CORBA Based Architecture for Large Scale Workflow.

Master thesis defended in August 21st, 2000.
Institute of Computing – UNICAMP – Campinas - SP

Autor: Roberto Silveira Silva Filho, Jacques Wainer, Edmundo R. M. Madeira
IC – Institute of Computing
UNICAMP – State University of Campinas
13083-970 Campinas - SP - Brazil
{ robsilfi, wainer, edmundo}@ic.unicamp.br

Abstract

Standard client-server workflow management systems have an intrinsic scalability limitation, the central server, which represents a bottleneck for large-scale applications. This server is also a single failure 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, following a process plan, while the case 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 its limitations, advantages and project decisions described. This work present the experiments performed using the WONDER architecture, analyzing its overhead and scalability under an increasing number of cases, and an increasing volume of data exchanged. This master thesis initially generated a poster in the SBRC'99 and an article in the ISADS'99 conferences. The project presented in these conferences originated an article published in a special edition of the IEEE/IEICE Transactions on Communications journal in 2000.

Resumo

Sistemas de Gerenciamento de Workflow Tradicionais possuem uma limitação intrínseca em sua escalabilidade, o servidor central. Este servidor representa um gargalo de desempenho para aplicações de larga escala. Este servidor representa ainda um único ponto de falhas que pode desabilitar todo o sistema distribuído. Este trabalho propõe uma arquitetura complemente distribuída para sistemas de gerenciamento de workflow. Esta arquitetura é baseada na idéia de casos (instâncias de processos) móveis e autônomos que migram de nó a nó do sistema distribuído, seguindo um plano (definição de processo). Uma implementação baseada em CORBA desta arquitetura é apresentada, sendo discutidas as limitações, vantagens e decisões de projeto envolvidas em sua implementação e projeto. Este trabalho apresenta ainda experimentos realizados com um protótipo da arquitetura WONDER, onde é analisado o impacto da arquitetura no sistema distribuído assim como o comportamento da arquitetura na presença de um grande número de casos, e durante o aumento do volume de dados trocados entre atividades consecutivas. Esta tese de mestrado gerou inicialmente um pôster no SBRC'99 e um artigo no ISADS'99. A proposta apresentada neste último congresso evoluiu para um artigo publicado numa edição especial do IEEE/IEICE Transactions on Communications em 2000.

1. Introduction

Workflow Management Systems (WFMSs) are used to coordinate and sequence business processes, such as loan approval, insurance reimbursement, and other office procedures. These 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, parallel activities, synchronization points and so on. This description may also express constrains and conditions such as when the activities should be executed, a specification of who can or should perform each activity, and which tools and programs are needed during the activity execution [1].

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

Recently, there has been a surge on research prototypes that are not centralized, but distributed to different degrees. There are many issues that motivate the distributed WFMS architecture research. First, it is assumed that the centralized nature of standard WFMS will present a bottleneck for large-scale systems. The centralized storage of application and workflow data and the centralized interpretation of the workflow represent a single failure point that may severely paralyze the whole system and the whole business. Finally, workflow execution is itself a clear example for distributed systems: not only the execution of the activities are inherently distributed, because they are executed by users working on personal machines, but organizations are becoming more and more geographically dispersed.

This research presents a radically distributed architecture for WFMS in comparison with other proposed architectures. In the WONDER, (Workflow ON Distributed EnviRonment), the granularity of the distributed objects are the finer we are aware of. Moreover, the WONDER system addresses requirements and specifications that some of the other distributed architectures have not addressed. We will discuss these requirements and their impact on the architecture as follows.

1.1. Terms

We will use, from now on, the following definitions. A **process definition**, or a **plan**, is described in terms of the WFMC primitives: sequencing, and-join, and-split, or-join, and or-split [2]. A **case** is an instance of a process. Processes are defined in terms of **activities** or **tasks**, which are atomic actions performed by a single person or by a program. **Role** is the generic description of a set of abilities required to a person in order to perform an activity. People or programs that perform the activities are called **users** or **actors**. A particular user can perform many roles. If the user is a person she has a **preferential host**, a computer to where all her work related notifications and activities are send. In particular, the notifications are send to her task list.

1.2. Requirements for Workflow Systems

The following workflow management systems were addressed in this project.

- **Scalability:** The WFMS should not have its performance degraded due to the increase of: processes, cases, activity instances within a workflow, application data and actors.
- **Failure recovery:** The WFMS should deal with both, software and hardware failures with the

least intervention of users.

- **Availability:** The WFMS must not get unavailable/unreachable for long periods.
- **Monitoring:** The WFMS should be able to provide information about the current state of all cases and activities in execution.
- **Traceability:** The WFMS should be able to provide the history (trace) information of the current and terminated cases for auditing and recovery proposes.
- **Late binding (of executors):** WFMS should be flexible enough to allow the determination and allocation of users and applications, over demand, at runtime.
- **Interface with executors:** Users that will execute the activities must have a mechanism to select which activity to execute in the present, and which to execute later. They must be able to communicate to the system about their eventual unavailability and to customize their preferences.
- **Support for external applications:** The WFMS should be able to start external applications and to manage the data read and produced by these applications.

2. The Distributed Model

The WONDER architecture is based on the idea that each case is a mobile agent that migrates from host to host as the case activities are performed. The agent encapsulates both the application data and the plan for that case (workflow control data). The case 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 agent "figures out" another user to perform the next activity and migrates to his/her host. This mobile agent architecture copes with the scalability requirement, since there is no central control or data server, and there is no centralized performance bottleneck.

The WONDER uses the activity instance granularity that is, there are as many active distributed objects, as the number of instances of activities in execution, for all cases. There is no code mobility, only data and the object state are mobile. The program, or object, responsible for controlling the execution of an activity instance is the Activity Manager.

Each activity manager coordinates the execution of an activity in each case. When the next activity of a case needs to be performed, a new activity manager is created at the preferential host of the user assigned by the role coordinator to that activity. This activity is, then, configured with its specific data, and the previous activity case state. The plan interpretation is resumed and the activity is executed using the appropriate applications, through the use of application wrappers. The activity manager waits until the user finishes and then computes who should execute the next activity (by interpreting the plan that came along with the case state, and by querying the appropriate role coordinator). If the next activity is to be performed by a human actor, the activity manager sends the appropriate information to that user's task list, notifying the case coordinator that the current activity finished, informing who is the selected user to perform the next activity. After that, the current activity transfers the case information to the new activity manager.

2.1. Architecture Components

Some components were defined in order to deal with other WFMS requirements. A complete list of these components and their respective function is described in more detail in [3].

2.1.1. Late Binding - Role Coordinator.

In order to cope with the Late Binding requirement, it was defined a Role Coordinator component. There are as many role coordinators as there are roles defined in all processes of an organization.

Each role coordinator contains information about all the users that can perform that particular role.

2.1.2. Monitoring and fault tolerance - Case coordinator

A Case Coordinator component was defined to keep track of the case as it moves along. Each time the 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. This information is used during failure recovery procedures.

2.1.3. Fault tolerance - Backup Server and Ckeckpointing

The backup server (or servers) is the repository of the intermediary state of the active cases. For the fault tolerance sake, the past state information about a case is stored in some of the hosts where it executed (checkpointing). This information is used by the case coordinator to perform common failure recovery procedures. If a host failure during an activity execution, the previous checkpoint can be recovered and a new activity started. However, these users' hosts are neither trusted to hold the past state information indefinitely, nor to be active when this information is needed. The backup server runs in a more reliable and powerful machine. It receives the data and state of the past activities of an active case, under the command of the its case coordinator. Once the backup is performed, the state information can be erased from the users' hosts.

2.1.4. External Activities - Application Wrapper

The application wrappers are objects that control the execution of a particular invoked application. It launches the application with initial parameters and data and collects the application output. It is a bridge between specific programs and the activity manager. When the task finishes, the Wrappers notify the corresponding Activity Manager.

2.1.5. Interface with executors (actors) - Task List

The user interface is implemented as a task list, similar to a mailbox. The task list notifies the user of new activities that she is supposed to perform. This allows the user to accept or to reject the incoming activity according to the current specified policy. 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, change of the user's preferential host, selection of policies for sorting the incoming activities, and so on. It also keeps information about the user's workload, to be queried by the role coordinators.

2.1.6. Process coordinator

The Process Coordinator is responsible for the creation and management of case coordinators. If one needs to locate all instances of a process, the process coordinator also keeps track of all of its case coordinator instances, eventually handling the failure recovery procedure of these servers.

2.1.7. Synchronization Points

And-joins and Or-Joins are a particular problem in fully distributed architectures. Each join of a case must be created before the case beginning, otherwise a mobile agent would not know where to go when it needs to synchronize with other mobile agents that are executing 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 (output) activity.

2.1.8. Data on demand and proactive delivery of data

In order to improve efficiency, not all data is moved when the case moves from an activity manager to another. Instead, only a reference to the data that will be used by the next Activity Manager is transferred. The other case data is passed as references to the place it was last modified.

3. The use of CORBA

The WONDER architecture was implemented using the CORBA (Common Object Request Broker Architecture). This distributed communication framework was chosen for providing transparences of access and a high-level interface to the development of distributed applications. During the implementation of the architecture, some inadequacies of the CORBA 2.0 standard had to be worked around. The lack of object persistence services and the non-persistence of the CORBA IOR object references were the main problems.

The WONDER architecture does not rely on any standard CORBA naming service due to its centralization aspect and the problem with the IOR references. The Event, Notification and Transaction services are also not used due to the complexity and centralization of these services to the solution implemented. Instead, each host executes a locator that resolves markers (OrbixWeb user-friendly object names) to IOR object references. It works as a local name service. This locator, operating with the LOA (Local Object Activator), created specially for this architecture, is also used to implement the objects activation, deactivation and persistence.

4. Performance Tests

Although all elements of the architecture were specified in this project, for the performance tests, a simplified version of the architecture was implemented. This prototype implemented the core migration and activity execution mechanisms and some of the auxiliary components of the architecture, in order to simulate the actual system with the maximum fidelity. The WONDER prototype was submitted to performance tests in order to evaluate the overhead introduced by the architecture. It was also investigated which distribution configuration is more appropriate for different system loads, that is which WONDER objects should run in the same or different machines, according to the number of concurrent cases and the volume of data exchanged.

4.1. Overhead Tests

This text studied the impact of the architecture components and their migration mechanism in the performance of a distributed system. No case data and application were executed in this test. The test concluded that: Due to the use of the same mechanism of communication (IIOP over Sockets), implemented by OrbixWeb, for both local and remove method invocations, there is no significant performance difference between these two operations. The network latency delay is also not very expressive in the implementation. Hence, the times to create the objects and to send data are very close for centralized and distributed environments. The time spent in message exchange operations, negotiation and configuration do not represent more than 20% of the total activity time. The biggest latency is associated to the CORBA objects creation, specially, the start-up of the Java virtual machines that execute these servers, one for each server.

4.2. Scalability Tests

There were performed two different sets of scalability tests. In the first, it was studied the behavior

of the architecture in the presence of concurrent cases and, in the second, it was exploited its behavior with the increase of the data volume exchanged between consecutive activities. Both tests were run in distributed and centralized configurations.

A) Study of the influence of the variation of the number of concurrent cases. During these tests, the distributed execution of cases performed better than the centralized one for instances in which the number of concurrent cases was bigger than five. These tests allowed the conclusion that: There is no significant difference in the total execution time between cases running the experiments with two or four distributed hosts. The test also shows that the loss of performance, due to the processing of the asynchronous event notifications, in the distributed case, is not significant.

B) Variation of Data Volume. This test allowed the conclusion that the increase of the data volume exchanged between consecutive activities does not influence performance of the system as it did in the increase of the number of concurrent cases. In this test, the behaviors of the centralized and distributed executions were very similar. There is also no significant delay difference between the transfer of data within a single machine or between two distributed machines. Neither CORBA nor WONDER makes any optimization regarding local transfer of data, since the CORBA framework does not distinguish between local and remote operation invocation. Hence, independent of the location of the server object, the data exchange between client and server is performed using the IIOP over the TCP/IP stack.

The tests also allowed to conclude that: The use of CORBA objects, written in Java, executing in different virtual machines, do not have a good performance in centralized environments, in which the number of concurrent cases is big. In distributed scenarios, however, in which the number of servers executing in one node is smaller, its performance is acceptable. The biggest delay, associated to the mobility of the architecture agent, is the creation overhead of these objects. This procedure consumes memory and CPU. It also has direct influence in the performance of the other objects executing in the same host.

4.3. Conclusions

The overall impression of the tests allowed the conclusion that the WONDER architecture was not designed to be executed in a centralized way. For instances with more than five concurrent cases, the architecture presents a better performance if a distributed configuration is used. The growth of the execution time, given the number of concurrent cases, is less than half for the distributed configuration in comparison with the centralized one. Finally, the scalability of the system can always be improved by increasing the number of hosts in the system.

Future extensions will include support for dynamic change of process definitions and ad-hoc workflows. This last feature can benefit from the fact that the WONDER distributed and autonomous approach facilitates the change of the plan during the case execution. This is also facilitated due to the on-demand allocation of workflow activities and users, which is performed at runtime, using the process definition enacted by the mobile object.

5. References

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

[2] **WFMC-TC-1011**. *Terminology Glossary.* WFMC. 1996.

[3] **Silva Filho, R. S. , Wainer J. , Madeira E. R. M. and Ellis C.** CORBA Based Architecture for Large Scale Workflow. *Special Issue on Autonomous Decentralized Systems. IEEE/IEICE Transactions on Communications. Tokyo, Japan,* E83-B (5), pages 988-998. 2000.