WONDER: A Distributed Architecture for Large Scale Workflow using CORBA

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

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

1. Introduction

Workflow Management Systems (WFMS) are used to coordinate and sequence business processes. Such processes are represented as workflows, computer interpretable description of activities (or tasks), and their execution order, the data available and generated by each activity and synchronization points. It also expresses 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 invoked applications are needed during the activity execution [1].

Current 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 executed in server machine that typically stores both the workflow and the application data. 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 possibly the whole business itself.

The WONDER (Workflow on Distributed Environment) [3] is a distributed WFMS architecture implemented in CORBA (OrbixWeb 3.0 and Java). It addresses, in special, the scalability and availability requirements of Large Scale Workflow applications. Other requirements such as fault recovery, monitoring, support for external applications and traceability are also addressed. In the WONDER architecture, the control, the storage of the data, and the execution of the activities are all distributed into a network of workstations and servers.

2. General description of the WONDER Architecture

The WONDER architecture is based on the idea that each case is a "mobile object" that moves from hosts to host as the activities are performed. The case (instance of a process) encapsulates both the application data and the plan (workflow schema) for that case (named the state of the case), 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 finishes, the case "figures out" a user to perform the next activity and moves to her host. This "mobile object" 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 actor of an activity, but only a role. In order to cope with this requirement, a role coordinator component, containing information of each role, is defined: the case queries the particular role coordinator for an user to perform an specific activity, satisfying some requirements. Once figured out the user, the case moves to the user´s host and is inserted in the her task list.

Monitoring is also an issue in our 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 message to its case coordinator. Therefore the case coordinator knows where and at which process stage is the 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 backup servers, 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, history server, and others. These servers or coordinators, however, are not likely to be a bottleneck in the performance of the system as only notifications from the mobile object are sent to these servers. 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.

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] S. Jablonski, C. Bussler. *Workflow Management - Modelling Concepts, Architecture and Implementation*. International Thomson Computer Press, 1996.
[2] Alonso, D. Agrawal, A. El Abbadi, C. Mohan. "Functionality and Limitations of Current Workflow Management Systems". *IBM Technical Report,* IBM, 1997.
[3] Filho, R. S. S. Wainer, J. Madeira, E. R. M., Ellis, C. "CORBA Based Architecture for Large Scale Workflow". *The Fourth International Symposium on Autonomous Decentralized Systems*, March 21-23, Tokyo, Japan, 1999.