction "down") or two output parameters (direction "up"). Horizontal data flow relates two WF-SubWF Relationship objects, one Input Parameter object and one Output Parameter object.

In Figure 2(b), the sample workflow model is enhanced with data flow (represented by dotted arrows; for better readability, data connectors are drawn between workflow models rather than between parameters of workflow models). In the example, complex workflow 1 passes data to the leftmost instance of sub-workflow 2, and it receives data from the second instance of sub-workflow 2. The leftmost instance of sub-workflow 2 is represented by WF-SubWF Relationship object o_1, while the rightmost occurrence is represented by object o_4, as shown in Figure 2(a). The data flow from 1 to the leftmost occurrence of sub-workflow 2 – assuming input parameter ip_1 of workflow object 1 is connected to input parameter ip_2 of sub-workflow 2 – is represented by Vertical Data Connector object (ip_1, ip_2, o_1), such that ip_1 is the source parameter, ip_2 is the destination parameter, and o_1 is the WF-SubWF Relationship object. The second example of vertical data flow is given by (op_2, op_1, o_4), assuming that op_2 and op_1 are the output parameters of the rightmost occurrence of workflow 2 and of workflow 1, resp.

Horizontal data flow is specified by a triple of the form (Output Parameter object, Input Parameter object, pair of WF-SubWF Relationship objects). An example of horizontal data flow connects workflow models 2 and 4 in complex workflow model 1, which is represented by $(op_2, ip_4, (o_1, o_2))$, assuming that op_2 is an output parameter of workflow model 2 and ip_4 is an input parameter of workflow model 4. Notice that parameter op_2 participates in different data flows. Hence, for each occurrence of a workflow model in a complex workflow model, new data flow constraints can be defined, supporting the modular design and reusability of workflow models. The other horizontal data flows are represented by $(op_4, ip_2, (o_2, o_4))$, $(op_3, ip_2, (o_3, o_4))$, $(op_2, ip_3, (o_5, o_6))$.

For each workflow object a start condition specifies if and when the workflow object can be started during a particular workflow execution. This condition may use the value of any input parameter and information on the termination of other workflows. Control flow is implemented by a special form of data flow, which is called control flow relevant data flow; it works as follows:

Assume there is a data flow $i \rightarrow j$ between workflow models i and j, defining the sequential execution of the two workflows. Then on the termination of workflow i a control flow relevant data flow is generated by i. The respective parameter is then passed to j. This information is used by j's start condition as follows. Each start condition is composed of two parts, the first one of which consists of a list of control flow relevant input parameters; the second part is based on traditional data flow. (An example of the latter is "start the workflow if the value stored in input parameter ip_j is less than x".) Control flow is implemented by the flow of data values "not-signaled", "true-signaled" and "false-signaled". Start conditions can be evaluated only if the control flow relevant input connectors are either "true-signaled" or "false-signaled", i.e., workflow instances cannot be started until predecessor workflow instances have terminated ("true-signaled"), or until the system determines that they will not be executed ("false-signaled") due to skipping activities or dead-path-elimination [13].

We remark that the behavioral aspect is covered by control connectors as specified in the control connector class and by start conditions. Notice that by allowing each workflow model to have different start conditions in different complex workflows, flexible composition of complex workflow models from existing workflow models is supported. The organizational aspect is specified in the Role class, which maintains information on the roles and – using the relationship to the agent class – the persons in the organization which are skilled and ready to perform the activity. The operational aspect is defined in the atomic workflow class; this class holds for each atomic workflow an application program which is executed to perform the atomic workflow.

Workflow instance objects are created using workflow model objects. We now discuss a workflow instance created using workflow model 1, as shown in Figure 2. The workflow instance is depicted in Figure 3(a). For each occurrence of a workflow model, a new workflow instance object is created. For example, a workflow instance object $1'$ is created using workflow model 1. Workflow model object 2 appears three times in the example; hence, three workflow instance

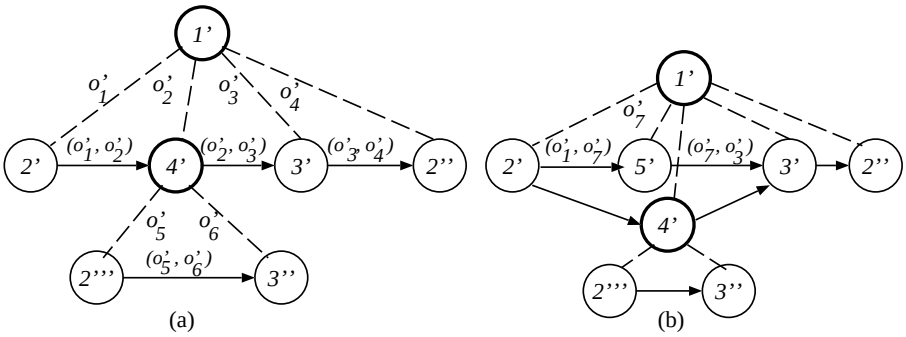

Fig. 3. Complex (a) and Dynamically Modified Workflow Instance (b)

objects are created during the execution of complex workflow $1'$, referred to by $2', 2''$, and $2'''$, resp. Besides the workflow instance objects, instance-objects of other classes are also created, for instance in the WF-SubWF Relationship class. While objects o_1 through o_6 of that class represent the relationships between workflow model objects, new objects of the WF-SubWF Relationship class are created to reflect the relationships between specific workflow instance objects. In the example, objects o'_1 through o'_6 of the WF-SubWF Relationship class are created during the execution of complex workflow instance $1'$, such that $o'_1 = (1', 2'), o'_2 = (1', 4'), o'_3 = (1', 3'), o'_4 = (1', 2''), o'_5 = (4', 2'''), o'_6 = (4', 3'')$. It also applies for data connectors: For each data connector object connecting workflow model objects, a new data connector object is created, which represents the data flow between workflow instances (Due to better readability, data flow instance objects are not shown in Fig. 3.). We remark that the Parameter Instance Reference class specifies the link between parameters and data objects, which are maintained in the Data Object class.

4 Modeling Dynamic Behavior

After discussing how workflow models and workflow instances are represented in our object model, we now describe how workflow instances are executed. This description is based on the sample workflow instance $1'$, as shown in Figure 3(a). The toplevel workflow instance is created following an external event which triggers the start of a business process, e.g., a customer order arrives in a company. The creation of workflow instance $1'$ is followed by the creation of its immediate sub-workflow instance objects $2', 4', 3', 2''$. At this point, the sub-workflows of complex workflow instance $4'$ are not yet created. Creating workflow instances as they are needed ('on the fly') presents a nice way to reduce overhead (in some workflow instances, certain paths may not be executed and, consequently, certain workflow sub-trees may not be needed) and to react in a flexible manner to changes in the workflow environment, as will be discussed in Section 5.

Workflows are executed in a distributed fashion. In our example, the execution of the complex workflow instance $1'$ is done distributed and without centralized control: Toplevel workflow instance $1'$ sends a message to the workflow instance $2'$, which is the only sub-workflow instance without any incoming control connector. $2'$ evaluates its start condition and if it evaluates to true, its execution starts. Since it is an atomic workflow, a work item representing the atomic workflow will appear on the work item list of the person which is selected by role resolution to perform this activity. The person selects the work item, and the application program is started and provided with data, as defined by the vertical data flow from the toplevel workflow to $2'$. When the person completes the activity, the workflow instance $2'$ is notified. It sends a message to $4'$, etc. Finally, the toplevel workflow instance is notified about the completion of its sub-workflows, completing the toplevel workflow.

Notice that there is no centralized workflow engine controlling the execution of the complex workflow. In contrast, each complex workflow instance instantiates its immediate sub-workflows. The sub-workflows are not controlled by the complex workflow; workflows objects communicate by sending and receiving messages, which is a natural way of describing communication in object oriented systems. This approach is very appropriate in workflow environments, since these are inherently distributed, such that centralized solutions will suffer from the central workflow engine being the bottleneck of the system or a single point of failure.

5 Flexibility Issues

Recently the need for enhanced flexibility of workflow management system has emerged in the context of workflow applications in the natural sciences [10, 14, 24, 7], in engineering, and in advanced business applications [5, 19]. We now show how flexibility requirements have shaped the object modeling and state modeling as presented above. While important flexibility operations can be classified in dynamic modification operations and user intervention operations [27], this paper concentrates on dynamic changes of workflow instances, using our example.

Again consider the workflow instance shown in Figure 3(a). Assume while executing $2'$, the user (or the workflow administrator) wants to perform a dynamic change by inserting another workflow instance based on an atomic workflow model 5, to be executed concurrently with workflow instance $4'$. A sample reason for this situation is that the user requires additional information, for which no activity has been defined in the workflow model. In this case, the user may want to include another sub-workflow which gets hold of the information by, e.g., accessing a database. If the information is required to execute workflow instance $3'$ and the inserted workflow instance is independent from the workflow instance $4'$ then the two can be performed concurrently. This dynamic change results in the workflow instance shown in Figure 3(b). From a system's perspective, the dynamic change is reflected by (i) creating a workflow instance $5'$ using

a predefined workflow model 5, (ii) creating a new WF-SubWF Relationship object $o'_7 = (1', 5')$, (iii) creating control flow objects $(o'_1, o'_7), (o'_7, o'_3)$ to reflect the control flows $2' \rightarrow 5'$ and $5' \rightarrow 3'$, resp., and (iv) creating the respective data flow connectors. We remark that these operations can be executed while the workflow runs. However, in order to make sure that the parts which are changed by a dynamic modification are not started early in the course of the workflow execution, workflow instances $3'$ and $4''$ can be suspended temporarily. After the dynamic change operations are completed, workflow instance $2'$ sends messages to workflow instance $4'$ and to (the newly inserted) workflow instance $5'$. When both are completed, the execution of the toplevel workflow can continue with sub-workflows $3'$ and $2''$.

If changes to the data flow occur during dynamic change operations, additional steps have to be carried out. Assume in our example there is a data flow from workflow instance $2'$ to the inserted workflow instance $5'$. This is reflected by creating a horizontal data flow object $(op'_2, ip'_5, (o'_1, o'_7))$. After $2'$ terminates, the data in op'_2 is passed to ip'_5 to implement the defined data flow. Assume there is a data flow defined from $5'$ to $3'$. This data flow can be represented by a horizontal data flow object $(op'_5, ip'_3, (o'_7, o'_3))$. Adding this data flow constraint requires workflow instance $3'$ to be updated to make use of this data flow.

Notice that this section discusses dynamic changes from a system's perspective, especially focused around how the insertion of a sub-workflow can be performed and which objects are involved. There are additional issues related to dynamic changes, ranging from correctness criteria of dynamic change operations to authorization management (who is allowed to perform dynamic change operations on which workflow instances) and user interface requirements; these questions are topics of ongoing research.

6 Related Work

Important topics in recent workflow research include enhancing the flexibility of workflow management systems [7, 19, 27, 5] and object-oriented approaches [20, 2] to the development of workflow management systems, especially in the context of the specification of the OMG Workflow Facility [17], which is currently underway.

A number of submissions to the Request for Proposals of the OMG Workflow Facility were received, which were submitted by major workflow vendors [15, 1, 4, 9]. Generally speaking, these submissions are based on the designs of the workflow management systems already developed by the submitting companies. These systems were developed using mostly traditional techniques, like relational databases, a centralized workflow engine and a number of workflow clients accessing the centralized engine using a client/server approach. Since a CORBA-based re-implementation of their systems is not an option for most workflow vendors, the specifications as described in the submissions are tailored more towards the running systems than towards object modeling techniques, which may be a reason for the deficiencies of the submissions, as far as object modeling and

embedding of the Workflow Facility in the CORBA Common Object Service [16] context is concerned. A detailed critique of the initial submissions is given in [21].

The Meteor_2 project [20] at the University of Georgia allows the definition of workflow models and the automatic generation of workflow code from workflow models which then is executed in a distributed environment using CORBA infrastructure. While the approach allows the distributed execution of workflows, flexibility issues are not considered in that project. In fact, adding flexibility operations like changing the control flow or adding new activities to workflow instances may be complicated by the fact that workflow code is already distributed throughout a distributed system and running.

In terms of flexibility issues in workflow management, the ADEPT framework and the FUNSOFT approach are discussed now. In the ADEPT framework, workflow models are defined by symmetric graphs with specific workflow relevant nodes [19]. Branching is modeled by AND split, OR split, XOR split nodes, which are followed by the respective join nodes. Based on this framework, a set of change operations are specified to allow controlled dynamic changes of workflow instances by users. For instance, adding an activity to a workflow instance involves embedding the added activity into the workflow model and performing a number of operations on the resulting workflow graph to make sure the structure of the graph is consistent with the symmetric structure of workflow model graphs as specified in the ADEPT framework. While ADEPT presents a mathematically founded framework for flexible workflow management, design considerations of a workflow management system providing this functionality are not yet described. In the FUNSOFT approach to modeling and executing workflows, workflow models are specified by FUNSOFT nets, a variant of higher Petri nets geared towards workflow management. Flexibility is achieved by allowing sub-nets to be defined while the workflow runs, achieved by a technique called late modeling. The FUNSOFT net explicitly contains black boxes, i.e., sub-nets which have to be defined during runtime of the workflow [7]. This form of flexibility is rather limited since the position to be specified during runtime has to be known in advance. Dynamic changes due to unanticipated situations in the execution environment of the workflow cannot be handled by that approach. The functionality discussed in [7] is implemented in the CORMAN prototype.

7 Conclusions

As was shown in this paper, there are other good reasons for using distributed object technology to develop a workflow management system. First of all, the development of complex software systems in general may benefit from distributed object technology, especially in distributed environments. In addition, workflow-specific properties arise. In particular, using standardized interfaces of workflow services can provide (i) integration of existing domain-specific tools to be used in workflow executions, (ii) interoperability between different workflow management systems. Furthermore, (iii) data flow can be implemented by passing object references between workflow activities, which will become even more important

when implementations of the CORBA Business Object Facility [3] will be available.

This paper discusses an object model to design a flexible, distributed workflow management system based on object technology. Its key features are a strict object-oriented approach, the modeling of workflow models and workflow instances in a single class and providing a high degree of flexibility in re-using workflow models. For further information on a design object model and on additional flexibility issues, the reader is referred to [25]. Based on the material presented in this paper, we are currently implementing a prototype of a flexible workflow management system using the Orbix [11] Object Request Broker.

As discussed in the section on flexibility, other important issues in flexible workflow management are in the center of ongoing work and will be the topic of future work as well, for instance correctness criteria of dynamic change operations, authorization management and issues in user interface design. We believe that the object model presented in this paper will provide an adequate basis for these research and development activities.

Acknowledgment: The author appreciates the work of colleagues and students involved in the WASA project at the University of Münster.

References

1. CoCreate Software, et al. *jFlow. Submission to Request for Proposals OMG Workflow Facility*, 1997. OMG Document bom/97-08-05, 1997. (available from www.omg.org)
2. Dogac, A., Gokkoca, E., Arpinar, S., Koksal, P., Cingil, I., Arpinar, B., Tatbul, N., Karagoz, P. Halici, U., Altinel, M.: *Design and Implementation of a Distributed Workflow Management System: METUFlow.* NATO ASI Workshop, Istanbul, August 12–21, 1997. To appear in Springer ASI NATO Series
3. Data Access Technologies, Inc., et al. *Combined Business Object Facility Proposal.* OMG Business Object Domain Task Force BODTF-RFP 1 Submission. OMG Document bom/97-11-09, 1997.
4. Electronic Data Systems Corporationate Software. *Submission to Request for Proposals OMG Workflow Facility*, 1997. OMG Document bom/97-08-06, 1997. (available from www.omg.org)
5. C. Ellis, K. Keddara, G. Rozenberg. *Dynamic Change Within Workflow Systems.* In Proc. Conference on Organizational Computing Systems (COOCS), Milpitas, CA 1995, 10–22.
6. D. Georgakopoulos, M. Hornick, A. Sheth. *An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure.* Distributed and Parallel Databases, 3:119–153, 1995.
7. J. Hagemeyer, T. Herrmann, K. Just-Hahn, R. Striemer. *Flexibility in Workflow Management Systems (in German).* Software-Ergonomie '97, 179–190, Dresden, March 1997.
8. IBM. *IBM FlowMark: Modeling Workflow, Version 2 Release 2.* Publ. No SH-19-8241-01, 1996.

9. Intelligent Systems Technology Inc. *Submission to Request for Proposals OMG Workflow Facility*, 1997. OMG Document bom/97-08-07, 1997. (available from www.omg.org)
10. Y. Ioannidis (ed.). *Special Issue on Scientific Databases.* Data Engineering Bulletin 16 (1) 1993.
11. Iona. *Programming Guide Orbix 2.* Iona Technologies PLC, March 1997
12. Jablonski, S., Bußler, C.: *Workflow-Management: Modeling Concepts, Architecture and Implementation* International Thomson Computer Press, 1996
13. F. Leymann, W. Altenhuber. *Managing Business Processes as an Information Resource.* IBM Systems Journal 33, 1994, 326–347.
14. C.B. Medeiros, G. Vossen, and M. Weske. *WASA: A workflow-based architecture to support scientific database applications (Extended Abstract).* In *Proc. 6th DEXA Conference*, London, Spinger LNCS 978, 574–583, 1995.
15. Northern Telecom. *Submission to Request for Proposals OMG Workflow Facility.* OMG Document bom/97-08-04, 1997. (available from www.omg.org)
16. OMG. *CorbaServices: Common Object Services Specification.* (available from www.omg.org)
17. OMG: *Workflow Management Facility: Request for Proposals.* OMG Document cf/97-05-06, 1997 (available from www.omg.org)
18. Rational Software et al. *Unified Modeling Language – UML Notation Guide. Version 1.1*, September 1997. (available from www.rational.com/uml)
19. M. Reichert, P. Dadam. *Supporting Dynamic Changes of Workflows Without Loosing Control.* To appear: Journal of Intelligent Information Systems, Special Issue on Workflow and Process Management, Vol. 10, No. 2, 1998.
20. Sheth, A., Kochut, K.J.: *Workflow Applications to Research Agenda: Scalable and Dynamic Work Coordination and Collaboration Systems.* NATO ASI Workshop, Istanbul, August 12–21, 1997. To appear in Springer ASI NATO Series
21. W. Schulze. *Evaluation of the Submissions to the Workflow Management Facility RFP.* OMG Document bom/97-09-02, 1997.
22. A. Sheth, D. Georgakopoulos, S.M.M. Joosten, M. Rusinkiewicz, W. Scacchi, J. Wileden and A. Wolf. *Report from the NSF Workshop on Workflow and Process Automation in Information Systems.* Technical Report UGA-CS-TR-96-003 University of Georgia, Athens, GA, 1996.
23. J. Siegel. *Corba – Fundamentals and Programming.* John Wiley, 1996.
24. G. Vossen, M. Weske. *The WASA Approach to Workflow Management for Scientific Applications.* NATO ASI Workshop, Istanbul, August 12-21, 1997. To appear in: Springer ASI NATO Series.
25. M. Weske, D. Kuropka, J. Hündling, H. Schuschel. *Design of a Flexible Workflow Management System for Corba Architectures. (in German)* Technical Report Angewandte Mathematik und Informatik 18/97-I, Universität Münster, 1997.
26. M. Weske, G. Vossen. *Workflow Languages.* To appear in: P. Bernus, K. Mertins, G. Schmidt (Editors): Handbook on Architectures of Information Systems, Springer, 1998.
27. M. Weske. *Flexible Modeling and Execution of Workflow Activities.* In Proceedings of 31st Hawai'i International Conference on System Sciences, Software Technology Track (Vol VII), 713-722. IEEE Computer Society Press, 1998.

This article was processed using the LaTeX macro package with LLNCS style