TINA Architectures And Performance: A Telepresence Case Study

P.P.Jogalekar[*], G. Boersma[*], R. MacGillivray[**], C.M.Woodside[*]
[*] Department of Systems and Computer Engineering,
Carleton University, Ottawa.
[**]Bell Northern Research Ltd., Ottawa.

Abstract

TINA is a distributed computing framework based on Open Distributed Processing which is targeted partly towards management of telecommunications services. This paper is partly a TINA case study and partly an examination of how performance can be addressed during design of a TINA system. The case study part examines a recently developed software suite which manages a telepresence system with video connections, video services and sources, and conference management services. The software was designed without using TINA, but here it is re-interpreted within TINA definitions and concepts. Then a performance assessment framework is described, with the results of performance tests, a model, and an analysis of system scale-up properties based on the model. From the example some lessons are drawn about how performance assessment should be done. Our long-term goal is a method for assessment of performance problems at an early stage in development of distributed software.

1. Introduction

TINA represents a fundamental shift in the way telecommunications services may be managed, towards a distributed computing paradigm.This paper analyzes the service management software in a teleconferencing system from the perspective of the TINA framework. The architecture of this teleconferencing system, which is a part of the Ontario Telepresence Project[1], is at present experimental and fluid. We describe it using TINA concepts, in order to obtain a single, unified view of telecommunications service management and distributed processing, and then consider performance aspects of the software design, and how to address performance questions during design.

We will view the control of a telepresence conference as a small-scale example of service management, one of the domains for which TINA was developed. TCC has been developed as part

1. The Ontario Telepresence Project is managed by the Information Technology Research Institute and the Telecommunications Research Institute of Ontario.

of the Ontario Telepresence Project (OTP), which investigates the technical and social issues of implementing telepresence into business environments. Telepresence is the application of computing and telecommunications to support a sense of social proximity among collaborators, in spite of separation by distance and time [integrating]. The TCC software [nsi] is a new software architecture based on recent experience using the software in the OTP lab [iiif] which is based on software developed by Xerox Parc [media]. Telepresence Conference Control (TCC) manages multipoint conferences between automated conference rooms, and service management issues within this narrowly defined context are the issues of conference control. TCC provides a framework which allows independent distributed applications, such as device controllers, to co-operate within the context of a teleconference. TCC and the applications make use of the Network Software Infrastructure (NSI), which provides a multipoint communication service, dynamic resource allocation, and a directory service.

We present a TINA-like engineering viewpoint model (with eCOs) of the system. We further show how the Stochastic Rendezvous Network model (SRVN) and the TimeBench tool can be used for analyzing the performance of the system. The purpose of this case study is: (1) to explore the architectural issues in a distributed telecommunications system in the TINA framework, (2) to describe an approach for performance modelling of TINA architectures, (3) to study possible restructuring of the architecture by analyzing the bottlenecks, (4) to study scalability of the architecture.

2. Overview: Telepresence Conference Control (TCC)

Telepresence Conference Control (TCC) manages multipoint conferences between automated conference rooms. Each conference room is equipped with a number of devices, such as cameras and VCRs, which are connected to a workstation by a RS232 serial connection. Software on the workstation can control the devices by issuing commands to and receiving responses from the devices via this connection. Audio and video signals are sent between conference rooms using a coder/decoder (CODEC) which compresses the signal using H.261 or a proprietary compression algorithm and ships the signal via connections provided by the phone company, such as switch-56 or ISDN lines.

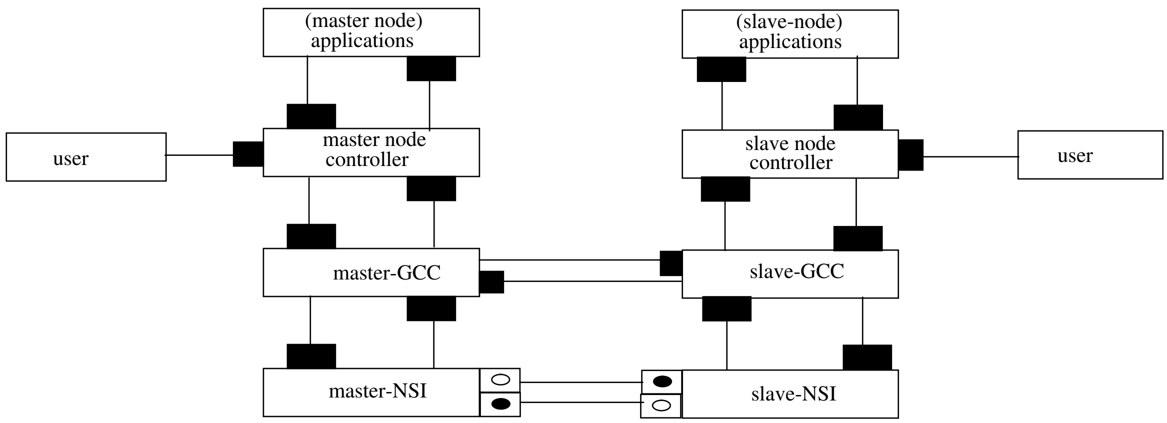

Fig. 1. TINA Engineering Model of TCC

Figure 1 illustrates the architecture of TCC and the type of communications between each of the components, using TINA notation.

The GCC server acts as a coordinator between applications in the context of a conference. Each application presents its own interface to the user, with its functionality restricted to the scope of the application. A node consists of a set of applications which participate in a conference as a single entity. A node controller is used by the user to request services from the GCC server. Peer-level GCC servers, each responsible for coordinating applications within their own node, communicate using the services of the NSI server. Applications use the NSI server directly for communication between distributed peer-level entities.

2.1 Applications

Applications assume one of two different flavours: the first type of application is a physical resource controller, which provides services for the control of devices from any node participating in a conference. The second type of application is a logical resource controller, which provides services to access logical resources from any node participating in a conference. An application which allows a user at any node to control a VCR or a camera is an example of a physical resource controller, while an application which allows a user at any node to access a distributed database or naming service is an example of a logical resource controller.

Each application is instantiated at every node participating in the conference. The peer-level application entities then communicate via the services of the NSI server to provide a unified service to users at all nodes within the conference. For example, an application to control a camera is instantiated at every node within a conference. Each instance of the application accepts requests from the user to control the camera and forwards the request to the node where the physical device is located. This node sends the appropriate command to the camera and then broadcasts the change in the status of the camera to all nodes participating in the conference.

A node may also request ownership of a device, preventing all other nodes from controlling the device until ownership is released. The protocol for grabbing a device is implemented among peer-level instances of an application, and may be different for each device.

Instances of applications at a node are clients of a node's GCC server. Instances of applications may be dynamically created and destroyed without affecting the state of the conferences in progress, or the state of the GCC server.

2.2 Node Controller

The node controller is the user interface to the GCC server. The node controller informs the user at a node of changes in the state of conferences, which are communicated to the node controller by the GCC server. A set of services which manage the policy of how conferences are maintained in the particular conferencing environment is translated into a set of standardized services from the GCC server.

The node controller is a client of a node's GCC server. Therefore, the node controller may be terminated and restarted without affecting the state of the conferences in progress.

2.3 GCC Server

The state of all conferences in progress is maintained by the GCC server, while each node in the conferencing environment has a separate instantiation of the GCC server. The conference server implements a subset of the Generic Conference Control (GCC) standard, as developed by the ITU telecommunications standardization committee [GCC].

A master GCC server maintains the state of a conference. It is the GCC server at the node where the user requested to initially create the conference. The slave conference servers request the current state of a conference from the master GCC server in order to complete a service request made by the user at the node. The GCC server at the master node may then inform the user of the master node through its node controller of the request by the slave GCC server. The user at the master conference node may then allow or disallow the request by the slave GCC server, and the result is returned to the user of the slave GCC server. Any changes to the state of the conference are broadcast to all nodes in the conference.

One node is selected as the administrator, which keeps track of all conferences in progress and responds to requests by other nodes for this information. Other nodes make these requests by first joining an administrative conference, where the administrator node is designated as the convener of the administrative conference. The administrative conference is a special conference, which manages the list of conferences in progress. All nodes participating in the administrative conference are informed of new conferences being created, or of conferences being terminated. Therefore, all nodes which require notifications of new conferences must be joined to the administrative conference.

Typical operations that the node controller can request of the conference server include conference establishment and termination, conference conductorship, roster management of current conference participants, and a roster of current conferences. Applications can request the current state of a conference, but may not change it. Applications are informed in changes in the conference state.

The conference server is multi-threaded. Therefore, clients may make any number of requests from the conference server at a time, before the result to a previous request has been returned to the client.

2.4 NSI Server

The NSI server provides a multicast communications abstraction based on the Multipoint Communications Service (MCS) standard, as developed by ITU telecommunications standardization committee [mcs,mcs2]. Nodes communicate in domains, which reflect interactions between a set of nodes over a period of time. Each domain has a set of channels, where a channel may be a point-to-point channel or a multicast channel. A domain also has a set of tokens, which provide a method of ensuring only one node accesses a centralized resource.

Nodes communicate on channels. Communication on a channel may be unordered or ordered. Multiple source ordering [spauter] is provided by sending all messages on a channel to a master NSI node, which then retransmits the messages to all nodes connected to the channel. Therefore, all nodes will receive messages in the same order regardless of which node is the source of the message. The master node also maintains the current state of all tokens.

The conference server uses multiple source message ordering for broadcasting changes in the conference state to all participant nodes. Therefore, all conference servers at all nodes will maintain the same conference state since they will all receive the messages in the same order despite the source of the message.

The NSI servers currently communicate with peer NSI servers using TCP/IP connections. However, the NSN could also use a true multicasting service, such as IP multicast.

3. TINA Representation of the Teleconference Control System

Fig 1 describes the TCC system in terms of TINA engineering computational objects (eCOs). The diagram illustrates the objects that form a cluster at a master node, and at a typical slave node.

There could be many slave nodes interacting with one master node in a conference domain, but they interact with the master in a similar fashion, hence only one slave node is shown.

The master GCC server and the master NSI server are the manager objects which manage the slave GCC servers and the slave NSI servers respectively.

The user at the master node and the user at any slave node interact in a similar way with their respective node controllers, as shown in the diagram. The user is shown as a TINA object to account for the think-and-type time. The node controllers and the GCC servers at the master and the slave sites communicate with each other using the request-indication-response-confirm protocol. The operational interfaces provided by the objects in this diagram are interrogations. The master NSI and the slave NSI's have a stream interface between them, in each direction. Thus a channel object exists between master and slave NSI servers, providing a multicast abstraction.

In the existing telepresence system, there is no exact equivalent of a trader object (e.g. a name/directory server) but it has been planned for the future versions of the TCC system. There is also no equivalent of a notification server in the existing system, but can be suggested for future developments based on TINA guidelines. Notification is a function of the master server.

The binding between the master and the slave objects in this system is "implicit", or fixed by the infrastructure. Explicit bindings occur between remote objects (like GCC servers), during change in the states of conferences.

The TINA concepts of binding factory and negotiable quality of service during a binding are not yet supported in the current implementation of the tcc system.

The application objects control the devices being used in a conference, for example, the video cameras and the codecs. The data transfer between the participants of the conference occurs via these codecs and is not shown in the TINA architecture since it is not a part of the communication services management software. The teleconference control occurs over the TCP/IP connection between the nodes.

4. Performance Assessment Framework

Software designs and specifications clearly have performance consequences, but it is a puzzle how to evaluate them without first completing the implementation and deployment. We are developing a model-based approach (e.g., [maps and paths]) which uses a framework based on software paths or "operational profiles". An operational profile has three parts:

- the scenarios to be evaluated. Each one is triggered by a particular class of input event or of user-level end-to-end service request, and provides a part of the total functionality of the system. Twelve scenarios were used in this case.

- the relative frequency of the scenarios, in an assumed operational situation. This describes how the user community is expected to use the system.
- an execution environment which describes the deployment of the software in processes and processors, resource scheduling, the number of nodes and users, and other logical and physical resources.

Performance parameters should ideally be measured for each scenario separately, so they can be combined freely. Then the parameters are combined in a model with the relative frequencies and an environment definition, and the model is solved to give a prediction. We are using the Rendezvous Nets/Method of Layers model, which gives a high level of abstraction but also includes queueing effects for software servers as well as for processors [refs, include new tools paper]; more detail is given below.

Typically we are interested in a range of scenario mixtures and environments, which characterize the “operational envelope” of the system. In this work we assume the operational envelope is compact enough to be described by example, that is by a reasonably small set of models. For example here we will evaluate over a range of sizes of the system, scaling up the number of users.

This section now describes the scenarios, the tests that were performed and the performance parameters; then the following two sections describe the analysis based on these parameters.

4.1 Test Configuration

TCC was implemented as a set of processes running under UNIX, with a thread package for internal concurrency. The implementation tested here ran under SunOS, although it is intended to be portable to other platforms. Each object (capsule) shown in figure 1 was implemented as a UNIX process, with communication by TCP and sockets.

The tests used a configuration of two nodes. One node was operated as the administrator node, or the master of the administrative conference, while the other node was a slave of the administrative conference. The slave is the convener (or master) of all the other conferences created during the test. Each node has a node controller but no applications; the time for services to be completed on behalf of the applications is the same as for the node controller. The service times for the node controller are also treated as the service times for applications in the final model.

The slave node creates several conferences, and the master node approves the creation of each conference. The slave node then requests GCC services from the master node, such as queueing the list of conferences in progress. The slave node then leaves each conference.

The individual operations of the tests map to different services offered by the GCC server as defined by the GCC standard. Each service of the GCC server is called once for each conference.

4.2 Test Results

The results of the test were a set of processing times (CPU times) for the different operations of each process. The measurements which could be obtained gave the average time for an operation following a request arriving at an identified port. This averaged together the times for different operations occurring within different scenarios, however it was judged from the design that the operation times being averaged would in any case be very similar.

Table 1: Operation Times

Input from Object:	Input to this Object	Operation Time by Second Object (in msec)
m_applications	master_nc	8
master_nc	m_applications	400
master_nc	master-gcc	7
master_gcc	master_nc	400
master_gcc	master_nsi	2
master_gcc	slave_gcc	7
slave_gcc	slave_nc	384

The symmetrical operations by substituting master for slave take the same amount of time, except for the one given explicitly in the last row of the table. The rates of flow of requests are causally related to the activities of the users, and to the way that processes propagate requests to each other. The rates of flows, relative to the rate of user requests, were measured as follows:

Table y

Table 2: Flow Rate Factors in the Test Results

Flow From	Flow To	Rate Factor, Relative to the User Event Input Rate
m_application	master_nc	0.16
master_nc	m_application	0.583
master_nc	master_gcc	0.521
master_gcc	master_nc	0.380

Table 2: Flow Rate Factors in the Test Results

Flow From	Flow To	Rate Factor, Relative to the User Event Input Rate
master_gcc	master_nsi	1.26
master_gcc	slave_gcc	1.72
slave_gcc	master_gcc	1.687
slave_nc	slave_gcc	0.356
slave_gcc	slave_nsi	0.914

Using these flow factors requires an assumption that the mixture of events in a conference is similar to that used in the test. A more developed procedure, as outlined at the beginning of the section, would measure the scenarios separately and obtain these flow factors for each one, and then use data on conference operation to give the total flows. At the present state of the project there is little user data yet, so this rather crude assumption seemed to be appropriate.

From the above parameters the performance model was constructed.

5. Performance Analysis

The TINA model employs client_server relationship between its components which matches the Stochastic Rendezvous Network performance model [WNMP94]. We have embedded this model in a graphical software design tool called TimeBench [Tools95]. Figure 2 shows the TimeBench display corresponding to the TINA model in Figure 1. TimeBench offers a windows based user interface for graphical description of this model and automatically generates the code to simulate the model.

The parallelograms are the processes, with multiple symbols to indicate either a pool of processes (in the case of the applications) or a multi-threaded process (elsewhere). The prefix m_ indicates a master node and a prefix s_ indicates a slave node. User interactions arrive at the entries m_user and s_user respectively. Processing load parameters and event rate parameters are derived from the values in Tables 1 and 2.

The model could be solved for mean delays for reqeust-reply interactions, throughputs at different points and the utilizations of the processes. Here, we shall examine the scalability of the system when it is deployed with all the processes on one side of Figure 2 being allocated to a single processor. To judge scalability we will examine the processor utilization.

5.1 Two Nodes, Many Users

The first analysis is for a scenario similar to the test conditions, with two nodes and one conference domain. The number N of users attached to the second (slave) node varies; we asume they are loggin remotely to access the conference control. This corresponds to a deployment with two sites, with one GCC server per site and to a conference in which all the participants except the conference conductor are attached to the slave site. Our interest is to see how many users can be attached to the slave node. With N = 1 the model parameters are taken to be the same as those found in the test. The following request flows will be proposrtional to N, the number of users at the slave side:

- between gcc's, because of the requests from the users to the master_gcc and the replies,

-from gcc to nsi on both sides, to carry the traffic,

-betwen each gcc and the corresponding nc, to convey notifications of changes made by the users

The rate of input of service requests by each user must be estimated. We took a notional value of one request per minute, per user, giving an input rate of 0.0167 requests per sec at the master and 0.0167N at the slave node controller

The processor utililization results are shown in Figure 3. As N increases, the utilizations climb steadily (unevenness in the curves are due to simulation result variance). The slave side is considerably more heavily utilized, as expected because of the large number of users. Given that operating system overheads ar not all accounted for in our measurements, we might take a practical estimate of the capacity of this configuration as 50 users, which is quite satisfactory.

5.2 Scalability over Many Nodes with One User Each

The second analysis considers a conference with M slave nodes and one user per node. The model will again represent one master node and one slave node.The additional request traffic from the M-1 other slave nodes is modelled by an additional traffic input into the entry m_gcc_e2, and by additional notification traffic to all slaves. Again for M = 1 the model is assumed to be the same as for the tests; the input rates attached to the flows labelled in Figure 2 are:

- m_nc_e2 and s_nc_e2: 0.1/sec

- m_nc_e1 and s_nc_e1: 0.00633 M

- m_gcc_e2: 0.028 M

- s_gcc_e2: 0.0s87 M

The other flows are determined by the same propagation relatinoships as before.

The utilization results are in Figure 4 The master node utilization is slightly higher, because there are now more application tasks running in the system (remember that applications control the communication pathways needed to connect these far-flung users). The slave node utilization is much less because each one only handles node management for one user, plus conference-wide notifications. The practical limit for a conference size appears to be about 70 users, again quite satisfactory.

We can interpret these results to predict the effect of the administrative conference. It is a mechanism for tying potential users together and all active and would be users must belong to it. Only a few events (eg., joining and leaving a conference - say three per conference on average) are processed by it. If a user participates in 10 conferences per day (a bit like useing the telephone) this would generate about 0.06 events per minute per user, or about 17 times less than the nominal rate we took for a conference. On the same basis as before this indicates a nominal limit of about 1200 active users in the overall system. However a caveat is immediately evident: to use the system, a node must simultaneously belong to a conference and to the administrative conference. Therefore these two activities must divide the node's capacity: we are cut to about 35 users in a single conference and about 600 users altogether.

These predictions are made to indicate the analyses we would like to be able to do

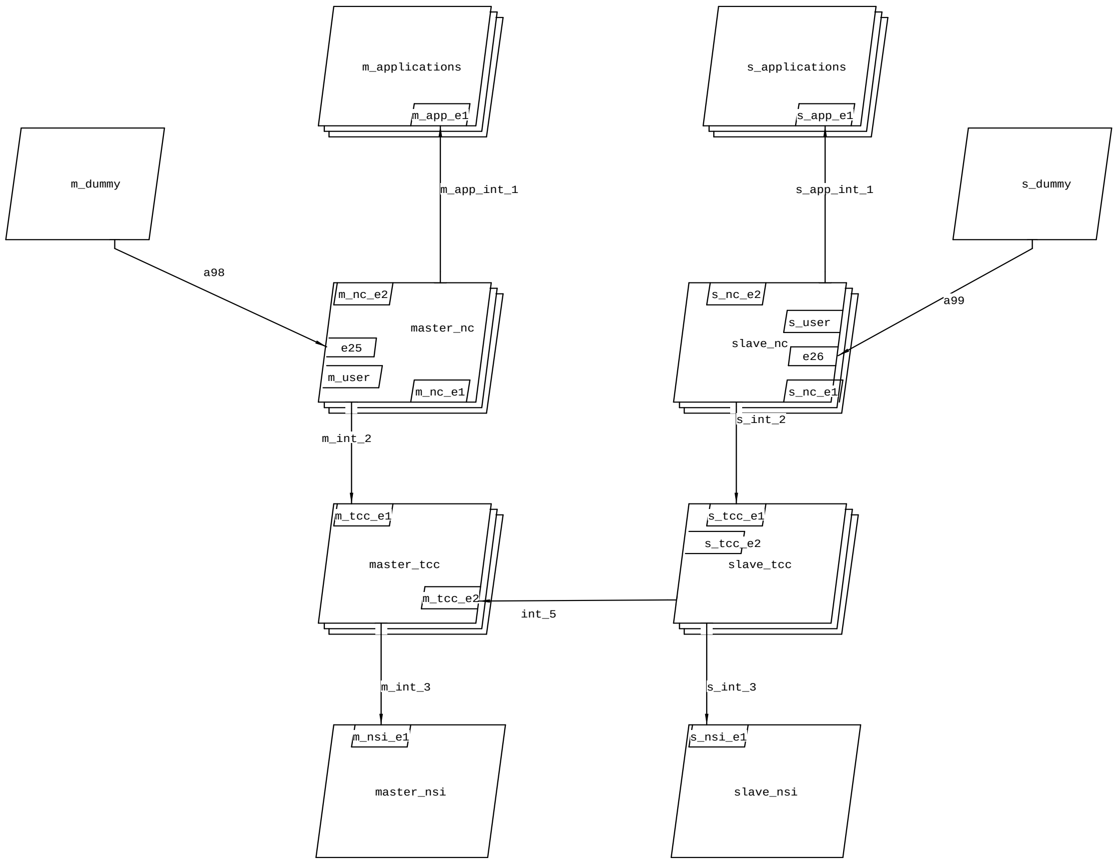

Fig. 2. Model2.ps

6. Architectural issues for performance

Out of a host of possible questions to address, we will consider the important question of scale-up. We have seen that the limitation to scale up of the existing design seems to be at the administrative conference controller cluster. Can we obtain hints about how to overcome this and how effective the solution will be? From this discussion we will go on to show that a performance characterization of the computational model (rather than just the engineering model, as we have done here) would be useful in providing truly performance -adaptable solutions.

Section 5 concluded that the master GCC limits the number of conference participants that can be managed to about 100. Two ideas will be considered for improving this:

• multiple masters, arranged so each one deals with conferences originating in a certain domain;

• separation of the state inquiry function (scenario number X) from the conference control functions.

The single master conference controller is a classic single point-of-control bottleneck in large systems. To break this bottleneck, we suppose that a certain number of users and slave controllers form a domain with their own master controller, which handles management of all conferences led by a user in the domain, and all resources in the domain. If the leadership moves, the conference control state will be moved too. The masters notify each other of changes so each one has knowledge of the global system state, to be used for answering inquiries, for instance.

The effect is that each domain master deals with control and allocation in its domain, plus negotiations for resources in other domains, and the volume of these operations are of the order of the domain size. It also deals with all notifications, and this volume is still of the order of the system size.

To remove this last burden we may introduce specialized inquiry servers to hold the global state. All notifications go from the domain master to one of these and they forward these to other inquiry servers. Now the only non-scalable object is the inqu8iry server which must receive notification and store state updates for every operation in the system, as well as deal with its own set of users for inquiries. This can be improved by partitioning the information, if necessary.

To reconstruct the Master Controller into domain masters requires some changes to their logic, while the introduction of inquiry servers depends on partitioning the controller object into two parts, one for contr9ol, and one for notifications and inquiries. Both changes require modifications in the computational model, or at the very least place conditions on it.

The analysis just preceding makes a point with general application:

- it is meaningful to assess an engineering model against a range of environmental and workload profiles,
- changes to overcome limitations may require rather deep changes to the model, (such as modifying the internal structure of capsules.)

References

[BUX92] W. Buxton, “Telepresence: Integrating Shared Task and Person Spaces”, Proceedings of Graphics Interface’92, Vancouver, B.C., May 1992.

[Tools95] R.G. Franks, A. Hubbard, S. Majumdar, J.E. Neilson, D.C. Petriu, J. Rolia and {C.M. Woodside, *A Toolset for Performance Engineering and Software Design of Client-Server Systems*, Dept. of Systems and Computer Engineering, Carleton University”, Report SCE-94-14, June 1994 (To appear in Performance Evaluation, 1995.)

[WNPM94] C.M.Woodside, J.E.Neilson, D.C.Petriu, S.Majumdar, *The Stochastic Rendezvous Network Model for Performance of Synchronous Multi-tasking Distributed Software*, Technical Report SCE-93-09, Department of Systems and Computer Engineering, June 94. (Accepted for publication in IEEE Transactions on Computers, to appear Jan. 1995).

[spauter] H. Garcia-Molina and A. Spauster,*Ordered and Reliable Multicast Communication*, TOCS, vol. 9, n. 3, pp. 242-271, August 1991.

[media] S. Bly, S. Harrison and S. Irwin, *Media Spaces: Bringing People Together in a Video, Audio, and Computing Environment*, CACM, n. 1, pp. 28-47, January 1993.

[GCC] ITU Study Group 8, *T.GCC: Generic Conference Control for Audiovisual and Audiographic Terminals and Multipoint Control Units*, May 23-5, 1994 (ITU Telecommunications Standardization Sector).

[mcs] ITU Study Group 8, T.122: Multipoint Communications Service for Audiographics and Audiovisual Conferencing Service Definition, October 1993, (ITU Telecommunications Standardization Sector).

[mcs2] ITU Study Group 8, T.125: Multipoint Communication Service Protocol Specification, October 1993, (ITU Telecommunications Standardization Sector).

[iiif] T. Milligan, The Telepresence Integrated Interactive Intermedia Facility (IIIF), Ontario Telepresence Project, 1993, OTP-93-04, Toronto, Canada

[nsi] G. Karam and B. McLeod and G.L. Boersma, A Network Software Infrastructure for Telepresence Applications, in *Second International Symposium on Autonomous Decentralized Systems (ISADS)*, 25-27 April 1995, Phenix, AZ (to appear.)

[otp] R. Reisenbach, The Ontario Telepresence Project, in *CHI '94 Conference Companion, ACM Conf. on Human Factors in Computing Systems*, pp. 173-174, April 1994, Boston, MA.

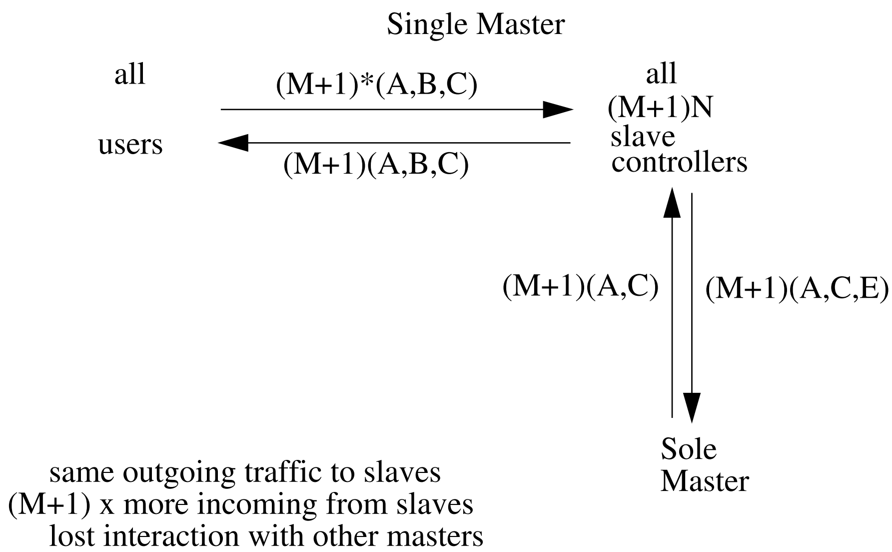

Fig. 3. Traffic Pattern in the Base Case (Sole Master) and M+1

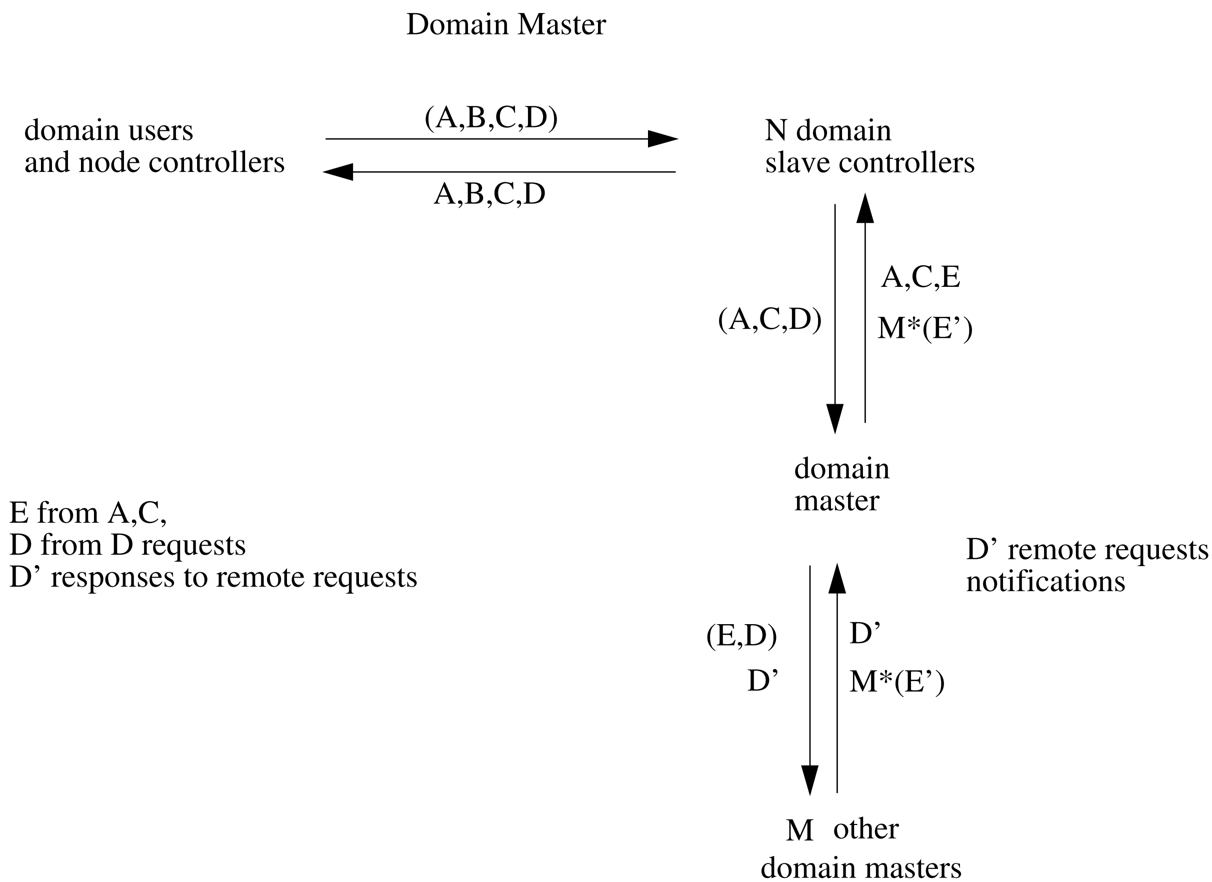

Fig. 4. Traffic Pattern with M+1 Domain Masters

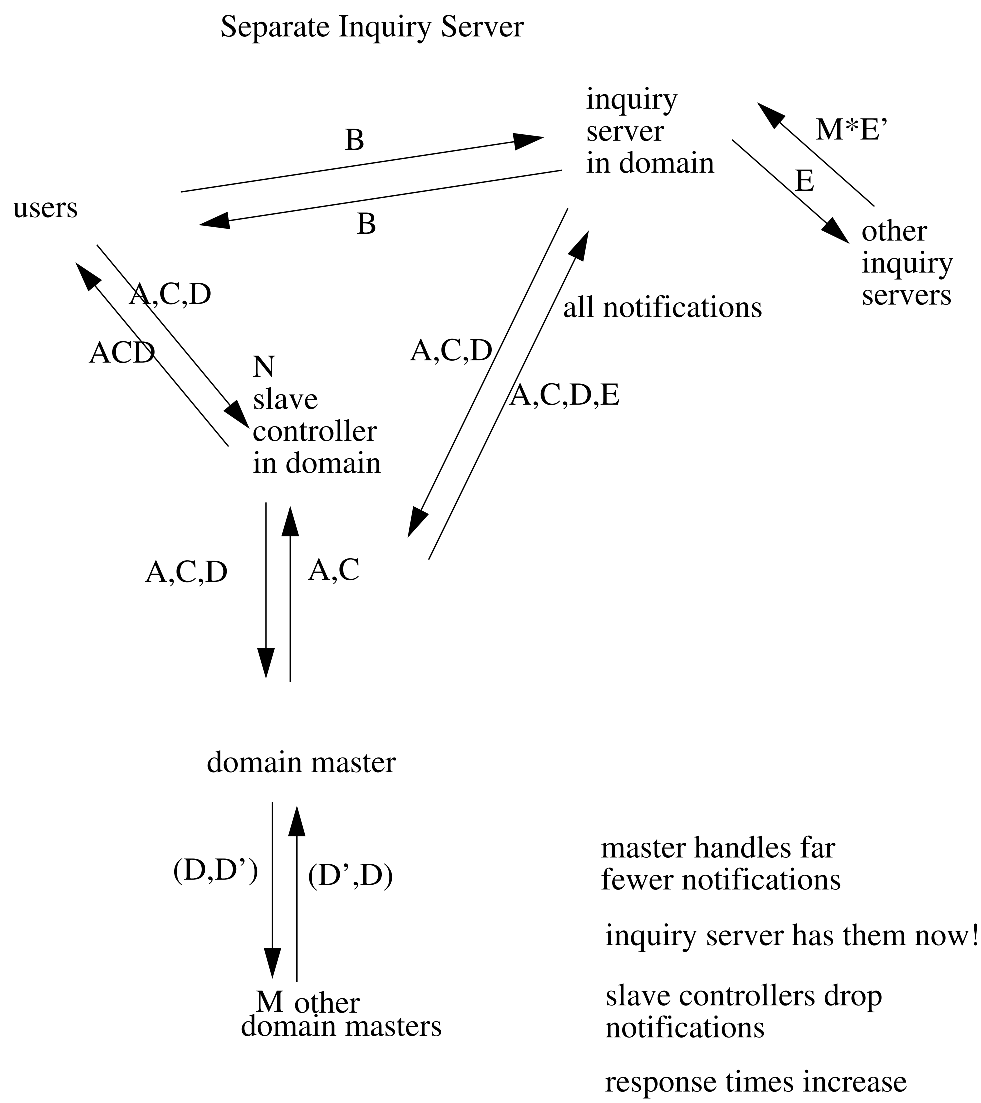

Fig. 5. Traffic Pattern with M+1 Domain Masters and with Inquiry