Minoro Toi / Photonica

Optimizing Workflow

Using a CORBA-based, fully distributed process to create scalable, dynamic systems | KRYS J. KOCHUT, AMIT P. SHETH, AND JOHN A. MILLER

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

Workflow management is the automated coordination, control, and communication of work that is required to satisfy workflow processes.[1] A Workflow Management System (WfMS) is a set of tools that provide support for the necessary services of workflow creation (which includes process definition), workflow enactment, and administration and monitoring of workflow processes.[2] The developer of a workflow application relies on tools for the specification of the workflow process and the data it

This project began as an academic exercise. Its commercial applications were uncovered as the process continued.

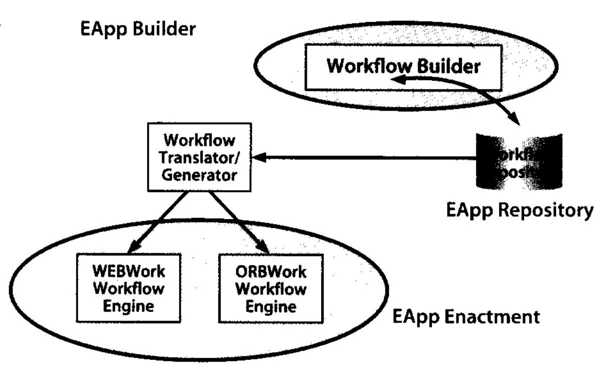

Figure 1. ***METEOR architecture.***

manipulates. The specification tools are integrated 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 intertask dependencies, scheduling tasks, managing workflow data, and ensuring a reliable execution environment. Administrative and monitoring tools are used for managing user and work group roles, defining policies (e.g., security, authentication), audit management, process monitoring, tracking, and reporting data generated during workflow enactment.

METEOR provides open-systems-based, high-end workflow management, and enterprise application integration infrastructure

A number of applications posing substantial challenges to the currently available WfMSs are discussed in "Workflow Applications to Research Agenda: Scalable and Dynamic Work Coordination and Collaboration Systems."[3] 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 it must 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 introduce changes dynamically to the whole workflow process definition (schema), or even just to individual workflow (instances). The system must also include a flexible enactment system that is capable of supporting scalability, where new resources (computers, database servers, end users, etc.) can be incorporated easily within the workflow system; and adaptive workflows, where the workflow specification can be changed or extended, including the addition or modification of tasks and intertask dependencies.

The METEOR project is represented by both the research system (METEOR), and a suite of commercial offerings—the METEOR Enterprise Application Suite of tools and services (EAppS)[4]—that addresses these 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, METEOR's enactment service that exploits CORBA, Java, and Web technologies to meet these challenges.

METEOR Architecture

METEOR's architecture includes a collection of four services: EAppBuilder, EAppRepository, EAppEnactment, and EAppManager. EAppEnactment includes two services—ORBWork and WebWork. Both ORBWork and WebWork use fully distributed implementations. WebWork,[5] an entirely Web-based enactment service, is a comparatively lightweight implementation that is well suited for a variety of enterprise workflow process applications that involve limited data exchange and do not need to be changed dynamically. ORBWork is better suited for more demanding, mission-critical enterprise applications that require high scalability, robustness, and dynamic modifications. The overall architecture of the system is shown in Figure 1.

Workflow Builder Service This service consists of a number of components that are used to design and specify a workflow graphically, and in some cases it leaves 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, the data objects to be manipulated by the workflow, and 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 the modeling of complex workflows that consist of varied human and automated tasks in a conceptual manner with 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, this service places very few restrictions on the designer with regard to the specification of the workflow.

The workflow specification created using this service includes all the predecessor/successor dependencies among the tasks as well as the data objects that are passed among them. It also includes definitions of the data objects, and the task invocation details. The specification may be formatted to be compliant with the Workflow Management Coalition's Workflow Process Definition Language (WPDL).[2] This service assumes no particular implementation of the workflow enactment service (i.e., the runtime system). Its independence from the runtime system 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 (which was previously referred to as METEOR Designer, or MTDes,) is given in works by Lin and Zheng.[6,7]

Workflow Repository Service The METEOR Repository Service is responsible for maintaining information about workflow definitions and their associated workflow applications. The graphical tools in the workflow builder service communicate with the repository service and retrieve, update, and store workflow definitions. The tools are capable of browsing the contents of the repository and incorporating fragments (either subworkflows or individual tasks) of existing workflow definitions into the one that is currently being created. The repository service is also available to the enactment service, and it provides it with the information it needs about workflow applications that are to be started.

ORBWork exploits CORBA, Java, Web, and emerging interoperability specifications to manage enterprise applications

The current implementation of the repository service implements the Interface I API, as specified by the WfMC. A detailed description of the first design and implementation of this service is presented in works by Yong.[8]

Workflow Enactment and Management Services The task of the enactment service is to provide an execution environment for processing workflow instances. At present, METEOR provides two different enactment services: ORBWork and WebWork. Each of the two enactment services has a code generator that is suitable for building workflow applications from the workflow specifications generated by the building service or from those stored in the repository. With ORBWork, the code generator outputs specifications for task schedulers, 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 the data objects created by the data designer component of the builder. 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 works by Miller, et al.[5] With WebWork, the management service supports monitoring and administering workflow instances as well as configuring and installing the enactment services.

Overview of ORBWork

The current version of ORBWork, the implementation of the METEOR EAppS enactment services that was designed to address a variety of shortcomings in today's workflow systems, supports the following capabilities:

- It provides an enactment system that is capable of supporting dynamic workflows;
- It allows the enactment service to be significantly scalable;
- It supports execution over distributed and heterogeneous computing environments within and across enterprises;
- It provides developers with the ability to utilize or integrate with new and legacy enterprise applications and databases* in the context of processes;
- It utilizes open standards, such as CORBA, because of its emergence as the infrastructure of choice for developing distributed, object-based, interoperable software;
- It utilizes Java for portability and HTTP network accessibility;
- It supports existing and emerging workflow interoperability standards, such as JFLOW and SWAP; and
- It provides standard Web browser-based user interfaces, both for the workflow end users and participants as well as administrators of the enactment service and workflows.

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

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

Scalability The scalability of the enactment system is becoming increasingly important for enterprises that wish to entrust their workflow management systems 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 allows us to place various components of the enactment service or other runtime components of the workflow instances, such as task schedulers, task managers, data objects, and even actual tasks on different hosts, at the same time keeping their locations transparent.

Adaptability and Dynamic Workflows Recently, there has been an increasing interest in developing WfMSs that are capable of supporting adaptive and dynamic workflows. The majority of current work addresses relevant issues at modeling and language levels,[9–12] with few efforts on implementations underway.[13–15] A different approach to supporting adaptive workflow (workflow that is capable of reacting to the changes in local rules and other conditions) is being developed using the notion of migrating workflows.[16] Related issues of integrating workflow or coordination technologies and collaboration technologies are investigated in works by Guimaraes and Sheth.[17,18]

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 that is 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 is discussed Han and Sheth[16] and the relevant issues involving organizational changes appear in works by Ellis and Hermann.[10,20] However, 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 maintains the full workflow specifi-

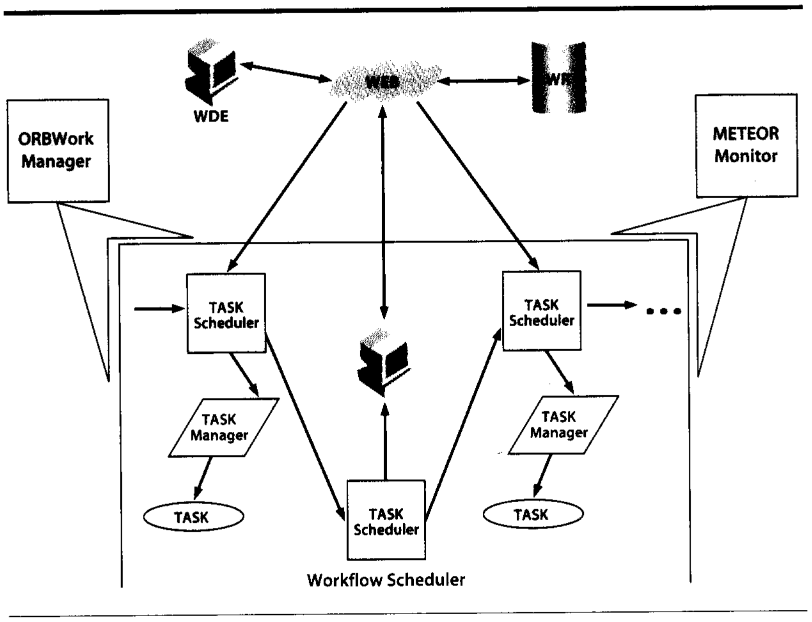

Figure 2. ***ORBWork organization.***

cation. The workflow administrator can modify the workflow schema easily at runtime by acquiring new information from the workflow repository, or even by modifying the specification by interacting directly with the scheduler.

Enactment Services

ORBWork provides a fully distributed, scalable enactment service for METEOR. The enactment service has been implemented to support enterprise applications involving workflows in heterogeneous, autonomous, and distributed (HAD) systems. It utilizes the World Wide Web to provide a consistent interface

ORBWork implements fully distributed scheduling and fault-tolerant workflows

to end users and workflow administrators from commonly available Web browsers, and also utilizes the HTTP protocol to distribute task definitions and task routing information.

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

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 that are 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 controls the existing workflows and the structure of the scheduler using the available interface. 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 because 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 the 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.

ORBWork Scheduler ORBWork's scheduler is fully distributed 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. For a human task (a task that end users perform directly), the specifications include an HTML template of the end user interface page(s). For nontransactional automatic tasks (tasks that are typically performed by a computer program), the specifications also include a task description and the details of its invocation. Finally, for 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.

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 setup naturally supports a mobile user.

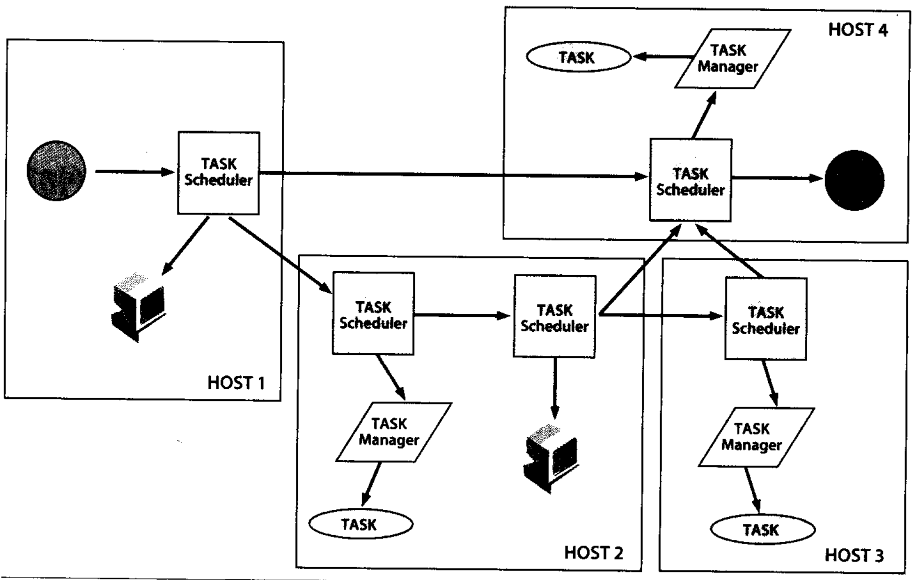

Figure 3. ***ORBWork's distributed scheduler.***

Support for Dynamic Workflows One of the design goals of ORBWork has been to provide an enactment framework that is 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 subcomponents of the METEOR's builder services, or by standalone 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 support dynamic changes easily and serve as a platform for conducting research in the areas of dynamic and collaborative workflows.

Since ORBWork uses a fully distributed scheduler, it must be easy to provide the scheduling information 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, it configures itself accordingly. Furthermore, the configuration of a workflow that has already been deployed is not fixed and it can be changed dynamically. At any given time, a workflow designer, or in some cases an authorized 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.

Support for Scalability and Fault Tolerance The fully distributed architecture of ORBWork yields significant benefits in the area of scalability. 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 schedulers, managers, and 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.

ORBWork's administrator may move any component from one node to another anytime

The features of ORBWork that were designed to handle dynamic workflows are also very useful for supporting scalability. As the 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 shut down 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, if it is a legacy application), the associated task manager may be left in place while only the scheduler is transferred.

If a group of task schedulers is deployed to the same host, the ORBWork administrator has the option of combining them into a single "master" scheduler. A master scheduler controls a number of individual task schedulers that share the same heavyweight process. This allows the administrator to control the utilization of the participating host even further, where having

CORBA/Orbix features used in ORBWork

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

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 executing.

All of the ORBWork components are implemented as CORBA objects

The error handling and recovery framework for ORBWork (the initial design has been described in "An Error Handling Framework for the ORBWork Workflow Enactment Service of METEOR"[18]) has also been defined in a scalable manner. All errors are organized into error class hierarchies by partitioning the recovery mechanism across local hosts, by 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.

ORBWork Implementation

One of the most important considerations in the design of 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. Table 1 summarizes the features used.

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 it also shuts down 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, which will reduce the overhead on the host systems to the necessary minimum.

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

ORBWork uses the object loader capability supported by OrbixWeb to save and restore the state of a task scheduler. The state includes the necessary information about forthcoming instances (those with 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 reloaded automatically.

Task Managers Task managers control the execution of all nonhuman tasks (human tasks have no associated task managers). Depending on the task type, a task manager is classified as nontransactional 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 it is activated, the task manager stays active until the task itself completes or it 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 activates the task manager automatically, only when it is 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 uniform access to a wide variety of database management systems (including those on mainframes) from a single task manager.

Data Objects Data objects are implemented as CORBA objects, and they provide 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 that hosts 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.

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

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. 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 because if all of the schedulers are placed within a single server, they will be able to utilize all of the available processors.

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. Also, 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 with it directly. The URL 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.

To provide easy access to task schedulers, the ORBWork Manager also functions as a URL redirector when end users wish to access their task worklists. This is necessary because 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 end users connect to their designated tasks for the first time. The manager does not participate in any task scheduling activities.

Conclusion

ORBWork provides a flexible, fully distributed implementation of the workflow enactment and application integration service for the METEOR 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 increase the number of workflow hosts incrementally, migrating and/or replicating some of the scheduling functions to the new hosts.

ORBWork has been implemented entirely in Java and is therefore available on a wide range of computer systems. (ORBWork has been used to implement a number of workflow applications, mainly in the area of health care[22]), we have used Sun Solaris and Windows NT as ORBWork hosts. We were able to integrate disparate distributed and heterogeneous computing environments with ease.

We expect to integrate our workflow research with that of collaboration to develop a new generation of systems

The current ORBWork implementation has been based on open standards. It will also provide support for workflow interoperability standards (such as SWAP and JFLOW), once they stabilize. We are currently in the process of creating 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. Δ

Acknowledgments

We thank IONA Technologies and I-Kinetics for the donation of all their products to the LSDIS Lab at the 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 cosponsored by the Healthcare Open Systems and Trials, Inc. consortium.

Bibliography

1. Sheth, A., *et. al.* Report from the NSF Workshop on Workflow and Process Automation in Information Systems, Technical Report UGA-CS-TR-96-003, Dept. of Computer Science, University of Georgia, Oct. 1996, http://lsdis.cs.uga.edu/lib/lib.html.
2. Workflow Management Coalition Standards, http://www.aiim.org/wfmc/mainframe.htm.
3. Sheth, A., and K. Kochut. "Workflow Applications to Research Agenda: Scalable and Dynamic Work Coordination and Collaboration Systems," in *Workflow Management Systems and Interoperability*, A. Dogac, L. Kalinechenko, T. Ozsu, and A. Sheth, Eds., NATO ASI Series F, Vol. 164, Springer Verlag, 1998.
4. Infocosm home page, http://infocosm.com.

continued on page 57