

ORBWork: A CORBA-Based Fully Distributed, Scalable and Dynamic Workflow Enactment Service for METEOR

Krys J. Kochut, Amit P. Sheth, John A. Miller
Large Scale Distributed Information Systems Lab
Computer Science Department, University of Georgia
Athens GA 30602-7404 USA
<http://lsdis.cs.uga.edu>

Second generation workflow management systems need to deal with heterogeneity of platforms within and across cooperating enterprises along with legacy applications and data. At the same time, there is increasing demand for advanced features for supporting mission-critical processes, including adaptability through dynamic changes and scalability. The workflow management system METEOR is based on open systems and standards and utilizes CORBA and Java. This allows METEOR to provide high-end workflow management combined with application and data integration capabilities in increasingly network-centric environments.

1. Introduction

Workflow management is the automated coordination, control and communication of work as is required to satisfy workflow processes [Sheth et al. 96]. A Workflow Management System (WfMS) is a set of tools providing support for the necessary services of workflow creation (which includes process definition), workflow enactment, and administration and monitoring of workflow processes [WfMC]. The developer of a workflow application relies on tools for the specification of the workflow process and the data it manipulates. The specification tools cooperate closely with the workflow repository service, which stores workflow definitions. The workflow process is based on a formalized workflow model that is used to capture data and control-flow between workflow tasks.

The workflow enactment service (including a workflow manager and the workflow runtime system) consists of execution-time components that provide the execution environment for the workflow process. A workflow runtime system is responsible for enforcing inter-task dependencies, scheduling tasks, managing workflow data, and ensuring a reliable execution environment. Administrative and monitoring tools are used for management of user and work group roles, defining policies (e.g., security, authentication), audit management, process monitoring, tracking, and reporting of data generated during workflow enactment.

A number of applications posing substantial challenges to the currently available WfMSs have been discussed in [Sheth and Kochut 98]. The applications demand that a WfMS be easily scalable and able to handle dynamic workflows. Moreover, a WfMS must be able to operate on a wide variety of hardware and software platforms and be able to incorporate legacy applications and data sources within the administered workflows. Such a WfMS must include suitable design and development tools that can be used to design a workflow and then dynamically introduce changes to the whole workflow process definition (schema), or even just individual workflow (instances). The system must also include a flexible enactment system, capable of supporting scalability, where new resources (computers, database servers, end-users, etc.) can be easily incorporated within the workflow system, and adaptive workflows-where workflow specification can be changed or extended, including addition or modifications of tasks and inter-task dependencies.

The METEOR project is represented by both the research system [METEOR], and a suite of commercial offering - METEOR¹ **Enterprise Application Suite** of tools and services (**EApps**)

¹ METEOR = Managing End-To-End Applications

[Infocosm], that addresses the above challenges by providing an open-systems based high-end workflow management solution as well as an enterprise application integration infrastructure. This article focuses on ORBWork, a METEOR's enactment service that exploits CORBA, Java and Web technologies in meeting the above challenges. In Section 2, we provide a brief overview of the METEOR WfMS by focusing on its architecture. In Section 3, we provide an overview of the ORBWork enactment service. Section 4 presents technical features of ORBWork, followed by a description of ORBWork's implementation in Section 5. The article ends with conclusions.

2. METEOR Architecture

METEOR's architecture includes a collection of four services: **EApp>Builder**, **EApp>Repository**, **EApp>Enactment**, and **EApp>Manager**. **EApp>Enactment** includes two services- **ORBWork** and **WebWork**. Both **ORBWork** and **WebWork** use fully distributed implementations. **WebWork** [Miller et al 98], an entirely Web-based enactment service, is a comparatively light-weight implementation that is well-suited for a variety of enterprise workflow process applications that involve limited data exchange and do not need to be dynamically changed. **ORBWork** (discussed in this article) is better suited for more demanding, mission-critical enterprise applications requiring high scalability, robustness and dynamic modifications. The overall architecture of the system is shown in Figure 1.

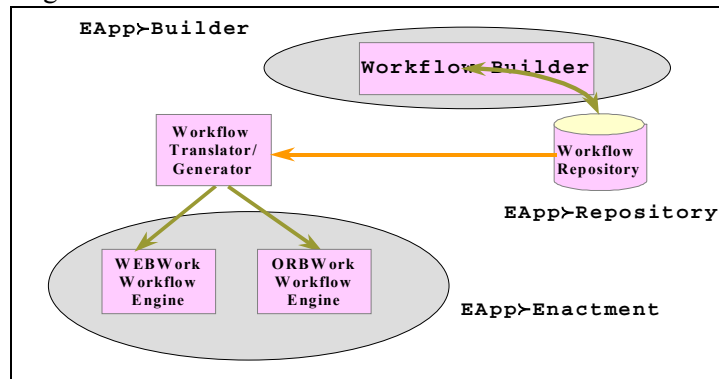


Figure 1: METEOR Architecture

2.1 Workflow Builder Service

This service consists of a number of components that are used to graphically design and specify a workflow, in some cases leaving no extra work after a designed workflow is converted to a workflow application by the runtime code generator. Its three main components are used to specify the entire map of the workflow, data objects manipulated by the workflow, as well as the details of task invocation, respectively. The task design component provides interfaces to external task development tools (e.g., Microsoft's FrontPage to design the interface of a user task, or a rapid application development tool). This service supports modeling of complex workflows consisting of varied human and automated tasks in a conceptual manner using easy to use tools. In particular, the designer of the workflow is shielded from the underlying details of the infrastructure or the runtime environment. At the same time, very few restrictions regarding the specification of the workflow are placed on the designer.

The workflow specification created using this service includes all the predecessor-successor dependencies between the tasks as well as the data objects that are passed among the different tasks. It also includes definitions of the data objects, and the details of the task invocation details. The specification may be formatted to be compliant with the Workflow Process Definition Language (WPDL) of the Workflow Management Coalition [WfMC]. This service assumes no particular implementation of the workflow enactment service (runtime system). Its independence from the runtime supports separating the workflow

definition from the enactment service on which it will ultimately be installed and used. Workflow process definitions are stored in the workflow repository.

Detailed information concerning this service (earlier referred to as METEOR Designer, MTDes, is given in [Lin 97, Zheng 97].

2.2 Workflow Repository Service

The METEOR Repository Service is responsible for maintaining information about workflow definitions and associated workflow applications. The graphical tools in the workflow builder service communicate with the repository service and retrieves, updates, and stores workflow definitions. The tools are capable of browsing the contents of the repository and incorporating fragments (either sub-workflows or individual tasks) of already existing workflow definitions into the one being currently created. The repository service is also available to the enactment service (see below) and provides the necessary information about a workflow application to be started.

The current implementation of the repository service implements the Interface I API, as specified by WfMC [WfMC]. A detailed description of the first design and implementation of this service is presented in [Yong 98].

2.3 Workflow Enactment and Management Services

The task of the enactment service is to provide execution environment for processing workflow instances. At present, METEOR provides two different enactment services: ORBWork, presented in this paper, and WebWork. Each of the two enactment services has a suitable code generator that can be used to build workflow applications from the workflow specifications generated by the building service or those stored in the repository. In the case of ORBWork, the code generator outputs specifications for task schedulers (see below), including task routing information, task invocation details, data object access information, user interface templates, and other necessary data. The code generator also outputs the code necessary to maintain and manipulate data objects, created by the data designer. The task invocation details are used to create the corresponding “wrapper” code for incorporating legacy applications with relative ease. Details of code generation for WebWork are presented in [Miller et al. 98]. The management service support monitoring and administering workflow instances as well as configuration and installation of the enactment services.

3. Overview of ORBWork

The current version of ORBWork, the one of the two implementation of the METEOR **EApps** enactment services been designed to address a variety of shortcomings found in today’s workflow systems by supporting the following capabilities:

- provide an enactment system capable of supporting dynamic workflows,
- allow significant scalability of the enactment service,
- support execution over distributed and heterogeneous computing environments within and across enterprises,
- provide capability of utilizing or integrating with new and legacy enterprise applications and databases² in the context of processes,
- utilize open standards, such as CORBA due to its emergence as an infrastructure of choice for developing distributed object-based, interoperable software,
- utilize Java for portability and Java with HTTP network accessibility,

² Data integration capability is supported by integrating METEOR’s enactment services with I-Kinetics’s DataBroker/OpenJDBC, a CORBA and Java based middleware for accessing heterogeneous and legacy data sources.

- support existing and emerging workflow interoperability standards, such as JFLOW [JFLOW] and SWAP [SWAP], and
- provide standard Web browser based user interfaces, both for the workflow end-users/participants as well as administrators of the enactment service and workflows.

In this article, we emphasize two of the features—scalability and support for adaptive workflows. Other important issues including improved support for exception handling for robust and survivable execution are not discussed for brevity.

Scalability

Scalability of the enactment system is becoming increasingly important for enterprises that wish to entrust their workflow management system with mission-critical processes. The number of concurrent workflows, the number of instances of the workflows processed during a given time period, and the average number of tasks in a workflow, all have an impact on the architectural issues.

We have leveraged the functionality offered by Iona's OrbixWeb and Name Service that allow us to place various components of the enactment service or other run-time components of the workflow instances, such as task schedulers, task managers, data objects, and even actual tasks on different hosts, at the same time providing transparency of their locations.

Adaptability and Dynamic Workflows

Recently, there has been an increasing interest in developing WfMSs capable of supporting adaptive and dynamic workflows. The majority of current work addresses relevant issues at modeling and language levels [Krishnakumar and Sheth 95, Ellis et al. 95, Jablonski et al. 97, Han 97], with few efforts on implementations underway [McClatchey et al. 97, Taylor 97, Reichert and Dadam 98]. A particularly different approach to supporting adaptive workflow (capable of reacting to the changes in local rules and other conditions) is being developed using the notion of migrating workflows [Cichocki et al. 97]. Related issues of integrating workflow or coordination technologies and collaboration technologies are investigated in [Guimaraes et al. 97, Sheth 97].

Developing systems that are able to support dynamic and adaptable workflow processes stands out as one of the difficult new challenges in the future evolution of WfMSs. Such systems must be uniquely sensitive to a rapidly changing process execution triggered by collaborative decision points, context-sensitive information updates, and other external events. Some research issues in this area that have been raised in the context of modeling and specification aspects appear in [Han and Sheth 98] and the relevant issues involving organizational changes appear in [Ellis et al. 95, Hermann 95]. However, the literature that addresses some of the enactment service issues is scarce.

The ORBWork scheduler and its supporting components have been designed in such a way that the enactment service can be used to support a variety of dynamic changes both to the workflow schema and to the individual workflow instances. The fully distributed scheduler (described later) maintains the full workflow specification. The workflow administrator can easily modify the workflow schema at runtime by acquiring new information from the workflow repository, or even by modifying the specification by direct interaction with the scheduler.

4. Enactment System of ORBWork

ORBWork provides a fully distributed, scalable enactment system for the METEOR workflow management system. The enactment system has been implemented to support workflows in heterogeneous, autonomous and distributed (HAD) systems. It utilizes the World Wide Web in providing a consistent interface to end-users and workflow administrators from commonly available Web browsers, and also utilizes the HTTP protocol for distribution of task definitions and task routing information.

4.1 ORBWork Architecture

ORBWork's architecture includes the scheduler, workflow specification repository, workflow manager, and the monitor. An overview of the ORBWork system organization is depicted in Figure 2.

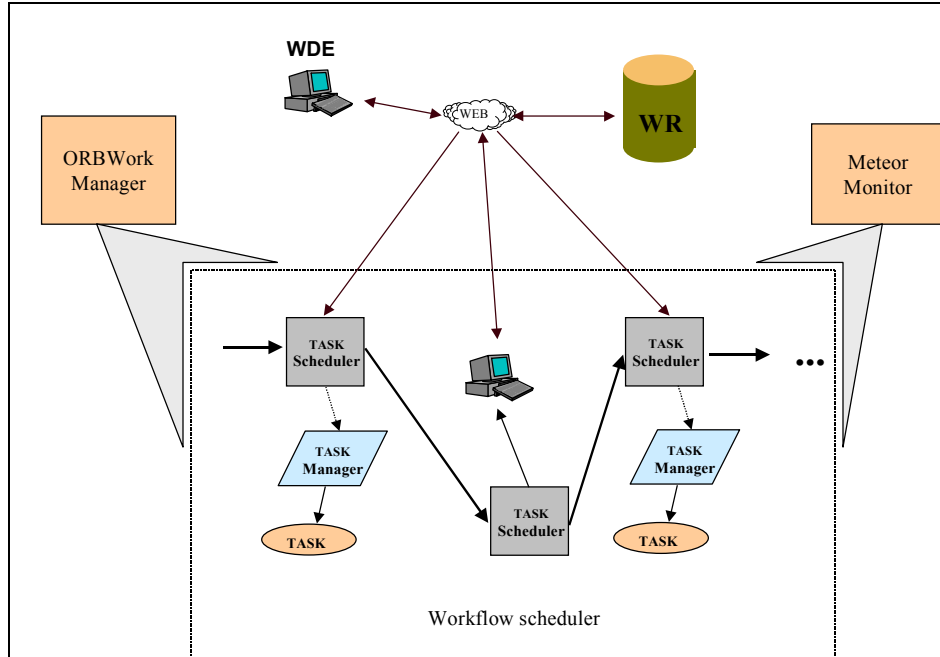


Figure 2: **ORBWork organization**

The *scheduler* accesses workflow specifications through the HTTP protocol, directly from the repository. The *monitor* records all of the events for all of the workflows being processed by the enactment service. It provides a user interface for the workflow administrator, who can access the information about all of the current workflow instances. The *workflow manager* is used to install new workflow processes (schemas), modify the existing processes, and keep track of the activities of the scheduler. The workflow administrator, using the available interface, controls the existing workflows as well as controls the structure of the scheduler. The structure of the scheduler can be altered by adding more resources, or by migrating fragments of the scheduler to other hosts, for example with lower processing loads. Some schedulers may be replicated, in case the load of workflow instances is too high for a host running just a single scheduler.

ORBWork's scheduler is composed of a number of small schedulers, each of which is responsible for controlling the flow of workflow instances through a single task. The individual schedulers are called *task schedulers*. In this way, ORBWork implements a fully distributed scheduler in that all of the scheduling functions are spread among the participating task schedulers that are responsible for scheduling individual tasks. In this sense, the ORBWork scheduler is composed of a network of cooperating task schedulers. Each task scheduler controls the scheduling of the associated task for all of the workflow instances "flowing" through the task. Each task scheduler maintains the necessary task routing information and task invocation details (explained later).

As a workflow instance progresses through its execution, individual task schedulers create appropriate task managers that oversee execution of associated tasks. Each workflow instance receives its own task manager, unless the task has been designed to have a worklist, in which case all of the instances are processed by the same task manager.

A workflow is installed by first creating an appropriate workflow context in the Naming Service. (The context is used for storing the object references for all of the participating components.) Then the installation continues by activating and configuring all of the necessary task schedulers and registering them with the Naming Service. All of the component task managers are also registered with the Interface Repository of the underlying ORB.

4.2 ORBWork Scheduler

ORBWork utilizes a fully distributed scheduler in that the scheduling responsibilities are shared among a number of participating *task schedulers*, according to the designed workflow map. Each task scheduler receives the scheduling specifications at startup from the Workflow Repository (currently, the repository service sends the specifications via the HTTP protocol). Each set of task specifications includes the input dependency (input transitions), output transitions with associated conditions, and data objects sent into and out of the task. In case of the human task (performed directly by end-users), the specifications include an HTML template of the end-user interface page(s). In case of a non-transactional automatic task (typically performed by a computer program), the specifications also include a task description and the details of its invocation. Finally, in case of a transactional task, the specification includes the details of accessing the desired database and the database query.

When a task is ready to execute, a task scheduler activates an associated task manager. The task manager oversees the execution of the task itself. Figure 3 presents a view of the ORBWork's distributed scheduler. Note that scheduling components and the associated tasks and task managers are distributed among four different hosts. The assignment of these components to hosts can be modified at runtime by the workflow administrator.

The partitioning of various components (scheduler's layout), including task schedulers, task managers and tasks, among the participating hosts is flexible. An ORBWork administrator may move any of the components from one host to another. In the fully distributed layout, it is possible to place all of the workflow components on different hosts.

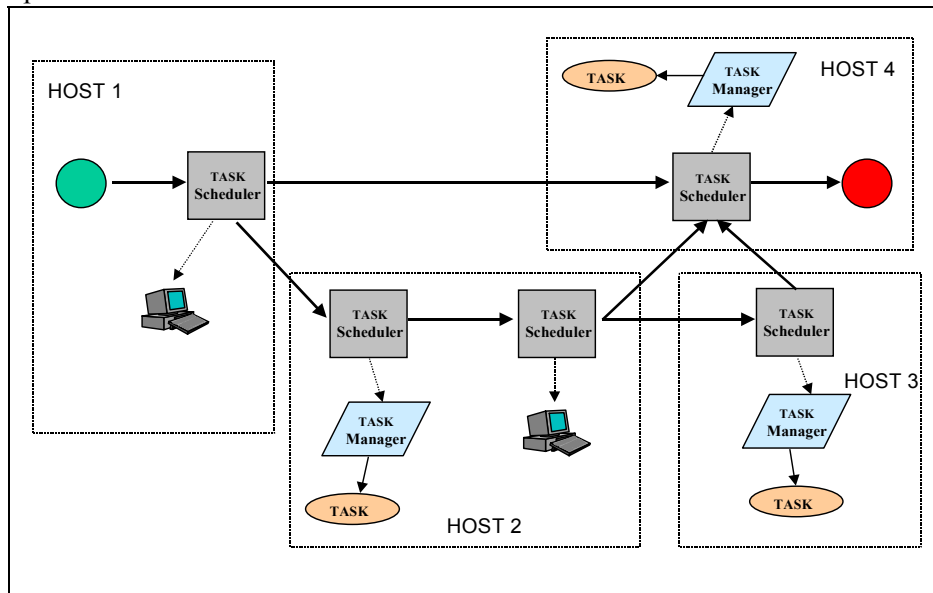


Figure 3: ORBWork's Distributed Scheduler

Each task scheduler provides a well-constrained subset of the HTTP protocol, and thus implements a lightweight, local Web server. This enables an ORBWork administrator to interact directly with a selected task scheduler and modify its scheduling specifications from a common Web browser. It also enables the end-user to access workflow instances residing on the task's worklist. This set up naturally supports a mobile user.

4.3 Support for Dynamic Workflows

One of the design goals of ORBWork has been to provide an enactment framework suitable for supporting dynamic and adaptive workflows. However, we must point out that the issues concerning the correctness of the dynamically introduced changes are handled outside of the enactment system by sub-components of the METEOR's design services, or by stand-alone correctness verification tools. The

ORBWork's enactment system performs only basic validation of deployed workflows. Nevertheless, the architecture of the enactment system has been designed to easily support dynamic changes and serve as a platform for conducting research in the areas of dynamic and collaborative workflows.

Since ORBWork uses a fully distributed scheduler, the scheduling information must be easily provided to all of the participating task schedulers at runtime. Each scheduler receives the information about the transitions leading into and out of it. In addition, the scheduling information includes the list of data objects to be created (a task may originate a data object).

At startup, each task scheduler requests scheduling data and upon receiving it configures itself accordingly. Furthermore, the configuration of an already deployed workflow is not fixed and can be changed dynamically. At any given time, a workflow designer, or in some permitted cases an end-user, may decide to alter the workflow. The introduced modifications are then converted into the corresponding changes in the specification files and stored in the repository. A "reload specification" signal is then sent to the affected task schedulers. As a result, the schedulers reload their specifications and update their configurations accordingly, effectively implementing the desired change to the existing workflow.

As one possibility, the changes introduced to a workflow may include adding a new task and connecting it to an already installed and active workflow application. Such a change must also include modifications of output transitions in the predecessor task schedulers and input dependencies in the successor task schedulers.

4.4 Support for Scalability and Fault Tolerance

The fully distributed architecture of ORBWork yields significant benefits in the area of scalability. As already mentioned, all of the workflow components of a designed and deployed workflow (this includes individual task schedulers, task managers, and task programs) may be distributed to different hosts. However, in practice it may be sufficient to deploy groups of less frequently used task scheduler/manager/programs to the same host. At the same time, heavily utilized tasks may be spread out across a number of available workflow hosts, allowing for greater load sharing.

The features of ORBWork designed to handle dynamic workflows are also very useful in supporting scalability. As load increases, an ORBWork administrator may elect to move a portion of the currently running workflow to a host (or hosts) that become available for use in the workflow. The migration can be performed at the time the deployed workflow is running. Simply, the workflow administrator may suspend and shutdown a given task scheduler and transfer it to a new host. Because of the way task schedulers locate their successors, the predecessors of the moved task scheduler will not notice the changed location of the task. If the associated task must be executed on a specific host (for example it is a legacy application), the associated task manager may be left in place, while only the scheduler is transferred.

In case a group of task schedulers is deployed to the same host, the ORBWork administrator has an option of combining them into a single "master" scheduler. Such a master scheduler controls a number of individual task schedulers that share the same heavy weight process. This allows the administrator to control the utilization of the participating host even further, where having many individual operating system-level processes (task schedulers) could potentially burden the host system.

The distributed design of ORBWork offers no single point of failure for an ongoing workflow instance. Since the individual task schedulers cooperate in the scheduling of workflow instances, a failure of a single scheduler does not bring the whole system down, and other existing workflow instances may continue execution.

The error handling and recovery framework for ORBWork (initial design has been described in [Worah et al 97]) has also been defined in a scalable manner. All errors are organized into error class hierarchies, partitioning the recovery mechanism across local hosts, encapsulating and handling errors and failures as close to the point of origination as possible, and by minimizing the dependence on low-level operating system-specific functionality of the local processing entities.

5. ORBWork Implementation

One of the most important considerations while designing the ORBWork workflow management system has been its flexible and easily modifiable distributed architecture. The current version of the system has been implemented in Java and OrbixWeb 3.0, Iona's CORBA system with Java binding. In addition, Iona's Naming Service has been utilized as a way of providing location transparency for all of the ORBWork components.

Using CORBA, and especially Iona's OrbixWeb and Naming Service, as the underlying middleware system offers a number of advantages for implementing a distributed workflow enactment system. In addition to the obvious features provided by CORBA, ORBWork relies on a number of specific services that proved extremely useful in implementing ORBWork. The following table summarizes the features used.

Feature	Application
Dynamic Object Activation	Allows for automatic activation and deactivation of ORBWork components, reducing the load on the host system(s)
Dynamic Invocation Interface (DII)	Only object references are transferred; data object are accessed dynamically, according to their interfaces
Object Loaders	Data objects, task schedulers, and other ORBWork components use loaders to automatically save/restore state
Naming Service	Task schedulers are located with the use of the Name Service; this allows for flexible and transparent placement of the schedulers and their possible migration at runtime

Table 1: CORBA/Orbix features used in ORBWork

All of the ORBWork components are implemented as CORBA objects. ORBWork relies on the Orbix Activator to start the necessary server when its functions are necessary for the activities of the distributed scheduler and also shutdown the servers once no services have been requested within a specified time interval. In this way, certain portions of a large, distributed workflow (for example those less frequently used) may become inactive, reducing the overhead on the host systems to the necessary minimum.

5.1 Task Schedulers

A task scheduler is implemented as a CORBA object. The IDL interface presented to clients (other task schedulers and other ORBWork components) enables them to invoke various scheduling functions according to the currently loaded specifications. The interface also enables dynamic modifications of the scheduling specifications by reloading from the specification server (repository) or by a direct modification of the specification within the task scheduler.

A task scheduler relies on Orbix Name Service to locate its successors. This enables the ORBWork administrator to dynamically reconfigure the runtime layout of the scheduler by shifting some components between hosts, without introducing any changes to the remaining task schedulers, or workflow instances administered by them.

ORBWork uses the object loader capability supported by OrbixWeb to save/restore the state of a task scheduler. The state includes the necessary information about forthcoming instances (those with still unfulfilled input dependency) and those already on the worklist. As the CORBA object representing a task scheduler is activated (because one of its task predecessors attempts a transfer of the next workflow instance), the necessary scheduling data is automatically reloaded.

5.2 Task Managers

Task managers control execution of all non-human tasks (human tasks have no associated task managers). Depending on the task type, a task manager is classified as non-transactional or transactional, and is implemented as a CORBA object. A task manager's IDL interface allows it to be invoked by the corresponding task scheduler. Once activated, the task manager stays active until the task itself completes or generates an exception. Once the task has completed or terminated prematurely with a fault, the task manager notifies its task scheduler. The task scheduler then continues the flow of the workflow instance. Orbix Activator automatically activates the task manager, only when needed. The communication between the task scheduler and the associated task manager is accomplished by asynchronous (one way) method calls.

A transactional task manager uses JDBC to access the requested data source. Currently, ORBWork provides specific task managers for accessing Oracle and Mini SQL databases, as well as one for the Open JDBC driver from I-Kinetics. The last of the mentioned task managers allows a uniform access to a wide variety of database management systems (including those on mainframes) from a single task manager.

5.3 Data Objects

Data objects are implemented as CORBA objects, providing an IDL interface for accessing all of the defined attributes and methods. As in the case of a task scheduler, the data object implementation uses the object loader to load and save the state of each data object. The CORBA server hosting the data objects is automatically shut down if no data read/write requests arrive within a specified time period, and the dynamic loader saves the state of the object.

As task schedulers implement flow of control within a workflow instance, data objects must be made available at the successor tasks. Instead of the whole object, only its object reference is sent to the task scheduler. When preparing to run the task, the task scheduler accesses the necessary data object(s) (using the Dynamic Invocation Interface) and extracts the relevant attribute values.

5.4 ORBWork Servers

Typically, a single ORBWork host runs a number of task schedulers, each of which is implemented as a separate CORBA object. A CORBA object must reside within a CORBA server that typically runs as a single operating system process. In order to save computer resources, a group of ORBWork task schedulers may be placed within a single CORBA server that functions as an ORBWork server. Each ORBWork server is designed to control any number of task schedulers.

A workflow installed on the ORBWork enactment system may utilize any number of heterogeneous hosts (of course, OrbixWeb must be available on each one of them; clients/browsers may be used anywhere). Each of the hosts may have any number of ORBWork servers. However, the most common approach is to keep the number of ORBWork servers close to the number of available processors. Nowadays, some of the available Java virtual machines are able to take advantage of the available processors to run threads. Since the implementation of an ORBWork task scheduler is multithreaded, the question of the number of ORBWork servers may be less critical in that if all of the schedulers are placed within a single server, the schedulers will be able to utilize all of the available processors.

5.5 ORBWork Manager

The ORBWork Manager is used to install workflows (schemas) and activate all of the necessary task schedulers. In addition to registering with Orbix Name Service, each task scheduler registers with ORBWork Manager and notifies it of its precise location. In addition, since each task scheduler provides a subset of the HTTP protocol, the scheduler notifies the ORBWork Manager of the precise URL address that the end users and the administrator can use to interact directly with it. The URL address is created when the scheduler is initially installed and it contains the port number that has been assigned to the HTTP server.

The manager is implemented as a CORBA object. It has an IDL interface that allows ORBWork clients to install and administer a workflow (schema) as well as create workflow instances. The manager provides an HTTP protocol, so that the same administrative functions can be performed via the Web, from a common browser.

In order to provide an easy access to task schedulers, the ORBWork Manager also functions as a URL redirector, when an end-user wishes to access her task's worklist. This is necessary since the port number on which the task scheduler's HTTP server is listening is assigned by the system at the time the task scheduler is activated. The port number is not fixed and cannot be known beforehand.

It is important to note that the role of the ORBWork Manager is necessary only at the time a new workflow is installed or modified, or when an end-user is connecting for the first time to her designated task. The manager does not participate in any task scheduling activities.

6. Conclusion

ORBWork system provides a flexible, fully distributed implementation of the workflow enactment service for the METEOR Workflow Management System. The ORBWork scheduler has been designed and implemented to support dynamic workflows. The scheduler offers significant potential for scalability, since the workflow administrator can incrementally increase the number of workflow hosts, migrating and/or replicating some of the scheduling functions to the new hosts.

The ORBWork enactment system has been implemented entirely in Java and is therefore available on a wide range of computer systems. In our workflow application development (the ORBWork enactment service has been used to implement a number of workflow applications, mainly in the area of healthcare [Sheth et al 97]), we have used SUN Solaris and Windows NT as ORBWork hosts. We were able to integrate disparate distributed and heterogeneous computing environments with ease.

The current ORBWork implementation has been based on open standards. It will also provide support for workflow interoperability standards (such as SWAP [SWAP] and JFLOW [JFLOW]), once they stabilize. In fact, we are currently in the process of creating prototype implementations to the two mentioned interoperability interfaces.

On the research front, we expect to increasingly integrate our workflow research with that of collaboration to develop a new generation of coordination and collaboration system.

Acknowledgement

We thank Iona Technologies and I-Kinetics for the donation of all their products to the LSDIS Lab, University of Georgia. This research was partially done under a cooperative agreement with the National Institute of Standards and Technology Advanced Technology Program (under the HIIT contract, number 70NANB5H1011) and co-sponsored by the Healthcare Open Systems and Trials, Inc consortium.

Bibliography

[Cichocki et al. 97] A. Cichocki and M. Rusinkiewicz, Migrating Workflows, Advances in Workflow Management Systems and Interoperability, Istanbul, Turkey, August 1997.

[Das et al. +97] S. Das, K. Kochut, J. Miller, A. Sheth, and D. Worah, ORBWork: A Reliable Distributed CORBA-based Workflow Enactment System for METEOR₂, Technical Report UGA-CS-TR-97-001, LSDIS Lab, CS Department, Univ. of Georgia, February 1997.

[Ellis et al. 95] C. Ellis, K. Keddara, and G. Rozenberg, Dynamic Changes within Workflow Systems in Proc. of the Conf. on Organizational Computing Systems (COOCS'95), 1995.

[Guimaraes et al. 97] N. Guimaraes, P. Antunes, and A. Pereira, The Integration of Workflow Systems and Collaboration Tools, Advances in Workflow Management Systems and Interoperability, Istanbul, Turkey, August 1997.

[Han 97] Y. Han, "HOON - A Formalism Supporting Adaptive Workflows," Technical Report #UGA-CS-TR-97-005, Department of Computer Science, University of Georgia, November 1997.

- [Han and Sheth 98] Y. Han and A. Sheth, "On Adaptive Workflow Modeling," the 4th International Conference on Information Systems Analysis and Synthesis, Orlando, Florida, July, 1998
- [Hermann 95] T. Hermann, Workflow Management Systems: Ensuring Organizational Flexibility by Possibilities of Adaptation and Negotiation, in Proc. of the Conf. on Organizational Computing Systems (COOCS'95), 1995.
- [Jablonski et al. 97] S. Jablonski, K. Stein, and M. Teschke, Experiences in Workflow Management for Scientific Computing, Proceedings of the Workshop on Workflow Management in Scientific and Engineering Applications (at DEXA97), Toulouse, France, September 1997.
- [Krishnakumar and Sheth 95] N. Krishnakumar and A. Sheth, "Managing Heterogeneous Multi-system Tasks to Support Enterprise-wide Operations," Distributed and Parallel Databases Journal, 3 (2), April 1995
- [Infocosm] Infocosm home page, <http://infocosm.com>
- [JFLOW] OMG jFlow Submission, <ftp://ftp.omg.org/pub/docs/bom/98-06-07.pdf>
- [Lin 97] C. Lin, "A Portable Graphic Workflow Designer," M.S. Thesis, Department of Computer Science, University of Georgia, May 1997.
- [McClatchey et al. 97] R. McClatchey, J.-M. Le Geoff, N. Baker, W. Harris, and Z. Kovacs, A Distributed Workflow and Product Data Management Application for the Construction of Large Scale Scientific Apparatus, Advances in Workflow Management Systems and Interoperability}, Istanbul, Turkey, August 1997.
- [METEOR] METEOR project home page, <http://lsdis.cs.uga.edu/proj/meteor/meteor.html>
- [Miller et al. 98] J. Miller, D. Palaniswami, A. Sheth, K. Kochut, H. Singh "WebWork: METEOR's Web-based Workflow Management System", Journal of Intelligent Information Systems, (JIIS) Vol. 10 (2), March 1998.
- [Reichert and Dadam 98] M. Reichert and P. Dadam, ADEPT flex: Supporting Dynamic Changes of Workflows Without Losing Control, Journal of Intelligent Information Systems, 10 (2), March 1998.
- [Sheth 97] Sheth, "From Contemporary Workflow Process Automation to Adaptive and Dynamic Work Activity Coordination and Collaboration," Proceedings of the Workshop on Workflows in Scientific and Engineering Applications, Toulouse, France, September 1997.
- [Sheth et al. 96] A. Sheth, D. Georgakopoulos, S. Joosten, M. Rusinkiewicz, W. Scacchi, J. Wileden, and A. Wolf, Eds. Report from the NSF Workshop on Workflow and Process Automation in Information Systems, Technical Report UGA-CS-TR-96-003, Dept. of Computer Sc., University of Georgia, October 1996. <http://lsdis.cs.uga.edu/lib/lib.html>
- [Sheth et al 97] A. Sheth, D. Worah, K. Kochut, J. Miller, K. Zheng, D. Palaniswami, S. Das, "The METEOR Workflow Management System and its use in Prototyping Healthcare Applications", Proceedings of the Towards An Electronic Patient Record (TEPR'97) Conference, April 1997, Nashville, TN.
- [Sheth and Kochut 98] A. Sheth and K. Kochut, "Workflow Applications to Research Agenda: Scalable and Dynamic Work Coordination and Collaboration Systems," in A. Dogac, L. Kalinechenko, T. Ozsu and A. Sheth, Eds. Workflow Management Systems and Interoperability, NATO ASI Series F, Vol. 164, Springer Verlag, 1998.
- [SWAP] Simple Workflow Access Protocol home page, <http://www.ics.uci.edu/~ietfswap/index.html>
- [Taylor 97] R. Taylor, Endeavors: Useful Workflow Meets Powerful Process, Information and Computer Science Research Symposium, University of California at Irvine, February 1997. URL: <http://www.ics.uci.edu/endeavors/>
- [WfMC] Workflow Management Coalition Standards, <http://www.aiim.org/wfmc/mainframe.htm>
- [Worah et al 97] D. Worah, A. Sheth, K. Kochut, J. Miller, "An Error Handling Framework for the ORBWork Workflow Enactment Service of METEOR," Technical Report, LSDIS Lab. Dept. of Computer Science, Univ. of Georgia, June 1997.
- [Yong 98] J. Yong, "The Respository system of METEOR workflow management system", Master Thesis, Department of Computer Science, University of Georgia, March 1998.
- [Zheng 97] K. Zheng, Designing Workflow Processes in METEOR₂ Workflow Management System M.S. Thesis, LSDIS Lab, Computer Science Department, University of Georgia, June 1997.