# Efficient Communication in a Design Environment*

Idalina Videira+, Paulo Veríssimo++, Helena Sarmento+

INESC - Instituto de Engenharia de Sistemas e Computadores

Rua Alves Redol, 9, 1000 Lisboa, Portugal

**Abstract**

*This paper presents a new communication service. The novelty of the work resides in the distributed architecture adopted which is based on communication agents in every tool and in every host of the design environment. The importance of the work is demonstrated by the results achieved: improved performance, reduced network traffic and fault-tolerance to host and network failures.*

## 1 Introduction

Current Design Systems are composed of an increasing number of very specialised tools, supported by a software infrastructure providing another large set of tools and services, with all of these components distributed over a network. To achieve the interoperability needed in such an environment, the Communication Service plays a fundamental role.

Three key aspects have to be considered in a Communication Service for the achievement of good tool interoperability support. Firstly the performance, because it has a large influence on the overall performance of Design Systems. Then, the completeness of the communication mechanisms offered, which is essential to facilitate the task of the tool developer/integrator. Finally, the dependability, which is gaining importance as Design Systems become more complex and distributed. For dependability, at least continuous availability should be provided, in order to prevent tools from becoming isolated. Also, security authentication mechanisms are needed to protect data from being accessed by non-authorised users. Until now these aspects have not been considered in the communication services used in EDA.

In this paper a new communication service, addressing the aspects referred to above, is presented. This service offers three basic communication mechanisms: efficient point-to-point data exchange, remote data access with different degrees of location transparency and multicast of events within Design Sessions and Design Teams.

* This work was partially supported by European Commission Esprit Programme under Project JESSI-COMMON-FRAME (#7364) and by JNICT under Project STRDA/C/TIT/78/9.

+ Also affiliated with Instituto Superior Técnico, Technical University of Lisbon, Portugal.

++ Also affiliated with Faculdade de Ciências, University of Lisbon, Portugal.

The novelty and the importance of the work presented resides mainly in the approach followed and in the results achieved. Having dependability and performance as driving goals, a distributed architecture was adopted, with communication agents in every host and in every tool of the environment. The group orientation paradigm was followed, with the environment structured in groups where multicast communication is used.

As will be demonstrated later on, the advantages of the approach followed over current centralised solutions are: a significant improvement in performance, a reduction in network traffic, and fault-tolerance to host and network failures.

In this paper we start by analysing currently available communication technologies. Then, we describe the communication mechanisms necessary in a Design Environment. We present the Communication Service developed and the results achieved so far. Finally, we draw some conclusions and describe future work.

## 2 Communication technologies

Conventional communication technologies include message passing and remote procedure calls (RPC). The evolution of distributed systems gave raise to other communication paradigms, like multicast and group communication. The following is a short overview of relevant available communication technologies. In [21] a more detailed description can be found.

RPC has been a very popular technique for building distributed systems, because it makes a remote procedure call look like an ordinary local call. Several RPC implementations are currently available. Some are widely used, such as the ONC RPC [16] from Sun and DCE RPC [12] from the Open Software Foundation, based on the previous NCS RPC [5] from HP/Apollo.

Though, the simplicity of the RPC programming model is achieved at the expense of flexibility: by definition a remote procedure call is a blocking and asymmetric operation. Flexibility can be achieved with 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 [18], [25], [20] have been developed, offering higher level interfaces without losing too much of the flexibility provided by lower levels. The main advantage of using messaging systems is the possibility of building highly concurrent systems in a relatively easy way, having better control over the information exchange and flow than with RPC. Nevertheless, for purely blocking calls, RPC technology is still easier to use than messaging systems.

In an attempt to overcome the limitations of the RPC model, several features have been added to RPC systems, such as the usage of threads packages for simulating non-blocking calls [12] or the possibility of calling multiple instances of a procedure in parallel [19]. However, in our opinion, these features contradict the RPC model.

**33rd Design Automation Conference ®**


DAC 96 - 06/96 Las Vegas, NV, USA

Other communication mechanisms based on models addressing more naturally distributed applications' needs should be used/ developed.

One of the most popular paradigms used to build distributed systems is the "object-orientation". Management of distributed objects and communication mechanisms to support them, are now emerging technologies. Those mechanisms are mainly based on object brokers or traders that forward requests sent by an object to the appropriate providers. The Common Object Request Broker Architecture (CORBA) [2] from the Object Management Group (OMG) is the best known of these communication architectures. However, the communication model used in CORBA is nothing more than a generalisation of the remote procedure call paradigm.

The technologies seen so far only permit point-to-point communication and invocation of procedures/methods on remote processes/ objects, which may not be adequate in all situations. For instance, in a group of cooperating tools, where state information about a shared resource needs to be passed to all tools, it is necessary to have mechanisms providing one-to-many communication (multicast). Also, to build fault-tolerant systems, when replication techniques are used, many-to-many communication mechanisms between groups of component replicas are needed.

In Electronic and Software Design environments the need for multicast communication led to the concept of "multicast message server", using the policy of "sending to whom it might concern". For that purpose, a centralised server is used, through which all messages are routed. The message server keeps a record of interests, previously registered by tools, and delivers the received messages according to tools' interests. For CASE environments and based on this concept, several commercial products are available, such as ToolTalk from SunSoft, Broadcast Messaging Service from HP and Multicast Messaging Service from DEC. In the EDA area, CAD Framework Initiative (CFI) is recommending an Inter-Tool Communication Service [4] also based on the multicast message server concept. This concept can also be found, at the commercial level[1], in Cadence Communications Manager [1] (from the former Valid Logic), which is part of Cadence Design Framework II.

ToolTalk [7] is the most important of the products using the multicast message server approach because it is recommended both as a COSE (Common Operating Software Environment) and CFI standard. In spite of this, we believe that it is not the definitive solution for communication in distributed Design Environments: it does not provide all the required mechanisms[2], which are described in section 3, its performance is poor and it is not dependable. Although, the main ToolTalk limitation is due to its implementation, which is based on an RPC package. Building a multicast messaging system on top of an RPC, which is implemented on top of messages, would obviously result in an inefficient product.

In the distributed systems world, the concept of group orientation has lately been receiving great attention. Following this concept, distributed systems are structured in groups of cooperating tools. This type of organisation raises problems, such as maintaining connectivity, managing group membership, doing reliable multi-participant communication, etc. ISIS [8] is one of the most important works using the group orientation paradigm. It provides mechanisms to form and manage process groups and to build group based software. As the objective of ISIS is to help building tightly coupled distributed systems, the stress was mainly on dependability and less on performance. Consequently, the performance obtained is not good. For instance, multicasting a message to a group with one member has performance similar to an RPC call [8].

Considering the relative importance of the standardisation bodies and the foreseen impact of their recommendations, the most relevant standardisation activities in the area of Communication Services are the ones being carried out by OSF, OMG and COSE[3]. Therefore, the relevant standards are, respectively DCE, CORBA and ToolTalk, being each of them solutions to different types of communication requirements.

# 3 Communication requirements in a Design Environment

In a Design Environment (as in other distributed environments) several communication mechanisms are required 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. A possible scenario where this mechanism is needed is a group of cooperating simulators. These tools are highly inter-dependent and require a very efficient mechanism to allow synchronisation and share of results. The communication should be private because, normally, a group of simulators behaves conceptually as a single entity.
- A mechanism providing location transparent access to data, for loosely coupled tools. To make tools independent from each other, this mechanism should allow them to exchange information without needing to know who owns it or where it is located. A possible scenario is the request of data (e.g., a cell's structure) by a simulator. Data can be provided, for instance, by a schematic editor, a design database or a synthesis tool. The communication mechanism should be able to find data and to deliver it to the simulator.
- A multicast mechanism to publish information about important events. It should be able to multicast events not only to the set of tools of the Design Session, but also to Design Teams, or to the whole environment, if required. This mechanism can be used to advertise the completion of certain design tasks, or database updates. This kind of information can be useful to other tools being executed in the same Design Session, to other members of the same Design Team, or even to other teams.

Currently, there is no single communication service simultaneously fulfilling these requirements. Therefore, a tool developer needs to be familiar with several communication services in order to integrate his tools and to make them interoperate with the environment. In particular in the EDA area, where tool developers are more concerned with their own CAD algorithms than with networking details, it is important to provide a uniform and complete solution to the communication requirements.

As designers are always concerned with shortening the design time, and as the design process implies of a considerable amount of inter-tool communication, performance of the communication mechanisms used is of utmost importance.

1. Other commercial design frameworks and systems were investigated (such as Mentor Graphics, Synopsis and Viewlogic) and no particular inter-tool communication system is referred. Either files or, specially in simulation backplanes, point-to-point communication are currently used.

2. See page 5 of [3] for a description of the kinds of problems ToolTalk does not solve.

3. We left CFI out of this list because, in this area, CFI is following COSE.

Furthermore, as the operability of the Design Environment is increasingly relying on the communication between its components, it is necessary to have a dependable communication service. At least, this service should present continuous availability. Also, authentication mechanisms are needed in order to protect data from being accessed by non-authorised users. Until now these aspects have not been addressed in the EDA area.

# 4 The Message Switcher

This section presents the communication service developed, starting by describing the conceptual architecture and then presenting the implementation aspects.

## 4.1 Conceptual architecture

A Design System consists of groups of tightly coupled tools working together with loosely coupled tools. However, it can be considered a loosely coupled system, viewing a group of tightly coupled tools as a single entity.

As “sending messages to whom it might concern” is a good principle to achieve the tool independence required in loosely coupled distributed systems, this principle was adopted in the architecture of our communication service. This architecture is based on an abstract entity, designated by **Message Switcher** (MS) [22], which acts as a virtual resource provider for requests, through a Request/Response Service, and as a filter for notifications, through a Notification Service. The MS also provides efficient point-to-point communication mechanisms to fulfil the needs of tightly coupled tools.

The Notification Service has basically the functionality of the CFI ITC Message Server, as it is able to forward messages to tools, based on message contents and on tools previously registered interests. A Tool Information Centre (TIC) is used to keep information about tools, such as status, communication end-points and list of message interests. The TIC can be loaded statically (reading files) or dynamically (receiving information from tools). Providing static tool registration allows the Message Switcher to start the execution of a tool in reaction to an event. Although it is typically the Session Manager that starts tools, the Message Switcher can do it, if such a manager does not exist.

The Request/Response (R/R) Service provides the mechanisms by which tools make requests and receive responses in a transparent manner. It uses a Provider Information Centre (PIC) to store and retrieve the location of remotely callable routines or data (objects). Location transparency is permitted but not enforced. Therefore, this service allows clients to specify (or not) providers for their requests, or to characterise the desired types of providers. The PIC can also be statically or dynamically loaded. Static loading allows the Message Switcher to choose a provider that is not running to serve a particular request.

The Message Switcher uses a Message Logger to record messages, which can be used, for instance, to update new tools. A garbage collector is used for removal of unneeded messages.

We have seen how the Message Switcher uses the policy of “sending to whom it might concern”. However, this policy is not always adequate as the flexibility introduced is achieved at the expense of efficiency. Closely cooperating tools, which obviously have a deep knowledge of each other, and frequently exchange large amounts of information, need very efficient communication mechanisms and have no need for a policy promoting tool independence. This is the case, for instance, of two cooperating simulators. To provide an efficient solution the Message Switcher is able to establish and maintain point-to-point connections.

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

With the conceptual architecture presented, the Message Switcher is able to provide the required mechanisms referred in section 3. The following section describes the implementation, where performance and dependability requirements were addressed.

## 4.2 Implementation

Mainly for fault-tolerance and performance reasons, we decided to adopt a distributed solution in the implementation of the Message Switcher. As so, the communication service is distributed by two types of entities: an agent in every host of the Design Environment - the **Host Agent** (MSHost), and an agent linked with each tool - the **Tool Agent** (MSTool) (Figure 1.)[4].

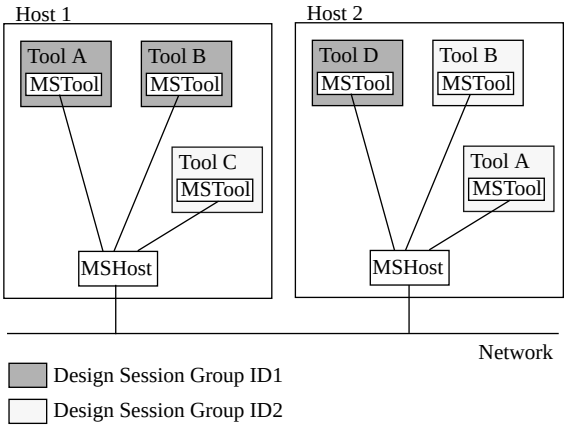


Figure 1. The Message Switcher agents.

The group orientation paradigm is followed. Several groups can be established, such as the Design Session group, the Design Team group or the Design Environment group. Multicast communication is used within groups. Most messages are sent within the scope of Design Sessions. However, messages from a tool in a Design Session under a given user ID, can be multicast to any group to which the user belongs. Assignment of users to groups (others than Design Sessions, which is automatically done) is done by the system administrator or project manager.

Host Agents are responsible for forwarding the notifications received within a multicast group to the interested local tools, for finding adequate providers for requests and for forwarding requests and responses. Each Host Agent only serves and maintains information about tools running in its host. This policy makes the Message Switcher invulnerable to host and network failures: a host may become isolated but its running tools can continue to communicate with each other; other Host Agents do not need any information residing in the isolated Host Agent to proceed. If, instead of isolated, the host had crashed, communication between the tools running in the remaining hosts would not be affected.

The information kept in the Host Agent’s TIC about a particular tool is replicated in the Tool Agent. This strategy allows that:

4. This architecture may resemble the one used in the old NCS Location Broker [5], which anyway used an extra entity managing information about all the resources in the environment. However, the functionality provided by this old product was merely of a location broker, which was supposed to be used in conjunction with the NCS RPC.

- When the Host Agent suffers a process crash failure and is restarted, the Tool Agents can supply it the information about tools, remaining the tools unaware of the failure.
- Messages the sending tool also receives are delivered immediately by the Tool Agent.

The Tool Agent provides the application interface to send notifications and requests to multicast groups, and to establish and maintain point-to-point communication channels. The Tool Agent and its local Host Agent[5] have a direct connection[6], which is used for:

- announcing joins and leaves from a Design Session by tools;
- registering/unregistering interests by tools;
- advertising exportable data or routines by tools;
- publishing communication end-points for accepting point-to-point communication, as well as the protocols served by tools;
- sending all messages to tools by local Host Agents;
- acknowledging reception of messages to tools by local Host Agents in the low level protocol.

Host Agents are the true group members. Tool Agents can send multicast messages but can not receive them. Messages only arrive to the Tool Agent via the local connection with its Host Agent (after filtering), or from point-to-point connections.

Figure 1. shows the path followed by a notification. In step (1) the notification is sent by Tool Agent of Tool A to the multicast address of the Design Session Group and is received by all Host Agents in that group. Notification forwarding to the interested tools is then done in parallel by those Host Agents - step (2).

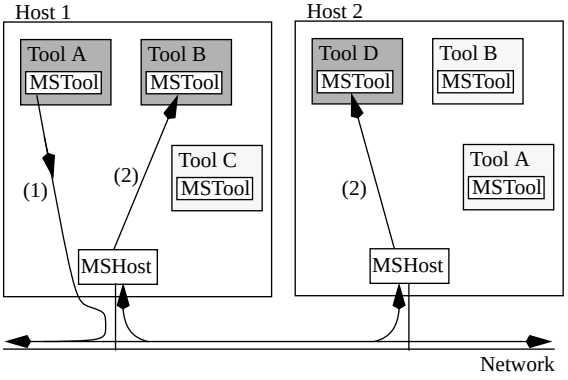


Figure 2. Notifications path.

To deal with requests, and when a provider is not specified, a negotiation occurs between Host Agents in order to find the best provider. When a provider is found, its local Host Agent forwards it the request. The reply is later multicast to the group, and, can be seen by other tools. When a reply does not arrive within a certain time interval the requester is notified by its local Host Agent.

Tools able to serve point-to-point connections advertise to MS their communication end-points and protocols served. A tool requiring to establish a private connection asks MS for the end-point of the communication partner. The MS behaves in this case as a name server. Then, tools communicate privately using the MS Tool Agent interface. Point-to-point communication is based on SPOOK [18], a communication mechanism built on top of TCP/IP and Unix IPC.

This implementation of the Message Switcher is inherently fault-tolerant to network failures, because there is an agent in every host capable of working 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.

**Current status**

Multicast communication was implemented with IP Multicasting [10] through the 4.3BSD socket interface. Since the delivery of multicast datagrams is not reliable (as in unicast IP datagrams) we implemented a reliable protocol [23] on top of it. This protocol, implemented at the process level, includes: 1) sequences numbers to detect lost, duplicated and out of order messages; 2) retransmission and message reordering; 3) window based flow control. The highest quality of service level provided is source ordered delivery.

The Message Switcher is written in C++. The Notification and the Request/Response Services correspond in total to around 9000 lines of source code. The CFI ITC interface specification [4] was implemented on top of the Message Switcher in order to allow its integration in the JESSI-COMMON-FRAME framework[7] [15]. This interface corresponds to 1100 lines of source code. The size of the Tool Agent class library with the CFI interface is 95 Kbytes.

# 5 Evaluation

In order to evaluate the performance of the Message Switcher, a series of tests were done to each of the communication mechanisms offered. All tests were conducted on SUN SPARC IPX workstations with SunOS 4.1.3 on a lightly loaded 10 Mbit/s Ethernet.

**Notification Service**

To allow a comparison with published ToolTalk performance values [6], similar test methods were adopted. In each test, a tool sends 5000 notifications of 200 bytes. All tools run in the same machine and each tool has two registered interests. The callback routine to be called at message arrival is a dummy routine. Notifications are sent within a Design Session.

| # receiver tools | Message Switcher | | ToolTalk | |
|---|---|---|---|---|
| | elapsed time (ms) | performance degradation (%) | elapsed time (ms) | performance degradation (%) |
| Self | 0.6 | -- | -- | -- |
| 1 | 4.8 | -- | 19.4 | -- |
| 2 | 6.2 | 29 | 30.8 | 59 |
| 3 | 7.2 | 50 | 41.6 | 114 |
| 4 | 8.2 | 71 | 53.2 | 174 |

Table 1. Comparison of Message Switcher and ToolTalk performance.

Table 1. shows the elapsed time between the sending of a notification and its arrival to all receivers, as a function of the number of

5. The “local Host Agent” is the Host Agent residing in the same host as the tool being referred.

6. Using Unix IPC.

7. This framework uses the CFI ITC interface.

receivers, with both the MS and ToolTalk. The different results are mainly due to the different underlying mechanisms used by MS and ToolTalk. The multicast mechanism of MS leads to a much better performance than the RPC used by ToolTalk, as expected.

Table 1. also shows the performance degradation when increasing the number of receiver tools. Values express the degradation obtained always with respect to the case of one receiver tool. As expected, less degradation exists with the Message Switcher because for each notification only one message is sent to the net[8] whereas ToolTalk sends a number of messages equal to the number of receiver tools plus one.

Table 2. compares the performance of the Message Switcher protocol layers. Row 1 shows the time required for sending a message of 200 bytes directly using IP multicast. Tests of the Unix IPC mechanism showed that it takes 0.6 milliseconds to transmit a message of 200 bytes between two processes. If the same process sends messages to $n$ tools then the total time is about $n \times 0.6$ milliseconds[9]. Row 2 shows the total time taken by the underlying communication mechanisms used during a notification. Row 3 shows the performance of the previous version of the Message Switcher, without the reliability layer (results published in [22]). Row 4 shows the performance of the current version with the reliable protocol [23].

| Protocol layers (time in ms) | 1 tool | 2 tools | 3 tools | 4 tools |
|---|---|---|---|---|
| 1 - IP mcast | 2.6 | 3.2 | 3.6 | 4.4 |
| 2 - IP mcast + Unix IPC | 3.2 | 4.4 | 5.4 | 6.8 |
| 3 - MS without reliability | 3.8 | 5.0 | 6.0 | 7.3 |
| 4 - MS with reliability | 4.8 | 6.2 | 7.2 | 8.2 |

Table 2. Performance of the Message Switcher protocol layers.

Subtracting the values of row 2 from row 3 we obtain the time taken by the Message Switcher filtering and forwarding operations, which is about 0.6 ms. As we can see, this value is independent of the number of receiver tools. This means that the increase in machine load when the number of receiver tools increases is the main cause for performance degradation.

Subtracting the values of row 3 from row 4 allow us to evaluate the cost of introducing reliability. In case of one receiver tool the overhead obtained is 26% and this value decreases as the number of receivers increases, being only 12% with four receivers.

When tools (sender and receivers) reside each in a different host the sending of a notification takes only 3 ms, regardless the number of receivers. This test demonstrates the claim that delivery time in one host is independent of the number of receivers in other hosts.

Comparing with other reliable multicast protocols (such as [9], [11], [17]) the figure of 3 ms is excellent. To our knowledge only protocols implemented at the kernel level (such as [13], [24]) exhibit better performance.

**Request/Response Service**

Tests done to the Request/Response Service showed an average latency[10] of 9 ms (with 200 bytes requests in a SPARC IPX), which a much smaller value than the one obtained, for instance, with the DEC implementation of CORBA - 644 ms [14] (in a SPARCstation 10). With MS, serving a request means sending two IP multicast messages, two Unix IPC messages and the negotiation messages exchanged among Host Agents. The time taken by the transport mechanisms to perform these operations, in the absence of negotiation messages, is two times 3.2 ms[11] which makes 6.4 ms. Therefore, 2.6 ms are spent by the Message Switcher in the reliability layer and in filtering and forwarding operations. When requester and replier reside in different hosts the performance measured is approximately the same, because the time taken by negotiation messages is compensated for by the lower load of the machines.

We could expect that a request/reply in these conditions would take a time equivalent to twice the time taken by a notification. Actually, it takes a bit less because there is one less filtering operation and also because optimisations are done with the I/O multiplexer mechanism that result in lower idle times.

**Point-to-Point Communication**

The point-point communication mechanism presents a latency of 3 ms and a throughput of 500 Kbyte/s for 1 Kbyte messages and 450 Kbyte/s for 10 Kbyte messages. For tools running on the same machine a throughput of 720 Kbyte/s is obtained for 10 Kbyte messages. Using TCP directly under the same conditions we obtain a throughput of 700 Kbyte/s for 1Kbyte messages and 650Kbyte/s for 10 Kbyte messages.

# 6 Conclusions and future work

In this paper a new communication service using a distributed architecture is presented, which offers considerable advantages over the current centralised solutions:

- A great reduction in network traffic: each multicast message is sent only once to the net and arrives at every Host Agent in the Design Session.
- A great improvement in 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.

Experimental results confirm the superior performance of our approach over others.

With the CFI ITC interface the Message Switcher is part of the JESSI-COMMON-FRAME (JCF) framework last version[12]. The software was also transferred to ICL and it is being considered as a possible solution for the communication between ICL CAD tools.

As future work, we intend, firstly, to improve the flow control algorithm by reducing the imposed idle times, in order to reduce the overhead introduced. Then, we intend to further develop the Message Switcher. Better support for groups, such as mechanisms for group definition and management will be addressed. The order requirements in the delivery of multicast messages will be further analysed and mechanisms to enforce the required order will be developed. Furthermore, we intend to improve the service in general, by introducing new features, such as integration of a load bal-

8. Delivery to the tools is done through a local (Unix IPC) connection and does not go to the net.

9. Actually, it is a bit less because optimisations can be done when using the Unix *select* mechanism.

10. The elapsed time since the request is issued until the reply is received by the requester tool.

11. Value taken from Table 2. in the case of one receiver tool.

12. As the second phase of JESSI-COMMON-FRAME project was not approved the last version of the framework did not result in a product.

ancer to the request/reply mechanism, configuration mechanisms, message priorities and support for communication at the user level.

# References


[1] *Cadence Communications Manager Reference Manual 2.0*, Cadence Design Systems, December 1993.

[2] *The Common Object Request Broker: Architecture and Specification*, OMG and X/Open, December 1991.

[3] *Designing and Writing a ToolTalk Procedural Protocol - A White Paper*, Sunsoft Inc., June 1992.

[4] *Inter-Tool Communication Programming Interface*, Version 1.0.0-112592, CFI Release 1.0, January 1992.

[5] *Network Computing System (NCS) Reference*, Apollo Computer Inc., 1987.

[6] Results published in CFI mailing list "cfi-itc@cfi.org", June 1993, and USENET newsgroup "alt.sys.tooltalk" FAQ, 1993.

[7] *The ToolTalk Service - a SunSoft White Paper*, Sunsoft Inc., June 1991.

[8] K. Birman, R. Renesse, *Reliable Distributed Computing with the ISIS Toolkit*, IEEE Computer Society Press, 1994.

[9] J. Chang, N. Maxemchuk, "Reliable Broadcast Protocols", *ACM Transactions on Computer Systems*, Vol. 2, No. 3, August 1984.

[10] S. Deering, *Host Extensions for IP Multicasting*, RFC 1112, August 1989.

[11] D. Dolev, S. Kramer, D. Malki, "Early Delivery Totally Ordered Multicast in Asynchronous Environments", *23rd Annual International Symposium on Fault-Tolerant Computing (FTCS)*, France, June 1993.

[12] B. Johnson, *A Distributed Computing Environment Framework: An OSF Perspective*, Technical Report DEV-DCE-TP6-1, OSF, January 1992.

[13] M. Kaashoek et al, "An efficient Reliable Broadcast Protocol", *Operating Systems Review*, Vol. 23, No. 4, October 1989.

[14] F. Kuhl, W. Neal and H. Cohen, "Object Request Broker: Foundation for Distributed Simulation", *Software - Practice and Experience*, Vol. 24, No. 12, December 1994.

[15] B. Steinmüller, "The JESSI-COMMON-FRAME Project - A Project Overview", Proc. of the 3rd Int. IFIP Workshop on Electronic Design Automation Frameworks, March 1992.

[16] Sun Microsystems, "RPC: Remote Procedure Call Specification" (RFC 1057), *Internet Network Working Group Request for Comments*, NIC, 1988.

[17] T. Montgomery, *Design Implementation, and Verification of the Reliable Multicast Protocol*, MSC Thesis, West Virginia University, 1994.

[18] P. Santos, H. Sarmento and L. Vidigal, "Ghost-Spook: user interface and process management in the PACE framework", *Proc. of the European Design Automation Conference*, March 1990.

[19] M. Satyanarayanan and E. Siegel, "Parallel Communication in a Large Distributed Environment", *IEEE Transactions on Computers*, Vol. 39, No. 3, March 1990.

[20] D. Schmidt, "An object-oriented interface to IPC services", *The C++ Report*, November-December 1992.

[21] I. Videira, *Report on the analysis of various communication services*, ESPRIT 7364, JCF, Deliverable D1.ITC-1, November 1992.

[22] I. Videira, P. Veríssimo and H. Sarmento, "Communication in a Distributed Environment", *Proc. of the 4th Int. IFIP Workshop on Electronic Design Automation Frameworks*, December 1994.

[23] I. Videira, *Reliable Multicast Communication with IP Multicastin*, Internal Report, INESC, September 1995.

[24] A. Weaver, "The Xpress Transfer Protocol", *Computer Communications*, Vol.17, No. 1, January 1994.

[25] M. Zelenick, "A portable, network-transparent communication system for message-based applications", *Proc. of the 6th International Conference on Distributed Computing Systems*, 1986.