The MENTOR Workbench for Enterprise–wide Workflow Management

Dirk Wodtke[1], Jeanine Weissenfels[1], Gerhard Weikum[1], Angelika Kotz Dittrich[2], Peter Muth[1]

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

[2] Union Bank of Switzerland
P.O.Box 2336, CH-8033 Zurich, Switzerland
E-mail: kotz-dittrich@ubs.ch

1 Overview of the Project

MENTOR ("Middleware for Enterprise–Wide Workflow Management") is a joint project of the University of the Saarland, the Union Bank of Switzerland, and ETH Zurich [1, 2, 3]. The focus of the project is on enterprise–wide workflow management. Workflows in this category may span multiple organizational units each unit having its own workflow server, involve a variety of heterogeneous information systems, and require many thousands of clients to interact with the workflow management system (WFMS). The project aims to develop a scalable and highly available environment for the execution and monitoring of workflows, seamlessly integrated with a specification and verification environment.

For the specification of workflows, MENTOR utilizes the formalism of state and activity charts. The mathematical rigor of the specification method establishes a basis for both correctness reasoning and for partitioning of a large workflow into a number of subworkflows according to the organizational responsibilities of the enterprise. For the distributed execution of the partitioned workflow specification, MENTOR relies mostly on standard middleware components and adds own components only where the standard components fall short of functionality or scalability. In particular, the run–time environment is based on a TP monitor and a CORBA implementation.

2 Workflow Specification

In virtually all commercially available WFMS the specification of a workflow is done via high–level graphical interfaces by drawing "bubbles and arcs". This specification must be mapped into an internal representation in order to execute it. In most systems, the underlying internal representation uses an ad hoc model and thus lacks capabilities for formal correctness reasoning and interoperability between different WFMS since exchanging workflow specifications across different execution platforms is not possible. In contrast, general–purpose specification formalisms for dynamic systems such as Petri nets, state charts, temporal logic, or process algebras, which are pursued in various research projects and a few products, come with a rich theory and thus provide an excellent basis for formal proofs. For the MENTOR prototype we have adopted the method of state and activity charts [3] which is perceived by practitioners as more intuitive and easier to learn than Petri nets yet has an equally rigorous semantics.

SIGMOD '97 AZ,USA

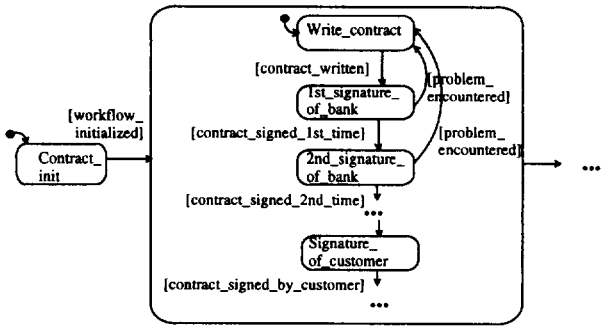

Fig. 1 : State chart for the processing of a loan agreement

An example for a simplified state chart describing a small portion of a workflow for the processing of a loan agreement in a bank is given in Fig. 1.

2.1 Verification of Workflow Specifications

With regard to the verification of workflow specifications, the validation techniques that are currently pursued in most of the workflow products are animation and simulation [9]. However, if the number of possible states of the workflow is very large, simulation techniques face inherent limitations as exhaustive coverage becomes computationally infeasible. An alternative verification technique that has achieved significant results in the area of reactive systems is model checking, especially symbolic model checking [8]. The major benefits that can be obtained by applying symbolic model checking are as follows: First, the complete set of necessary test patterns is generated automatically. Second, from the user's point of view the formal correctness proof is carried out in a push–button fashion, thus, avoiding tedious developing of test patterns and testing. Finally, by generating only a symbolic, i.e., very compact representation of the model the size of the models that can be verified increases remarkably.

Since in general state chart specifications are amenable to symbolic model checking critical workflow properties can be formally verified. However, before the validity of properties can be checked the interesting properties have to be specified formally. In addition to the specification of generic properties (e.g., reachability of states, existence of nondeterminism, etc.), temporal logic is an adequate specification language for expressing properties. Temporal logic extends propositional and/or predicate logic by temporal operators. For example, a temporal formula may state that if property p was true in the past, property q will be true sometime in the future.

Among the spectrum of temporal logic variants that have been studied in the literature, the computational tree logic (CTL) [4] offers a good compromise between expressive power and complexity of the model checking problem. Therefore, the MENTOR proto-

```
sc2bdd -f credit.sc
...
mc4sc -sc credit.sc
...
ctl> A G (Signature_of_customer => ( A G ¬
( 1st_signature_of_bank | 2nd_signature_of_bank )));
...
tautology
(time : 0.356982 sec.)
```

Fig. 2 : Example of a model checking session

type uses CTL to express properties for verification purposes. CTL formulas consist of atomic propositions. Each proposition corresponds to a variable in the model, boolean connectives, $\neg$ and $\wedge$, and operators. Each operator consists of a path quantifier, A or E, and a temporal operator. The universal path quantifier A stipulates that a temporal property must hold over all possible computation paths; the existential path quantifier E stipulates that a temporal property holds along some computation path. Each temporal operator (X, F, G, U) is preceded by one of the two quantifiers. If p is a formula in CTL, then Xp ("neXt p") means that p holds in the next state, Fp ("Finally p") means that p holds sometime in the future, Gp ("Globally p") means that p holds globally in all future states, pUq ("p Until q") means that q holds in the future, and p holds in all states until the state in which q holds.

An interesting and critical property of the loan processing workflow would state that before a customer is allowed to sign the loan agreement, it has to be signed by two authorized bank clerks, i.e., the signatures of the bank clerks must not be given after the customer's signature. Basically, this is an order dependency between two events, *e1* and *e2*. This order dependency is expressed by the generic CTL macro, "AG (*e2* => AG $\neg$ *e1*)". The macro states that whenever event *e2* has been generated event *e1* can never be generated. By instantiating this macro the CTL formula given in Fig. 2 can be obtained. As it is shown in the figure, this CTL formula serves as the input for the model checking tool. The output of the model checking session which finally produces the (correct) test result "tautology" is illustrated in the figure. The test result "tautology" means that the property that the signatures of the bank clerks must not be given after the customer's signature holds on all execution paths.

2.2 Design Method for Workflow Specifications

Being well aware of the fact that state and activity charts may be perceived as too formal and therefore, inappropriate for wide use in a business environment, the organizational staff of an enterprise would rather prefer high–level design tools for business process engineering or re–engineering (BPR). These tools are geared for ease of use in the design process itself but cannot produce executable workflows. With this in mind, the MENTOR prototype implements the following workflow design method (which may be iterated if necessary):

(1) Workflow or business process specifications are initially developed using a BPR tool (ARIS Toolset in our prototype) or the specification tools of the user's favorite commercial workflow system (FlowMark in our prototype).
(2) The specification is automatically converted into a state and activity chart specification.
(3) Critical properties of the state and activity chart specification are verified by applying model checking techniques.
(4) The workflow specification is partitioned for distributed execution according to the method given in [3].

Thus, the role of state and activity charts in a practical business environment, as we view it, is that of a canonical representation for the underlying execution engine (with rigorously defined semantics), into which other specifications can be converted and which may even serve as an exchange format across different workflow engines. Mapping various specification methods onto the state and activity chart formalism is a contribution towards decoupling the specification environment from the execution environment as it is suggested by the reference model of the Workflow Management Coalition (WfMC) [10]. Reconciling the concepts of business process reengineering and of workflow management is a significant step towards increasing the efficiency and thus productivity in the specification of workflows.

3 Distributed Workflow Execution

Presently marketed WFMS are designed for only a small number of concurrent workflow executions. However, actual requirements for complex, large–scale applications are orders of magnitude higher [5, 7, 9]. Such applications may involve a large number of concurrent workflow instances which impose a high load on the workflow engine. Consequently, scalability and availability considerations dictate that the overall workflow processing is distributed across multiple workflow engines running on different servers. This workload partitioning may itself require the partitioning of individual workflows.

Therefore, the MENTOR execution environment is based on a distributed client–server architecture. Each workflow instance is executed by a number of appropriately configured *MENTOR workflow servers*. On each workflow server there exists a *MENTOR workflow engine* which is responsible for interpreting the control flow specification of the workflow or executing the compiled control flow specification which has been loaded into the workflow engine. Complementary to the workflow engine, the *invoked applications* of a workflow's various activities are run at the client sites, where the applications may in turn issue requests to other servers (regardless of whether an application is invoked within a workflow or not). The data that the invoked applications are working on is not handled by the workflow engine. Thus, references to data objects (e.g., URLs) rather than the actual data are passed between activities, unless an individual data element drives the control flow (e.g., the amount of a credit request, but not the entire credit dossier). This design decision of MENTOR is due to scalability, security, and manageability reasons. Note that this does not preclude data sharing between activities, but it is up to the invoked applications to initiate the necessary remote server accesses to obtain the data.

In addition to the workflow engine, each workflow server consists of a set of commercially available standard middleware components (as suggested in [5]) along with our own "glueing" software. As depicted in Fig.3, the standard components are as follows:

- To cope with heterogeneity among the invoked applications, a CORBA–style *object request broker* is used to invoke application programs and pass parameters between the workflow engine and the applications. In the MENTOR prototype, this component is implemented with Orbix. All applications that are invokable during workflow executions must have been "wrapped" and provided with IDL interfaces.
- For fault tolerance reasons, all communication between servers is performed via a *TP monitor*, which is Tuxedo in the case of the MENTOR prototype. The TP monitor is needed for two reasons: First, we have to ensure the consistency of the global workflow state, even in the event of a failure. Secondly, we need a facility to monitor the components of our execution environment. The first problem is solved by using reliable message queues and ACID transactions embracing write and read operations on multiple local databases and message queues. The second problem is solved in that each workflow engine and all other components of the execution environment register with the TP monitor when they are started. In the event of a failure of such a registered server, the server is automatically restarted by Tuxedo.

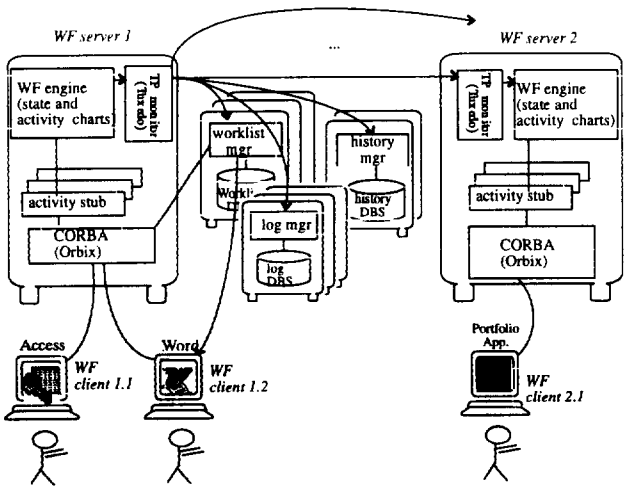

Fig. 3 : Distributed Execution Environment of MENTOR

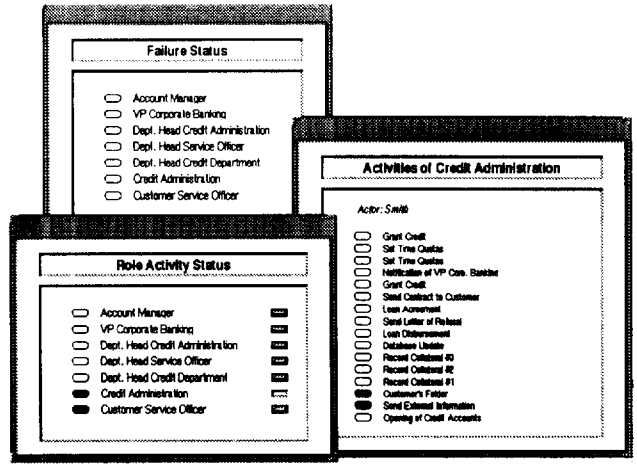

Fig. 4 : Screendump of the User Interface for Workflow Monitoring

As shown in Fig. 3 , three additional functional components are factored out from the actual workflow servers, as they may have to be configured in a more flexible and demanding way (e.g., enterprise–wide access also by non–workflow applications such as special analysis and assessment applications). Note that these components are part of many commercial workflow engines, but are typically supported only in a rudimentary form that is not appropriate for enterprise–wide workflow management.

- The *MENTOR log manager* records all changes to a workflow's state, events, and control–flow–relevant data elements. Multiple outgoing messages (e.g., if more than one "remote" activity is spawned) and the writing of log records are combined into a single atomic unit by using transactional calls to the TP monitor. In the case of a failure the log manager is responsible for recovering the crashed workflow engine and playing back the already performed steps of the workflow engine up to the last consistent state.
- The *MENTOR worklist manager* assigns the work items (i.e., activities that are ready for execution) among those actors that can fill the corresponding role. This assignment is based on information stored in the worklist database. The worklist manager maintains its own worklist database where information about role resolution policies, vacation periods, etc. is stored. Since the worklist manager should support also more advanced functionality like load balancing, time and priority management, it needs on–line access to the workflow history data, too. The counterpart of the worklist manager at a client site is the worklist itself, along with functions to select, reject, or suspend and resume work items.
- For monitoring purposes, state information about currently running as well as completed workflow instances is stored in a workflow history database where it is kept for both status inquiries and long–term evaluation and made accessible through the *MENTOR history manager.*

4 Monitoring of Workflow Execution

Currently, the WFMS on the market lack support for storing and querying the history of both ongoing and past workflow executions. However, data in this category is a major source of identifying inefficiencies and time bottlenecks from a business process reengineering perspective and are helpful for assessing the efficiency, accuracy, and timeliness of the enterprise's business processes. For example, a process engineer would be keen to learn about the (average, maximum, and total) duration of various activities, in order to identify opportunities for improvement. Workflows with a high turnaround time could then be reorganized by streamlining the control flow and the organizational responsibilities. In addition, systems lack a clean way for a processing entity that is involved in an ongoing workflow instance to get informed about the current status of the workflow instance. For example, a bank clerk who participates in a credit request workflow may want to check the current status of parallel activities, in order to assess the criticality with respect to deadlines.

Therefore, in MENTOR much effort has been invested in developing a concept for monitoring workflow executions. For monitoring purposes, state information about currently running workflow instances is stored in a workflow history database where it is kept for both status inquiries and long–term evaluation, and made accessible through the history manager. Moreover, the history database will serve as an information repository for the MENTOR worklist manager.

In order to support monitoring of workflow executions, all or a subset of the current information on states, events, and data items need to be visualized whenever an update occurs. The visualization is based on monitoring panels which mainly consist of displays that indicate relevant state information. MENTOR generates these user interfaces from the workflow specification and additional information on roles and other organizational aspects.

Figure 4 shows the realization of three monitoring panels in MENTOR for an example role that is involved in the execution of a workflow "capital investment loan": The first panel "Failure Status" indicates that none of the roles involved in the execution of the workflow has run into a failure situation. The second panel "Role Activity Status" indicates which execution roles are potentially involved in the execution of the workflow. Note that the set of execution roles that are actually involved in the execution of the workflow instance can only be determined during the execution of the workflow, since it may depend on the data that is fed into the execution of the corresponding workflow instance. The third panel "Activities of Credit Administration" shows the activities that are under the responsibility of the execution role "credit administration" and indicates which of these activities are currently executing there.

These displays help to increase transparency, flexibility and overall work quality. Note that only those displays would be shown for which the actor is indeed authorized, and that the display of information is subject to security policies as well. For example, it could be the case that if the worklist manager had selected a different actor (i.e., not "Smith") for resolving the role "credit administration" in this workflow instance, only a subset of the information given in the

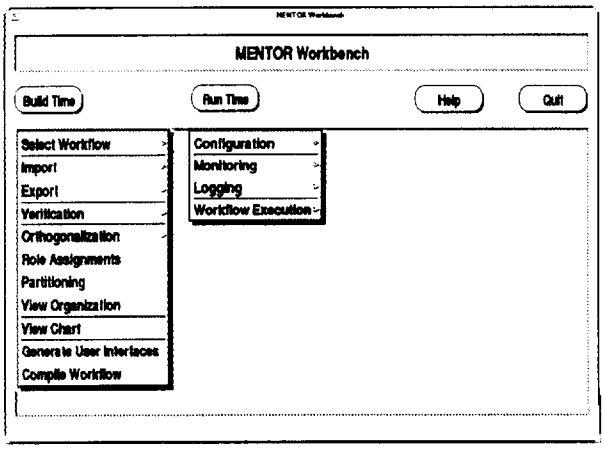

Fig. 5 : Mentor Workbench

figure should be displayed in the monitoring panels whenever the new actor has less access rights.

5 Project Status and Outlook

The aforementioned functionalities, i.e., automatic conversion of FlowMark specifications into state and activity charts, coupling with a model checking tool, distributed workflow execution and monitoring facilities have been implemented in a prototype of the MENTOR Workbench (see Fig. 5). The prototype has been implemented on SUN Sparcstations with Solaris 2.x. It integrates a model checking tool developed at the University of Oldenburg [6]. In addition, it integrates the commercial state chart tool STATEMATE which both performs simulations of state and activity charts and works as a code generator for the workflow engine. The prototype is based on the middleware components Tuxedo 5 and Orbix 1.3 and on the database system Oracle 7.2.

We are currently working on extending the query functionality of the history manager beyond standard SQL in order to run complex queries against the workflow history for on–line inquiries, alerting purposes and mining–style evaluation. Furthermore, we are developing an enhanced and more flexible worklist management component for the support of advanced work assignment policies.

Acknowledgement: We wish to thank Ralf Schenkel, Holger Bielenstein, Jörg Bur, Ulrich Hoffmann, Andreas Kläser, Katja Stichter, and Mark Wehrmann for their help in implementing this demo and the Union Bank of Switzerland for the support of the MENTOR project.

6 References

[1] D. Wodtke, J. Weissenfels, G. Weikum, A. Kotz Dittrich, The MENTOR Project: Steps Towards Enterprise–Wide Workflow Management, IEEE International Conference on Data Engineering, New Orleans, 1996

[2] J. Weissenfels, D. Wodtke, G. Weikum, A. Kotz Dittrich, The MENTOR Architecture for Enterprise–wide Workflow Management, in: 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

[3] D. Wodtke, G. Weikum, A Formal Foundation for Distributed Workflow Execution Based on State Charts, in: Proceedings of the International Conference on Extending Database Theory, Delphi, Greece, 1997

[4] E.A. Emerson, Temporal and Modal Logic, in: J. van Leeuwen (ed.), Handbook of Theoretical Computer Science, Elsevier, 1990

[5] D. Georgakopoulos, M. Hornick, A. Sheth, An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure, Distributed and Parallel Databases, 3(2), 1995

[6] J. Helbig, P. Kelb, An OBDD–Representation of Statecharts, Proceedings of the European Design and Test Conference, 1994

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

[8] K. L. McMillan, Symbolic Model Checking, Kluwer Academic Publishers, 1993

[9] C. Mohan, State of the Art in Workflow Management Research and Products, Tutorial Notes, ACM SIGMOD Conference, 1996

[10] Workflow Management Coalition, various publications, available at: http://www.aiai.ed.ac.uk/WfMC/