CORBA Based Architecture for Large Scale Workflow

Roberto Silveira Silva Filho, Jacques Wainer, Edmundo R. M. Madeira
IC -Institute of Computing
UNICAMP - University of Campinas
13081-970 Campinas - SP - Brazil
{robsilfi, wainer, edmundo}@dcc.unicamp.br

Clarence Ellis
Department of Computer Science
University of Colorado, Boulder, CO 80309
skip@colorado.edu

Abstract

Standard client-server workflow management systems have an intrinsic scalability limitation: the central server is a bottleneck for large scale applications. It is also a single fault point that may disable the whole system. We propose a fully distributed architecture for workflow management systems. It is based on the idea that the case (an instance of the process) migrates from host to host, according to the process definition specification, as the corresponding activities are executed. This basic architecture is improved so that other requirements for Workflow Management Systems, besides scalability, are also contemplated. A CORBA-based implementation of such architecture is discussed, with both its limitations and positive points described.

Key Words*: Large Scale Workflow, Distributed Objects, CORBA, and Mobile Agents.*

1. Introduction

Workflow Management Systems (WFMS) are used to coordinate and sequence business processes, such as loan approval, insurance reimbursement, and other office or assembly line procedures. Such processes are represented as workflows: computer interpretable description of activities (or tasks), and their execution order. The workflow also describes the data available and generated by each activity, synchronization points and so on. This description should also express constrains and conditions such as when the activities should be executed, a specification of who can or should perform each activity, and what tools and programs (such as word processors, CAD and CASE systems, spreadsheets and others) are needed during the activity execution. [3].

A WFMS can be seen as a set of control applications and interfaces (to other tools and applications) that allow the project, definition, execution, and monitoring of workflows. The Workflow Management Coalition (WFMC) specifies a set of terms, definitions, and interfaces that standardizes most of the main aspects of a WFMS [1,2].

Many academic prototypes and commercial WFMS are based on the standard client-server architecture [2]. In such systems, the Workflow Engine, the core of a WFMS, is a server machine that typically stores both the workflow data (the definition of the workflow and its activities, the state and history information about each instance of the workflow, and any other data related to the workflow execution) and the application data (the data that is used and generated by each activity within the workflow).

Such client-server centralized architecture represents a limiting barrier for large-scale applications with (possibly) many instances of a workflow being executed concurrently. Furthermore, the use of a central database in these systems represents a single failure point that can paralyze the whole system and possible the whole business itself. Therefore, WFMS based on centralized client-server architectures are limited in providing appropriate levels of scalability, fault tolerance and availability, which may hind their use on an important set of applications [4].

In this paper we introduce the WONDER (Workflow on Distributed Environment) architecture, a WFMS that addresses, in special, the scalability and availability issues. Other requirements of a WFMS, such as fault recovery, auditing and traceability are also addressed. In the WONDER architecture, the control, the storage of data, and the execution of the activities are all distributed into a network of computers.

1.1. Terms

We will use, from now on, the following definitions: A process (or process definition) is a workflow (the description of a business process). A case is an instance of a process. Thus, if purchase of office supplies is a process, then Joe's Friday request to purchase 500 paper

clips is a case. Processes are defined in terms of activities, that is, predefined tasks usually performed by a single person, or by a program. Role is the generic description of a set of abilities required to perform certain activities. Thus secretary, programmer, and reviewer, are roles. People or programs that perform the activities are called users or actors, and a particular user can fulfill many roles. The process definition, which we will call a plan, is described in terms of the Workflow Management Coalition primitives: sequencing, and-joint, and-split, or-joint, and or-split [2].

1.2. Requirements for Workflow Systems

In this paper, we will address the following requirements of a WFMS:

Scalability. The WFMS should not have its performance degraded with the increase of: processes, cases, activities within a workflow, volume of application data, and number of actors that perform the activities.

Fault recovery. The WFMS should deal with both software and hardware failures with the least intervention of users as possible. Furthermore, the re-execution of activities due to some failure should be avoided.

Availability. The WFMS must not become unavailable for long periods of time, specially if mission critical workflows are being executed.

Monitoring. The WFMS should be able to provide current state information about all cases being executed.

Traceability. The WFMS should be able to keep all history information about current and terminated cases.

Interoperability. Different WFMS should be able to inter-operate.

Support for the use of external application tools. The execution of a particular activity may require external tools (such as word processors, spreadsheets, CAD systems, expert systems, and so on). The WFMS should be able to start such external applications and determine when these tools have been terminated, managing the data read and produced by these applications.

1.3. Paper Description

The next section discusses the general idea of the WONDER architecture, describing its main components. Section 3 discusses the implementation of that architecture using CORBA. Section 4 presents some implementation issues and Section 5 describes some related work.

2. The Distributed Model

2.1. General description of the model

In general, and using informal terms, our architecture is based on the idea that each case is a "mobile agent" that moves from hosts to host as the activities are performed. The case encapsulates both the application data and the plan for that case (workflow control data), and "moves" to a particular user's host once it "figures out" that the next activity will be performed by that user at that host. Once the activity is finished, the case "figures out" the user who will perform the next activity and moves to her host. This "mobile agent" architecture copes with the scalability requirement: there is no central control or data server, therefore there is no performance bottleneck.

To deal with the other requirements, some components are added to the architecture. For example, it is usually the case that the plan of a process does not specify a particular user as the performer of an activity, but only a role. Consider a credit checking activity example, the plan will state that the activity of credit checking should be performed by a credit evaluator, but not a specific actor. Hence, the plan does not provide enough information for the "mobile agent" (the case) to "figure out" where to it should migrate. In order to cope with this requirement, a role coordinator component, containing information of each role coordinator, is defined: the case queries the particular role coordinator, in the example above, the credit evaluator coordinator, and asks it for an user to perform that activity, satisfying some requirements. Once figured out the user, the case moves to that user's hosts.

Monitoring is also an issue in our "mobile agent architecture": how do we find out, without broadcasting, what is the current state of a case, since it may be in any of the hosts in the network? A case coordinator component that keeps track of the case as it moves along was defined: each time a case moves to a new user's host, it sends a notification to its case coordinator. Therefore the case coordinator knows where and at which process stage is a case.

Another important problem for the mobile agent architecture is failure recovery. The distributed characteristic of our architecture introduces many failure points, but keeps the failure isolated from other processes. What happens to a case if it is at a user's host that breaks down? To deal with that, a requirement to the moving protocol was defined: the case, when moving to the next host, keeps a copy of itself in a stable storage on the source host. If the destination host, where the case is being executed breaks down, the case state at the previous activity can be accessed by the case coordinator as soon as the failure is detected, and another host/user is elected to restart or continue the stopped activity, using the state stored in a previous host. Furthermore, to avoid the replication of previous states of the case throughout the network hosts, the case coordinator may direct hosts to

transfer its old case state to a backup server, deleting its data from these hosts.

In general, the "mobile agent" architecture is augmented with components that hold a specific domain information, like the case coordinator, role coordinator, backup server and others. These servers or coordinators, however, are not likely to be a bottleneck in the performance of the system. The case coordinator, for example, receives only very short notifications from the "mobile agent", which mainly informs where the case is moving to. On the other hand, the backup server may receive large amounts of data, but this transfer can be done asynchronously when network and server load allows for it. The only standard server, in a client-server sense, is the role coordinator which receives a query and must return an answer for the processing to continue. However, in this case, the amount of information exchanged is small: a short query and the identity of a user as answer. The information stored in the role coordinator, in most cases, is also the product of small notifications.

2.2. Main Components of the Architecture

We will now describe the WONDER components in detail. The architecture is composed of a set of autonomous distributed objects that together execute the activities that compose the active cases. These objects are described below.

2.2.1. Process Coordinator. The process coordinator is the object that manages the description (template) of a particular process. This object is responsible for the case coordinator creation. Upon a request of a new office supplies purchase, for instance, the "purchase of office supplies" process coordinator will create a new case coordinator for that order, instantiating and transferring the plan to that object.

In case one needs to locate all instances of process, the process coordinator also keeps track of all case coordinators that it created and that are still active. For example, if the definition of the process is changed, say to introduce a new activity, the process coordinator will propagate such changes to all its cases.

In order o achieve maximum distribution, the coordinator for each process in the enterprise can be each located at a different host, it can also be replicated among various hosts. On the other extreme, in a more centralized police, they can all be located at the same host. This decision is outside the scope of our architecture and should be based on the conflicting goals of maximum resource (hosts) use or performance and management considerations.

2.2.2. Case coordinator. The case coordinator centralizes all information concerned with a particular case. It is responsible for managing the activities of that case, detecting failures and coordinating its recovery procedures, performing the garbage collection of activities and data, storing summary data in the history server, answering to queries about the case, notifying the process coordinator when a case is terminated, among other things.

Similar to the process coordinator, each case coordinator may be located on a different host, or many of them can be executed at a single host, based on decisions that balance performance and resource use.

The case coordinator is created by the process coordinator, with a copy of the process plan. The case coordinator creates a synchronization activity for each and-join specified in the process definition, adding their addresses (or names) to its plan.

2.2.3. Role Coordinator. The role coordinator is responsible for the management of the users that can perform a particular role and their state information. One of such user state information is, for example, the number of cases that the user is currently executing. With this information, the "programmer" role coordinator can answer queries like "Which is the least loaded programmer?" or "Who are all the programmers available?".

The role coordinator may also have access to the History Server, which stores information about completed cases, and to corporate databases. This information allows the role coordinator to answer queries like: "Who is the programmer with most experience in that kind of system?" or "Who was the programmer that implemented the previous version of that code?".

2.2.4. Synchronization activity. And-joins and Or-Joins are a particular problem in "mobile agent" architectures. The join must be defined before the case start by the case coordinator otherwise, a "mobile agent" would not know where to go when it needs to synchronize with other "mobile agents" that executed in different branches of the same plan. The synchronization activity will wait for all notifications (and-join) or the first notification (or-join) from its input activities before starting the following activity. Once all "mobile agents" from its input activities have moved to the synchronization activity (and-join), it merges all case and application data, and composes a new single "agent" that is moved to the host of the user that will perform the and-join output activity. In an or-joint, the first agent to arrive will produce a synchronization activity output.

2.2.5. Task Lists. The user interface is implemented as a task list, similar to a mailbox. This task list is controlled by an object executing in the user's host. The task list notifies the user of new tasks that he/she is supposed to execute or, if the allocation policy is one that offers activities to all people that may execute it, the task list allows the user to accept or reject the incoming task. Furthermore, the task list is the user's main interface to the WFMS itself, so it should also allow for some customizations such as selection of preferred external applications (say a particular text processor), change of the user's preferential host, policies for sorting the activities in the task list, and so on. It also collects information about the user's work load, which is useful for the role coordinators.

2.2.6. History Server. The history server (or servers) is a front-end for the repository of completed cases. When a case coordinator finishes its work, all relevant data concerned with the case is stored in the history repository. Such procedure allows the cases to be audited and the memory of the cases to be kept and further queried.

2.2.7. Backup Server. The backup server (or servers) is a front-end for the repository of the intermediary state of the active cases. As we mentioned above, the past state information about a case is stored in some of the hosts where the "mobile agent" have been before. Such user's host may not be trusted to hold the past state information indefinitely, nor to be active when this information is needed. The backup server executes in a more stable machine to which a user host will transfer, under the command of the case coordinator, the past state of the case.

There may be many backup servers in the systems, one per case, one for a group of cases, or even many for a single case. The identity of the backup server and the moment in which the backup will be performed is chosen by the case coordinator based on many considerations such as network and server loads. Once the backup is made, the user host can erase the past state information of that case.

2.2.8. Activity Manager. So far, we have been using the idea of a "mobile agent" as an intuitive description of the distributed nature of a case. However, the case is not really implemented as a "mobile agent", but as data that is transferred between two specific objects, the activity manager instances located on both the source and the destination hosts. There is no code mobility.

Each host that executes an activity has an activity manager that "implements" our architecture. It receives a new case for a specific user, activates the appropriate tool (using the activity wrapper), waits until the user finishes the activity and computes who should execute the next activity (by interpreting the plan that came along with the case and by querying the appropriate role coordinator). If the next activity is to be performed by a user, the activity manager sends the appropriate information to that user's task list, notifying the case coordinator that the activity has ended and who is the selected user to perform the next activity. After that, it transfers the case information to the (next) activity manager. It also receives requests from the case coordinator to transfer its case data to a backup server.

2.2.9. Wrapper Activity. The wrapper activities are objects that control the execution of a particular application program. It invokes the application with initial data or files and receives the application output files or data. It is a bridge between specific applications and the activity manager.

2.2.10. Gateway Activity. In order to cope with the WFMC Interoperability requirement, the gateway activity was defined. This component is responsible for bi-directional workflow data, control and process definition conversions between different WfMS. Thus, process executing in different WfMSs can be used in our architecture**.**

3. CORBA Implementation

The CORBA distribution environment [9] provides a set of functionality and transparencies that improves the distributed applications development. It implements an object-oriented bus, providing access transparencies (independence of hardware, language or operating system) and location transparencies (independence of the host where the object is executing). It offers benefits as inheritance, information hiding, reusability, polymorphism and other object-oriented features. It also allows the use of legacy applications, developed for different hardware and software platforms, through the IDL language usage, defining a common communication interface.

3.1. References to CORBA objects

The main problem using CORBA as the support environment for the distributed workflow architecture is its object reference. CORBA references as the standard IORs (Interoperable Object References) are too transitory for our application. These references contain information such as the hostname, where the object is located, and a port number that identifies the object in that machine.

Assuming that the completion of a case may take up to many months, or even years, one cannot assume that object will keep itself active, on the same port it was created, during the case whole execution life. Since the OMG CORBA specification still does not have an object persistence service, we had to create our own persistent CORBA object references. In our scheme, the objects are locally stored (made persistent), and identified using the following naming structure: (host, process, case, actor, activity, file) for files; (host, process, case, actor, activity) for activities; (host, process, case) for case coordinators; (host, process) for process coordinators; (host, backup-server) for backup servers, and so on. In order to provide transparent object persistence, each host has a Local Object Activator (LOA). The LOA executes as a daemon and saves the object's state in a local storage, the Object Repository.

For example, the case coordinator for Joe's request for the purchase of 500 paper clips (case C4375), of the process "purchase of office supplies" (process P12), in the host abc.def.com is identified by (abc.def.com, P12, C4375). To access such object (or formally to bind to such object), a process must send the reference (P12, C4375) to the LOA in machine abc.def.com, and it will instantiate and restore the state of that case coordinator, based on the information stored in the object repository. The LOA then returns the IOR of the newly restored object to be immediately used.

The implementation uses many CORBA services. The transference of the case information between two activity managers uses the Transaction Service, so that the data exchange is transactional, guarantee error-free. Authentication of servers and clients, cryptography of the data, and such can be implemented using the Security Service in CORBA.

3.3. Hierarchy of Interfaces

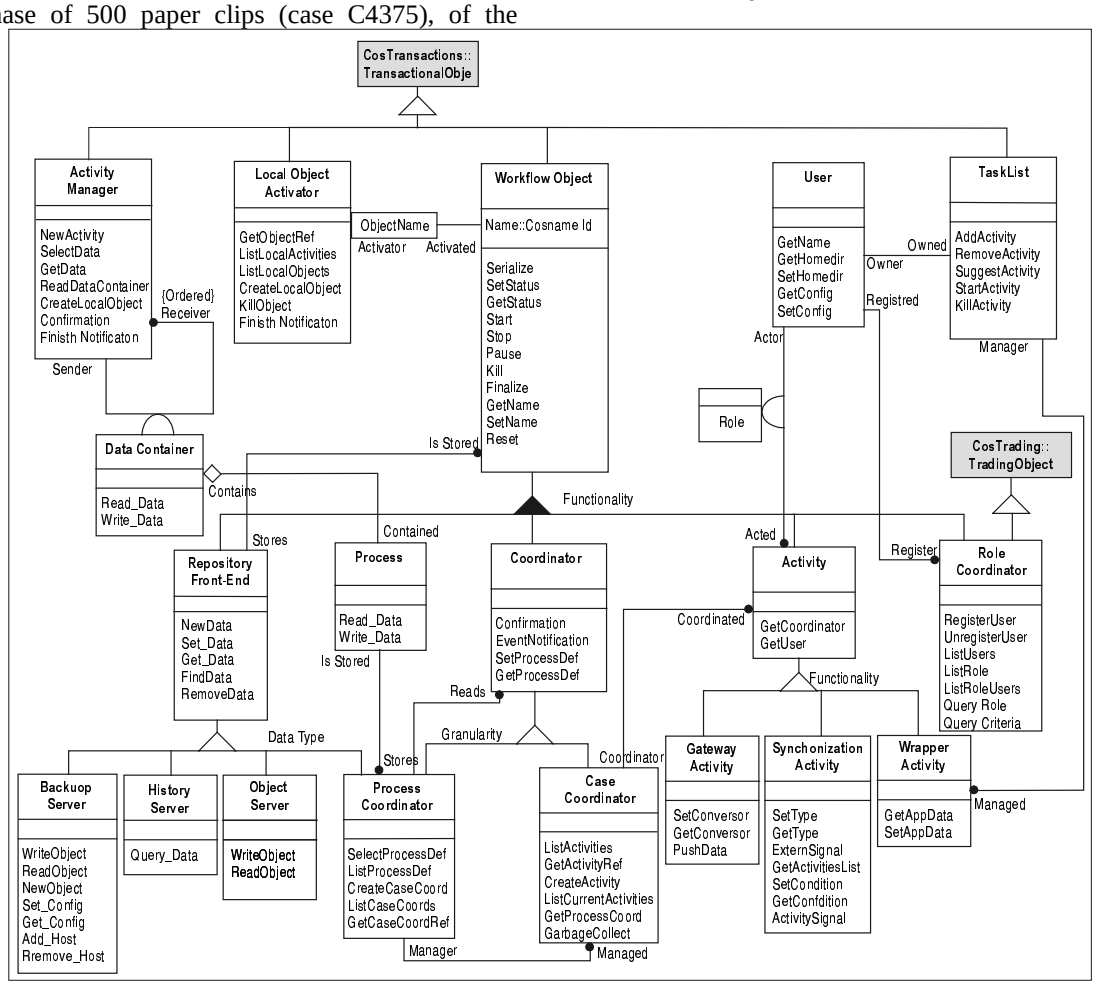

Figure 1. Interface hierarchy using Rumbaugh OMT methodology.3.2. Other CORBA services

We describe in Figure 1 the main aspects of the mapping between the components of the architecture and the CORBA environment, representing the hierarchy of IDL interfaces, according to the OMT methodology [13].

Gray rectangles represent inherited CORBA Object Services interfaces.

The interface hierarchy described in Figure 1 is composed of three interface groups (superclasses): Repository Front-Ends, Coordinator and Activity interfaces. Repository Front-Ends represents interfaces implemented by objects that manages data repositories. There are four kinds of data repositories: Backup, History, Objects and Process. Coordinators manage the execution of other system objects. There are Case and Process coordinators. Process Coordinators, through the multiple inheritance mechanism, are still used to control Process Definition data repositories. Activities are objects that control the task execution. They are managed by Case Coordinators, activating applications (Wrapper Activity), implementing synchronization points (Synchronization Activity) and allowing the interconnection among different WfMS (Gateway Activities). All the three groups, along with the Role Coordinator, are sub-interfaces of the Workflow Object. The Role Coordinator manages dynamic and history information concerned with the system users (User Interface). It inherits the CORBA Trading Service interface, allowing more sophisticated queries to be accomplished. Each user has an associated role. Workflow Objects are uniquely identified, being able to be controlled, located and stored. Local Object Activators are responsible for implementing the object persistence. The Activity Manager implements the activity sequencing and executing. All these interfaces inherit characteristics from the CORBA Transaction Service, allowing fail-safe data and messages interchange among them. The object sequencing uses Data Containers which stores data and process definitions. These containers are exchanged among Activity Managers. Task Lists stores information concerned with user allocated activities.

3.4. Execution Scenarios

In this section, some execution examples are presented. They emphasize the main objects of the architecture, showing the communication and relations among them. For simplicity reasons, we will leave out of the explanations and figures the interaction with the LOA and Object Repository.

3.4.1. Activity Sequencing. Figure 2 shows a typical example of activity sequencing. The wrapper activity 70, when finished, makes the Activity Manager 34 start the new activity creation process. The case coordinator 12, executing in a different host, is notified during whole sequencing procedure. The procedure starts with the role coordinator query, selecting an actor to perform the next activity, and sends a notification to the task list. If the activity is accepted by the selected user, the sequencing procedure starts. The activity manager 34 contacts the activity manager 35 in the user's host, and transfers all necessary data, together with the process definition, wrapped in a data container. Finally the wrapper activity 78 is created by the activity manager 35 and then started.

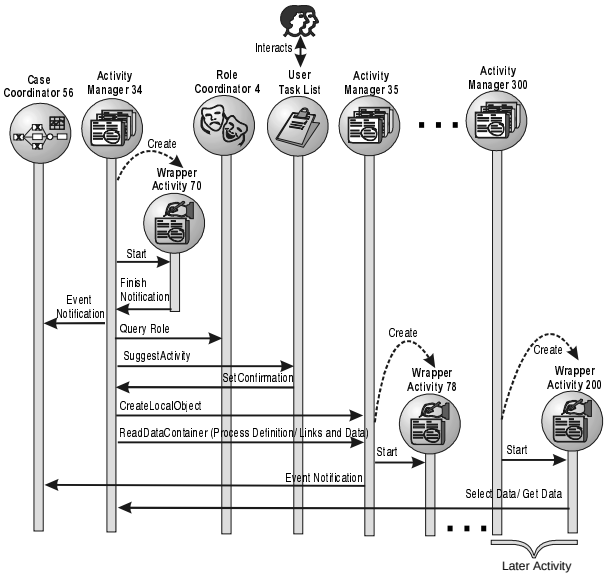

Figure 2. Activity Sequencing Temporal Diagram.

For performance reasons, only data necessary for the created activity is transferred. The reminder data are passed by reference, in order to be retrieved by subsequent activities.

3.4.2. Case Creation. The case creation procedure, shown in Figure 3, is started by a user request in the process coordinator 6 interface, which reference was previously obtained. This request results in the case coordinator 65, the wrapper activity 234 and the synchronization activity 345 creation, according to the process definition description. The role coordinator 23 is queried in order to get an actor satisfying the role of the activity to be created. The activity state and data are stored in the local object repository.

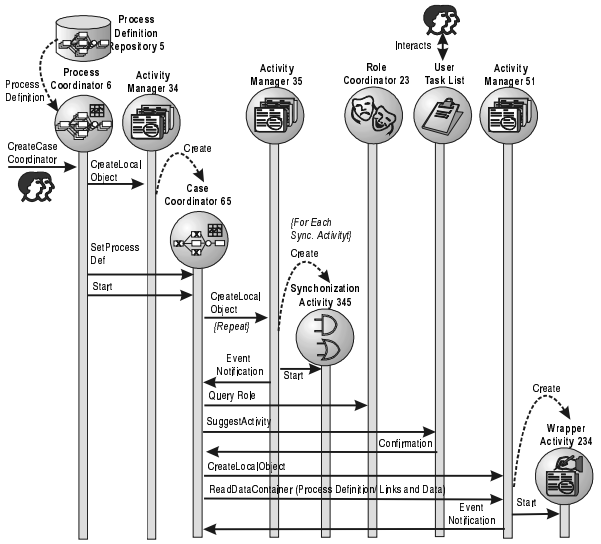

Figure 3. Case Creation.

3.4.4. Activities Synchronization. The synchronization procedure of the activities 70 and 71 are described in Figure 4. When finalized, each activity notifies the synchronization activity 5. After the evaluation of the programmed condition (and/or join), the wrapper activity 78 is created in the selected actor's specified object repository. The actor to perform the next activity is dynamically determined by querying the role coordinator 4. The case coordinator 12 is notified during the whole procedure.

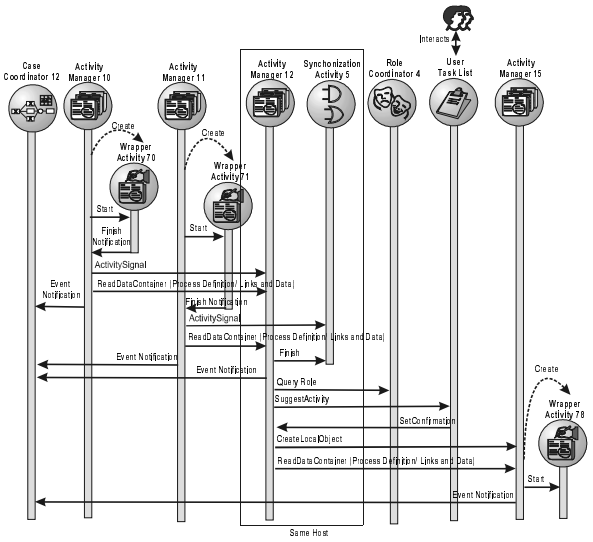

Figure 4. And Join Synchronization.

3.4.5. Case Finalization. The Figure 5 shows the temporal diagram of a case finalization procedure. By the end of each case, data stored in the object repositories and in the backup servers are removed in a garbage collection procedure, while synchronization activities are finalized. An execution summary containing relevant data for subsequent queries is stored in the history server 7. The whole process is managed by the case coordinator 56.

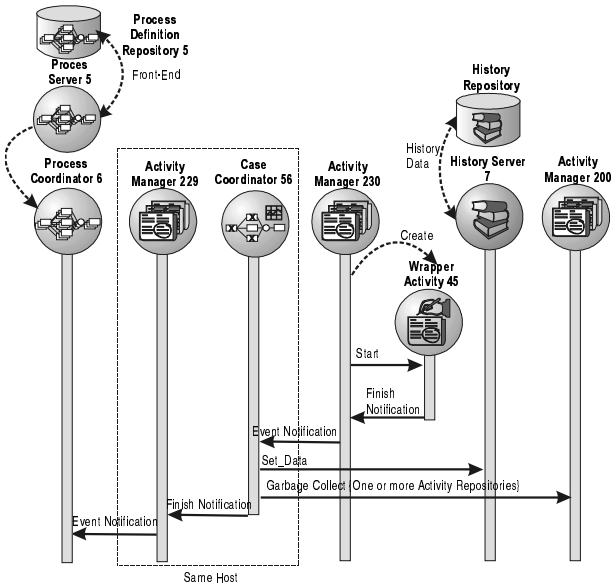

Figure 5. Temporal Diagram of a finalizing case.

3.4.6. Failure Recovery. The failure recovery process consists of stopping the current process (current executing activities), restoring the system to a previous stable state, modifying its process definition adding compensation activities and resuming the process by the end. The process is managed by the case coordinator, using data stored in the object repositories and backup servers spread over the system.

4. Implementation Issues

The system is being developed in the Institute of Computing at UNICAMP. It is being written in Java (Sun JDK1.1), using the Iona OrbixWeb3.0, a CORBA implementation written in Java. The local distributed system is composed of Unix Workstations, NCD Diskless X-terminals and Windows NT PC Workstations. All computers are integrated by a Local Area Network.

The development process is concentrated, initially, in the LOA and Activity Manager implementation.

The LOA is loaded by initialization scripts, just after the orbixd (daemon that implements the ORB in the OrbixWeb implementation) in the host operating system of each workstation in the system. This object has an infinite timeout, being configured to accept requests in one predefined Port of the operating system. As all the architecture objects are written in Java, their persistence are implemented using the j*ava.io.Serializable* interface. All the objects managed by the LOA are periodically

serialized and stored in their respective Object Repositories.

5. Related Work

Some of the components of the Exotica project [5,6,7,8], developed at IBM Almaden Research Center, have similarities to our proposal. In particular the Exotica/FMQM (Flowmark on Message Queue Manager) architecture is a distributed model for workflows, using a proprietary standard (MQI - Message Queue Interface) of persistence queues. The case is a message that is stored in these persistent queues, which are fault tolerant. Nevertheless, the proposal is not very detailed on how to deal with all the other requirements for a WfMS.

There are two OMG proposals for the Workflow Management Facilities under consideration [10,11]. These proposals are based on the WfMC standards and define a set of basic objects and interfaces, without giving more details about implementation or architecture. These submissions uses the not yet implemented CORBA facilities such as the Persistence and the Meta Object Facilities.

The Mentor project [12] of the University of Saarland aims at developing a scalable, traceable workflow architecture. Fault tolerance is achieved by using TP-Monitors and logs. CORBA is used as a communication and integration support for heterogeneous commercial components. Scalability is achieved by replicating the data in backup servers. Similar to our architecture, the data and reference to data are exchanged between Task List Managers as the activities are being executed and terminated. A limited first prototype was implemented and future extensions should include support for dynamic change of processes and the rollback of cancelled or incomplete workflows.

6. Conclusions

In this paper, we have presented WONDER, a distributed architecture for WfMS that satisfies all important requirements of a workflow management system including scalability and availability. The architecture is based on the idea that the case moves from user host to user host, following the process definition. A set of coordinators and servers were added to the basic architecture so that all other requirements for a WfMS could be also contemplated.

The use of CORBA as the supporting environment for such architecture has problems related to the persistence of objects. The standard CORBA references may not work in a domain in which objects representing activities and cases can take months or even years to be accomplished. A new form of referencing was proposed, and it mediates the accesses to CORBA objects. With the overcome of CORBA persistence problem, it seems to be an appropriate platform for such applications.

Acknowledgments. The first three authors would like to thank FAPESP (Process 98/06648-0) , CNPq, CAPES (Process 027/98), and the Pronex - SAE project - MCT/Finep for their support.

7. References

[1] "The Workflow Reference Model", Version 1.1, WfMC-TC-1003, Nov. 1994

[2] "Terminology & Glossary", Version 2.0, , WfMC-TC-1011, Jun. 1996.

[3] S. Jablonski, C. Bussler. *Workflow Management - Modelling Concepts, Architecture and Implementation*. International Thomson Computer Press, 1996.

[4] Alonso, D. Agrawal, A. El Abbadi, C. Mohan. "Functionality and Limitations of Current Workflow Management Systems". *IBM Technical Report*, IBM, 1997.

[5] Kamath, G. Alonso, R. Günthör, C. Mohan. "Providing High Availability in Very Large Workflow Management Systems", In *Proceedings of the Fith International Conference on Extending Database Technology (EDBT'96)*, Avignon, France, march 25-29,1996.

[6] Alonso, D. Agrawal, A. El Abbadi, C. Mohan, R. Günthör, M. Kamath. Exotica/FMDC: "A Persistent Message-Based Architecture for Distributed Workflow Management", *Proceedings of the IFIP WG8.1 Working Conference on Information Systems Development for Decentralized Organizations*, Trondheim, Norway, August, 1995.

[7] Mohan, G. Alonso, R. Günthör, M. Kamath, B. Reinwald. "An Overview of the Exotica Research Project on Workflow Management Systems", *Proc 6th Int'l Workshop on High Desempenho Transaction Systems*, Asilomar, Set. 1995.

[8] Mohan, D. Agrawal, G. Alonso, A. El Abbadi, R. Güthör, M. Kamath. "Exotica: A project on Advanced Transaction Management and Workflow Systems", *ACM SIGOIS Bulletin*, Vol. 16, No. 1, August, 1995.

[9] "The Common Object Request Broker: Architecture and Specification" - *OMG* - Revision 2.0 July 1995.

[10] "OMG Business Object Domain Task Force BOTF-RFP 2 join Submission - jFlow. Workflow Management Facility, revised submission". *OMG*, bom/98-03-04.

[11] Notel, "Workflow Management Facility Specification Submission", OMG Supported by University of New Castle upon Tyne, bom/98-03-01.

[12] Weissenfels, J., Wodtke D., Weikum G. e Dittrich A. K.- "The Mentor Architecure for Enterprise-wide Workflow Management", University of Saarland, Department of Computer Science, 1997.

[13] Rumbaugh, J. Et. Al. - *Object Oriented Modeling and Design*, Pretice Hall, 1991.