Communication in a Distributed Environment

I. Videira, P. Veríssimo, H. Sarmento

IST/INESC

R. Alves Redol, 9, P-1000 Lisboa

Phone: +351 1 3100242 Fax: +351 1 525843

e-mail: imv@inesc.pt

Abstract

In this paper a new communication service is presented. A distributed architecture with communication agents in each tool and in each host is adopted. The environment is modelled as a set or a hierarchy of groups of components, and multicast communication is used inside and between those groups. This approach resulted in significant improvements in throughput, network traffic and fault-tolerance when compared to centralised approaches. For throughput, a comparison with ToolTalk demonstrates the better performance.

1 Introduction

Communication is a key issue in any kind of distributed system. System components may need to share data in order to perform a task. Management tools need to pass control information to other tools. When using replication, replicas need to communicate in order to maintain consistency. Users of a certain system may need to be able to share information.

Nowadays, specially with the migration from the “client-server” paradigm to the “object-oriented” paradigm, which is resulting in software systems being made up of a large number of objects spread over networks, the communication service is becoming even more important. The latter has to provide, not only the necessary interoperation, but also the ability to locate objects implementing the services required by other objects, and to activate them, if necessary. In addition, in certain systems, communication may need to be authenticated – objects may only grant access to some other specific and correctly identified objects.

This dependency on communication makes critical its reliability. Therefore, it is necessary to have dependable communication mechanisms, providing, namely, fault-tolerance and security authentication.

As we will see, in the CAD area, the communication services available do not present a satisfactory solution, mainly because they do not provide all the required mechanisms. Also, the current solutions in this area do not consider dependability.

It is the objective of our work to develop a new communication service offering the mechanisms needed in the Design Environment. Dependability is a major concern, and greatly influenced the adopted architecture.

Along the text we will refer to a *Design System* as a set of tools and data. A *Design Session* is a running instance of the Design System. A *Design Environment* is a set of Design Systems in an organisation.

In this paper we start by giving a short overview of currently available communication technologies. Then, we describe the communication mechanisms necessary in a Design Environment, which is seen as a particular case of a distributed environment. Finally, we present the Communication Service developed and the results achieved so far.

2 Communication technologies

In this section we present a brief overview of currently available communication technologies. In [5] a more detailed description can be found.

The conventional technologies for communication are message passing and remote procedure calls (RPC). Communication based on message passing mechanisms presents application interfaces at very different levels, most commonly at the OSI transport level. RPC technology hides the communication from the programmer, making a remote procedure call look like an ordinary local call.

RPC has been a very popular technique for building distributed systems, by the fact that it insulates the application from the networking details. However, this is done at the expense of flexibility. This flexibility can be achieved with the messaging systems, mainly with those providing low level interfaces. Since programming at low level is hard and requires a deep knowledge of the underlying protocols, some new messaging systems have been developed that offer a high level interface without losing too much of the flexibility provided by the lower levels. The advantage of using messaging systems is that

the application developer can build highly concurrent systems in a relatively easy way, having more control over the information exchange and flow than with the RPC services. However, for purely blocking calls, RPC is still easier to use than messaging systems.

The “object-oriented” paradigm is now becoming popular and people are finding out that it is adequate for building distributed systems. Management of distributed objects, including the communication mechanisms that support them, are now emerging technologies, and are mainly based on object brokers or traders that forward requests sent by an object to the most adequate providers. The Common Object Request Broker Architecture (CORBA) [1] from the Object Management Group (OMG) is the most known of these communication architectures.

The technologies seen so far permit point-to-point communication and invocation of procedures/methods on remote processes/objects, which may not be adequate in all the cases. For instance, in a group of loosely-coupled applications, where state information about a shared resource has to be passed to all applications, it is necessary to have mechanisms providing one-to-many communication (multicast). If the group can shrink and grow dynamically, it would be very difficult to implement this with the already described communication technologies. Another example where the conventional technologies are not adequate is in fault-tolerant systems. These systems normally use replication to achieve fault-tolerance, and, thus, they need other communication mechanisms like many-to-many between groups of component replicas.

In the areas of CAD and CASE the need for multicast communication led to the concept of “multicast message server” – a centralised server through which all messages are routed. This server keeps a record of interests previously registered by tools and it delivers the messages received to the tools according to those interests. This communication architecture allows the interoperability of independently developed tools, as long as they use the same message set.

The concept of a multicast message server (in the context of CASE) was firstly introduced by Steven Reiss from Brown University and was implemented as part of the Field environment [3]. Several commercial products are now available using this concept, like ToolTalk from SunSoft, Broadcast Messaging Service from HP and Multicast Messaging Service from DEC. These products are used in CASE applications, respectively Sun SPARCworks, HP SoftBench and DEC Fuse. In the area of CAD the CAD Framework Initiative (CFI) is recommending an Inter-Tool Communication Service [2] also based on the concept of a multicast message server.

In the distributed systems world the concept of group orientation has been lately receiving great attention. Following this concept, distributed systems can be seen as sets or

hierarchies of groups of cooperating tools. This kind of organisation raises some problems to its participants, like maintaining connectivity, managing group membership, doing reliable multi-participant communication, etc. In the recent years several solutions have appeared, such as group communication and group management.

3 Communication in a Design Environment

In a Design Environment (like in many other distributed environments) several communication mechanisms are required, with different qualities of service, to serve the specific needs of the components of the environment:

- A point-to-point mechanism with high throughput and/or low latency for closely cooperating tools (like a group of simulators).
- A communication mechanism with weaker requirements on performance, but simple to use, for loosely coupled tools. To make tools independent from each other, this mechanism should allow tools to exchange information without needing to know who owns the information or where it is located.
- A multicast mechanism to spread information about important events. It should be able to multicast events not only in the set of tools of the Design Session, but also in design teams, or in the whole environment, if required.
- A transactional mechanism for database access.

Currently, there is no single communication service fulfilling these requirements, which means that a tool developer needs to be familiar with several communication mechanisms in order to integrate his tools and to make them interoperate with the environment.

Furthermore, as the operability of the Design Environment is relying more and more on the communication between its components, it is necessary to have a dependable Communication Service. Therefore, this service must present continuous availability. Also, some authentication mechanisms need to be provided in order to protect data from being accessed by non authorised users. Until now these aspects were not considered in the CAD area.

The CFI Inter-tool Communication architecture is a first approach to the Communication Service required in a Design Environment, and, therefore, it is the basis for our work. We extended the used concepts in order to fulfill the requirements expressed above and to add dependability.

4 The Message Switcher

The conceptual architecture of the Communication Service is based on a component, designated by Message Switcher, which provides two distinct services: a Notification Service and a Request/Response Service (Fig. 1). Tool messages are delivered to the appropriate service by the Service Selection mechanism.

Figure 1: The Message Switcher conceptual architecture.

The Notification Service has basically the functionality of the CFI ITC Message Server, as it is able to forward messages to tools, based on the message contents and on the tools' interests. The Tool Information Center (TIC) is used to keep information about the tools, like status, communication end-points used and the list of message interests. The TIC can be loaded statically or dynamically. The objective of providing static tool registration is to enable the Message Switcher to start a tool, if necessary.

The Request/Response Service handles all the requests for data and routine/method execution. The Provider Information Center (PIC) maintains a record of the location of remotely callable routines or data (objects). This service is able to find the adequate provider for requests or to locate a specific provider, if requested to. The PIC can be statically or dynamically loaded. Static service (provider) registration allows the Message Switcher also to choose a provider that is not running, to serve a certain request.

The basic idea of the Message Switcher is to provide, through the same application interface, a notification service and a request/reply service, following, for the latter, the same philosophy of the former. This means that a tool willing to offer a certain service or to share some piece of data, uses the Message Switcher to advertise that. The information about what is provided and who provides it, is stored and used when a request is issued.

It is also possible to establish point-to-point connections using the Message Switcher. A tool willing to establish a direct link with another tool can ask that to the Message Switcher. Functions for dealing with point-to-point messaging are provided.

Being able to control all the services/data in the environment, the Message Switcher can also be used to control the access to those services/data using authentication mechanisms.

5 Implementation

For fault-tolerance and performance reasons mainly, we decided to adopt a distributed solution in the implementation of the Message Switcher. The service is distributed by two types of entities: an agent in every host of the Design Environment – the Host Agent (MSTHost), and an agent linked with each tool – the Tool Agent (MSTool) (Fig. 2).

Figure 2: The Message Switcher agents.

Communication is done inside multicast groups. Several different groups are possible, like the Design Session Group, the Design Team Group, or the Design Environment Group. Messages from a tool in a Design Session under a certain user ID, can be multicast to any group to which the user belongs. Assignment of users to groups (others than the Design Session Group) is done by the system administrator or the project manager.

Tools register in the Host Agent using functions of the Tool Agent. The Tool Agent is responsible for sending private messages directly to some specific tool, or to a multicast group. Notifications or requests that do not specify a provider are always multicast to the Design Session Group, and will be received by every Host Agent having tools registered in that Design Session. A response to a request is also multicast to the Design Session Group and can, therefore, be seen by the interested tools.

A Host Agent is responsible for forwarding the messages received within the group to the interested tools in its host and to find adequate providers for requests. Each Host Agent keeps information only about tools running in its host (and about tools that registered statically).

Figure 3: Notifications path.

Figure 3 shows the path followed by a notification: 1) it is sent by the Tool Agent of Tool B to the multicast address of the Design Session Group; 2) each Host Agent that serves tools in that group receives that notification; 3) each Host Agent consults its TIC to verify if any of the tools it serves is interested in that notification; 4) each Host Agent

forwards the notification to the tools interested. It is important to highlight that message forwarding is done in parallel in both Host Agents.

Inter-Session communication can be easily done by explicitly multicasting messages to specific Design Session Groups, provided that the tool has permission to do it. Otherwise, following the principle of sending “to whom might be interested”, tools can register interest in receiving specific messages from other Sessions. The Host Agent will forward them the inter-session messages that match with any of their interests.

This distributed implementation of the Communication Service is inherently fault-tolerant to network failures, because there is an agent of the service in every host that can work isolated. Also, host failures are not a problem, since a Host Agent only serves tools in its host. Therefore, it is only necessary to mask process failures of the Host Agent, which is very easily achieved.

Multicast communication is implemented with IP Multicasting through the BSD socket interface. As the delivery of multicast datagrams is not reliable (as in ordinary IP datagrams) it is necessary to implement a reliable protocol on top of it. In a LAN (which is currently the environment of a Design Session) the loss or duplication of packets is not very frequent. Therefore, the introduction of reliability will have a minor impact on performance, because there will be few retransmissions.

6 Performance evaluation

Some tests were done, in order to compare the performance of our approach with the available results for ToolTalk [4]. These tests only evaluate the performance of the Notification Service for two reasons: first, at the time the Request/Response Service was not yet entirely finished and second, there are no published figures for ToolTalk performance in serving requests.

We adopted the same test scenario used for obtaining the ToolTalk results:

- Tests were done on a networked SUN IPX workstation
- In each test a tool sends 5000 notifications
- Each notification has a size of 200 bytes
- Each tool receiving a message has two interests registered
- The tool supplied callback routine, to be called at message arrival, performs no operation

- Notifications are sent within the scope of a Design Session

In Table 1 it is shown the elapsed time between sending the first notification and receiving the last one in each tool, as a function of the number of tools receiving the messages. Figures are shown for the Message Switcher and for ToolTalk.

		Sender	Receiver 1	Receiver 2	Receiver 3	Receiver 4
1 tool	Mes. Switcher	19 s	19 s			
	ToolTalk	97 s	97 s			
2 tools	Mes. Switcher	24 s	24 s	25 s		
	ToolTalk	153 s	153 s	154 s		
3 tools	Mes. Switcher	28 s	28 s	29 s	30s	
	ToolTalk	206 s	206 s	208 s	207 s	
4 tools	Mes. Switcher	29 s	29 s	30 s	31s	34s
	ToolTalk	266 s – <i>Average value for all columns</i>				

Table 1: Comparison of the Message Switcher and ToolTalk performance.

Another test was conducted, in order to evaluate the performance when tools are running in different hosts. For this test we had two tools running in different hosts, where one is the sender and both receive the burst of notifications. We verified that, as expected, the obtained throughput is exactly the same as if only one receiving tool exists (5000 messages in 19 s). Delivery is done locally in each host, not affecting delivery in other hosts.

7 Conclusions

In this paper a new Communication Service using a distributed architecture that resulted in considerable advantages over the current centralised solutions, is presented. Those advantages can be summarised as follows:

- A great reduction in network traffic: each multicast message is only sent once to the net and arrives to every Host Agent in the Design Session.
- A great improvement on performance: due to network traffic reduction and to the fact that message filtering and forwarding is a distributed task, performed locally in each Host Agent.
- Achievement of fault-tolerance to host and network failures.

Performance evaluation experiments could confirm the expected improvement in throughput of our approach when compared with a centralised one. The other two referred advantages are obvious (do not need confirmation).

Currently, the Notification Service and the Request/Response Service implementations are finished. We are implementing the CFI ITC interface specification on top of the Message Switcher. This will allow tools using CFI ITC to also use our component, and, namely, it will allow the integration of the Message Switcher in the JESSI-COMMON-FRAME framework.

As future work, we intend to further develop the Message Switcher. Several new features that will provide better support for group communication will be added, like support for group definition and management. Furthermore, other features that will, in geral, improve the service will also be added, such as the integration of a load balancer to the request/reply mechanism, configuration mechanisms, message priorities and support for communication at the user level.

Acknowledgments

This work has been partially funded by the European Commission Esprit Research and Development Programme in the sector of Microelectronics under Project JESSI-COMMON-FRAME (#7364) and by Junta Nacional de Investigação Científica e Tecnológica under Project STRDA/C/TIT/78/92.

References

- [1] *The Common Object Request Broker: Architecture and Specification*, Object Management Group and X/Open, December 1991.
- [2] *Inter-Tool Communication Architecture, Version 0.9.1-012292*, CAD Framework Initiative, January 1992.
- [3] Steven Reiss, "Connecting tools using message passing in the Field environment", *IEEE Software*, July 1990.
- [4] Results published in CFI mailing list "cfi-itc@cfi.org", June 1993, and USENET newsgroup "alt.sys.tooltalk" FAQ, 1993.
- [5] Idalina Videira, *Report on the analysis of various communication services*, ESPRIT 7364 Jessi-Common-Frame, Deliverable D1.ITC-1, November 1992.