Flexible Worklist Management in a Light-Weight Workflow Management System

Jeanine Weissenfels, Peter Muth, Gerhard Weikum

University of the Saarland, Department of Computer Science
P.O. Box 151150, D-66041 Saarbrücken, Germany
E-Mail: {weissenfels, muth, weikum}@cs.uni-sb.de, WWW: http://www-dbs.cs.uni-sb.de

Abstract

Building a universal workflow management system, if feasible at all, is achievable only at the expense of creating a heavy-weight system that needs to provide a very broad spectrum of functionality. For many application scenarios, such a system is too expensive, creates too much overhead or simply does not perform well enough. In this paper, we advocate a light-weight system architecture, consisting of a small system kernel on top of which extensions to the kernel are implemented as workflows themselves. This approach provides a seamless integration of kernel extensions like worklist management, history management etc.. The functionality of the kernel such as distributed workflow execution and interoperability interfaces can be used by all extensions. We show the feasibility of our approach by presenting the implementation of worklist management in our workflow specification language, based on state and activity charts, on top of our light-weight kernel, coined Mentor-lite.

1 Introduction

Many of the workflow management systems today are designed with a given application class in mind. Others claim to be well-suited for virtually every kind of workflow application. However, such universality, if feasible at all, is achievable only at the expense of building a heavy-weight system that provides a very broad spectrum of functionality. In this paper we advocate a light-weight system architecture that is based on a “slim” workflow management kernel, yet can be adapted to the full spectrum of applications through appropriate extensions. To retain the light-weight nature also for the extended systems as much as possible, we pursue an approach that makes extensive use of the kernel functionality to implement the higher-level extensions. In particular, we argue that workflow administration functionality like worklist management should itself be implemented as a workflow. This novel technique distinguishes our approach from earlier proposals towards light-weight and adaptable architectures.

Since there is a large variety of workflow application classes such as banking and financial trading, health care, or cooperative design, posing fairly different requirements on workflow management systems, it is necessary to identify and categorize dimensions of the workflow management systems architecture:

- **Functionality:** The functionality of a workflow engine is essentially defined by the scope of the workflow specifications. While some specifications minimize their scope to the aspects that are relevant for the enactment of activities by focusing on the control flow between activities, other workflow specifications extend the scope by including data flow, organizational aspects, detailed information on application programs, and possibly further dimensions of a comprehensive enterprise model.
- **Dynamics:** The functionality of the workflow engine also depends on whether processes are completely defined by static “production workflow” specifications, or on the extent to which dynamic changes of the specification need to be supported at runtime. The spectrum of dynamics in workflow specifications ranges from dynamically choosing the number of parallel branches that

This work was performed within the research project “Architecture, Configuration, and Administration of Large Workflow Management Systems” funded by the German Science Foundation (DFG).

are instantiated with a given subworkflow (e.g., a credit approval requiring two vs. three authorized signatures) to user-generated additional work items which in turn may contain additional control flow information for a dynamically created subworkflow.

- **Interoperability and Decentralization**: Some workflows demand decentralizing the workflow execution for organizational reasons, for example, when a workflow spans multiple, largely autonomous business units within the same or even different enterprises. It may therefore be necessary that different workflow engines cooperate on behalf of such a distributed workflow. When these engines are heterogeneous (i.e. come from different vendors, support different specification languages, etc.), the engines must meet high demands for interoperability. In simple cases, the capability for remote invocations of subworkflows may be sufficient, whereas more advanced applications would require tighter forms of synchronization such as one engine waiting for an event created by another engine.
- **Monitoring and Administration:** A convenient workflow management system must also provide monitoring and administration tools. But again there is a wide spectrum of possible requirements, upon which the necessary system support (and overhead) would depend. For example, worklist management may be as simple as sharing a single pool of work items among the employees of a small startup company, whereas large enterprises may wish to enforce sophisticated policies for the assignment of work items (e.g., taking into account competence levels, expertise and experience, availability, current workload, etc. of the employees).

The above discussion of the architectural dimensions of a workflow management system clearly shows that there are major tradeoffs. We cannot expect a single, monolithic system to be ideal for the full spectrum of requirements. So rather than aiming to build a panacea, we advocate an architectural approach that starts out with a small kernel engine and allows extension components to be added when necessary. This allows adapting the overall system to the specific demands and constraints of the application, trading off advanced system features for lower overhead when appropriate.

As mentioned above, most workflow management systems, both products and reseach prototypes, are rather monolithic and aim at providing full fledged support for the widest possible application spectrum [1, 7, 8, 13, 20]. On the other hand, the idea of extensible and adaptable system architectures is not new. Extensible database systems have been pursued in research and development for a decade, with mixed success but wide agreement that this is the right architectural paradigm [5]. A similar paradigm is being persued for constructing highly adaptable, distributed middleware [2, 3]. In the area of workflow management, however, the need for extensible and adaptable architectures has not received much attention so far, with the most notable exceptions being the Mobile [12, 13], Meteor$_2$ [6, 18], INCA [4], and WASA [21] projects. Our specific approach towards such an extensible, light-weight system, namely, viewing major extension components as workflows themselves, is, to the best of our knowledge, a novel idea. This paper provides an initial discussion of the approach; further research and especially implementation efforts along these lines are in progress.

The rest of the paper is organized as follows. Section 2 presents our approach to a light-weight workflow management kernel system, coined *Mentor-lite*. As an example for extending the kernel, we present the design and implementation of worklist management. Section 3 discusses the tasks of worklist management including some basic strategies. Section 4 shows how worklist management is implemented on top of the Mentor-lite kernel. Section 5 concludes the paper.

2 Mentor-lite - A Light-weight Workflow Management System

In this section we present our approach to a light-weight and tailorable workflow management system. It is based on the Mentor workflow management system [14, 15, 22, 23], but aims at a simpler architecture. The main goal is to provide only kernel functionality inside the workflow engine, and consider system components like history management and worklist management as extensions on top of the kernel. The key point to retain the light-weight nature is that these extensions are im-

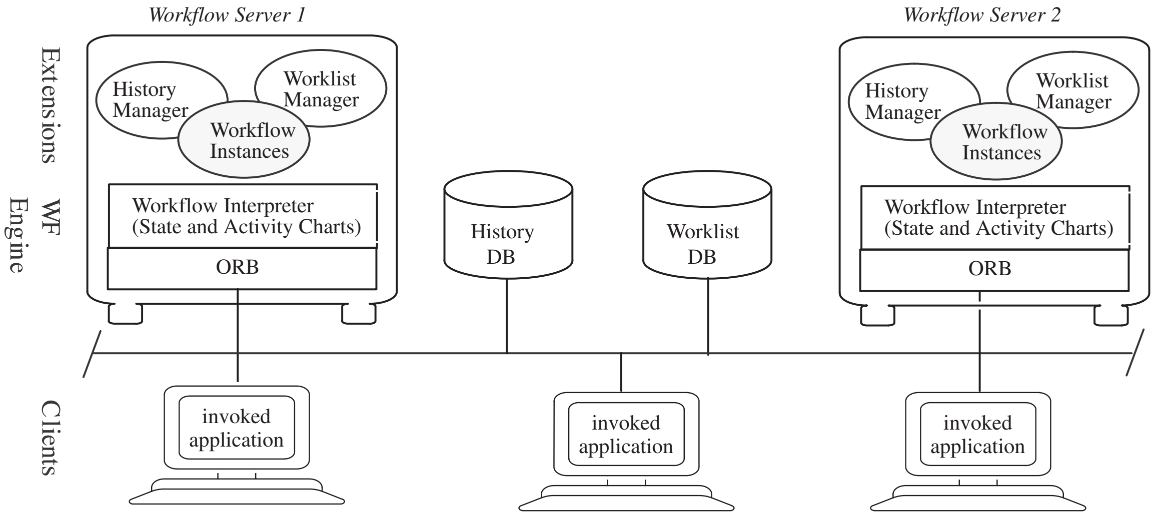

Fig. 1 : Components of the Mentor execution environment

plemented as workflows themselves. We have coined our new approach Mentor-lite.

As shown in Fig. 1, a workflow is orchestrated by appropriately configured servers, while the invoked applications of a workflow's various activities are run at client sites (from which they can in turn access other servers). The basic building block of Mentor-lite is an interpreter for workflow specifications. The specifications are given in terms of state and activity charts, the specification language already used in Mentor. The interpreter executes the workflow specification stepwise according to its formal semantics. For each step, the activities to be performed by the step are determined and started. In contrast to Mentor, which initially included a code-generation phase, specifications in Mentor-lite can be dynamically modified.

The Mentor-lite interpreter for state and activity charts employs standard middleware, specifically CORBA [16], to invoke applications and to exchange control flow information with other Mentor engines on remote nodes.

2.1 State and Activity Charts

In this section we briefly describe the formalism of state and activity charts [9, 10, 11]. This specification formalism has been adopted for the behavioral dimension of the UML industry standard [19]. State and activity charts comprise two dual views of a specification.

Activities reflect the functional decomposition of a system and denote the "active" components of a specification; they correspond directly to the activities of a workflow. An activity chart specifies the data flow between activities, in the form of a directed graph with data items as arc annotations.

State charts capture the behavior of a system by specifying the control flow between activities. A state chart is essentially a finite state machine with a distinguished initial state and transitions driven by Event-Condition-Action rules (ECA rules). Each transition arc between states is annotated with an ECA rule. A transition from state *X* to state *Y*, annotated with an ECA rule of the form *E[C]/A*, fires if event *E* occurs and condition *C* holds. The effect is that state *X* is left, state *Y* is entered, and action *A* is executed. Conditions and actions are expressed in terms of variables, for example, those that are specified for the data flow in the corresponding activity chart. In addition, an action *A* can explicitly start an activity, expressed by *st!(activity)*, and can generate an event *E* or set a condition *C*. Each of the three components of an *E[C]/A* triple may be empty. Every state change in a state chart execution is viewed as a single step; thus, state changes induce a discrete time dimension.

Two important additional features of state charts are nested states and orthogonal components. Nesting of states means that a state can itself contain an entire state chart. The semantics is that upon entering the higher-level state, the initial state of the embedded lower-level state chart is automatically entered, and upon leaving the higher-level state all embedded lower-level states are left. The capability for nesting states is especially useful for the refinement of specifications during the design process and for inserting predefined macros. Orthogonal components denote the parallel execution of two state charts that are embedded in the same higher-level state (where the entire state chart can be viewed as a single top-level state). Both components enter their initial states simultaneously, and the transitions in the two components proceed in parallel, subject to the preconditions for a transition to fire.

2.2 Extensions on Top of the Workflow Engine

The Mentor-lite workflow engine only provides a basic building block for workflows. It executes workflow specifications given as state and activity charts. Tools for managing worklists, for monitoring workflows, for alerting administrators when deadlines are violated, etc. are implemented on top of the engine as state and activity charts themselves. This allows us to use the functionality of the Mentor-lite workflow engine through a single interface, namely the state chart and activity chart interpreter.

Consider, for example, the worklist manager. If complex strategies like load balancing are used, assigning work items to actors is a workflow by itself. For documentation purposes, maintaining a history of this workflow might be required, and if multiple departments are involved, the workflow might even have to be decentralized. Instead of reimplementing all the necessary functionality in the worklist manager, the functionality of the Mentor-lite workflow engine can be used. This makes the worklist manager much easier to design and to implement. In addition, all components on top of the engine immediately benefit from enhancements of the engine, e.g., in terms of scalability and availability, and the underlying middleware components.

If necessary, components on top of the workflow engine may use the engine in different modes. For example, if an electronic commerce application requires the best possible security, special encryption schemes can be used for communication. In other applications, the resulting overhead could be unacceptable and a non-secure communication mode may be sufficient.

In the next section, we discuss worklist management as a detailed example for implementing extensions on top of the Mentor-lite workflow engine.

3 Towards Flexible Worklist Management

Worklist management is a key component of workflow management systems that execute complex workflows involving a large number of actors. The major task of worklist management is to assign work items to actors based on a specific assignment strategy, also known as "role resolution". The lifecycle of such a work item comprises three phases:

(1) *Work item creation*: A work item is created when the workflow engine launches an activity specified in the *E[C]/A* label of a state transition, and the activity has to be performed by a human actor. The new work item is inserted into the list of work items handled by the worklist manager.

(2) *Work item assignment:* In the second phase, the work item is assigned to one or more actors. The actors are determined by the worklist manager based on the worklist management strategy defined for the work item.

(3) *Work item selection and execution:* In the third phase, an actor selects the work item from her worklist and finally executes the corresponding activity. After successful execution, the work item is removed from the list of work items maintained by the worklist manager.

The assignment of work items to actors as performed in phase 2 is subject to specific assignment strategies. Work items can be categorized such that all work items of a given category are subject to the same strategy, or can be treated individually. A simple example strategy might be:

For a given work item, find out all actors who are capable of performing the role, have the necessary permissions, and are currently available (e.g., logged in to the system, as opposed to being on vacation).

Usually, strategies are more complex and implement specific business requirements such as the principle of dual control. If a dual control strategy for signing a document is used, it must be guaranteed that two different actors are chosen for two signatures.

It is impossible to model all conceivable assignment strategies in advance. In an enterprise there may exist as many different strategies as workflow types. Because we model the assignment strategies as workflows, different strategies can easily be implemented in Mentor-lite. The strategies may depend on various parameters such as workflow type, the organizational structure of the enterprise, the actors' expertise, the actors' load etc.. In Mentor-lite, a library of basic assignment strategies is made available to the workflow designer. By combining basic strategies, complex ones can be implemented. Even building specific strategies from scratch is conveniently supported, as a single specification language for both workflow specifications and assignment strategies is used.

Because other components like the history manager are also implemented in terms of workflows, using history data for assignment strategies is straightforward. This allows choosing actors based on their work on past worfklows, for example to find the most experienced actor for an important, but rare workflow type.

Assignment Strategies

In the simplest case, we could define a single assignment strategy for each workflow type, such that all work items created in a workflow of this type are subject to this strategy. In more complex workflow types however, each work item can be subject to a separate strategy. Strategies can be assigned statically at the time the workflow is specified, or at runtime when the work item is created.

Depending on the actual strategy, different parameters have to be considered.

- **Organizational Structure of the Enterprise:** In business processes we have to deal with organizational structures (e.g. departments or branch offices) and a hierarchy among the employees (e.g. simple employee vs. department head). The assignment of work items to actors has to consider the organizational structure. This is usually modelled by the concept of roles. Each work item carries a set of roles. An actor can only be chosen to perform a work item if she can perform the role assigned to the work item. Most organizations already use some kind of organizational database. This database maintains the assignment of roles to actors and can be accessed by the worklist management.

- **Actors' Expertise and Experience:** The organizational structure of an enterprise only determines an actors' formal competence, but not her expertise and experience. An important work item should be performed by the actor with the highest level of expertise, whereas less important work items can also be performed by rather inexperienced actors. The history of workflows is a good source of information for that purpose. It allows the worklist management to determine the number of times an actor has performed a similar work item in the past, and how successful the actor was in performing work items of that class.

- **Actors' Availability and Load:** In workflows where quick processing of work items is crucial for the enterprise, the worklist management must take the availability of actors into account. Illness and vacations have to be considered. An advanced worklist management must also provide load balancing facilities, i.e. there must be a way to distribute work items evenly among a set of candidate actors. The load balancing might be as simple as round robin but will more likely be based on complex information, e.g. on the individual efficiency of the actors or on former work item assignments.

- **Workflow-Instance-Specific Restrictions:** A powerful worklist management must be able to handle restrictions based on workflow instance specific data like customer names or product-IDs. For instance, based on bad experiences in the past a customer may wish not to be served by a certain actor.

The worklist manager must be able to implement strategies which take all these parameters into account. Implementing the strategies as workflows provides for a seamless integration of the worklist management into the system. In addition, it supports the administration of the worklist management by using tools of the specification environment.

4 Implementation of Worklist Management Strategies

In this section we present a worklist management implementation based on the architecture presented in Section 2.

4.1 Building Assignment Strategies

Assignment strategies for work items will be specified as state and activity charts, i.e. the assignment strategies are interpreted by the workflow engine. The activity chart defines activities used in the strategy and the data flow between them. The activities usually consist of queries against organizational databases in order to select qualifying actors. In general, all kinds of activities are allowed, even if they create new work items. This should happen only for rare cases where determining an assignment requires human activities, e.g., when a good assignment is extremely critical and should be done manually by a manager. The state chart corresponding to the activity chart defines the control flow for the assignment strategy. Choosing the appropriate assignment strategy for each activity requires human interaction, while the task of integrating the assignment strategies into the original workflow specification can easily be automated. In the following, we present the implementation of three example strategies.

1.) Load Balancing

The state and activity charts of Fig. 2 and Fig. 3 implement the following assignment strategy, which takes the current load of actors into account:

Example 1: *Find all actors that are capable of performing the activity and are currently available. Among those, assign the work item to the actor with the lowest current workload.*

The activity chart of Fig. 2 consists of activities *PQ1* to *PQ4*. They are implemented by predefined queries against an organizational database in order to select an appropriate actor. Such queries constitute a generic toolset for implementing assignment strategies. The database schema can be sketched as follows:

Actors (ActorID, Actor, Status, Workload, ...)
PlaysRole (ActorID, Role, WFType, ...)

The first query *PQ1* determines all actors who are able to perform the role *role* that is required by the work item. Relation *PlaysRole* of the organizational database contains the necessary information. A set of *ActorIDs* is returned (in terms of a temporary relation *Temp1*):

insert into *Temp1* (*ActorID*) select *ActorID*
from *PlaysRole* where *Role*=$role

The second query *PQ2* checks the availability of a set of actors stored in relation *Temp1* by checking the status attribute in the *Actors* relation and returns a set of *ActorIDs*:

insert into *Temp2* (*ActorID*) select *ActorID*
from *Actors, Temp1*
where *Status*="available" and
Actors.ActorID = Temp1.ActorID

The values of the *Status* attribute may themselves be the result of appropriate queries against an employee-vacation database; so one may regard relation *Actors* as a view.

Query *PQ3* determines the actor with the lowest workload among a set of actors. Query *PQ4* counts the number of tuples stored in a relation and stores the result in the variable *#tup*. For space reasons, we do not list the corresponding SQL statements.

The state chart of Fig. 3 shows how the queries are used in the assignment strategy. Queries *PQ1*, *PQ2* and *PQ3*

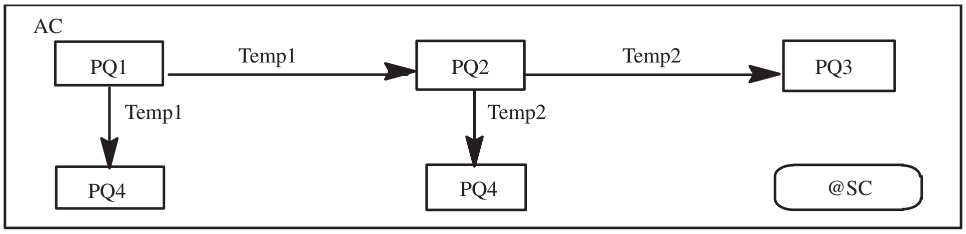

Figure 2: Activity chart for the assignment strategy of Example 1

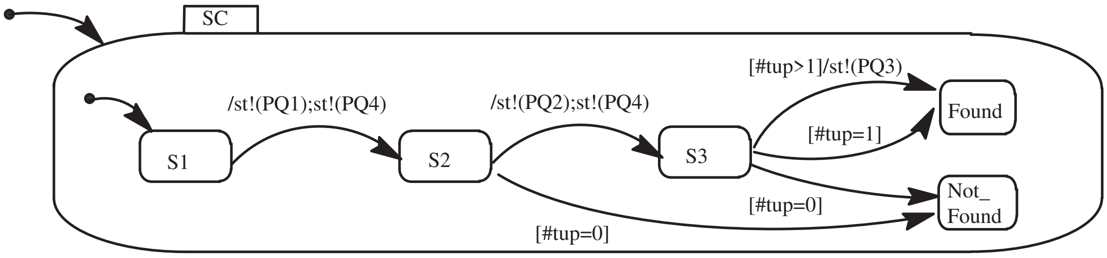

Figure 3: State chart for the assignment strategy of Example 1

perform the filtering of actors. *PQ3* is called only if more than a single *ActorID* is delivered by query *PQ2*. After the execution of queries *PQ1* and *PQ2*, *PQ4* is used to check whether the set of qualifying actors is already empty. In this case, state *Not_Found* is entered, indicating that no qualifying actor was found. If a qualifying actor was found, state *Found* is entered.

If a complex strategy is frequently used, its execution time might become a problem. In this case, the assignment strategy or at least parts of it can be transformed into larger SQL queries or stored procedures. This provides for a more efficient execution as less queries are executed, and the remaining ones can be better optimized by the query optimizer of the database. However, compared to using simple and generic queries inside of larger state and activity charts, such an implementation is harder to build and harder to maintain in case of modifications.

2.) Principle of Dual Control

Now assume that for signing an important document by two actors, we need to implement the principle of dual control, i.e., we have to make sure that the document is signed by two different actors. Each actor should be determined by the assignment strategy as shown above. We assume that the signatures have to be given by actors playing different roles, so only an actor allowed to fill both roles is a candidate for violating the principle of dual control. We assume this case to be rare. Hence, for performance reasons, it is beneficial to look for both candidate actors in parallel.

Example 2: *Find two actors according to the strategy of example1. Make sure that the resulting actors are different.*

We implement the strategy of example 2 by the state chart shown in Fig. 4. It consists of two copies of the state chart of Fig. 3, and a final check that the deter-

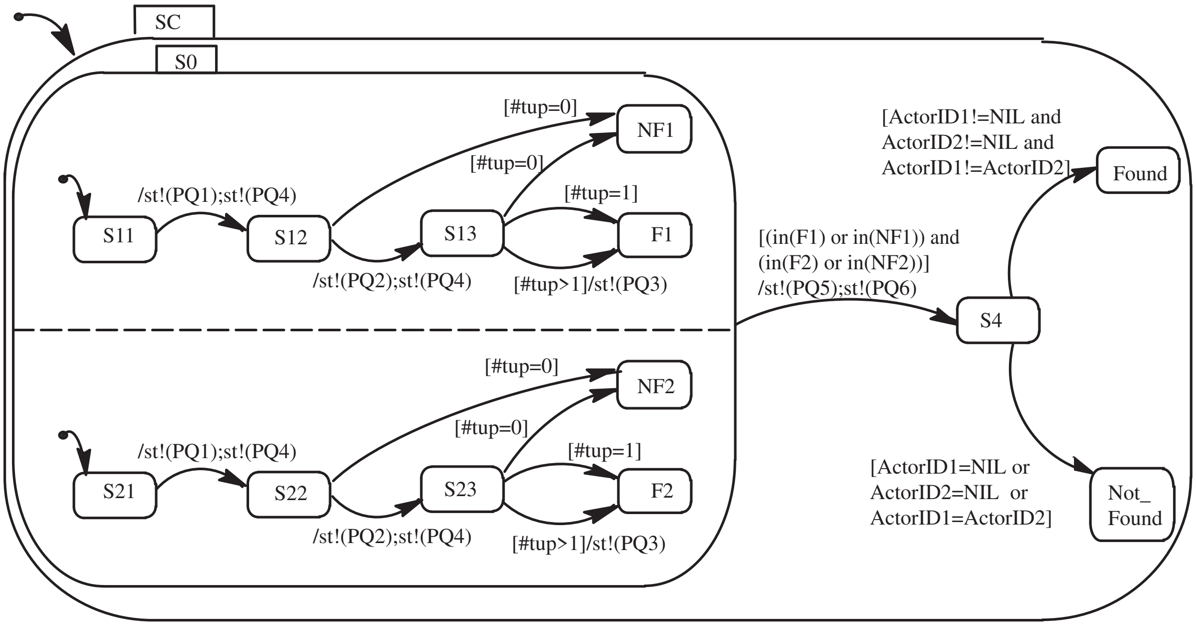

Figure 4: State chart for dual control

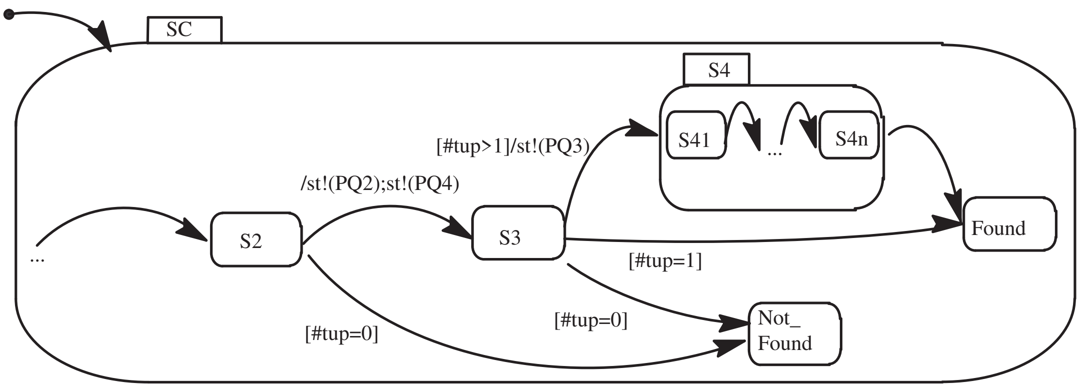

Figure 5: Extended state chart for the assignment strategy of Example 1

mined actors are different people. The dotted line indicates a parallel execution. The condition on the transition to state *S4* guarantees that the check for inequality is only performed after both actors are found, or after it turned out that there are not enough qualifying actors. Queries *PQ5* and *PQ6* return the *ActorID*s for the inequality test. For the sake of simplicity, we do not show how to proceed in case the inequality test failed. In general, we would then restart the search for one of the actors, excluding the other actor. Building the corresponding activity chart is straightforward, we do not present it for space reasons.

3.) Using the history manager

Finally assume that instead of performing load balancing on the actors load as done in the state chart of Fig. 3 by executing query *PQ3*, we want to choose the actor which has performed the least number of workflows of the same kind in the last three months. This could be reasonable in order to keep the actors at the same level of training. To determine this actor, we have to consult the history manager. If an appropriate SQL query is provided by the history manager, we simply substitute query *PQ3* by the new query. However, for complex queries it is unlikely that the history manager alone can provide the full solution. In this case, we have two choices. We can either implement the query as part of the history managers' generic toolset of activities, or use simple queries already in the toolset together with an appropriate state and activity chart. Instead of executing query *PQ3* in Fig. 3, we can insert a new state *S4* between state *S3* and state *Found.* State *S4* is itself a statechart, namely the one provided by the history manager. Fig. 5 shows a part of the resulting statechart.

4.2 Integrating Assignment Strategies into Workflows

As the assignment strategies are implemented in terms of state and activity charts, we integrate worklist management into workflow specifications by including the assignment strategies into the workflow specifications. Whenever a workflow specification contains an activity which has to be performed by a human actor, we have to make sure that the worklist management assigns the corresponding work item to an actor. Fig. 6 shows the basic scheme. Assume that activity *a* in Fig. 6a has to be performed by a human actor. The inclusion of an assignment strategy is shown in Fig. 6b. The target state *S2* becomes a nested state, with the state chart defining the assignment strategy inside of *S2,* represented by states *S21 ... S2n.* Activity *a* is no longer started when the incoming transition into *S2* fires. Instead, it is started inside of *S2*. In *S2*, after a qualifying actor has been assigned to the work item, the work item is inserted into her worklist. This is performed by activity *insertWL* of Fig. 6b. Finally, the actor selects the work item for execution by using a graphical worklist interface, which is not subject of this paper.

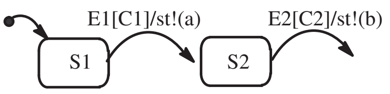

Figure 6a: Original specification

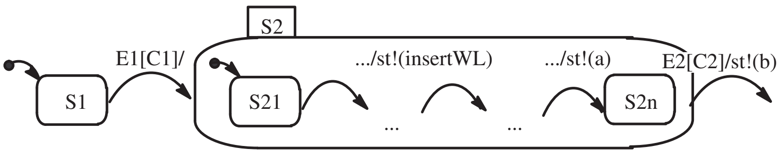

Figure 6b: Assignment strategy included

5 Conclusions and Future Work

In this paper, we have presented an approach for implementing workflow administration functionality on top of a light-weight system kernel. We argue that such extensions to the kernel should themselves be implemented as workflows. As an example, we have presented the implementation of worklist management in our prototype workflow management system, Mentor-lite. By using our approach, we gain the following advantages:

- Extensions to the kernel can easily be implemented in the same specification language as used for workflows.
- The extensions can be easily tailored to the needs of specific workflow types.
- There is only a single type of interface between the kernel and kernel extensions as well as between different kernel extensions: the workflow specification language.
- Functionality implemented in the kernel is automatically available to all extensions, e.g., distributed execution, reliable and fault tolerant execution, encryption etc.. This allows even a decentralized execution of assignment strategies for work items or a decentralized history management without additional implementation effort.
- The same holds for interoperability between different workflow management systems. Interfaces provided by the kernel are available to all extensions, providing interoperability, e.g., for the history management of different workflow management systems.

In Mentor-lite, we use state and activity charts as our specification language. However, the approach can, in principle, be carried over to workflow management systems using other specification languages.

In addition to worklist management and history management, other system components can be implemented and integrated in the same way, for example, an alerter indicating that during execution, certain conditions became true. Checking the conditions can be implemented by inserting orthogonal "watchdog" components into the state chart of the workflow.

Complex system extensions should provide a library of state and activity charts. This is the easiest way for the workflow designer to utilize their functionality. In the future, we will implement such libraries for Mentor-lite. We plan to investigate the feasibility of our approach for even more demanding system components such as alerting mechanisms based on time constraints.

Literature

[1] G. Alonso, D. Agrawal, A. El Abbadi, C. Mohan, Functionality and Limitations of Current Workflow Management Systems, IEEE Expert, Vol. 12, No. 5, 1997

[2] G. Alonso, C. Hagen, Geo-Opera: Workflow Concepts for Spatial Processes, in: M. Scholl, A. Voisard (ed.): Advances in Spatial Databases, 5th International Symposium, Lecture Notes in Computer Science, Vol. 1262, Springer, 1997

[3] G. Alonso, C. Hagen, H. J. Schek, M. Tresch, Towards a Platform for Distributed Application Development in [7]

[4] D. Barbara, S. Mehrotra, M. Rusinkiewicz, INCAs: Managing Dynamic Workflows in Distributed Environments, Journal of Database Management, Special Issue on Multidatabases, Vol. 7, No. 1, 1996

[5] M. J. Carey, D. J. DeWitt, Of Objects and Databases: A Decade of Turmoil, Proceedings of the 22nd VLDB Conference, 1996

[6] S. Das, K. Kochut, J. Miller, A. Shet, D. Worah, ORBWork: A Reliable Distributed CORBA-based Workflow Enactment System for $METEOR_2$, Technical Report #UGA-CS-TR-97-001, Department of Computer Science, University of Georgia, February 1997

[7] Ed.: A. Dogac, L. Kalinichenko, M. Tamer Ozsu, A. Sheth, Advances in Workflow Management Systems and Interoperability, NATO Advanced Study Institute, Springer, to appear 1998

[8] D. Georgakopoulos, M. Hornick, A. Sheth, An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure, Distributed and Parallel Databases, Vol. 3, No. 2 , 1995

[9] D. Harel, State Charts: A Visual Formalism for Complex Systems, Science of Computer Programming Vol.8, 1987, pp. 231-274

[10] D. Harel et. al, Statemate: A Working Environment for the Development of Complex Reactive Systems, IEEE Transactions on Software Engineering Vol. 16, No. 4, April 1990

[11] D. Harel, A. Naamad. The STATEMATE Semantics of State Charts, Technical Report, i-Logix Inc., October 1995

[12] S. Jablonski, MOBILE: A Modular Workflow Model and Architecture, Proc. of the 4th Int. Working Conference on Dynamic Modelling and Information Systems, Nordwijkerhout, 1994

[13] S. Jablonski, C. Bussler, Workflow Management, Modeling Concepts, Architecture, and Implementation, International Thomson Computer Press, 1996

[14] P. Muth, D. Wodtke, J. Weissenfels, G. Weikum, A. Kotz Dittrich, Enterprise-wide Workflow Management based on State and Activity Charts, in [7]

[15] P. Muth, D. Wodtke, J. Weissenfels, A. Kotz Dittrich, G. Weikum, From Centralized Workflow Specification to Distributed Workflow Execution, in: Journal of Intelligent Information Systems - Special Issue on Workflow Managament, Vol. 10, No. 2, March 1998

[16] OMG, The Common Object Request Broker: Architecture and Specification, Rev. 2.0, Technical Report, Object Management Group, 1995

[17] A. Sheth (ed.), Proceedings of the NSF Workshop on Workflow and Process Automation in Information Systems, Athens, GA., May 1996, http://lsdis.cs.uga.edu/activities/NSF-workflow/proc_cover.html

[18] A. Sheth, K. J. Kochut, Workflow Applications to Research Agenda: Scalable and Dynamic Work Coordination and Collaboration Systems, in [7]

[19] Unified Modeling Language (UML) Version 1.1, http://www.rational.com/uml/documentation.html

[20] G. Vossen (ed.), J. Becker (ed.), Busines Process Modelling and Workflow Managment, Models, Methods, and Tools, International Thomson Publishing, Bonn (in German), 1996

[21] G. Vossen, M. Weske, The WASA Approach to Workflow Management for Scientific Applications in [7]

[22] J. Weißenfels, D. Wodtke, G. Weikum, A. Kotz Dittrich, The MENTOR Architecture for Enterprise-wide Workflow Management. In: [17]

[23] D. Wodtke, J. Weißenfels, G. Weikum, A. Kotz Dittrich, The MENTOR Project: Steps Towards Enterprise-Wide Workflow Management, Proceedings of the International Conference on Data Engineering, New Orleans, 1996