

A Development and Runtime Platform for Teleconferencing Applications

Christian Blum, Philippe Dubois,
Refik Molva, and Olivier Schaller

Institut Eurécom
2229, route des Crêtes
F-06904 Sophia-Antipolis
{blum, dubois, molva, schaller}@eurecom.fr

Abstract—A communication platform is described that supports the fast implementation of networked multimedia applications with conference character and collaboration features. The platform exhibits the notion of a site as one of its main abstractions. A site is a collection of workstations, media input and output devices that are, in terms of control, tightly coupled. Connection and application control is centralized within one site, but distributed among different sites. The platform exports a programming interface with high-level abstractions for session and connection control, allowing application developers to concentrate on scenario and user interface design. The platform was implemented in the course of the European Beteus (Broadband Exchange for Trans-European Usage) project. A tele-meeting and a tele-teaching application were developed on top of it. Platform components and applications were tested on the European ATM pilot network over a period of nine months. The paper first describes platform architecture and programming interface; it then talks about the implementation of platform and applications, and their deployment in the harsh environment of a trans-national broadband pilot network.

I. INTRODUCTION

Today's collaborative teleconferencing systems are usually implemented as stand-alone applications with fixed interaction and communication scenarios. They establish a static audio and video connection structure among the conference participants and employ a specific tool for collaboration. The software architecture of such systems is often highly inflexible; since it is designed with the requirements of a single application in mind it does not automatically support the reuse of its components within other application scenarios. This means that there is a new software design and implementation process each time a new application needs to be developed, with code reuse being at the library level or lower.

The success of a tele-conferencing application depends to a high degree on its ability to make distances shrink and to bring people together in their daily work. It is impossible to develop a successful tele-conferencing application without having user feedback at all stages of system integration. Rapid prototyping is a must, and the design of an application needs to take into account that important application scenario changes, e.g., changes in connection structures, may occur at any time during

the implementation process.

While the situation is maybe not as drastic as it is depicted above, it is clear that the monolithic system approach to teleconferencing application development will give way to a platform approach; networked multimedia applications in general will be more and more implemented on top of programming interfaces that provide different levels of control for media acquisition, transmission and playout, for media storage and retrieval, for connection control, session management, or simply for whatever building block is likely to be used by a larger number of applications. Some of these interfaces will be standardized, allowing applications developed for one hardware architecture to be easily ported to other ones [1]. Authoring tools and application development platforms will further ease design and implementation of networked multimedia applications [2][3]. An application development platform is especially necessary in the case where an application is to be offered as a service in a larger public or private network. Much of the complexity there stems from the necessity to integrate the application into the network infrastructure and to make it interwork or coexist with other services. The development platform emulates the runtime platform and is service creation environment and test-bed at the same time. Such platforms do exist for the creation of interactive retrieval services on residential cable networks [4]. As the customer access link becomes symmetric, there will be a demand for multi-point and multi-user services like tele-conferences, having life-cycles maybe just as short as those of retrieval services. Development platforms for multi-user services will have to deal with dynamic connection structures and with multiple user interfaces, just to name two sources of added complexity.

The platform approach may also be of advantage in areas other than service provision. The particular problem we were faced with was to develop a set of applications for the European Beteus (Broadband Exchange for Trans-European Usage) project [5]. The project definition of Beteus focussed on the network communication aspects rather than the applications. The only requirements on the applications were that one of them be a tele-teaching application, and that they make the best use of the high bandwidth available on the ATM network that interconnects the project partners in France (Eurécom in Sophia-Antipolis), Switzerland (Cern in Geneva, Epfl in Lausanne,

Ethz in Zürich) and Germany (Tub Berlin). Two applications were vaguely envisaged, a tele-meeting application for informal group meetings, and a distributed classroom application that would allow to give a lecture at one site to a virtual classroom that is the combination of classrooms at the Beteus sites, including the local one. Since there was no clear vision for the applications at the beginning of the project, it was decided to build an application platform rather than stand-alone applications for everyone of the envisaged application scenarios. The platform would constitute the highest common denominator between the envisaged application scenarios and would allow to implement and to incrementally improve an application scenario with significantly reduced effort as compared to an approach based on stand-alone prototypes.

The Beteus platform and the initial application scenarios were designed and developed in the period from August 1994 to April 1995 [6]. In May 1995, the Beteus field trials started on the European ATM pilot network. Two application scenarios were successfully demonstrated to a commission of the European Union in July [7]. The field trials continued until the beginning of December 1995, with the second major event being the organization of a distributed conference November 16 and 17 between a main site in Madeira and attached sites in Madrid, Brussels and Sophia-Antipolis.

The ease with which the two application scenarios could be implemented justifies the extra effort that went into the development of the platform. The scope of the platform is well beyond the project for which it was developed; it is a valuable long-term investment around which many of our research activities can be centered. The deployment of the platform on an experimental trans-national ATM network turned out to be an important, but not always easy experience.

The paper is roughly divided into an architecture part and an experience part. The first part starts off with a section about the design issues and constraints that underlie our main architectural decisions (Section I). It then gives an introduction into the Beteus application model and a complete description of platform architecture and components (Section II), followed by a summary of the main session and connection related abstractions (Section III). Based on this, the application programming interface is presented. An extended example demonstrates its main features as well as the application development methodology (Section V). The beginning of the second part of the paper is marked by a section discussing implementation details of platform and applications (Section VI). The Beteus network configuration is described, with the main emphasis on how multicast is provided to applications (Section VII). The experience gained with the deployment of platform and applications as part of the network trials is summarized, and a non-exhaustive description of problems encountered during the tests is given (Section VIII). A final conclusion presents some ideas for the directions into which the Beteus platform can be developed in the future.

II. OBJECTIVES AND DESIGN ISSUES

Beteus is a follow-up to the Betel (Broadband Exchange over

Trans-European Links) tele-teaching project in which two of the Beteus partners, Eurécom and Epfl, were engaged [8]. Betel implemented a tele-tutoring application scenario: a professor in Lausanne supervises the laboratory exercises of students in a classroom at Eurécom in Sophia-Antipolis. Both professors and students have personal workstations that are equipped with microphone, speaker, camera and an external video screen. There is additional equipment on classroom level: a large screen in front of the student workplaces, a camera that overlooks the classroom, a speaker and a microphone. A typical session starts with the professor giving an introduction into the laboratory exercise. He is visible and audible at classroom level and overlooks the whole class. The students then start to work with some software, in this case an intelligent network simulator, and the professor will wait for questions to come up, or go from student to student. A student who has a question will push the question button in the user interface and wait for the professor to connect to him. The professor in turn will see the question on his interface and push the answer button, which makes him visible and audible at the student's personal workplace. The student may then share the simulator application with the professor, and the professor can guide the student remotely to the solution of his problem.

In Betel, Epfl and Eurécom were connected with a 34 Mbit/s ATM link via a cross-connect in Lyon. ATM was not directly visible to applications; a Betel end-station was connected via FDDI to a router, which in turn connected via HSSI to the network adapter that encapsulated IP packets in SMDS packets before transmitting them via AAL 3/4. At Eurécom, audio and video of the professor were distributed with an analog switch. People that attended the demonstrations were very much impressed by the dynamic connection structure changes in the classroom, i.e., by the ease with which the professor's virtual presence moved from classroom level to particular students and back. The analog switch was hiding the fact that there was at no point more than one digital audio and video stream in one direction between Lausanne and Sophia-Antipolis.

Beteus should build on the experience gained with Betel. It should improve Betel in various respects and provide

- true digital multi-point communication
- group collaboration on documents/software
- a minimum of two application scenarios

In addition to that, Beteus was to be employed for at least one event involving real users, as opposed to the theatrical demonstration of Betel. One proposition for this was to organize a distributed summer school for business students where the Beteus platform would be used for formal lectures, panels, group presentations, business games and informal meetings. It was clear that for such an event to be successful the platform would need to have a degree of maturity that allows non-skilled users to be let alone with it.

A. Application Scenarios

A key aspect of Beteus is that the project partners were to be interconnected via a broadband network. The Beteus applica-

tions are supposed to demonstrate the high quality of human communication and interaction that can be achieved when bandwidth is not a limiting factor. The people that are brought together by a Beteus application shall communicate and interact as freely as if they were sitting together around a table in a conference room [9]. This is only possible if the quality of the audiovisual communication and of the collaboration tools is such that geographically dispersed users are perceived as being present at every implicated location. In addition to that, a user must have the impression that he is seen and heard; he must trust the system to really convey his image and speech to other users.

High video quality requires:

- high frame rate
- high resolution
- low playout jitter
- low end-to-end delay
- low losses

Accordingly, high audio quality is achieved with:

- large pass-band
- low end-to-end delay
- low noise
- low echo
- no losses

For optimum communication, audio and video should be synchronized on playout, i.e., the end-to-end delay of an audio and a video stream originating at the same location should be identical at any playout location.

For the Beteus application scenarios the notion of a *virtual community* was retained. A virtual community can be described as a group of people that interact for some common purpose over a longer period of time by means of one or more networked applications. The term *community* implies that the members of the communicating group either already have a certain relationship with each other, or that such a relationship is developed through the multiple use of the communication platform. An example for a virtual community is the aforementioned group of business students that use the platform for preparation and presentation of assignments, or the Beteus people themselves that use the platform to discuss technical issues or to organize events.

It was foreseen that some of the application scenarios would be used by more than one virtual community, whereas others would need to be tailored to a specific virtual community. With a possible summer school event in mind, two application scenarios were envisioned at the beginning, a tele-meeting scenario and a distributed classroom scenario. The tele-meeting scenario is supposed to be a rather informal meeting environment where people could come together to talk and possibly collaborate on some document. The distributed classroom scenario should combine classrooms at different sites to a single virtual classroom. A professor could give a lecture in one classroom in which remote classrooms would participate. Every classroom would be equipped with multiple screens that show all other classrooms and possibly the slides of the professor, and people

within different classrooms could communicate with each other and the professor in a way similar to a panel discussion. The two scenarios differ fundamentally from each other in that the first one assumes a single user terminal as standard endpoint equipment, whereas the second uses a collection of workstations and media input and output devices to assemble a classroom. None of the scenarios was clearly specified at the beginning of the project. It was assumed that the application scenarios would evolve in the course of the project as tests are performed and experience is gained. Since the requirements of the target virtual community were only vaguely known it was not even clear if both of the scenarios would be retained, or if other ones were to be added.

These considerations led to the decision to implement a real conferencing platform rather than stand-alone systems for everyone of the envisaged application scenarios. The conferencing platform should offer a high-level programming interface with which the effort to implement an application scenario could be kept at a minimum, with this minimum being not much more than the effort to implement the graphical user interface.

It was also decided to build a light-weight directory service into the platform that would provide a run-time framework for the application scenarios. Users would formally log into the platform; once logged they could consult the directory service to see who is there, and then start or join applications or just remain passive.

At a later stage of the project it turned out that the official event of the project would be a distributed conference, and not the summer school. The decision was taken to postpone the implementation of the distributed classroom scenario and to implement instead of that a third scenario, the tele-tutoring scenario, which is a replication of the Betel application on the Beteus platform. The practical consideration behind this decision was that the tele-tutoring scenario could be demonstrated on the testbed without any changes to the hardware configuration.

B. Collaboration

One of the objectives of Beteus was to integrate a collaboration tool into the platform. The choice to be made here is between an application like a shared white-board that has integrated multi-user support, or a shared window system that replicates the user interface of a single-user application. The latter was chosen because it allows to use a wide range of applications for collaboration, and then also because one of the project partners had already developed such a system [10].

C. Network

At the beginning of the project it was not clear how exactly the project partners would be interconnected with each other. It was assumed that the majority of the project partners would have access to the European ATM pilot network, but at least in the case of Eurécom it looked a long time as if the access would be SMDS as in Betel. It was a clear objective to have ATM access for all project partners since such an access was sup-

posed to be more favorable for multipoint communication.

D. Basic Software Architecture

An important design issue was the control architecture of the platform. In terms of control, the platform could be completely centralized or completely distributed. The centralized solution was declined, first because there would be a single point of failure, then because of performance and scalability considerations. But it was felt that control should be centralized within the network of a project partner. This is because it was assumed that an application endpoint will not be a single multimedia workstation, but rather a logical unit that is assembled from a collection of resources including workstations, multiple screens, cameras, speakers and microphones, digital and analog switches. Some of this equipment, especially workstations and analog switches, would be shared by many logical application endpoints, making it necessary to have some central connection management entity. The scope of this central connection management entity is limited to the local network, with the exact composition of an application endpoint being hidden to the outside. The establishment of a connection between the networks of two project partners must therefore involve some communication between the respective connection management entities. The resulting control architecture is thus semi-distributed: control is centralized on the level of a local network, but distributed on the level of the interconnection network.

A combination of equipment is found in the distributed-classroom scenario, but it is also interesting for the realization of personal workplaces like in the tele-meeting scenario where a single workstation processing multiple audio and video streams may easily run into performance problems.

III. PLATFORM ARCHITECTURE

This section describes the platform architecture. It introduces the main abstractions and an application model and discusses the main building blocks of the platform from a functional point of view. The description of implementation details is left to a later section.

A. Sites and Nodes

For the total amount of tightly coupled equipment within the network of a project partner the abstraction of a *site* is introduced. The abstraction of a *node* is introduced as the application dependent mapping of equipment onto a logical application endpoint. Connection and session control within a site is performed by a central entity that knows about the application specific node mapping from a configuration file. Fig. 1 shows a possible node mapping for the personal workplace. The node shown uses different workstations for audio and video processing and for the actual application process. The user interface of the application is displayed on a terminal rather than a workstation screen. Video is displayed separately from the user interface on a second screen. The media input and output devices in Fig. 1 have the logical names *PersMicrophone*, *PersSpeaker*, *PersCamera* and *PersScreen*. Such names are used by the appli-

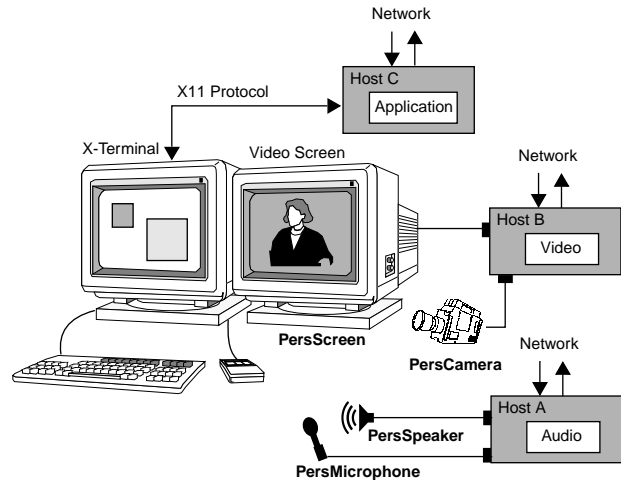


Fig. 1. Node mapping example

cation to denominate connection endpoints. The site configuration file contains for every node a list of endpoint entries, with each entry containing a logical name and its mapping onto a physical address. This configuration information is used by the connection management for the establishment of audio and video connections. Logical device names are in general application specific; they describe the context in which a device like a camera or a microphone is used within a specific application, and they can be as exotic or unique as the application itself. The logical device names shown in Fig. 1 are likely to be employed by more than one application, simply because the node configuration itself is quite common. To illustrate the concept of logical device names, we could add to this node a camera that captures the view from a laboratory window, and call it *WindowCamera*. During the Beteus project, window views were sometimes transmitted when the laboratory personal took a break.

B. Application Model

The Beteus application model introduces the abstractions of a *session*, a *session vertex* and a *session application*. A session is the abstraction for one instance of a distributed application that runs on top of the platform. A session comprises, from a logical point of view, a set of nodes as session members. From a computational point of view, a session consists of a set of session endpoints, called session vertices, which are processes that run on the session nodes. The ensemble of session vertices within a session constitutes the session application. In the following, we will use the term session application interchangeably with application or application scenario. If we want to refer to a process running at a node within the framework of a session application, we will explicitly refer to it as session vertex.

Participants are humans or groups of humans that register their name and node with the platform. Once registered they can participate in sessions. For every session in which they participate there will be a session vertex running at their node. Note that it is the session vertex rather than the person that is the actual session participant; the human participant appears as an

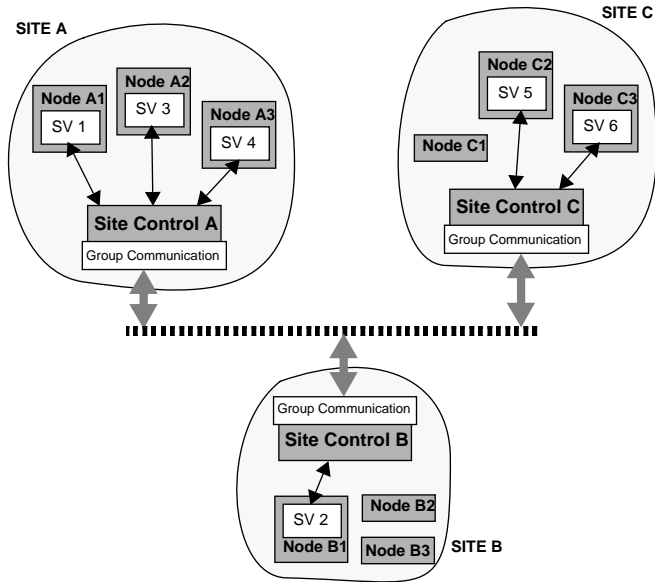


Fig. 2. Application model.

attribute or name tag of the session vertex.

Fig. 2 shows three sites with each of them having three nodes defined in its site configuration file. An application is indicated that spans all three sites, with three nodes being involved at site A, one at site B, and two at site C. In fact, there is no limitation on the location of the nodes that form a session; they can be all within a single site, or all within different sites. It is therefore also completely hidden to the session vertex on a node if the session in which it participates spans remote sites or if it is local. Session vertices always interact with their local site control, but the processing of a session vertex request may trigger inter-site communication, which is the case whenever connections need to be established in-between sites. The group communication module indicated in Fig. 2 provides the messaging services required for inter-site communication.

C. Site Architecture

The three principle layers of the site architecture are depicted in Fig. 3. The top layer is an application layer containing a generic control panel and application processes - the session vertices. In the middle there is the site control layer which comprises the site manager, the connection manager and the station agents. The site manager implements the functionality offered at the high level interface towards the applications, whereas the connection manager performs physical connection establishment in collaboration with the station agents. The communication layer finally contains the audio, video and application-sharing software as well as the group communication entity that supports the exchange of control messages between site managers and between connection managers.

The shaded architecture components in Fig. 3, i.e., the site manager, the connection manager and the group communication entity, have only one instantiation within a site. Station agents

on the contrary are daemons that are found on every machine on the site network that may be source or sink of audio or video connections or that may run application-sharing software.

D. Site Management

The site manager offers the platform services to the session vertices that run on top of it. Platform services are

- session management
- connection management
- endpoint control
- application sharing
- messaging service
- directory service

Some of the services offered by the platform will only be used by the control panel, and others only by the session vertices, although there is theoretically no such limitation.

Participants register with the platform via the control panel. The site management keeps a list of all registered participants and of all ongoing sessions. Participants can create new sessions or join ongoing sessions. When a participant creates or joins a session the control panel forks the session vertex that corresponds to the session application. In case the session is created, the forked session vertex will automatically become the *session master*. The session master has certain rights with respect to the session that other session vertices do not have. This includes for instance the right to delete nodes from a session, or to kill the session. The session master is also the coordination center for the distributed application; it is the session master who configures the session at the beginning and who initiates connection structure changes later on. Other session vertices communicate with the session master via the messaging services of the platform. Most of what the session master does will be in direct response to a message received from another session vertex, or to input from the local user interface. The session master role can be transferred to another session vertex, which is especially

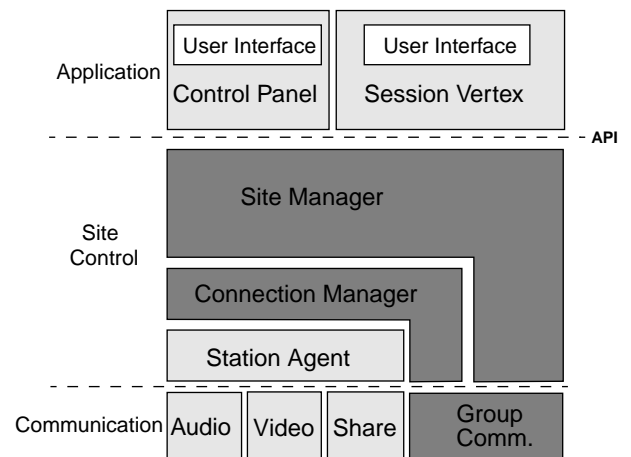


Fig. 3. Site Architecture.

necessary when the participant that is behind the session master wants to leave the session. Note that the session master functionality is not necessarily visible at the user interface of the respective session vertex - this is an application design choice.

E. Connection Management

The connection manager receives connection control and endpoint control requests from the site manager. Connection control comprises connect and disconnect requests and is only performed by the connection control of the session master; endpoint control stands for the setting of device parameters like audio volume and video saturation. The connection manager maps in a first step the logical endpoint names in the site manager requests to physical addresses. Endpoint control requests are then forwarded to the concerned audio or video processes. Connect requests result in immediate connection establishment if all of the connection endpoints are local. If an endpoint is remote, the connection manager of the session master asks the remote connection manager to establish the respective endpoint. Sink endpoints are established before source endpoints. In the case of unicast connections, the request for the creation of a sink endpoint returns an Internet Protocol (IP) sink address that is given to a source endpoint as target address. In the case of multicast connections, it is the connection manager that assigns an IP multicast address and port number to source and sink endpoints.

The site manager requests only point-to-point connections from the connection manager, but every connect request is accompanied by a hint as to whether the respective connection is part of a multipoint connection structure, in which case the connection manager may use IP multicast if available.

F. Inter-Site Communication

Both the site manager and the connection manager communicate with remote peer entities, as is indicated in Fig. 3.

Site managers need to communicate as part of the directory service, the messaging service and the session management service. The communication between site managers is asynchronous and consists of the reliable transfer of a message from one site manager to one or more other site managers. As part of the directory service, a site continuously broadcasts to all other sites a list of logged participants and a list of sessions of which it is the master site. This broadcast is also used for liveliness detection; if a site manager times out on the directory service broadcast of a remote site it will delete this site from all sessions of which it is the master, and it will locally terminate all sessions of which the missing site is session master. The messaging service of the site manager allows session vertices to send messages to one or more other session vertices. This requires the reliable delivery of a message from one site manager to a limited set of remote site managers. The session management service deals with session membership. A site indicates a new session vertex with a join message to the site management of the session master. The session master returns a message containing the session membership list to the joining

site and an update message to the other sites in the session. The procedure for leaving session vertices is similar. More complicated is the transfer of the master role from one site to another. The site manager of the actual session master first sends a message containing the session state to the new master and enters a state where it waits for the new master to announce himself to the session members as the new session master. Once this has happened it forwards messages for the session master that it received in the meantime to the new master. Some additional care has to be taken to avoid race conditions and deadlocks.

Connection managers need to communicate to establish inter-site connections. As for now, connection endpoints are established sequentially with remote procedure calls: the connection manager sends an establishment request and receives an acknowledgment once the remote connection manager has established the endpoint.

The communication requirements raised by the site manager and the connection manager are optimally addressed by the Reliable Multicast Protocol (RMP) [11]. RMP supports the reliable delivery of messages to all members of a group with different levels of service ranging from unreliable delivery to totally resilient delivery. It runs efficiently on IP multicast, but allows group members that are not multicast capable. RMP was integrated into the Beteus platform at a later stage, which is why its advanced features, like the totally ordered delivery of packets, were not exploited. The Beteus sites form one group, and messages sent to more than one site are actually delivered to every site. Messages sent to a single site are delivered with RMP's remote procedure call mechanism. This mechanism is also used for connection manager requests.

G. Evaluation

The platform, as it is implemented now, supports conference-style communication among a small number of sites with some sort of static relationship to each other. Examples for such groups of sites are

- universities with a common tele-teaching program
- laboratories working on a common project
- administrative units of an international enterprise.

The platform is not designed for ad-hoc communication on a network with a large number of sites. Such a deployment would require a redesign of the directory service and the group communication component.

IV. MAJOR CONNECTION ABSTRACTIONS

This section presents the abstractions that are used at the application programming interface to describe connections and connection structures. Connection types are audio, video and application-sharing.

A. Roles

An application scenario is implemented within a single executable. The session vertices of an application are therefore identical in terms of code, but they behave according to dynam-

ically taken or assigned *roles*. The already introduced master role and a general *participant* role are the only roles which exist by default - all other roles are defined by the application itself. An application may define as many roles as it wishes to, and session vertices may also hold multiple roles at the same time. A session vertex will adapt the user interface that it produces towards the human user to the role or roles that it takes. Roles fall into two categories: *static* roles and *transient* roles. A static role determines the main behavior of the session vertex and is usually not transferred to another session vertex. Examples for such roles would be the professor role and the student role in the Betel tele-tutoring scenario. Transient roles are created, assigned and deleted as needed; they model whatever ephemeral position a session vertex may have with respect to other members of the session. An example for this would be the role of a momentary speaker in a panel discussion. The application programming interface itself does not differentiate between static and transient roles. This is more a concept that the application designer needs to have in mind when analyzing an application scenario.

Applications bind connection endpoints to roles and let the site infrastructure do the mapping of the given role to a session vertex. All roles, including the master role, can be reassigned to other session vertices. This allows an application to specify the audio, video and application sharing connection structures once on session start-up; later on it will transfer roles in-between session vertices when it wants to change the connection structure. A typical example for this would be the speaker role mentioned above at the root of an audio and a video multicast connection. The infrastructure will automatically rebuild this multicast connection whenever the speaker role is passed from one session vertex to another.

B. Bridges

The introduction of the role abstraction already provides considerable comfort for application development in that it allows to group connection endpoints. In addition to this the platform provides abstractions for connection structures. A *bridge* is a single-medium connection structure among session vertices. A bridge can be a point-to-point connection, a point-to-multipoint connection, a multipoint-to-multipoint connection, or a multipoint-to-point connection. The endpoints of a bridge are given as role names; the cardinality of a role decides about the nature of the bridge. Up to now it was not necessary to introduce another endpoint addressing scheme than the role-based one. The role-based addressing scheme might become awkward when an application scenario employs an excessive number of point-to-point connections, but no such scenario has been identified until now.

The concept of a medium bridge hides the underlying network from the application. The connection management realizes bridges with whatever transport the network offers. It knows the connection types and is thus able to handle media specific endpoint issues. In a multipoint-to-multipoint audio bridge it will automatically establish an audio mixer at every sink node, whereas it will launch separate receiver processes for

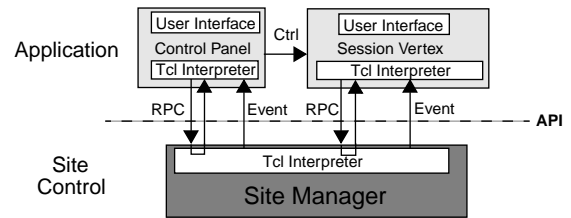


Fig. 4. Procedure calls and event notification at the platform interface.

every stream in the case of an equivalent video bridge.

The bridge abstraction can also be applied to X11 application sharing. The majority of shared window systems intercept the traffic between an X11 client application and server, which allows them to replicate the user interface of the application at various displays by duplicating the client's drawing requests towards the connected servers and by combining events evolving from these servers into one event stream towards the client [10]. A bridge models the group of endpoints on which the user interface of an application is replicated, with the client application as source endpoint and the remote displays as sink endpoints. The connection structure it creates is a combination of point-to-multipoint (drawing requests) and multipoint-to-point (events).

C. Bridge Sets

A number of bridges, typically an audio and a related video bridge, can be assembled to form a *bridge set*. An application configures the platform on session start-up with a description of the bridge sets that it uses. During the session, only one bridge set can be active at a time. If an application changes the active bridge set, the infrastructure will tear down any connection of the old bridge set that is not included in the new one, and establish the connections that are missing.

The number of bridge sets an application defines corresponds to the number of fundamental application states which in turn corresponds to different temporal phases a session traverses during its lifetime. The programming interface does not directly support the notion of application state, but application state is, like static and transient roles, a concept that the application designer has to be aware of.

V. APPLICATION PROGRAMMING INTERFACE

The application programming interface (API) is based on synchronous remote procedure calls (RPC) and asynchronous event notifications as is indicated in Fig. 4. The RPC package chosen for the communication between session vertices and the site manager is Tcl-DP [12], the distributed programming package for Tcl/Tk [13]. Since simple applications will mainly deal with user interface issues, it is possible to implement them completely in Tcl. More complex applications will have C or C++ code in addition to the Tcl/Tk user interface script; they will use the C library of Tcl-DP to call site manager procedures or to register callback functions for event notification.

The API procedure calls [14] are grouped into the following categories:

- **Registration:** user registration and deregistration
- **Endpoint handling:** audio and video device control
- **Session directory:** directory service related calls
- **Session startup:** session initialization and startup
- **Session information:** convenience calls
- **Session control:** session membership and lifetime control
- **Bridge set handling:** changing the active bridge set
- **Messaging:** communication among session vertices
- **Role handling:** role assignment and removal
- **Application sharing:** X11 application sharing

The convenience calls allow session vertices to query the session configuration. Session vertices do not maintain records about actual role assignment, actual bridge set or session participants; they retrieve this information from the site manager as they need it.

The main event notifications are

- **Receive:** a message from another session vertex
- **Join:** there is a new session vertex in the session
- **Left:** a session vertex left the session
- **Kill:** the session got killed or disrupted
- **RoleAdd:** a role is assigned to the session vertex
- **RoleDel:** a role is removed from the session vertex

API procedure call usage and event occurrence are illustrated in Fig. 5. Two session vertices A and B are shown; A creates a session that is joined by B. This session is killed by A when B leaves again. The lifetime of a session stretches from the point of time when it is announced to the point of time when it is killed. The three principal states of the session are *announced*, *initializing* and *ongoing*. An announced session is a session that is scheduled for a certain date and time in the future. Announced sessions are visible via the directory service and

TABLE 1
THE API BRIDGE DEFINITION CALL

DefineBridge pid sid type ginfo gtime srcepname rslst sinkepname rrlst		
pid	Integer	participant identifier
sid	Integer	session identifier
type	Enum{1,2,3}	1=audio,2=video,3=sharedXapp
ginfo	Integer	information granularity [0..100]
gtime	Integer	time granularity [0..100]
srcepname	String	source endpoint name
rslst	Integer	list of source role identifiers
sinkepname	String	sink endpoint name
rrlst	IntegerList	list of sink role identifiers
returns: bid	Integer	bridge identifier

help people discover each other's activities. The announcement phase can be skipped by calling `SessionInit` right after `SessionAnnounce`. The `SessionInit` call marks the beginning of the initializing phase where the creator of the session configures the site manager for the actual session application. Initialization comprises role, bridge and bridge set definition. Roles have to be defined before bridges since role identifiers are necessary to specify bridge endpoints. For the same reason, bridges are defined before bridge sets. With the `SessionStart` call the session enters the state ongoing where it can be joined by other session vertices. This call contains as parameter the initial bridge set identifier. The session creator becomes the first session member and gets automatically the session master role assigned. If he takes additional roles he will assign them to himself with `AddRole` calls. The session master then has to wait for others to join the session. As is indicated in Fig. 5, B finds out about A's session via a `SessionOngoingQuery` call. B joins the session with a call to `SessionJoin`, which is indicated to A with a `Join` event notification. Connections other than those defined for the general participant role are not established before the session master A assigns a first role to B. The connection structure that is then established between A and B depends on their respective roles and the active bridge set. The example in Fig. 5

continues with a message transfer from B to A that prompts A to change the active bridge set. When B leaves the session, the site management tears down all connections between A and B. The session is formally finished with A's call to `SessionKill`.

Table 1 shows as an example for an API procedure call the parameter fields of `DefineBridge`. The first two parameter fields identify participant and session. The type field marks the bridge as audio, video or shared application bridge. Information granularity is interpreted as window size in the case of video and as sample encoding in the case of audio. Similarly, time granularity is interpreted as frame rate in the case of video and sample rate in the case of audio. Source and sink endpoint names define the logical devices that terminate the connections of the bridge. The call allows further to define a list of role identifiers for sources and one for sinks. The connection type is determined by the cardinality of source and sink roles within the session:

- **no connection:** no session vertex holds any of the source roles, or no session vertex holds any of the sink roles.
- **point-to-point connection:** one session vertex holds one of the source roles, and one session vertex holds one of the sink roles.
- **point-to-multipoint connection:** one session vertex holds one of the source roles, and multiple session vertices hold one of the sink roles.
- **multipoint-to-multipoint connection:** both source and sink roles are held by multiple session vertices.
- **multipoint-to-point connection:** only one session vertex holds one of the sink roles, and multiple session vertices hold one of the source roles.

The `DefineBridge` call returns an identifier that can consequently be used to include the bridge in one or more bridge sets.

A. Example Scenario

An example shall serve to illustrate how application scenarios are translated into role, bridge and bridge set definitions. Imagine a distributed school with professors and students all geographically dispersed. Professors have application scenarios for all kinds of teaching purposes at hand, among them a scenario that supports translation work on stage-plays written in a foreign language. The scenario has four states or phases. In a first phase, the professor gives an introduction into the translation assignment that was previously distributed by E-mail. Students see and hear the professor, and they hear each other, which allows them to hear questions asked to the professor by fellow students. In a second phase, the students start to work on the translation of the stage-play. The professor goes from student to student and answers their questions. The editor of the current student is automatically shared with the professor. The professor may return to phase one if one of the questions is of general interest. Once students have finished the translation, phase three begins where students present their results. The professor and the presently presenting student are visible to all other students and to each other. The editor of the student is automatically shared with all others, and audio is like in phase one. In phase four, students take roles in the stage-play and recite them. Their

TABLE 2
EXAMPLE BRIDGE DEFINITIONS

No.	Medium	Source Roles	Sink Roles
1	audio	participant	participant
2	audio	professor,studentSpeaker	professor,studentSpeaker
3	audio	studentSpeaker	participant
4	video	professor	student
5	video	student	professor
6	video	professor,studentSpeaker	professor,studentSpeaker
7	video	studentSpeaker	participant
8	sharedX	studentSpeaker	professor
9	sharedX	studentSpeaker	participant

image and voice is distributed to the professor and to the other students. The professor finishes the course with some remarks, with the application being again in phase one. During the whole session the professor has as the replacement of a classroom-view an icon-sized video image with low frame rate from every student.

The roles that can be identified in this scenario are:

- **professor:** static professor role
- **student:** static student role
- **studentSpeaker:** visible students in phase two, three, four
- **master:** held by the professor
- **participant:** professor and students

The transient role `studentSpeaker` is assigned to the visited student in phase two, to the presenting student of phase three, and to the acting students in phase four.

The bridges that need to be defined are shown in Table 2. The first audio bridge is the all-to-all audio of phase one and three. Audio bridge 2 and video bridge 6 form a bidirectional audiovisual connection for phase two. Audio bridge 3 and video bridge 7 form the virtual stage of phase 4. The multipoint-to-point bridge 5 represents the icon-sized classroom view.

Four bridge sets are defined according to the four phases of the application scenario:

- **bridge set one:** audio bridge 1, video bridges 4+5
- **bridge set two:** audio bridge 2, video bridge 5+6, sharedX bridge 8
- **bridge set three:** audio bridge 1, video bridges 5+7, sharedX bridge 9
- **bridge set four:** audio bridge 3, video bridge 5+7

Connection control during the session consists of changing between bridge sets and assigning the transient role `studentSpeaker`.

VI. IMPLEMENTATION

This section describes the implementation of the Beteus platform and of the applications that have been developed so far. Platform and applications run on Sun workstations under SunOs 4.1.3 and are developed in C++ and Tcl/Tk.

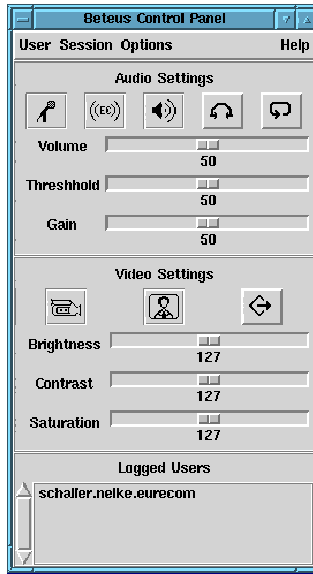


Fig. 6. Control panel user interface.

A. Audio and Video Transmission

Both audio and video are built on top of the User Datagram Protocol (UDP) and the Internet Protocol (IP). One of the biggest issues in Beteus was how to implement multipoint communication with audio and video. It turned out that the structure of the Beteus network is a hostile environment for IP multicast, which would be the natural choice for multipoint communication. This is why the audio and video components implement in addition to IP multicast simple sender based stream duplication. The connection managers that control the establishment of connections over the network may employ whichever scheme is possible.

B. Audio

The multipoint nature of the audio communication makes it necessary to use some sort of combination of multiple incoming audio streams at the receiving side. This combination could be done either by a stream selector or a mixer. Using a stream selector requires silence detection at the sending side and some support for talkspurt transmission at sender and receiver. A stream selector chooses an active audio stream from the set of incoming streams for output whenever the talkspurt of the currently chosen stream is finished. A mixer could manage continuous audio on all incoming streams as well as talkspurts, but using talkspurt audio avoids the situation where there is a sum of the background noise of multiple remote sites in the mixer's output signal. The audio component implements silence detection at the sending side with an adjustable threshold value and is built on top of the Real-time Transport Protocol (RTP) [15], which in turn uses UDP for transmission. The receiving side supports both stream selection and audio mixing. Both sender and receiver generate activity events that can be graphically displayed on the user interface. The sender indicates begin and end of talkspurt to the local user, whereas the receiver indicates activity for each of the incoming streams on which it listens.

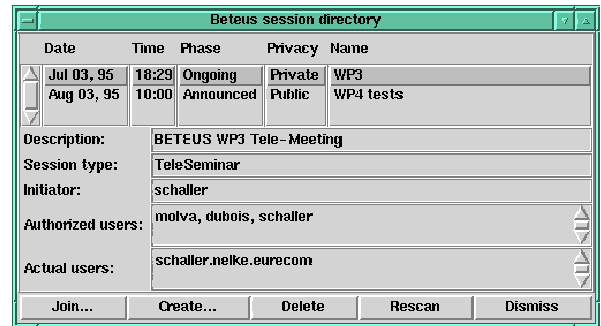


Fig. 7. Directory service window of the control panel user interface.

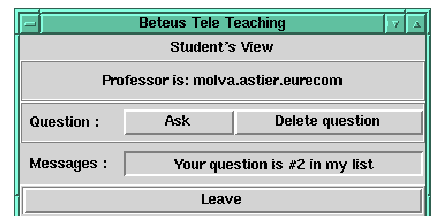
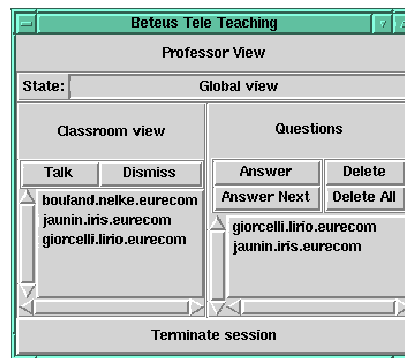


Fig. 8. The user interfaces for the tele-tutoring application (professor/student).

The two audio encodings that are supported are 8kHz sampling rate with 8bit resolution and 16kHz sampling rate with 16bit resolution.

C. Video

Video transmission is built around the XVideo board from Parallax. The compression of the Parallax board follows the JPEG standard for the compression of still images [16]. On connection setup, the video sender allows to specify a target data rate that is consequently enforced by means of a control loop in which maximum and measured data rate are constantly compared, with the JPEG compression factor being modified according to the result of this comparison. Such a mechanism is clearly necessary in cases where there are data rate restrictions per video stream and traffic policing within the network. The video receiver adapts automatically to the actual compression factor as it also adapts to frame rate and window size. Care was taken to have a constant frame rate within the receiver window because the human eye is extremely sensitive to frame rate irregularities. The benefit of this is that the subjective quality of the video component is excellent even at frame rates as low as five frames/s; people tend to overestimate the frame rate when

asked for a guess.

D. Application Sharing

The application sharing component of the platform is Xwedge from project partner ETH Zürich. Xwedge is a distributed shared window system that has agents running at all implicated client and server sites. X11 clients connect to the local Xwedge agent which in turn communicates via the Transmission Control Protocol (TCP) with remote agents and the local X11 server. The Beteus API offers three calls for application sharing control: a session vertex can get a list of sharable applications, which are the clients that are momentarily connected to the Xwedge agent, and it can share and unshare an application. Sharing means that the interface of the chosen application is replicated at the sink endpoints of the currently active X11 bridge. As for now, the platform does not implement the rich set of floor control mechanisms that Xwedge offers. The platform uses the default floor control mode where the floor follows mouse clicks and keyboard input.

E. Site Control

The persistent components of the platform are the site manager, the connection manager, the group communication and the station agents. A station agent launches audio and video processes and Xwedge agents and relays operation requests from the connection manager to these processes and operation results and asynchronous events back to the connection manager. The connection manager is forked by the site manager and establishes TCP control connections to all station agents when coming up. The site manager exists in two versions: the normal runtime version and a development version. The development version of the site manager forks a dummy connection manager that does not do anything else than reading the site configuration file and returning positive responses to site management connect and disconnect requests. This allows to test session vertices in faked sessions on a single display and without establishing audio and video connections.

F. Applications

The applications that have been developed so far are the generic control panel, a tele-meeting application and a tele-tutoring application.

Fig. 6 shows the main window of the control panel with the audio and video device control fields and the list of logged users at the bottom. From this window the directory service window shown in Fig. 7 can be invoked. The directory service window lists ongoing and announced sessions and allows, among other things, to join or create sessions.

The tele-meeting scenario is a simple framework for work meetings that can be used by many virtual communities. The audio and video connection structure is all-to-all, i.e., everybody sees and hears everybody else. There are simple user interfaces for a chairman and a normal participant, with the chairman being able to assign the role of a presenter to one of the session participants. The presenting person can share one of

its X11 applications with the other participants. The chairman interface allows to transfer the chairman role to another participant, in which case this participant gets his interface exchanged for a chairman interface. Within the tele-meeting scenario, connection management gets active only when a presenter shares an application, or when participants join or leave the session, in which case their connection endpoints are added or removed automatically from the audio and video bridges.

The tele-tutoring application is a remake of Betel on the Beteus platform with the difference, that students are geographically dispersed. Fig. 8 shows the user interfaces of professor and student. The application can be in the states *global* and *talk*. In the state *global*, the professor has a video window for every student, and can himself be seen and heard by all students. In the *talk* state, the professor talks to a single student, but audio and video of both professor and student are distributed to all other students so that everybody can follow their discussion. The student can also share an X11 application to show his work. The tele-tutoring application is still a simple application, but it already has much more connection structure dynamics than the tele-meeting scenario.

G. BeCool Application Compiler

It was realized that the development of more advanced applications than the ones implemented until now, i.e., applications using many different dynamically assigned roles and a larger number of bridge sets, may pose some problems for unexperienced programmers. It was therefore decided to build a development tool that compiles an application description language to a C++ session vertex skeleton that then has to be filled out by the programmer. The resulting BeCool (Beteus Cooperative Language) compiler [17][18] was used for the development of the tele-meeting and tele-tutoring applications, although both applications could have been easily implemented in Tcl/Tk alone.

VII. THE BETEUS NETWORK

The implementation phase which started in November 1994 went hand in hand with the planning of the network for the field trials. The Beteus partners in Germany and Switzerland had very quickly negotiated an access to their national ATM pilots, whereas Eurécom had a hard time convincing France Telecom that a direct ATM access was absolutely necessary for its participation in the field trials. The solution initially proposed by France Telecom was the one of Betel, i.e., FDDI access to an SMDS service with a translation to ATM in Paris. Apart from the negative impact that such a solution would have had on transit delay and transit delay jitter, it would have made an ATM-based solution to the multicast problem impossible right from the beginning. When it became clear that everybody else in the project was on the European ATM pilot, France Telecom gave in and offered Eurécom direct access to ATM three months before it actually wanted to provide such a service.

Fig. 9 depicts the Beteus network. All project partners have local ATM LANs on the basis of Fore Systems ASX-200

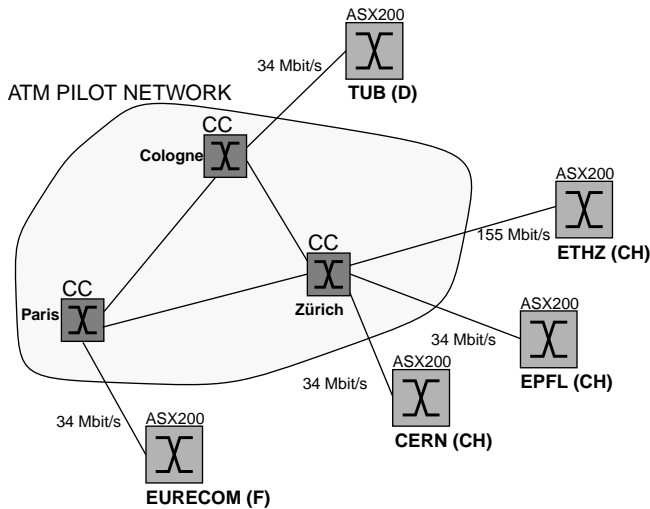


Fig. 9. The Beteus network (CC==cross-connect).

switches [19]. The switches of all project partners but Ethz connect to the ATM pilot via 34 Mbit/s E3 interfaces. Ethz is the only project partner that has a 155 Mbit/s STM 1 link to the ATM pilot. The ATM pilot itself is a collection of ATM cross-connects in various European countries. Beteus runs over cross-connects in Paris, Cologne and Zürich as far as can be judged from the scarce information provided by the network operators. The only service provided so far by the ATM pilot is the interconnection of bidirectional permanent virtual paths (PVP) or semi-permanent virtual paths (SPVP). The PVP service offers connectivity over a longer period of time, whereas SPVP requires occasional or periodic reservation. Beteus was most of the time using the SPVP service and had connectivity on Tuesday mornings from 8:00 to 12:00. In addition to that, the Swiss project partners were once in a while running tests on other week-days.

The issue that came up after the access problem was solved was the topology of the future Beteus overlay network. Two extreme proposals were full interconnection and serial alignment. It was imaginable to arrange the project partner LAN's in a chain starting at Eurécom and going over Cern, Epfl and Ethz to the Tub in Berlin. While such a configuration would have had a minimal number of PVP's on the ATM pilot and therefore a cost advantage as compared to other solutions, it would have forced every project partner to participate in every test and it would have made a possible point of failure out of every intermediate ATM switch on the line, which is why it was declined. The topology that was finally adopted was the fully meshed network where a given project partner has PVP connections to every other one.

Another issue was the dimensioning of the PVP connections. The calculation of the maximum bandwidth for a PVP is based on the following assumptions about the traffic originating from a Beteus site:

- one low quality video stream (12fps): < 1 Mbit/s

- one high quality video stream (25fps): < 2 Mbit/s
- one audio stream: 64 kbit/s or 256 kbit/s
- control and application sharing: negligible

It was decided to reserve a maximum bandwidth of 3 Mbit/s for every PVP.

The most important problem to be solved was multicast. It was clear that multicasting had to be done within the local networks of the project partners given that the ATM pilot does not support any form of multicasting. At the beginning there was the hope that Fore's proprietary SPANS signalling could be tunnelled through the ATM pilot, and that then either Fore's native ATM multicast or its implementation of IP multicast could be used. This does not work because the actual ForeThought 3.0 switch software does not support more than one signalling channel at a given network interface port. Since a Beteus ATM switch accesses the ATM pilot via a single E3 port it is not possible to establish signalling tunnels with more than one remote site, which is, there is no way to establish a fully meshed network of signalling channels between the Beteus switches.

ATM-based multicast itself turned out to be hard if not impossible to achieve. The ASX-200 switch does not allow an incoming cell stream to be duplicated into two streams that leave through the same output port, as it is required in Beteus. This is a natural restriction if one considers that output ports of switches are usually connected to input ports of other switches or to host interfaces. In both cases, there is no need for multicast in intermediate switches - multicast is only performed by switches situated at the endpoints of multicast branches. A solution proposed by Fore was to let the switch multicast an incoming stream to as many switch output ports as there are final destinations, and loop the streams back into the switch and then finally out via the E3 interface. This proposal was not further pursued because such a solution would have monopolized the Beteus switches of which some were used in parallel for other projects.

The only solution that remained after ATM-based multicast had fallen away was a multicast on application level. One possibility for this is to have one central multicast daemon running on one site, with each of the other sites having a point-to-point connection to the daemon machine. Such a configuration is similar to the multipoint control units found in N-ISDN conferencing systems. Another possibility is to distribute the multicast functionality onto all sites and have one multicast daemon per site. None of the two solutions was adopted simply because it was thought to be awkward to have delay sensitive media streams pass four times through the UNIX user space. The solution that was finally implemented was source-based stream duplication: audio and video senders transmit streams to more than one destination. While this multicast scheme misses any elegance, it seems to be the best of all possible compromises. It might appear to be a drawback of source-based stream duplication that it sends identical streams over the same network link, but note that this would have been equally the case in an ATM-based solution. A real drawback of source-based stream duplication is that it puts an extra load on the source machine, but this does not really matter in the case of Beteus where media

streams are sent to a very limited number of destinations. The platform architecture takes such performance issues into account and allows to have machines dedicated to audio and video transmission. Stream duplication avoids additional processes and thus additional points of failure, and it is in terms of end-to-end delay almost as optimal as ATM-based multicast

The site configuration during the field trials was to have two Sun Sparc 10 workstations connected to the ATM switch that together formed one node. Every machine had a permanent virtual channel connection to every other machine on the network.

VIII. TESTS AND EVENTS

The Beteus network was in place beginning of May 1995. The application platform and its components were tested until July and presented to a commission of the European Union on July 25. The tests restarted after the summer break at the beginning of September and had as main objective the preparation of the First International Distributed Conference IDC '95 which was held November 16 and 17 in Madeira. The tests then continued up to the beginning of December 1995.

A never ending source of problems was the big amount of configuration that was necessary to establish the Beteus overlay network. Every project partner had to manually configure four PVPs, sixteen permanent virtual channels and sixteen IP/ATM address mappings for a trial. At many sites, Beteus interfered with other projects that used the same equipment and that had their own configuration. It often happened that machines were not available for the tests, or were replaced at short notice by others, for instance because of hardware failures. The general configuration file that was used by the project partners was therefore often not up-to-date, or if it was it could happen that switch or interfaces were badly configured. Another error-prone procedure was the reservation of the ATM pilot connections that had to be done for every unregular trial. As an example, the cross-connect in Paris cut our transmission of IDC'95 right at the moment when the major event, a distributed panel discussion, should begin. The reason was that France Telecom had made a mistake in setting the expiration time of our connections. During the project we gained some experience in detecting and solving such connectivity problems, and made extensive use of network management tools to this purpose. There were also frequently problems with regular trials where reserved connections were not automatically established by the operators due to some cross-connect software problem.

The performance of the network was excellent. The round trip time measured between Eurécom in France and the Swiss sites was around 30ms. The Swiss sites among themselves measured round trip times of as low as 4ms [20]. A network management platform tailored to Beteus was monitoring performance at levels from the ATM layer up to the video and audio application processes. An example measurement for the bit loss rate at ATM level is 3.03×10^{-9} [21], which indicates a very reliable network. The quality of the network was directly visible at the application level. Audio and video had low losses and low latency. The video losses that were observed resulted

mostly from video window movements at sending sites; X11 window managers block video transmission while windows are moved on the screen.

The complete platform ran more often on the local test-bed at Eurécom than on the ATM pilot. This was mostly due to the aforementioned connectivity problems that took quite some time at the beginning of every trial. The result of this was that the main attention of the project shifted away from the applications to basic audiovisual communication. Although the platform was up and running, there was no time to really evaluate and improve the tele-meeting and tele-tutoring application scenarios, as it was foreseen at the beginning. The two scenarios were demonstrated to the European Union as they were originally conceived, i.e., without having traversed a cycle of gradual improvements as it is supported by the platform API. It is therefore also the audiovisual transmission that received most of the feedback during the network trials, and not the platform itself.

The major event of Beteus was the organization of the distributed panel discussion of IDC'95. The conference itself was held at Madeira, but all presentations were transmitted to two sites in Madrid, to Eurécom and to a host of the European Union in Brussels. The remote sites could actively participate in the discussions that followed the presentations. The distributed panel discussion would have been an excellent opportunity to create a tailored Beteus application, but it was finally decided to use only the audio, video and application sharing components of the platform for the event. The reason for this was a reduced complexity of the setup, considering also that the remote sites in Brussels did not have any experience in configuring the Beteus platform. The multicast solution during IDC'95 was a central multicast server in Madrid.

IX. CONCLUSION

The main emphasis of the design of the Beteus platform is on the application. The platform exports an API that considerably reduces the effort it takes to implement a tele-conferencing application. The platform is also a runtime environment for such applications; it allows applications to run in parallel and offers a directory service that informs logged users about what is happening on the platform.

The main emphasis of the field trials should also have been on the applications, but this goal was not achieved. The experience of Beteus is that the broadband network needs to be much more transparent than it is now in order to support advanced multi-point applications. The configuration effort necessary to allow for connectivity on IP level in a network as small as the one of Beteus was perceived as a major obstacle. What is needed for platforms like the one of Beteus to run successfully is signalling and network multicast support.

After Beteus, the platform will be deployed on France Telecom's ATM WAN in Sophia-Antipolis that is becoming operational at the end of 1995. Work on it will continue in various directions. In a general move to shorten communication paths and to increase distribution within a site, we started to reimplement our platform on top of CORBA [22]. Station agents are replaced by an object request broker that establishes direct com-

munication between the connection manager and the audio, video and application sharing processes. A future version of the API will be defined in IDL and allow direct connection endpoint control. An application stub may then provide Tcl access to API calls and events.

ACKNOWLEDGEMENT

The authors would like to thank all members of the Beteus consortium for their participation in the field trials and in the design of the network platform, namely Thomas Walter and Marcus Brunner from Ethz, Simon Znaty and Bruno Dufresne from Epfl, Christian Isnard from Cern, Jürgen Kawalek from Tub, Pierre de la Motte from Ifatec and Didier Loisel from Eurécom.

REFERENCES

- [1] Interactive Multimedia Association, "Multimedia System Services", *IMA Recommended Practice Draft*, available via <ftp://ima.org/pub/mss>, 1995.
- [2] IBM Lakes Team, "IBM Lakes: An Architecture for Collaborative Networking", R. Morgan Publishing, Chislehurst, 1994.
- [3] S. Wray, T. Glauert and A. Hopper, "The Medusa Applications Environment", *IEEE Multimedia*, vol.1, no. 4, Winter 1994.
- [4] Personal conversation at Telecom'95 with stand personal from Oracle Corp., Geneva, October 1995.
- [5] Beteus Report, "Functional Specification", Deliverable D2, July 1994.
- [6] Beteus Report, "Detailed Specification", Deliverable D6, November 1994.
- [7] Beteus Report, "Working Prototype of the Application Platform Specification", Deliverable D8, June 1995.
- [8] Y.-H. Puzstaszeri, E. Biersack, Ph. Dubois, J.-P. Gaspoz, M. Goud, P. Gros, J.-P. Hubaux, "Multimedia Teletutoring over a Trans-European ATM Network", *2nd IWACA Conference*, Heidelberg, September 1994.
- [9] S. A. Bly, S. R. Harrison and S. Irwin, "Media Spaces: Bringing People Together in a Video, Audio and Computing Environment", *Communications of the ACM*, January 1993.
- [10] Th. Gutekunst, D. Bauer, G. Caronni, Hasan and B. Plattner, "A Distributed and Policy-Free General-Purpose Shared Window System", *IEEE/ACM Transactions on Networking*, February 1995.
- [11] T. Montgomery, "Design, Implementation and Verification of the Reliable Multicast Protocol", Master's thesis at West Virginia University, <http://research.ivv.nasa.gov/projects/RMP/Docs/RMPdocs.html>, December 1994.
- [12] L. A. Rowe, B. Smith, and S. Yenftp, "Tcl Distributed Programming (Tcl-DP)", University of Berkely Computer Science Division, <ftp://mm-ftp.cs.berkeley.edu/pub/multimedia/Tcl-DP/tcl-dp-v1.0ak>, March 1993.
- [13] J. K. Ousterhout, "TCL and TK Toolkit", Addison-Wesley Publishing, 1994.
- [14] Christian Blum and Olivier Schaller, "The BETEUS API", Eurécom Technical Report, December 1995.
- [15] Internet Engineering Task Force, "RTP: A Transport Protocol for Real-Time Applications", Internet-Draft, March 1995.
- [16] G. K. Wallace, "The JPEG Still Picture Compression Standard", *Communications of the ACM*, April 1991.
- [17] J. Lindblad and O. Schaller, "BeCool and the BETEUS Application Programming Interface", Eurécom Technical Report, June 1995.
- [18] J. Lindblad, "The BeCool Thesis Project Report", Master's thesis at the KTH Sweden, 1995.
- [19] E. Biagionie, E. Copper, and R. Sansom, "Designing a Practical ATM LAN", *IEEE Network*, March 1993.
- [20] T. Walter, M. Brunner and D. Loisel, "The BETEUS Communication Platform", *Proceedings of the First International Distributed Conference IDC '95*, Madeira, November 1995.
- [21] M. Besson, K. Traore and Ph. Dubois, "Control and Performance Monitoring of a Multimedia Platform over the ATM Pilot", *Proceedings of the First International Distributed Conference IDC '95*, Madeira, November 1995.
- [22] Object Management Group, "The Common Object Request Broker: Architecture and Specification", John Wiley & Sons, Inc., 1992.