Technical Report: Eurecom-RR-96-020 December 1995

The Beteus Application Programming Interface

Christian Blum and Olivier Schaller

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

This technical report describes application development on top of the Beteus application platform. It contains a complete reference of the Beteus programming interface for the development of tele-conferencing applications.

1 Introduction to Beteus

The application programming interface (API) described in this technical report was developed as part of the collaborative teleconferencing platform of the European Beteus (Broadband Exchange for Trans-European Usage) project [1][2][3][4]. The Beteus platform and its applications were deployed in field trials over the European ATM pilot network which interconnects the Beteus partners in France (Eurécom, Sophia Antipolis), Switzerland (CERN in Geneva, EPFL in Lausanne, ETHZ in Zürich), Germany (TUB Berlin) and Sweden (KTH Stockholm). Beteus is a 16-month follow-up to the BETEL tele-teaching project [5] in which two of the Beteus partners, Eurécom and EPFL, were involved. The Beteus platform profited from the experience gained with the BETEL tele-teaching application which was one of the first collaborative applications to be run over cross-national ATM links in Europe.

Today's collaborative teleconferencing systems are usually implemented as stand-alone applications with fixed interaction and communication scenarios. They implement all the components of a teleconferencing system, from low-level media transmission and processing up to the user interfaces, from scratch. One drawback of this approach is that the interaction and communication scenarios may be hard to modify once implementation is reasonably advanced, or their modification may even turn out to be impossible. Another drawback is that software may be hard to reuse for other applications if its internal interfaces were not defined with this requirement in mind.

Ease of modification and software reuse were two prominent requirements imposed on the design of the teleconferencing system for Beteus. This is because the main focus of Beteus is not so much on the actual application than on a concept called the *virtual community.* A virtual community can be described as a group of people that interact with each other in a natural way by means of a networked application. One prerequisite for natural interaction is high quality multipoint audiovisual communication which in turn is only possible on a broadband network. Two types of interaction within the virtual community were vaguely envisioned for Beteus at the beginning of the design phase, one being a simple collaborative meeting, the other being tele-teaching. The final interaction scenarios were thought to evolve out of a series of prototypes that could be tested on the Beteus network. This situation led to the decision to construct a communication platform on top of which the various scenarios could be implemented with significantly reduced effort as compared to stand-alone prototype systems for every scenario. In March 1995, eight months after the initial design steps, work was advanced to a point that the platform and its components could be deployed for the field trials that started at that time. In July 1995, two application scenarios, tele-meeting and tele-tutoring, were successfully demonstrated to a commission of the European Union. The ease with which these application scenarios could be implemented justifies the extra-effort that went into the development of the platform.

The Beteus platform was designed to take as much functionality out of the applications as possible. Most importantly, it completely relieves the application from connection management. This simplifies application development to a point where there is not much more to be done than the user interfaces. The platform interface is based on Tcl/Tk [6], a scripting language that offers both support for user interfaces and support for network programming. The Tcl network programming package Tcl-DP [7] provides for the communication between application processes and the platform. Simple applications can be completely implemented in Tcl, which is ideal for rapid prototyping. More advanced applications will need some C++ code in addition and may profit from the application compiler that is implemented on top of the Beteus platform. This compiler takes as input an application description in the Beteus Cooperative Language (BeCool), from which it generates C++ skeletons for the application processes [8][9].

This technical report discusses the interface between the Beteus platform and the applications that run on top of it. In the remainder of this report we will refer to this interface as the Beteus API. The next section describes the platform and it presents the main concepts and abstractions of the API. This is followed by a more technical section about the architecture of the API and the remote procedure call package on which it is based. The remaining three sections give a complete and detailed reference of the API.

2 Main Concepts and Abstractions

This section discusses the main abstractions of the API and summarizes the platform services that are based on these abstractions.

2.1. Abstractions

The main abstractions of the platform are

- site
- node
- connection endpoint
- session
- session application
- session vertex
- participant
- role
- bridge
- bridge set

These abstractions are described in the following.

Sites and Nodes

A design issue in Beteus was the 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 equipment including workstations, multiple screens, cameras, speakers and microphones, as well as digital and analog switches. 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. Control within applications that span multiple sites is distributed among the central entities of the respective sites. The resulting platform is thus semi-distributed: it is centralized within a site, but distributed among sites.

Nodes and Connection Endpoints

The mapping of equipment onto nodes is specified in a site configuration file. A node appears in the site configuration file with a name and a list of connection endpoints. The names of these endpoints may be application specific. This allows applications to address endpoints, and with this devices, that have a special position within the list of endpoints that form the node. As a simple example, a distributed classroom application may use at one site two microphones, one for the

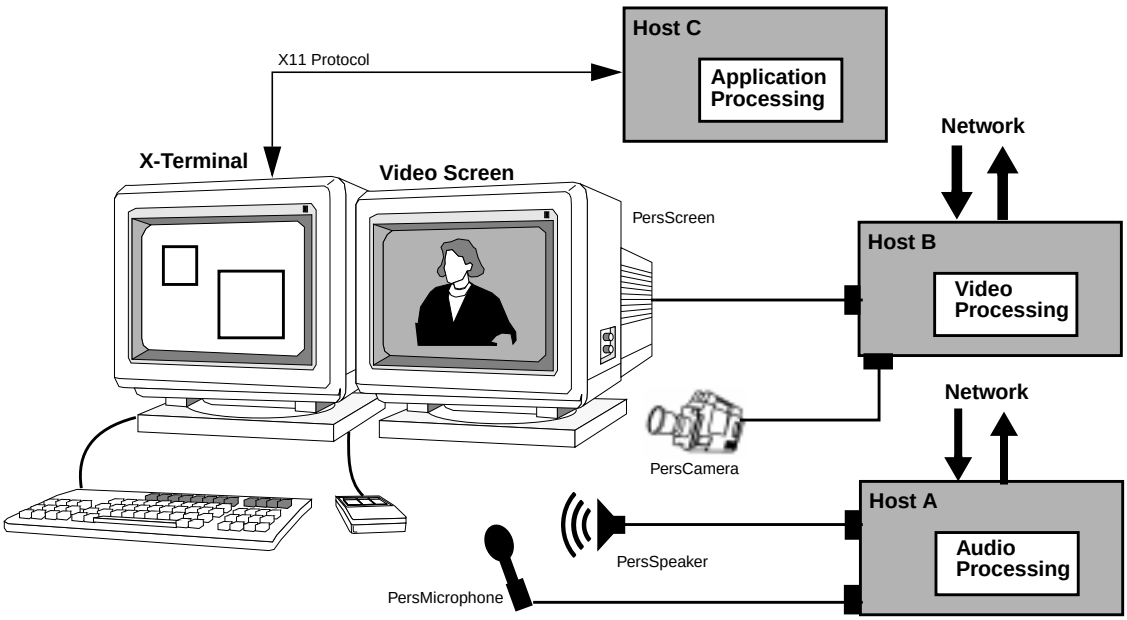

FIGURE 1. Node Mapping

professor and one in the audience for questions. The application will distinguish these microphones with different, meaningful endpoint names.

A prominent node configuration is the personal workplace, i.e., a node with personal camera, microphone, speaker, video screen and work screen. This configuration is likely to be used by more than one application.

Node and endpoint names is the only information in the site configuration file in which the application is interested. There is additional address information in the configuration file that allows the platform to establish connections to endpoints.

Figure 1 depicts a node that is configured as personal workplace. Audio, video and application processing is provided by three different machines. Host A controls the audio endpoints PersSpeaker and PersMicrophone. Host B controls the video endpoints PersScreen and PersCamera. The application software runs on Host C and generates a user interface on an X-terminal.

Applications and Sessions

At the heart of the Beteus platform is an 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 Beteus 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

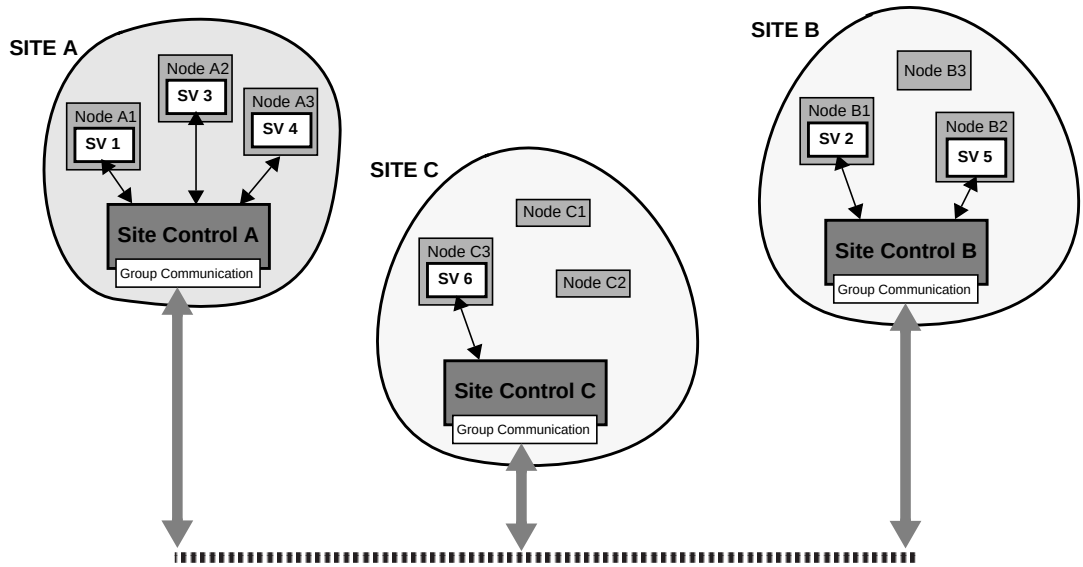

FIGURE 2. BETEUS application model (SV=Session Vertex)

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.

Figure 2 shows three sites with each of them having three nodes defined in its site configuration file. A Beteus application is indicated that spans all three sites, with three nodes being implicated at site A, two at site B, and one 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 Figure 2 provides the session broadcast, multicast and unicast messaging services required for inter-site communication.

Participants

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, from a computational point of view, it is the session vertex rather than the person that is the actual session participant; the human partici-

pant is an attribute or name tag of the session vertex.

Roles

The session vertices of an application are identical in terms of code, but behave according to dynamically taken or assigned *roles*. There is one prominent role within a session, which is the *session master*. The session vertex that is 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 or to kill the session. An application may decide for itself to which extent it offers this functionality on a user interface. It may also offer this functionality on other interfaces than on the one of the session vertex that holds the master role. The master role is the only role which exists per default - all other roles are defined by the application itself. Applications may bind certain 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 applications to specify the audio and video connection structure once on session start-up; later on it will only transfer roles in-between session vertices when it wants to change the connection structure. An evident example for this would be a speaker role that is 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. An application may define as many roles as it wishes to, and session vertices may also hold multiple roles at the same time.

Bridges and Bridge Sets

The introduction of the role abstraction provides already considerable comfort for application development. In addition to this the platform provides abstractions for connection structures. A *bridge* is a single-medium connection structure among session nodes. A bridge can either represent a point-to-point communication, a multicast communication, a broadcast communication, or an all-to-all communication. The concept of a medium bridge hides the underlying network to the application; a connection management entity realizes bridges with whatever transport the network offers. A set of bridges, typically an audio and a related video bridge, can be assembled to form a *bridge set*. An application configures the platform on start-up with a description of the bridge sets that it uses. During a session only one bridge set can be active at a time. If the application wishes to change the connection structure it will switch to another one of its bridge sets. The infrastructure will then tear down any connection that is not included in the new bridge set, and establish the ones that are missing.

The concept of bridges and bridge sets, along with the concept of roles being at the endpoints of bridges, allows to build applications that are almost stateless.

4.2. Platform Services

The services provided by the platform are

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

These services are summarized in the following.

Connection Control

As outlined above, the application specifies its connection structure in terms of roles, bridges and bridge sets. The platform establishes the endpoints of bridges, i.e., audio or video sender and receivers, and the necessary transport links that interconnect these endpoints. Endpoint parameters like audio volume or video brightness are controlled via the connection management.

Session Management

Nodes may create sessions, join them, leave them, and delete them. The session management interfaces internally to the connection control - connection structures are updated whenever nodes enter or leave the session.

Application Sharing

Collaboration within Beteus applications is provided by the possibility to share workspaces within the framework of the X11 windowing system. The interface of an X11 application running at one node can be replicated at other nodes without that the application would need to be prepared for this. This allows to visualize X11 applications at different nodes and further to share their control among the session members. Visualization is already an important aspect since it allows to communicate information, like it is contained in electronic documents or in the interface of a simulator, in a very convenient way. If the control of an X11 application is to be shared, there is a need for floor control. The platform offers the floor control features of the shared workspace system that it uses.

The application sharing component of Beteus is Xwedge from ETHZ [10].

Messaging Service

Session vertices use the messaging service of the platform to communicate among each other.

Directory Service

Nodes register with the platform when they are activated. One field in the information sent to the platform is the participant name under which the node wants to register. Once registered the presence of a node is visible to all other nodes within the network via the directory service. In addition

to registration information, the directory service informs about announced and ongoing sessions. This information is used by nodes to create or to join sessions.

The directory service enhances the platform from a development to a complete communication environment that could be used by a group of people scattered over different sites for their daily work.

5 Tcl-DP and the Interface Architecture

Figure 3 depicts the interface architecture. Session vertices, control panel and site manager are built around Tcl interpreters. Session vertices and control panel do remote procedure calls in the site manager, whereas the site manager sends asynchronous events back to these processes. The control panel launches session vertices.

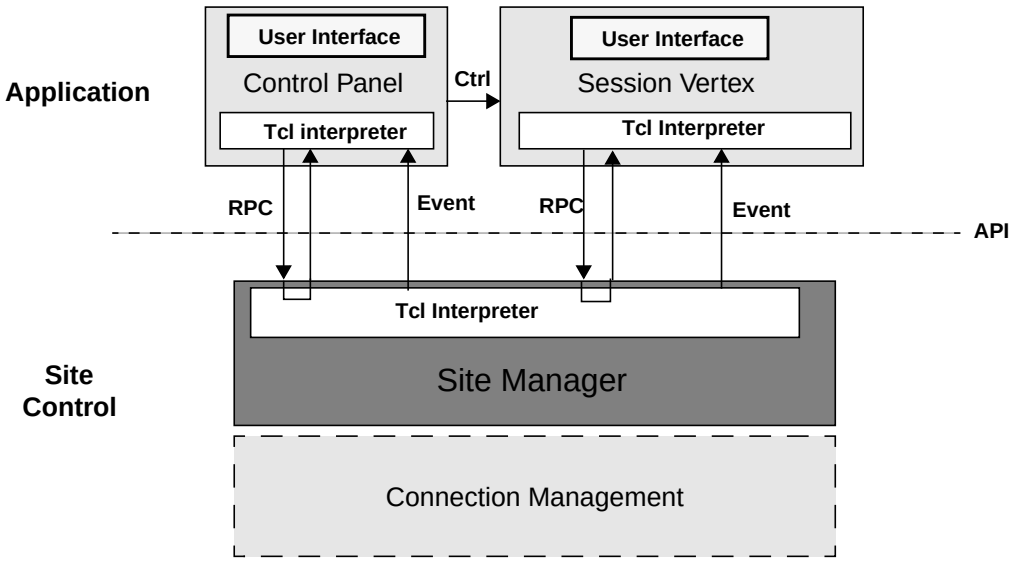

FIGURE 3. BETEUS Application Interface Architecture

A session vertex needs to register callback procedures for the events it wants to receive.

6 API Procedure Calls and Event Notifications

This section contains an overview of the API procedure calls and event notifications. Tables are given for procedure calls, event notifications, parameter types and important variables.

6.1. API Procedure Call and Event Notification Overview

API procedure calls are classified into the following categories:

Registration	user registration and deregistration
Endpoint Handling	audio and video endpoint device control
Session Directory	directory service related calls
Session Startup	session initialization and startup
Session Information	convenience functions
Session Control	session membership and lifetime control
Bridge Set Handling	application state/bridge set handling
Communication	communication among session vertices
Role Handling	role assignment and transfer
Application Sharing	X11 application sharing

The following section describes theses categories in more detail. Table 1 shows a summary of API function calls. There is an entry in the restriction column if usage of the call is restricted. Registration and deregistration are not really restricted to the control panel, but it can be assumed that only the control panel will call these procedures. A good portion of the procedure calls are restricted to the session master.

Table 2 show the event notifications send to the control panel. A control panel does not need to register these callbacks - the event names used correspond to control panel procedures that must be implemented by every control panel. Table 3 shows the event notifications sent to session vertices. Most event notifications to session vertices are the result of session master action. A session vertex that wants to receive them must explicitly register a callback function with the site management. The event identifier that is used in the registration call is indicated in the second column of Table 3.

Category	Call	Restriction	Description
Registration	Register	control panel	registers a user with the platform

Table 1: API Procedure Call Overview

Category	Call	Restriction	Description
	Deregister	control panel	deregisters a user from the platform
Endpoint Handling	GetEndpoints	-	gets the audio/video endpoints of the node
	SetDeviceParameter	-	sets a device (endpoint) parameter
	GetDeviceParameter	-	gets all parameter settings of a device
	Disable	-	disable an endpoint
	Enable	-	enable an endpoint that was disabled
	Loop	-	connect a local source to a local sink endpoint
	Unloop	-	disconnect a local sink from local sources
Session Directory	SessionAnnounce	-	announce a session
	SessionAnnouncementCancel	-	cancel a session announcement
	SessionAnnouncementGet	-	get the identifiers of all sessions
	SessionOngoingGet	-	get the identifiers of all ongoing sessions
	SessionAnnounceQuery	-	get the description of a certain session
	SessionOngoingQuery	-	get the description of a certain ongoing session
Session Startup	SessionInit	master	start session initialization
	SessionInitRole	master	initialize a session role
	DefineBridge	master	define a bridge
	DefineBridgeSet	master	define a bridge set
	RegisterCallback	session vertex	register an event notification callback
	SessionStart	master	start the session
Session Information	GetParticipantID	-	get the participant identifier of a participant
	GetParticipantName	-	get the participant name of a participant
	GetSessionVertexId	-	get the session vertex identifier of a session vertex
	GetSessionVertexName	-	get the session vertex name of a session vertex
	SessionGetMaster	-	get the session vertex of the session master
Session Control	SessionJoin	-	join a session
	SessionLeave	non masters	leave a session
	SessionKill	master	kill a session
Bridge Set Handling	ChangeBridgeSet	master	change the active bridge set

Table 1: API Procedure Call Overview

Category	Call	Restriction	Description
Communica-tion	Send	-	send a message to another session vertex
Role Handling	SessionGetRoleHolder	-	get a list of participants holding a certain role
	SessionGetRole	-	get the list of roles a participant holds
	SessionMaster	master	transfer the master role
	AddRole	master	add a role to a participant
	RemoveRole	master	remove a role from a participant
Application Sharing	GetXapps	-	get the list of sharable X11 applications
	ShareXapp	-	share an X11 application
	UnshareXapp	-	unshare an X11 application

Table 1: API Procedure Call Overview

Event	Description
Invitation	invite the participant to a session
BroadcastRcv	reception of a broadcast message
NewSession	notification for session directory update
UserNotify	notify an new registration or deregistration
NewAV	notify a audio/video parameter change

Table 2: API Event Notification To Control Panel

Event Type	Evid	Description
Receive	1	a message from another session vertex
Join	2	a new participant in the session
Left	3	a participant left the session
Kill	4	the session is killed
RoleAdd	5	a role got added to this session vertex
RoleDel	6	a role got removed from this session vertex

Table 3: API Event Notification To Session Vertices

6.2. API Types

Since the Beteus API is based on Tcl all variables come as strings and lists of strings. A string can have multiple interpretations which for themselves are not part of the weakly typed Tcl. The type interpretations that are used by the API Tcl-DP calls are shown in Table 3. This table has columns for type name, Tcl conversion function, Tcl string format and corresponding C/C++ type. The type names that are defined are String, Integer, Enum, Time and List. String, Integer, Enum and Time are string variables in Tcl; List is an arbitrary list that contains String, Integer, Enum, Time or again a List as items. Within C or C++ code the Tcl standard conversion functions should be used for Integer, Enum and List. Conversion is not necessary for strings, and for the Time type there exists a proprietary conversion function. After conversion to C/C++ types, String is `char*`, Integer and Enum are `int`, Time is `long`, and List is decomposed into an array of `argc char*` strings of which again each has to be converted.

The basic types in Table 3 are the building block for the composed types that are listed in Table 4. Table 3 and Table 4 contain all type interpretations currently be used for the API.

Type Name	Conversion	Tcl String Format	C/C++ Type
String	(unnecessary)	'/0' terminated character string	char*
Integer	Tcl_GetInt()	'/0' terminated numeric character string	int
Enum{1,2,3,...}	Tcl_GetInt()	'/0' terminated numeric character string	int
Time	proprietary function	/0' terminated character string	typedef long Time
List{*listItem*}	Tcl_SplitList()	'/0' terminated character string containing list items separated by whitespace characters - if a list item is itself a list it is limited by curly braces	int argc,char* argv[]

Table 4: Basic Types

Type Name	Decomposition	Description
StringList	typedef List{String} StringList	list of strings
IntegerList	typedef List{Integer} IntegerList	list of integers
Eplist	typedef String Epname typedef Enum{1,2} Fdir typedef List{Epname Fdir} Eplist	list of endpoint descriptions - an endpoint description contains an endpoint name and a flow direction (1=source,2=sink)
Rlist	typedef String Rname typedef Integer Rid typedef List{Rname Rid} Rlist	list of roles - a list item contains a role name and a role identifier

Table 5: List Types

Type Name	Decomposition	Description
Pmlist	typedef Integer Parid typedef Integer Value typedef List{Parid Value} Pmlist	list of parameter settings - a parameter setting consists of a parameter identifier and a value

Table 5: List Types

Variable	Type	Source	Scope	Description
pid	Integer	site mgr	network (u)	participant identifier
sid	Integer	site mgr	network (u)	session identifier
svid	Integer	site mgr	session (u)	session vertex identifier
rid	Integer	site mgr	session (u)	role identifier
bid	Integer	site mgr	session (u)	bridge identifier
bsid	Integer	site mgr	session (u)	bridge set identifier
epname	String	config file	node (u)	endpoint name
pname	String	user	network	participant name
rname	String	application	session (u)	role name
node	String	config file	site (u)	node name
svname	String	site mgr	session(u)	session vertex name
aname	String	application	network (u)	application name
sname	String	user	network	session name
xappname	String	user	site (u)	name of an X11 sharable application

Table 6: Names and Identifiers

6.3. API Names and Identifiers

The names and identifiers that are used in API calls and event notifications correspond to the main abstractions that are discussed in Section 4. Identifiers are of type Integer and are, with few exceptions, assigned by the platform; names are Strings that are assigned by the user, the application, or the configuration file. Most names have equivalences to identifiers. Names are important on the level of graphical user interfaces where they represent the equivalent identifiers in a user-friendly way; identifiers are the real control handles for the platform and remain us such hidden to the user. The exceptions to this rule are the names that denominate nodes and endpoints in the site configuration file.

Table 5 presents a list of the most important names and identifiers. For the specification of the API calls and event notifications variable names are used to circumvent the Tcl type problem. A vari-

able that corresponds to a major concept will therefore have the same name throughout the specification. Table 5 lists for each variable name, type interpretation, assignment source, scope and description. It is also indicated if a name or identifier is unique within its scope.

Identifiers

The participant identifier and the session identifier have network (platform) scope and are unique. Combined they would constitute a unique session vertex identifier. Nevertheless, it was decided to have a separate identifier for this. The session vertex, role, bridge and bridge set identifiers have session scope and are unique within the session.

Names

Node names have site scope and are unique within a site. Endpoint names have node scope and are unique within their node. Participant and session names have network scope and are visible via the directory service - they do not have to be unique, although it would make sense. A session application name has network scope and must be unique. A role name has session scope and should be unique within the session.

Some words need to be said about the session vertex name. It was felt that it is necessary to have unique session vertex names for the directory service and for display in the user interfaces of the applications. A session vertex name is assigned by the site manager and is a combination of participant name, node name and site name (participant.node.site). The session vertex name is unique within a session; it is unique on the network in case a user on a node can not participate in more than one session. Note that the session vertex name can also be used as a unique participant name.

7 API Procedure Call Reference

This section contains a detailed reference for the API procedure calls.

Calling from within a Tcl Script

The syntax to be followed when calling a procedure from a Tcl script is:

```
dp_RPC $socket ProcedureName par1 par2 par3.....parn
```

The socket that is used for the communication with the site management is obtained with a call to the Tcl-DP procedures `dp_MakeRPCClient`and `dp_connect`.

Calling from within a C/C++ Program

There are a couple of C routines that give C/C++ programs access to the Tcl-DP remote procedure call commands. The routine Tdp_RPC() contains a command parameter - this is the Tcl script that is to be evaluated by the remote Tcl interpreter. It will be set to

```
char* command = "ProcedureName par1 par2 par3.....parn";
```

See Section 3 or the Tcl-DP manual pages for more detail.

7.1. Registration

Prior to any other action a user must register himself with the platform. Once registered a user is visible to other registered users via the directory service of the platform. A user will deregister when he has finished all of his session activities and when he will no more be reachable. Registration and deregistration is a functionality offered by the user interface of the control panel.

Register

A user registers with the name under which he wants to be handled in the directory service and in

Register pname node		
pname	String	participant name
node	String	node name
returns: pid	Integer	participant identifier

following sessions. The control panel itself adds the name of the host or X11 display on which it is running as node name to the registration message. The node name must have a correspondence in the site configuration file. Only one user can be registered with a given node at a time, i.e., a second registration attempt on a node will fail. Note that it is possible that a user registers with the same name at more than one node, if for some reason he wants to participate in more than one session at a time. The registration call returns the unique participant identifier with which the control

panel and the eventual session vertices will identify themselves in consequent calls.

Deregister

For deregistration the control panel simply gives the participant identifier to identify itself. On

Deregister pid		
pid	Integer	participant identifier
returns: -		

deregistration the participant will be removed from the session directory and from all sessions in which he participates.

7.2. Endpoint Handling

Once a participant is registered his control panel and any subsequently launched session vertex can ask for the list of audio and video endpoints that are registered in the site configuration file for this node. The control panel can then let the user control the devices that are associated with these endpoints. Session vertices may offer such control in their user interfaces in parallel to the control panel, but it is more likely that they will control endpoint devices as part of the application scenario in a hidden way.

GetEndpoints

The participant identifier identifies the node for which the endpoint list is wanted. The flow type

GetEndpoints pid ftype		
pid	Integer	participant identifier
ftype	Enum{1,2}	flow type: 1=audio,2=video
returns: eplist	Eplist	list of ftype node endpoints

parameter specifies which endpoint list is wanted: audio or video. The return parameter is a list of audio or video endpoint devices. A list item contains the endpoint name and an indication if the endpoint is source or sink.

SetDeviceParameter

To set a device parameter the caller identifies the concerned node with the participant identifier

SetDeviceParameter pid epname parId value		
pid	Integer	participant identifier

SetDeviceParameter pid epname parId value		
epname	String	endpoint name
parid	Integer	parameter identifier
value	Integer	value
returns: -		

and the device parameter with an endpoint name and a parameter identifier. The parameter value is of type Integer.

GetDeviceParameter

This call returns the parameter settings for one endpoint. An item in the returned parameter list

GetDeviceParameter pid epname		
pid	Integer	participant identifier
epname	String	endpoint name
returns: pmlist	Pmlist	list of parameter id/value pairs

contains a parameter identifier and a parameter value.

Disable

This call allows to disable an endpoint. The interpretation of this action is endpoint specific: a

Disable pid epname		
pid	Integer	participant identifier
epname	String	endpoint name
returns: -		

video source will stop to transmit, whereas a video sink will stop to playout the received stream. Note that endpoints are per default enabled on startup.

Enable

This call allows to enable an endpoint that has previously been disabled.

Enable pid epname		
pid	Integer	participant identifier
epname	String	endpoint name

Enable pid epname		
returns: -		

Loop

An audio or video stream source can be looped back onto a sink endpoint within the same node.

Loop pid srcepname sinkepname		
pid	Integer	participant identifier
srcepname	String	source endpoint name
sinkepname	String	sink endpoint name
returns: -		

This allows a participant to see or hear how he his perceived by other session participants. In the implemented version, this has no effect outside a session since the audio/video processes are not yet running.

Unloop

This call is the inverse to the loop call; it disconnects the given sink endpoint from a local source

Unloop pid sinkepname		
pid	Integer	participant identifier
sinkepname	String	sink endpoint name
returns: -		

endpoints (assuming that a source can be connected to many sinks, but a sink get connected only with one loop at a time).

7.3. Session Directory

The session directory service offers access to information about registered users and announced and ongoing sessions. The calls listed here are typically used by the control panel, but there is no limitation on them being also used by a session vertex.

SessionAnnounce

Every session must be announced before it is actually started. The announcement call returns the

SessionAnnounce pid sname priv sched desc aname command plist		
pid	Integer	participant identifier

SessionAnnounce pid sname priv sched desc aname command plist		
sname	String	session name
priv	Enum{1,2}	session privacy: 1=public,2=private
sched	Time	the time for which the session is scheduled
desc	String	a textual description of the session
aname	String	the name of the session application
command	String	session vertex startup command
plist	StringList	a list of authorized session participants
returns:sid	Integer	session identifier

session identifier that is subsequently needed to start the session. The information given with the announcement call will be made visible to all registered users via the directory service. The session name is an informal attribute of the announced session that helps distinguishing different session announcements. The session name is complemented by a textual description of what this session is about. The privacy parameter indicates if the session is public or private. The announcement call contains further the name of the session application, the command with which a session vertex is launched, and a list of authorized participants in case the session is private.

A session remains in the state *announced* until it is created or cancelled. The session directory does not remove it automatically when the session is not created at the time for which it is scheduled.

SessionAnnouncementCancel

A session announcement can be cancelled by the participant who announced it.

SessionAnnouncementCancel pid sid		
pid	Integer	participant identifier
sid	Integer	session identifier
returns: -		

SessionAnnouncementGet

This call returns the session identifiers for all announced or ongoing sessions. These identifiers

SessionAnnouncementGet		
returns: slist	IntegerList	list of session identifiers

are subsequently used to query the session directory for sessions.

SessionOngoingGet

This call returns the session identifiers for all ongoing sessions. These identifiers are subsequently

SessionOngoingGet		
returns: slist	IntegerList	list of session identifiers

used to query the session directory for ongoing sessions.

SessionAnnouncementQuery

This call gives access to the session information that was previously given to the session directory within a session announcement. If the session is private and the querying participant is not in the

SessionAnnouncementQuery pid sid		
pid	Integer	participant identifier
sid	Integer	session identifier
returns:		
sname	String	session name
phase	Enum{1..5}	session state: 1=announced, 2=init,3=ongoing, 4=suspended,5=recovering
priv	Enum{1,2}	privacy: 1=public,2=private
sched	Time	the time for which the session is scheduled
desc	String	the textual description of the session
aname	String	the name of the session applications
command	String	session vertex startup command
plist	StringList	list of authorized participants

list of authorized users, the parameters scheduled time, description and participant list are not returned.

SessionOngoingQuery

This call retrieves information about ongoing sessions. The call returns the session name and a list

SessionOngoingQuery pid sid		
pid	Integer	participant identifier
sid	Integer	session identifier
returns:		

SessionOngoingQuery pid sid		
sname	String	session name
plist	StringList	list of current session participants

of current session participants.

7.4. Session Startup

The calls within the session startup category have to be called in a certain sequence. A session that has previously been announced will be started at some time. Following a respective user command the control panel will fork and execute the command line that is associated with the session announcement. The resulting session vertex will call SessionInit at the beginning and SessionStart at the end of the session startup phase. Inbetween these calls it will configure the session with the application specific roles, bridges and bridge sets, and it will register the event callbacks.

Note that only the participant which announced the session can actually initialize and start it. The session vertex of this participant automatically becomes the master of the session - a role that can subsequently be passed to another session vertex.

SessionInit

This call marks the beginning of the session initialization period. The session identifier parameter

SessionInit pid sid		
pid	Integer	participant identifier
sid	Integer	session identifier
returns: -		

must correspond to a session that is announced by the calling participant. The session changes from state *announced* to state *init*.

SessionInitRole

Roles need to be made known to the infrastructure before they can be used for bridge definitions.

SessionInitRole pid rname max plist		
pid	Integer	participant identifier
sid	String	session identifier
rname	String	role name
max	Integer	maximum number of role holders

SessionInitRole pid rname max plist		
plist	StringList	list of participants that may hold this role
returns: rid	Integer	role identifier

The role initialization call contains an application specific role name, the maximum number of session participants that may hold the role, and possibly a list of participants to which this role can be assigned to. This list of participant is a list of name, thus participants that register with these name can take the role. The participant list is empty if there no restrictions on role assignment.

The call returns an identifier for the role.

DefineBridge

A bridge can be a point-to-point, point-to-multipoint, multipoint-to-point or multipoint-to-multipoint connection depending on how the source and sink roles specified in the DefineBridge call are distributed within the session. A bridge is either an audio, a video or an application sharing connection as given in the type parameter. The priority level decides which of two or more active bridges that are competing for the same endpoint will prevail. The information and time granularity parameters are generic descriptions of stream parameters. In the case of video, time granularity is interpreted as frame rate and information granularity as window size. In the case of audio, time granularity is sample rate and information granularity is sample size. The source and sink endpoint names specify the logical devices that will terminate the network unicast or multicast connections of which the bridge is assembled. The source and sink role identifiers point to

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

the session vertices that are interconnected by the bridge. Note that there may be more than one sink role. This allows to build point-to-multipoint and multipoint-to-multipoint connections with different roles at the sinks. A multipoint-to-point or multipoint-to-multipoint connection is created if there is more than one session vertex holding the respective source role.

The call returns a bridge identifier that is subsequently used to add the bridge to bridge sets.

DefineBridgeSet

Bridges are combined to bridge sets. The call contains a list of bridge identifiers and returns a bridge set identifier. A certain bridge may be added to more than one bridge set.

DefineBridgeSet pid sid blist		
pid	Integer	participant identifier
sid	String	session identifier
blist	IntegerList	bridge identifier list
returns: bsid	Integer	bridge set identifier

RegisterCallback

The event identifier specifies the event for which a callback is to be registered (see Table 2). The

RegisterCallback pid sid evid tclcb		
pid	Integer	participant identifier
sid	Integer	session identifier
evid	Integer	event identifier
tclcb	String	TCL callback identifier
returns: -		

site management uses the TCL callback identifier when sending event notifications to the session vertex.

SessionStart

After a session has been initialized it is started. The session enters the state *ongoing* and estab-

SessionStart pid sid bsid		
pid	Integer	participant identifier
sid	Integer	session identifier

SessionStart pid sid bsid		
bsid	Integer	bridge set identifier
returns: -		

lishes the connection structure that corresponds the specified bridge set as soon as participants join the session. Note that the session master also has to join the session to become session participant.

7.5. Session Information

The following calls allow for some convenience in application implementation; they allow session vertices to retrieve identifers and names in an ad hoc manner, i.e., without memorizing them.

GetParticipantId

This call returns the participant identifier that corresponds to a participant name.

GetParticipantId pname		
pname	String	participant name
returns: pid	Integer	participant identifier

GetParticipantName

This call returns the participant name that is associated with a participant identifier.

GetParticipantName pid		
pid	Integer	participant identifier
returns:pname	String	participant name

GetSessionVertexId

This call returns the identifier of the session vertex that runs at a certain node.

GetSessionVertexID svname		
svname	String	node name
returns: svid	Integer	session vertex identifier

GetSessionVertexName

This call returns the node name that is associated with a registered participant.

GetSessionVertexName pid		
pid	Integer	participant identifier
returns:svname	String	session vertex name

SessionGetMaster

This call returns the name of the session vertex that holds the master role for the given session.

SessionGetMaster pid sid		
pid	Integer	participant identifier
sid	Integer	session identifier
returns:svname	String	session vertex name

7.6. Session Control

The calls in this category deal with session membership and session lifetime. A session can only be joined; there is a session invitation mechanism to notify the participant. Session vertices can leave a session or can be kicked out by the session master. The session master can kill the session.

SessionJoin

The SessionJoin call contains the initial specification of the session vertex name. The session ver-

SessionJoin pid sid svname		
pid	Integer	participant identifier
sid	Integer	session identifier
svname	String	session vertex name
returns: svid	Integer	session vertex identifier

tex name is usually the same as the participant name, but it could be another name in further implementation that allows a participant to be in more than one session at a time. A global session vertex name will be assigned by the site manager using a dotted notation consisting of the session vertex name, the node name and the site name. This forms a unique name that can be used for the identification of a session vertex within a session. Nevertheless, the site management also returns a unique session vertex identifier. The session vertex name is displayed in user interfaces, whereas the session vertex identifier is only used for control purposes.

SessionLeave

Any session participant but a master can leave a session. If a session master wants to leave a ses-

SessionLeave sid svname		
sid	Integer	session identifier
svid	Integer	session vertex identifier
returns: -		

sion he first has to turn over his role to another session participant. The session participant identifies himself with his session vertex identifier.

SessionKill

Only the session master can kill the session. The session master identifies himself with his session

SessionKill sid svid		
sid	Integer	
svid	Integer	
returns: -		

vertex identifier.

7.7. Bridge Set Handling

The number of bridge sets defined for an application corresponds to possible application states. The connection structures established among the session participants by two bridge sets are likely to be fundamentally different from each other.

ChangeBridgeSet

Only the session master can change a bridge set. The initial bridge set was specified in the SessionStart call. When changing the bridge set the master identifies himself with his session vertex

ChangeBridgeSet sid svid bsid		
sid	Integer	session identifier
svid	Integer	session vertex identifier
bsid	Integer	bridge set identifier
returns: -		

identifier.

7.8. Communication

The session vertices of a session are the endpoints of a distributed application; as such they must communicate with each other.

Send

This call sends a message to a list of session vertices. If the session vertex wants to send a mes-

Send svlist msg		
svlist	StringList	session identifier list
msg	String	message
returns: -		

sage to specific role holders he first has to find out about their session vertex identifiers (Session-GetRoleHolder).

7.9. Role Handling

Roles can be added or removed from session vertices. A session vertex may hold more than one role at the same time.

SessionGetRoleHolder

This call returns a list of the participants that hold a certain role.

SessionGetRoleHolder pid sid rid		
pid	Integer	participant identifier
sid	Integer	session identifier
rid	Integer	role identifier
returns: plist	StringList	list of session vertex global names

SessionGetRole

This call returns the list of roles that a participant holds within a certain session.

SessionGetRole pid sid		
pid	Integer	participant identifier
sid	Integer	session identifier
returns: rlist	Rlist	list of role id/name pairs

SessionMaster

This call transfers the master role to another session vertex. Only the master himself can transfer

SessionMaster sid svid newsvid		
sid	Integer	session identifier
svid	Integer	master session vertex identifier
newsvid	Integer	new master
returns: -		

his master role to another session vertex.

AddRole

With this call the master adds a role to a session vertex.

AddRole sid svid rid		
sid	Integer	session identifier
svid	Integer	session vertex identifier
rid	Integer	role identifier
returns: -		

RemoveRole

With this call the master removes a role from a session vertex.

RemoveRole sid svid rid		
sid	Integer	session identifier
svid	Integer	session vertex identifier
rid	Integer	role identifier
returns: -		

7.10. Application Sharing

The following calls control application sharing.

GetXapps

This call gets a list of sharable applications for a certain application.

GetXapps sid svid bid		
sid	Integer	session identifier
svid	Integer	session vertex identifier
bid	Integer	bridge identifier
returns: applist	StringList	application list

ShareXapp

This call shares a certain X11 application. The associated bridge determines with whom this

ShareXapp sid svid bid [xapp]		
sid	Integer	session identifier
svid	Integer	session vertex identifier
bid	Integer	bridge identifier
xapp	String	application name
returns: xapp	String	application name

application is shared. Note that if the specified bridge in not in the current bridge set, there will be no effect until the change of an appropriate bridge set.

UnshareXapp

This call unshares an application - a shared application is disassociated from a bridge.

UnshareXapp sid svid bid [xapp]		
sid	Integer	session identifier
svid	Integer	session vertex identifier
bid	Integer	bridge identifier
xapp	String	application name
returns: xapp	String	application name

8 Event Notification Reference

Callbacks are registered with the procedure call RegisterCallback. The event identifier that is used for this registration is given in the event description tables.

8.1. Callbacks to the Beteus Console

Invitation

An invitation notification is sent when a session starts or ends and the participant is in the list of

Invitation cmd sid		
cmd	Integer	new: cmd=1, delete: cmd=2
sid	Integer	session identifier

invited participants.

BroadcastRcv

A broadcast message is transmitted to the control panel. Svname is the global session vertex name

BroadcastRcv svname msg		
svname	String	session vertex name
msg	String	message

of the sender.

NewSession

A notification is sent when a session has changed state in order to update the session directory of

NewSession sid		
sid	Integer	session identifier

the control panel.

UserNotify

A notification is sent when a participant registers of deregisters.

UserNotify cmd pname		
cmd	Integer	register: cmd=1, deregister: cmd=2
pname	String	participant global name

8.2. Callbacks to Beteus SessionVertices

All callbacks contain a session identifier

Receive

A message is received (the message was sent with Send).

Receive sid msg		
sid	Integer	session identifier
msg	String	message
Event identifier: 1		

Join

A session vertex joined the session. Note that the session vertex that receives this notification is not interested in the session vertex identifier. It will probably just display the session vertex name of the new participant in its user interface.

Join sid svname		
sid	Integer	session identifier
svname	String	session vertex name
Event identifier: 2		

Left

A session vertex left the session. Note that the session vertex that receives this message is not

Left sid svname		
sid	Integer	session identifier

Left sid svname		
svname	String	session vertex name
Event identifier: 3		

interested in the session vertex identifier.

Kill

The session got killed.

Kill sid		
sid	Integer	session identifier
Event identifier: 4		

RoleAdd

The session vertex gets a role added by the session master.

RoleAdd sid svid rid		
sid	Integer	session identifier
svid	Integer	session vertex identifier
rid	Integer	role identifier
Event identifier: 5		

RoleDel

The session master removed a role from this session vertex.

RoleDel sid svid rid		
sid	Integer	session identifier
svid	Integer	session vertex identifier
rid	Integer	role identifier
Event identifier: 6		

9 Bibliography

[1] Beteus Report: "Functional specification", Deliverable D2, July1994.

[2] Beteus Report: "Detailed specification", Deliverable D6, November 1994.

[3] Beteus Report: "Working Prototype of the Application Platform Specification", Deliverable D8, June 1995.

[4] C. Blum, Ph. Dubois, R. Molva, and O. Schaller,"A Semi-Distributed Platform for the Support of CSCW Applications", *Proceedings of the 1st International Distributed Conference*, Madeira, November 1995.

[5] Y.-H. Pusztaszeri, E. Biersack, Ph. Dubois, J.-P. Gaspoz M. Goud, P. Gros, JP Hubaux :"Multimedia Teletutoring over a Trans-European ATM Network", *2nd IWACA Conference*, Heidelberg, September 1994.

[6] J. K. Ousterhout, "TCL and TK Toolkit", Addison-Wesley Publishing,1994.

[7] 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.

[8] J. Lindblad and O. Schaller:"BeCool and the BETEUS Application Programming Interface", Eurécom Technical Report, June 1995.

[9] J. Lindblad:"The BeCool Thesis Project Report", Master's thesis at the KTH Sweden, 1995.

[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*, Februrary 1995.