Real-Time Control and Management of Distributed Applications using IP-Multicast

P. Parnes, K. Synnes, D. Schefström
Luleå University of Technology/Centre for Distance-spanning Technology
Department of Computer Science
971 87 Luleå, Sweden
Peter.Parnes,Kare.Synnes,Dick.Schefstrom@cdt.luth.se

Abstract

A central issue within any distributed computer environment is how to control and manage running applications. This paper presents an implementation of a framework for control and management of distributed applications and components using IP-multicast. The framework allows for easy and scalable control of single applications, groups of applications or parts of applications using a new agent based architecture. Messaging is done using the Control Bus and the Scalable Reliable Real-time Transfer Protocol for reliable distribution of data. The paper presents how this framework is integrated into Java based applications and how developers specify access points. The paper also presents an application called multicast Manager - mManager, a Java implementation that provides a user interface to the framework. The mManager allows administrators to get an overview of currently running applications and if necessary control these applications. The paper presents example usage scenarios where the framework is used to create bandwidth adaptive applications and better group awareness.

Keywords

Distributed Systems and Applications Management, Intelligent Agent Technology, Integration of Network Control and Management, Quality of Service Management, Multimedia and Telecommunication Service Management, IP-Multicast, Java

1. Introduction

The number of deployed applications on the Internet and within organizations increases every day. This leads to a need for easy and scalable control and management of these applications. Reference [1] presents a scalable and secure framework for integrating remote control of distributed applications. This paper presents an implementation of this framework.

The rest if this section presents a summary of the control and management framework. Section 2. presents the architecture and implementation of the framework and

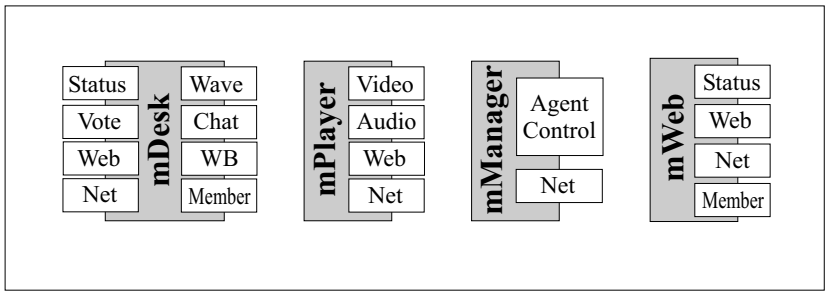

Figure 1: Example agent based applications. Note, that the same agents are reused in different applications.

the multicast Manager - mManager application. Section 3. presents how the work presented in this paper can be utilized in real world situations. Section 4. presents some related work and the paper is concluded with a summary in section 5.

1.1 The Management and Control Framework

Reference [1] presents a framework for control and management of software applications. It allows applications to distribute messages in a scalable way, with regard to both the number of applications currently running and the available control bandwidth. This is done using IP-multicast, a messaging platform called the Control Bus - CB (see below) [2] and a reliable multicast protocol - SRRTP [3]. Note, that the whole framework is designed without any special requirements from the underlying transport mechanism as long as it is transport reliable and uses IP-multicast (unicast can be used but the framework then becomes much less scalable).

The frameworks basic assumption is that applications are built following an agent based architecture, where applications consist of a number of different agents that each encapsulate a separate functionality. For instance, a video-agent would be responsible for capturing and displaying video data and a database-agent would act as an interface to a database-engine. Numerous agents can and are normally deployed within one and the same application. Figure 1 shows some example agent based applications (especially note, how the same agents are being reused in several different applications).

The novel usage of IP-multicast for management and control creates a mobile and scalable framework that can be used for a number of different applications.

Framework Requirements

A number of requirements were set up for the framework. This section presents these requirements and how they were addressed in the framework design.

- *The framework have to support large groups of applications, both for reporting and control.* This have been addressed by using IP-multicast for both control and reporting. Also, the delay between messages is dynamically calculated based on the current number of members in a session and the currently available bandwidth.

- *The framework have to support efficient control of groups of applications without the need to send control messages to each application explicitly.* The control bus allows single messages to be addressed to more than one application/agent.
- *The framework have to protect users from unauthorized control of their applications.* This have been addressed by including support for asymmetric cryptography and digital signatures.
- *The framework have to allow developers to easily insert access points in their software.* This have only been addressed from a Java perspective where a simple but powerful API have been defined.
- *The framework should be as independent of the underlying software and transport technology as possible.* The whole framework is independent of the underlying transport mechanism but requires IP-multicast for scalability.
- *The framework should allow for scalable resource discovery of both available control objects and controllable variables and methods in these objects.* This have been addressed by allowing controllable agents to announce themselves and provide information about available access points.

The Control Bus

The framework provides a so called *Control Bus - CB* [2] for agent messaging and interaction. This section presents the control bus as it is very central to the framework and the further work presented in this paper.

The CB is designed for transmitting messages both within and between different instances of applications. The control bus is not tied into any specific transport mechanism but it is designed to be used together with any underlying reliable transport protocol.

The format of the CB messages are based on the message formats presented in [4] and [5]. Each message is completely text based and a message consists of four sections:

1. The *from section* which identifies the sender uniquely. The address field consists of four parts: `Host / CB-id / Agent-type / Instance`.
2. The *to section* which identifies the destination of the message. The address format is the same as for the from section with the exception that any part can be replaced with a wild-card. This allows for simple and dynamic addressing of messages where a message can be directed to a single agent, all applications within a session or a group of agents independently of which application they reside in.
3. The *message id section* which together with the from section forms a unique identifier within the session. This section consist of a sequence number that is increased for each transmitted message.
4. The *message section* which contains the message itself.

The traffic on the CB can be both one-way and two-ways, i.e., messages can be of the information-type where an agent is announcing something and is not expecting any response or it can be a request for more information where the requesting agent is expecting a response.

To simplify the protocol only text based arguments and values are allowed. If a specific application needs another type, then that application has to take care of conversion to and from a text representation. The use of a simple and pure textual message format makes it easier to create a platform independent messaging system. It also makes it easier to integrate the control bus into existing applications. As the target of this work is to create a framework that should support developers in the process of creating new distributed applications, the choice of using a text based protocol also makes it easier for developers to debug their applications and make them more stable.

Of course, there is a cost of using text based messages, both in processing and in the amount of bandwidth used but remember that the focus here is to support the transportation of small information reports and occasional control messages. Therefore, it is argued that the overhead in converting the messages between application specific formats and their corresponding textual CB representation can be neglected. For an evaluation of the amount of bandwidth needed for control and management, see [1].

Resource Discovery

We now continue with discussing the resource discovery and actual basic messages needed in the framework. There are two aspects of resource discovery, finding the actual agents to control and finding out what control messages they support.

An important aspect is to be able to find which agents are available within a certain group. Administrators should not have to keep track of which users use which applications and when. This process is supported by the `alive` message which requests all agents to return a short answer if they exist and the `info-request describe` message for retrieving a textual description of the agent. This allows the administrator to dynamically build a list over currently active agents.

In difference from SNMP [6] reference [1] propose that every controllable agent should be able to supply information about which variables are controllable in that particular agent. This leads to a dynamic environment where every agent can evolve and be further supported with new control functionality as needed. The alternative would be to define all controllable variables and accessible methods in definition documents and very rarely change these. [1] argue that this creates a stale environment that puts to much demand on the design phase instead of allowing developers to extend the interface as needed and promoting an evolving environment. This discovery process is supported by the `info-request describe-all` message which requests a summary of all supported commands. Each command can be either write-enabled, read-enabled or both. The framework does not separate commands from get/set of variables in an agent. A command invocation is modeled as a set (with or without arguments).

Finally, the control and management framework must contain messages for actually invoking the get and set methods and these are called `get-request` and `set-request`.

Figure 2 show a number of example messages with corresponding control bus headers. Using these basic messages more information about the agent of interest and information about which methods it supports can easily be found.

2. The Management and Control Framework Implementation

This section presents the architecture and implementation of the framework and the multicast Manager application. The implementation is realized using the Java programming language and environment.

```
130.240.194.245/2345/mManager/45663 */2345/net/*
     7 Alive
130.240.64.42/2345/net/33664 130.240.194.245/2345/mManager/45663
     73 OK 7 Alive
130.240.64.54/2345/net/883243 130.240.194.245/2345/mManager/45663
     22 OK 7 Alive
130.240.64.97/2345/net/43526 130.240.194.245/2345/mManager/45663
     983 OK 7 Alive

130.240.194.245/2345/mManager/45663 130.240.64.42/2345/net/33664
     8 info-request describe
130.240.64.42/2345/net/33664 130.240.194.245/2345/mManager/45663
     74 info-reply 8 mStar network agent

130.240.194.245/2345/mManager/45663 */2345/audio/*
     9 set-request size-ms 40
130.240.194.245/2345/mManager/45663 */2345/video/*
     10 set-request fps 15
```

Figure 2: Example CB messages. **top**: resource discovery. **middle:** information request. **bottom:** invoking methods in several agents (note, that no answer or acknowledgment is sent as the underlying mechanism is expected to be reliable. Also note, that the Hollerith-encoding is omitted for readability reasons).

The internal structure of the control bus implementation is shown in figure 3. Every agent logically have its own *control bus (CB)* object that acts as an interface to the control bus itself. Messages to and from each agent is passed through this control bus object. Every CB object is registered with the *internal multicast emulator* which logically propagates messages from a sender, to any number of registered listeners. Next is the *control bus proxy* which checks if messages should leave/enter the application or if they should be stopped in the proxy. Messages being sent from one agent to e.g. all other agents within the same application should not be sent out over the network. Finally, we have the *network control interface* takes care of all the communication over the network and the Internet. Note, that the network control interface is actually an agent of its own and have its own CB object. This allows for control and monitoring of different network related parameters.

2.1 Control Bus Registration

Every agent have to register itself to the control bus manager to be accessible via the control bus. This is done by creating a new CB object and providing a textual description of the agent. For instance, the network agents registration looks as follows:

```
String desc = "This is the network agent that handles"
   + "incoming and outgoing messages. It parses  "
   + "and composes RTP and RTCP packets.";
cb = new CB(this, "cb", "net", this.hashCode(), desc);
```

The agent registers which object should be called and the call-back identifier that should be used when a non-handled control bus message is received.

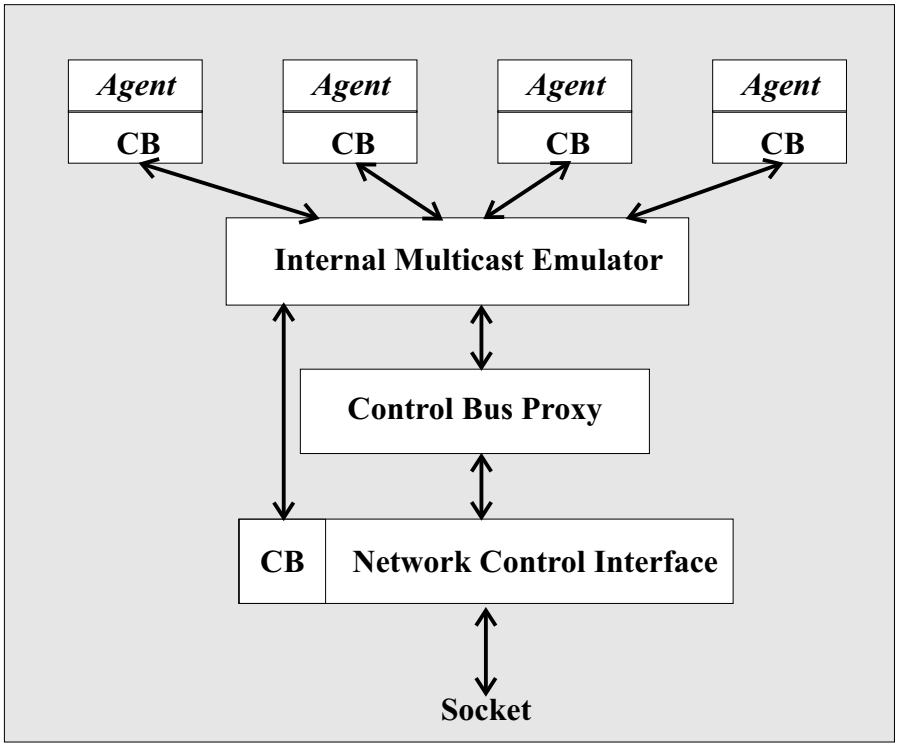

Figure 3: The internal architecture of the control bus Java implementation.

2.2 Agent Access Point Registration

Agents that provide access to certain access points have to register these through the Control Bus interface. For every access point, the agent have to provide seven different arguments: the name of the access point; a textual description; the object that contains an instance of the access point; the name of the get-method; the name of the set-method; the number of arguments that the set-method requires; and a textual description of which arguments the set-method requires. The following code-examples from the the network agent shows the registration of two different access points:

```
// First create a control bus access handler. Only one // needed per
agent.  CBAccessHandler cbacchandler = new CBAccessHandler();

// TTL: Set/Get of the current TTL used for a connection.
CBAccess cbacc = new CBAccess();
cbacc.setAll("TTL", "The current TTL used.", this,
   "getTTL", "setTTL", 1, "0-255");
cbacchandler.addCBAccess(cbacc);
```

```
// Address: Get of the current network destination
// address used for a connection. Note that there is
// no set-method as the destination address cannot
// be changed.
cbacc = new CBAccess();
cbacc.setAll("Address",
   "The current network address used.",
   this, "getAddress", null, 0, "IP-address");
cbacchandler.addCBAccess(cbacc);
```

These access-point methods will be called automatically when a corresponding control bus message is received. That is, if the following message is received

```
set-request 3HTTL 2H64
```

the `setTTL` method is called in the object specified when the method was registered. Note, that the arguments are Hollerith-encoded to allow for any characters in the access point names and arguments. A Hollerith string simply consists of three parts: 1: the length of the data; 2: the letter 'H'; 3: the actual data to encode.

Pre-defined Agent Access Points

There exists a number of pre-defined access points that can be accessed in any registered agent.

- `alive` which returns an ok-messages signaling that the agent exists and is available for interaction. This is used for resource discovery.
- `info-request variables` which returns information about either all available access points or one specific access point (if a third argument is provided).
- `info-request describe` returns a description of either the agent or a access point.
- `info-request numargs` returns the number of arguments a specific access point requires.
- `info-request values` return a textual description of which arguments a access point takes.
- `get-request` invokes the get-method if available in a access point.
- `set-request` invokes the set-method if available in a access point.

These different pre-defined access points are used by remote manager applications to build a view over a certain application and/or agent. This provides for a generic environment where information about available access points can be built on demand and agents can be further extended without updating some common definition document.

2.3 Example Access Points in the mStar Environment

Since 1995 we (CDT) have been developing the *mStar environment* [7] which is an environment for scalable and effective distribution of real-time media and data between a large number of users. mStar is a shared environment that can be used for a

number of different distance-spanning activities such as net-based learning (distance-education) and distributed collaborative team-work. It includes support for a number of different media including audio, video, shared whiteboard, distributed presentations using the World Wide Web and much more. It also supports on-demand recording and playback of sessions using either unicast or IP-multicast. The idea and need for a management framework came from the daily usage of the mStar environment at CDT. (Note, that the mStar environment is now being commercialized and sold under the name MarratechPro by the company Marratech AB in Sweden.)

Tables 1, 2 and 3 presents a number of example access points that are currently being used within different mStar applications.

Table 1: Example video grabber access points

Name	*Values*	*Description*
transmit	on/off	start/stop transmission
format	H261/H263/JPEG/NV	set video format
bps	0-	set the current bandwidth to be used
max_bps	0-	set the maximum allowed bandwidth
fps	0-30	set how frames per second should be sent
device	0-	set which device to use

Table 2: Example network access points

Name	*Values*	*Description*
address	IP-number	get the address of a connection
port	port-number	get the port-number of a connection
TTL	0-255	get/set the TTL of a connection
bytes_sent	0-	number of bytes sent on a connection
packets_received	0-	number of packets received on a connection
number_of_members	0-	number of known members on a connection

Table 3: Example mWeb access points

Name	*Values*	*Description*
current_url	URL	get the current URL being displayed
display_url	URL	request others to display a URL
history_back	none	go back in the URL history
history_forward	none	go forward in the URL history
set_history	list:URLs	set the contents of the URL history

These of course not all access points but some of the more used ones. These access points allow for easy integration of existing agents into new applications.

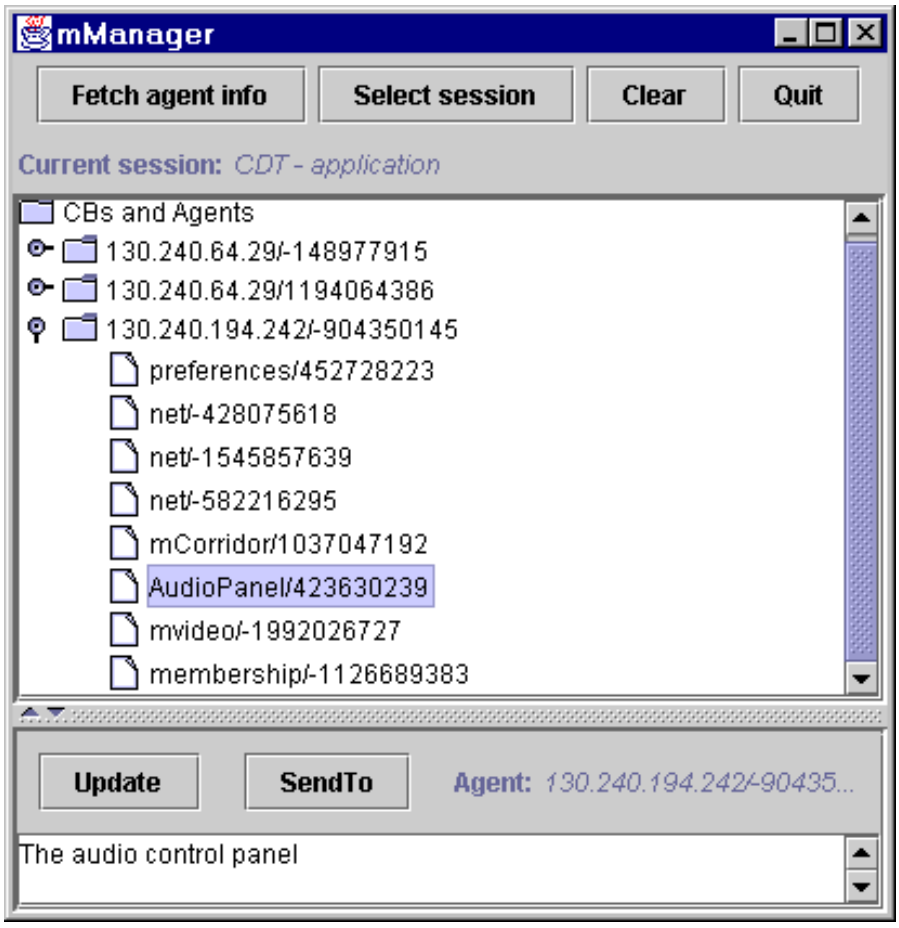

Figure 4: The mManager main user interface showing a number of active agents in a audio/video conferencing application.

2.4 The multicast Manager - mManager

To utilize the control and management framework and administrate running applications, a program called the multicast Manager - mManager have been developed. The mManager gives the administrator an overview over currently running applications and their agents. It also allows administrators to control these applications. The mManager application is also implemented using the Java programming language. Figure 4 show parts of the user interface of the mManager application.

The list over currently active agents within a specific session (different applications are members of different sessions and can by that be separated from each other) is gathered by sending out an alive-message to the network-agent of every application. As answers are received, information about all control bus active applications

are found and for each of these another directed alive-message is sent to each application. When the user requests more information about a certain agent that information is fetched first when needed. The information is of course cached locally for potential further access. As new users join the session (by starting instances of the application), information about active agents is gathered dynamically. This makes the list over active agents current and automatically updated over time.

From the mManager the user can interact with any access point in a single agent, groups of agents or all agents within a session.

3. Usage Scenarios

This section presents a number of usage scenarios of how the management and control environment can be and is being used.

Session Membership Management

The mStar environment can be used for light-weight desktop conferencing. It allows any user that have access to the application to transmit audio and video within a session.

Using the control and management framework, administrators can get an overview of the members of a session and they can control who within the session should be able to transmit. This can be controlled in real-time on a user by user basis.

Better Perceived Quality of Service using Adaptive Applications

The framework can be used to control the maximum allowed bandwidth within a session when it is noticed that the total utilization of the involved networks gets above a specific level. Together with automatic filters the framework can be used for creating a *better perceived Quality of Service environment* where a control process allocates bandwidth to session as needed and control applications so they cannot utilize more network bandwidth than allowed. Earlier this have only been possible using static configurations and control by an operator. Now it can be controlled dynamically in real-time depending on other concurrent sessions. This creates a set of adaptive applications that better utilizes the available bandwidth and scales to a larger number of simultaneous users *and* sessions as over-utilization by a greedy user/application can be prevented.

Imagine a network setup with four hosts where host A, B and C are running multimedia applications and X is the controller. The total bandwidth is set to 400 Kbps for all sessions on the network. Host A belong to session 1 and hosts B and C belong to session 2. Session 1 has 25% (max 100 Kbps) of the total bandwidth allocated to it. Initially only host A is active and it is transmitting with a bandwidth of 600 Kbps which is above the allowed maximum. The controller gets information about this via the regular reports and changes the transmit bandwidth of A to 400 (as nobody else is currently transmitting, A gets all available bandwidth allocated to it) Kbps via a control bus message. Next B and C joins and both start transmitting at the same time. The controller now lowers the bandwidth of A to 100 and sets the maximum allowed bandwidth for B and C to 150 Kbps each. After a short while the controller notices that host B is only transmitting with a bandwidth of 50 Kbps (the user selected to only transmit with this bandwidth) and therefore increases the available bandwidth for host C to 250 Kbps. When host A later leaves the session the controller reallocates the bandwidth from session 1 to session 2 and sets the maximum allowed bandwidth of C to 350 Kbps (note, that host B is still only transmitting with a bandwidth of 50 Kbps).

The framework can easily be used by applications to retrieve information about the currently allowed maximum bandwidth when the application starts. Instead of building this logic for adaptive applications into the applications themselves it can easier be controlled in real-time what policy should be used at a specific moment and installation.

Better Group Awareness

The mStar environment can be used as an *electronic corridor* where every user that wants[1] transmits a low bit-rate video stream from their office. This allows for their colleagues to peak into their offices and get a view of what they are currently doing. This can be used for seeing if the other party is available or not before trying to contact him/her. It can also be used to create better awareness in a distributed group environment where users geographically separated (e.g., working in different offices or from their homes) can see if their colleagues are working as well. In the existing mStar system every user sends the amount of video they decide themselves, independently if anyone is actually viewing the video-stream or not.

Using the control framework the receiving mStar instance should instead notify the other party's application when the local user is looking on a particular stream. When the other, remote application gets notified it should increase the amount of bandwidth used for that particular video stream. Further, the more users viewing the same video stream the higher bandwidth should be allocated to that stream.

The same concept can of course be used for real-time electronic meetings and lectures where the current number of viewers controls the relative bandwidth of a specific video stream. Note, that this example differs in its architecture from the earlier one in that it does not have any manager process controlling the session but instead all control is handled within the session itself.

4. Related Work

This section presents some related work to the work presented in this paper.

Simple Network Control Protocol - SNMP

The *Simple Network Control Protocol - SNMP* [8, 6] is designed for control of network elements and basic applications. It is designed with a 'polling' architecture in mind meaning that the managing software have to request information from each element to be monitored. This architecture allows managers to get information from a monitored object and set variables in that object.

SNMP includes support for so called trap's where a manager can request to be notified when some predefined situation occurs. A restriction with these trap's are that they cannot be defined dynamically but the manager has to rely on predefined trap's. Note, that although this trap mechanism exists the normally used part of the SNMP mechanism is still only the "get/set" functionality.

As SNMP is designed to control a single element at a time, it is not really suitable for controlling large groups of real-time applications. Another important aspect not supported by SNMP is resource discovery. Further, every user that want to control

[1]It is outside the scope of this paper to discuss why users would want to send video from their offices but experience have shown that camera shyness can be overcome after the added value to a distributed working environment is understood and that the purpose is *not* for the boss to monitor the employees but to support the daily teamwork.

and fetch information from a device have to have a corresponding definition document called a management information base to know what variables are actually available in the device to be controlled.

The Service Management System - SMS

The designers of *the Service Management System - SMS* [9] argue that the poll-architecture of SNMP causes managers to often notice problems too late as the object in trouble cannot notify its manager about its condition. The creators of the SMS system try to approach this problem by creating an architecture where each managed object is encapsulated by a SMS wrapper which marshals commands to and from the managed object. The wrapper has a SMS agent to aid it with automatic handling of certain situations. This SMS agent can be controlled by a set of rules (defined using the so called *Service Management Agent Programming language - SMAP*) to react on information provided by the SMS wrapper.

Although this architecture seems promising it does not include support for resource discovery, information reporting and scalable control of large groups of applications.

The Conference Control Channel Protocol - CCCP

Reference [4] presents the *Conference Control Channel Protocol - CCCP* which is a protocol for control of distributed multimedia applications. It consists of a text based message protocol for primarily control. The CCCP is similar to the work presented in this paper but differs in that we focus on a wider range of applications than just the group of conferencing applications and not only real-time control but also scalable reporting of information.

Reference [5] presents a similar message platform but that one is even more constrained and only target the problem of how to communicate between similar applications within a single host.

Java Specific Platforms for Distributed Applications

There exits a number of proposals for Java based architectures for distributed applications such as the InfoBus [10], JavaSpaces [11] and iBUS [12]. Some are very promising and flexible but they all make one large assumption on their programmatic environment and that is that they are very tightly integrated with the Java programming language and its runtime environment, They all assume that they have access to features that are very specific to the Java environment, such as Java-events and/or serialization (binary representation of Java runtime objects). This assumption makes these frameworks virtually unusable in other environments.

Other Potential Control and Management Technologies

Several other technologies could be used as the underlying mechanism (such as Corba [13], MPEG-4 DMIF [14], and SS7 [15]) but during our investigations we have found that all of these are either not scalable enough (centralized solutions where the central point becomes a bottle-neck) or to specific to their original design domain.

5. Summary and Conclusion

This paper presents the architecture and implementation of the framework for control and management of software applications as presented in [1]. It allows applications to distribute messages in a scalable way, with regard to both the number of applications currently running and the available control bandwidth. This is done using IP-multicast, a messaging platform called the Control Bus (CB) and a reliable multicast protocol (SRRTP). Note, that the whole framework is designed without any special requirements from the underlying transport mechanism as long as it is transport reliable and uses IP-multicast (unicast can be used but the framework becomes much less scalable then).

The novel usage of IP-multicast for management and control creates a mobile and scalable framework that can be used for a number of different applications. We presented an evaluation of the bandwidth usage and the total delay for doing control tasks.

The paper presents the Java based implementation and how programmers can interface the framework . To utilize the control and management framework and administrate running applications, a program called the multicast Manager - mManager have been developed. The mManager gives the administrator an overview over currently running applications and their agents. It also allows administrators to control these applications.

We presented how the framework can be used for creating better group awareness in a distributed real-time environment. Further, it was presented how the environment can be used to create a better perceived Quality-of-Service environment for distributed media applications where bandwidth is dynamically allocated between active sessions and the total amount of used bandwidth is controlled automatically by a control process. These different usage scenarios show that the framework is flexible and can be used for a number of different applications.

The framework is currently being evaluated in the industry so it is too early to draw any final conclusions about its practical usability but initial tests show that it works and is powerful enough to be used in real deployed application in the industry (note, that the CB and the SRRTP have both been used in large scale earlier in other contexts and applications for several years and should therefore be considered as stable).

The work presented in this paper is based on requests from both academia and industry. We believe that this framework simplifies the administration and control of large groups of distributed applications and real-time multimedia sessions.

Acknowledgments

Thanks to Claes gren and Serge Lachapelle (Marratech AB), the people at Ericsson Data and the anonymous reviewers for comments, suggestions, and feedback.

This work is supported by the Centre for Distance-spanning Technology (CDT), Marratech AB and the Swedish National Board for Industrial and Technical Development, NUTEK.

6. References

[1] P. Parnes, K. Synnes, and D. Schefström, “A Framework for Management and Control of Distributed Applications using Agents and IP-multicast,” in *Pro-*

ceedings of the 18:th IEEE INFOCOM Conference (INFOCOM'99), 1999. [2].

[2] P. Parnes, "Control bus specification." Work in progress[3], 1996.

[3] P. Parnes, "Scalable Reliable Real-time Transport Protocol - SRRTP." Work in progress[4], 1996.

[4] M. Handley, I. Wakeman, and J. Crowcroft, "The Conference Control Channel Protocol - CCCP: A scalabe base for building control applications," in *ACM SIGCOMM*, 1995.

[5] S. McCanne and V. Jacobson, "vic: A flexible framework for packet video," in *Proceedings of ACM Multimedia*, 1995.

[6] W. R. Stevens, *TCP/IP Illustrated*, vol. 1. Addison-Wesley, 1996. chapter 25.

[7] P. Parnes, *The mStar Environment - Scalable Distributed Teamwork using IP Multicast*. Luleå University of Technology, September 1997. Licentiate of Engineering Thesis.

[8] "Protocol operations for version 2 of the Simple Network Management Protocol (SNMPv2)." IETF RFC1905.

[9] Y. Izumi, T. Nakai, S. Yamaguchi, and Y. Oie, "Design and implementation of an agent system for application service management," in *Proceedings of ISOC INET'98*, 1998.

[10] M. Colan, "Infobus 1.1 specification," tech. rep., Sun Microsystems Inc. and Lotus Development Corp., March 1998. [5].

[11] "JavaSpace specification," March 1998. [6].

[12] S. Maffeis, "iBus - the Java Intranet Software Bus," tech. rep., SoftWired AG, February 1997. [7].

[13] "The OMG - Object Management Group web site." [8].

[14] I. J. S. . N. 2206, "Information technology - Generic coding of moving pictures and associated audio information - Part 6: Delivery Multimedia Integration Framework," 1998.

[15] "Common Channel Signaling System No. 7 - SS7." ITU Standard.

[2] <URL:http://www.cdt.luth.se/~peppar/docs/infocom99/>
[3] <URL:http://www.cdt.luth.se/~peppar/docs/>
[4] <URL:http://www.cdt.luth.se/~peppar/docs/rtp_srm/>
[5] <URL:http://java.sun.com/beans/infobus/infobus-1.1.pdf>
[6] <URL:http://java.sun.com/products/javaspaces/specs/js.pdf>
[7] <URL:http://www.softwired.ch/ibus/ibus_overview.pdf>
[8] <URL:http://www.omg.org/>