The Message Bus: A Platform for Component-based Conferencing Applications

Jörg Ott & Dirk Kutscher
TZI, Universität Bremen
Postfach 330440
28334 Bremen, Germany
{jo|dku}@tzi.uni-bremen.de

Colin Perkins
USC Information Sciences Institute
4350 N. Fairfax Drive #620
Arlington, VA 22203, USA
csp@isi.edu

ABSTRACT
We present the Message Bus (Mbus) [14], a lightweight, message oriented, infrastructure for ad-hoc composition of heterogeneous components. In this paper we describe the general Mbus architecture, the Mbus transport specification and higher-level services. It is shown how the Mbus fits into the multiparty multimedia conferencing architecture and can be used for decomposing conferencing applications.

Keywords
Message oriented middleware, coordination, conferencing, conference control

INTRODUCTION
Recent years have seen an explosion in the markets for remote teleconferencing, mobile telephony, and streaming media, with wide deployment of a range of increasingly sophisticated devices (e.g. PDAs, mobile telephones, and “web-devices”). There has also been a significant change in the underlying communication medium for many of these systems with increased reliance on IP-based transport, at the expense of earlier standards such as ISDN, H.320, GSM, ATM, etc.

These trends have resulted in a unique opportunity to develop a common infrastructure for ad-hoc composition of heterogeneous components. Such an infrastructure will leverage the increasing reliance of these devices on IP, to provide a common control and coordination channel, resulting in a large amount of experience that has previously been lacking.

This paper presents the Message Bus (Mbus): our vision of that common coordination infrastructure.

The Mbus developed out of the desire to support local conference control for multiparty multimedia conferencing in the Mbone community. In this community, a model has arisen whereby a set of loosely coupled tools is used to participate in a conference. A typical scenario is that audio, video and shared workspace functionality is provided by three separate tools (although some combined tools exist).

In [23] Henri ter Hofte introduces the notion of *grouplet services* that are the units of extension and composition in a groupware application. *Grouplet services* are used to implement certain functions within a groupware system, so-called *groupware services*. In the description of a reference architecture for component groupware the following guidelines for structuring *groupware services* into *grouplet services* are proposed (amongst others):

- Separate different media in different grouplet services
- Separate conference management services from other groupware services
- Separate coordination services from other groupware services

Multiparty multimedia conferencing is one specialization of groupware that is characterized by the use of synchronous collaboration between potentially large numbers of spatially separated particpants.

The decomposition of a conferencing application into separate media agents maps well onto the underlying media streams, which are also transmitted separately. Given such an architecture, it is useful to be able to perform some coordination of the separate media tools that perform protocol functions and present media data to the user. For example, it may be desirable to communicate playout-point information between audio and video tools, in order to implement lip-synchronization [10]. In addition, it may be useful to implement some degree of coordination between sessions, e.g. to arbitrate access to hardware devices, such as framegrabbers and audio input/output channels.

Usually, there also exist one or more user interface agents, which control the media engines, and provide a unified conferencing interface. This approach allows flexibility of user-interface, but clearly requires some inter-process communication channel by which a user-interface agent may communicate with the media engines.

Although current practice in the Mbone community is to rely on a lightweight session model, with little in the way of explicit conference control [3] [8], there are some situations where this is not appropriate, and a more tightly coupled wide-area conference control protocol must be employed. In [13] the notion of *vertical interoperability* within a conferencing endsystem has been introduced, which relied on a dedicated *conference management system* (CMS) controlling the set of media tools, user interfaces etc. in a user's conferencing system. The CMS, a conference controller, communicates with the local service entities *and* with controllers of other's users and can, for exmaple, implement the enforcement of conference policies and inform local entities about status changes in the conference.

Again, if a common local coordination channel is present, a separate conference control entity may be added to the session if desired to perform these functions that are required for tightly coupled conferencing.

Note also that these entities need not reside on the same device: you may wish to initiate a call from your PC, but participate via a traditional looking telephone, which is possible once the phone and PC are networked and intelligent.

Similar needs may be discovered when considering the kinds of distributed end-points that may become common with the increasing spread of technologies such as IrDA and Bluetooth that provide short-range wireless communication. For example, a PDA may wish to communicate with a cellular phone to dial a number from an address book, or the phone may wish to coordinate with your personal CD player to arbitrate access to your headset. Products that provide some of these features are already beeing developed [24].

- These applications point to the need for a local scope coordination protocol, yet this is an area where standards are lacking. For example (adapted from [7]) the tasks required for conference control can be broken down into five categories:
- Coordination and control of applications and components: tools need to be started with the correct initial state, but also may need to cooperate and be re-configured during the course of a session (e.g. lip-synchronization, voice switched video).
- Reporting and control of conference membership: who is currently in the conference, which media types they can receive. This may also incorporate admission control.
- Floor control: who may transmit at a particular time.
- Network management: comprising resource reservation, if required, reception quality reporting and congestion control.
- Meta-management: announcement, advertisement, initiation and teardown of conferences.

The Internet multimedia conferencing architecture provides protocol support in three of these five areas: membership reporting and control [17], network management [1] and meta-conference management [4] [6] [5] [18].

There is currently no provision within this protocol suite for distributed floor control, or for application control. It is this latter aspect, application coordination and control, that is addressed in this paper.

Roadmap

This paper is structured as follows: The next section provides an overview of the Message Bus infrastructure, followed by a detailed description of the transport services considering message structure, data transport, entity awareness and Mbus security.

Subsequently, we present a set of higher layer services that are used to realize more structured communication interactions and demonstrate the use of the Mbus by presenting a short example.

Finally, we provide a classification of the Mbus concepts comparing to related work followed by a final evaluation.

THE MESSAGE BUS INFRASTRUCTURE

Requirements

The Mbus provides an infrastructure for local coordination based upon mechanisms for inter-component communication. It is designed to satisfy the following needs:

- simplify the composition of building blocks from different sources through well-defined interfaces, thus supporting a modular system design;
- allow for simple, uncomplicated, and ad-hoc cooperation between dynamic groups of applications;
- be easy to implement and integrate into existing applications while enabling efficient independent development and testing;
- maximize re-usability of components in different systems as well as different system types;
- support separation of user interfaces from components providing protocol (or other) functions;
- provide easy system extensibility (at run time);
- be independent of programming language choices without prescribing system design or mandating certain components;
- allow for efficient and low-overhead communication; and
- be robust against partial system failures.

The Mbus infrastructure logically consists of two components: a transport infrastructure that provides message transfer, addressing, basic bootstrap and awareness mechanisms; and a semantic layer that is defined through the abstract services of the communicating modules. The following sections describe these in more detail.

TRANSPORT SERVICES

As its basic service, the Message Bus provides local (intra-system) exchange of messages between components that attach to the Mbus (Mbus entities). A system is typically expected to comprise exactly one host, but a may also extend across a network link and include several hosts sharing the tasks; a local system does not extend beyond a single link – the Mbus is *not* intended or designed for use as a wide-area conference control protocol. Wide-area conference (and device) control has significantly different requirements regarding trust-models and scalability and must deal with different problems regarding reliability and message transport delay.

The message transport service provides group- and point-to-point-communication. It does not offer all the features that are frequently found in protocols for coordinating distributed applications in general such as guaranteeing a global or causal ordering of delivered messages.

Given these deliberate restrictions in functionality it is important to note that the Mbus implies a component model where communication between components can be realized without services from a full-featured infrastructure for distributed applications.

Message exchange takes place using UDP (User Datagram Protocol): datagrams are sent either via unicast to a single entity or multicast to a host- or link-local group. For unicast communications, message delivery is optionally performed reliably, with acknowledgements and retransmissions taking place at the Mbus transport layer. All multicast communication is performed unreliably.

For unicast communications, message delivery is optionally performed reliably, with acknowledgements and retransmissions taking place at the Mbus transport layer. Point-to-point and multicast communication is in general not distinguished by the transmission mechanism employed at the IP layer but rather by the qualification of the Mbus destination address. There is however the possibility to use IP-unicast for messages that are directed to a single receiver. (See section *Entity Awareness*.)

Mbus Addressing

A key concept of the Mbus is its flexible and extensible addressing scheme. Mbus entities are identified by n-tuples with each component of the tuple represented as an attribute-value pair. The addressing scheme for conferencing applications includes address elements like conference (“conf”), media (“media”), module type (“module”), application name (“app”), and application identifier (“id”). For example:

```
(media:audio module:engine app:rat id:1035-
0@134.102.218.67)
```

Each Mbus entity responds to messages addressed to any subset of its own address. For example, the entity with the address illustrated above will respond to messages providing the following target addresses:

```
(media:audio module:engine app:rat id:1035-
0@134.102.218.67)
(app:rat id:1035-0@134.102.218.67)
(media:audio)
()
```

A fully qualified address (with all components of the tuple present) indicates a unicast Mbus address. Messages with incomplete addresses have the potential to be received by multiple entities.

Entity Awareness

An entity on the Message Bus can learn the existence of other entities (and their complete addresses) by listening to the self-announcement messages that all entities send periodically. The rate at which these periodic heartbeat messages are sent is adapted dynamically depending on the number of entities in a Mbus session, thus allowing the Mbus to scale to larger groups of entities.

This mechanism is used to locate entities and to monitor their liveness during a session but also to build a complete set of the addresses of all available entities – Mbus layer addresses and transport layer addresses. The latter information can be used for optimization strategies, e.g. for deciding whether a message can be sent via IP-unicast.

Based upon these awareness functions, a bootstrap procedure is defined that allows entities to determine whether all other entities they depend on are present (without bearing the risk of deadlocks).

Group Communication Mechanisms

Each entity has a unique Mbus address that can be used to identify it and that is composed of any number of named address elements. The types and values of address elements are application-specific and can be used to aggregate entities into address groups.

Address element names may be associated with semantics: by providing certain address elements, entities can signal the type of service functionality they are able to supply. The Mbus specification itself does not impose any restrictions on application specific address elements. The semantics are provided by application specific profiles. For example, the address element set for conferencing applications defines elements for distinguishing entities by the media type that is assigned to them allowing a local conference controller to address one or more audio, video and other media tools.

A message that is to be sent via the Mbus has a target address – at the application level – that is used to determine how to deliver the respective message. This allows for *subject-based addressing*-like message delivery semantics and different communication models represented by the different group addressing features:

Broadcast

When a target address list is empty the message is broadcast to all entities on the Mbus.

Unicast

A message that contains a target address that is a unique Mbus address of another entity will be processed by this entity only. The Mbus defines mechanisms allowing such messages to be sent directly via unicast to the specific entity.

Multicast

As mentioned above target addresses can be used to identify groups on a per-message basis. All entities that match the given target address will receive and process corresponding messages: In situations where a certain module requires a specific service functionality, that can be provided by more than one other module, it can use a multicast address specifying the group of service providers to locate the desired entity. This may facilitate the implementation of service clients significantly: addresses of service providers do not need to be hardcoded and the communication model can accommodate many different specific scenarios regardless of the number of potential service providers.

It is not necessary to know the exact addresses of all potential receivers of a message. Instead a sufficiently unambiguous address list can be used. For example, in order to reach all audio engines in a session the address list `(media:audio module:engine)` might be appropriate.

Mbus Messages

Mbus messages are text-encoded and consist of a header with protool information and a payload section that can carry several textual commands with parameter lists. Command names are hierarchical and a set of basic data types for command parameters is defined, including numeric types, character strings, lists and opaque data.

Message authentication and encryption are supported as inherent transport features to prevent malicious attacks and to provide privacy for the communication within an Mbus domain. This allows the Mbus to accommodate multiple sessions of a user per host (including cross-session coordination) as well as any number of users on the same host or link (preventing accidental cross-user interaction).

HIGHER LAYER SERVICES

Since the specification of the transport mechanisms does not include application-specific Mbus commands, these are expected to be defined in independent profiles. Mbus profiles normally do not only consist of a set of command definitions but also of the specification of the use of Mbus address elements, message interaction schemes and the description of message semantics.

In addition to the basic transport mechanisms, interaction schemes on a higher level of abstraction have been defined to support the creation of useful profiles. This includes abstractions like remote procedure call, event notification and publish/subscribe mechanisms that provide more structured communication schemes while still allowing for simple and straightforward messaging.

Furthermore, different types of control-relations between peers are provided. This allows for scenarios where entities need a dedicated control relation to another entity in order to operate properly. Again, mechanisms are provisioned that allow both the simple ad-hoc communication scenario without manual configuration and explicit control relations.

These higher layer services are defined in a description of guidelines for specifying Mbus profiles. One of the features of these services is the notion of a *default target address* that can be assigned to event notification commands. In an application without a tight coupling between entities the default target address is used to notify other entities of certain events. Interested application modules have to know this address and configure their Mbus interface accordingly in order to be able to receive notifications. By the means of entering an explicit control relation with an event generating entity, a controlling entity can *redirect* the target address. Depending on the type of the selected control relation, this also affects the right to send commands to the entity and may exclude other entities from concurrently controlling it.

These mechanisms allow for re-using the same application module in different scenarios with different requirements for the strictness of the control relations: In an ad-hoc Mbus session, say for prototyping a distributed system, an entity can accept commands from arbitrary entities and use a default target address for event notifications. In a tightly coupled setting, different levels of control can be employed in order to achieve a more formalized and structured communication style.

EXAMPLES

Reliable Audio Tool (RAT)

RAT [25] is an example for a media tool that is itself decomposed into several funtional units:

- a media-engine that is responsible for media transport and rendering;
- a user-interface-component that provides a graphical user interface; and
- a controller that coordinates the two other components.

Usually RAT is invoked by starting the controller that in turn starts new processes for the media-engine and the user-interface. On startup, the three entities synchronize by using the corresponding Mbus mechanisms and the audio-engine starts to report its capabilities and its status to the user-interface that uses this information to build appropriate menus etc. for the options that can be configured.

After this setup-phase, the user interface reacts to user-interaction by sending appropriate Mbus commands to the audio-engine and the audio-engine reports events to the user-interface.

The components that constitute the RAT application are implemented as independent programs and can be re-used for other applications. For example the media-engine can be used in integrated conferencing applications, where the RAT-user-interface is replaced by another, probably application-specific user-interface.

Another example is an IP-Telephony-gateway that employs the RAT-media-engine for receiving and sending audio.

These examples highlight the way that components of originally standalone applications can be re-used in new scenarios without having to re-configure or even re-compile them.

Multimedia Conferencing Application

The following example illustrates how the Mbus can be used to extend the functionality of a stand-alone IP-Telephony system to an audio/video conferencing application by integrating the telephony box and *re-using* its functional components as Mbus entities in a new application.

Figure1 shows a Mbus based IP-Phone that is used as an audio component in an audio/video conferencing system: The additional components on the left-hand side run on a co-located workstation and extend the functionality of the phone system by taking-over the control of some sub-components. The "Endpoint Controller" registers with the "Phone Controller" and coordinates the new application. The Mbus communication model allows for plugging-in the new components without prior re-configuration of the system. For example the events that are generated by the audio engine and visualized by the phone display can be received and processed by the new GUI running on the workstation. This can be accomplished by the Mbus group communication facilities and/or by registering the GUI as an additional controller at the audio engine.

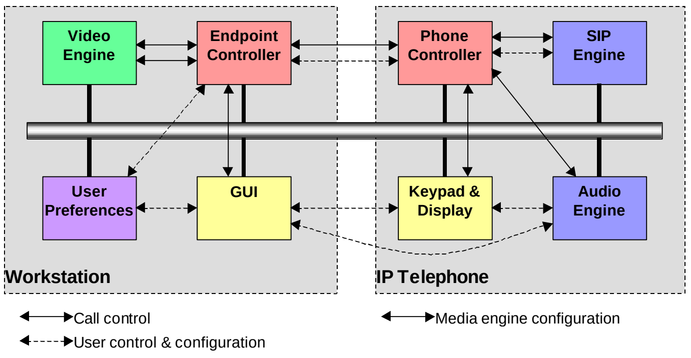

Figure1: Extending an Mbus-based IP-Telephony system

It can be noticed that decomposing applications using the Mbus mechanisms yields a certain level of *tailorability* of the final applications as well as the individual components that is achieved by

1. decomposing the application into independent, re-usable funtional units;
2. specifying Mbus interfaces for the components;
3. using the Mbus group communication mechanisms for potential extensions; and
4. deploying the higher level services in order to allow for structured extensibility and possible transition between uncoordinated and controlled operation modes.

Example Message

Figure 2 shows an example Mbus message that is sent unreliably to the group of user interface modules in a components-based application.

```
mbus/1.0 1 342545638 U (media:audio \
module:engine app:rat id:1035-      \
0@134.102.218.67) (module:ui) ()
rtp.source.name ("54e3" "unknown")
rtp.source.codec ("54e3" "GSM")
rtp.source.active ("54e3" 1000)
```

Figure 2: Sample Mbus Message

RELATED WORK

Concepts similar to those of the Mbus have been proposed on a number of occasions in the past. In this section, we review those proposals to show how they differ from our proposal, and highlight the relative advantages and disadvantages of each.

LBL Conference Bus

The most widely deployed local conference coordination protocol in the Mbone community is the LBL conference bus, used by the media tools vat [9] and vic [11] for arbitration of access to hardware devices (e.g.: framegrabber, audio hardware), voice switching of video, and other simple inter-process coordination.

The LBL conference bus protocol is simple, yet effective for the limited use to which it is put. Messages have a type field, which does not appear to be used, and a process-ID as the destination address. A single, free form, text command sent in each message (and interpreted directly as a Tcl command string by the recipient).

Messages on the LBL conference bus are sent using IP multicast, with zero time-to-live (restricting the multicast to a single host). Each process listens on two multicast groups, one for messages affecting all sessions, and one specific to the individual session. There is no reliability included in the LBL conference bus protocol.

The key features of the LBL conference bus are the use of IP multicast for local communication, and the use of multiple communication channels.

Conference Control Channel Protocol

The Conference Control Channel Protocol (CCCP) [7] is more ambitious in its scope, and is "intended for controlling conferences ranging from small, tightly coupled meetings, to extremely large loosely coupled seminars".

CCCP does not only provide a local coordination channel but is also targeted at conference wide control. Since IP-Multicast is used at the transport layer for both local and conference wide communication, CCCP has to address the problem of reliable multicast transmission. It therefore introduces different reliability and message ordering classes.

The addressing scheme provides address tuples (of fixed cardinality) and the possibility of "wildcarding" certain address elements in destination addresses. Although the set of address elements is fixed, CCCP still provides a mechanism for extensibility: the type element of an address can be hierarchical, so that the type of an CCCP entity can be very specific while the entity can be still be addressed by the major type prefix.

CCCP messages have a plain text addressing scheme and a plain text, free format text payload that usually consists of a function name and parameters.

Pattern Matching Multicast

The MInT toolset [19] uses a communication protocol named "pattern matched multicast" (PMM) [16] for coordination of media agents by a local controller. PMM supports three different transport methods: a centralized architecture with a message replicator that distributes messages to registered entities via UDP/IP or TCP/IP and a de-centralized model using host-local IP-Multicast without a central message replicator. The replicator process can also act as a message filter by sending certain messages only to entities that are interested.

PMM messages simply consist of the name of an object and a operation to be applied to the object. Object names are hierarchical descriptors containing identifiers for the conference, media, media instance and (optionally) the media session member. Operations are textual commands, "directly interpretable by a standard Tcl interpreter".

Applications can register for messages with the central replicator by sending a message with a wildcarded object and operation name.

The receiver directed filtering is only available in the presence of a central replicator.

Jini

Jini [21] is a Java based architecture enabling users to share services and resources over a network within the scope of a workgroup. Jini provides the concept of federating services and clients of services.

Services can be located using lookup services that serve as repositories of services – objects in the Java programming language that can be downloaded to clients during a lookup operation and act as proxy stubs to the service module. Services deploy a location server discovery and join protocol in order to become part of a Jini system. The discovery protocol relies on IP-Multicast. It is possible to distinguish types of services by a group identifier within in a lookup service. These identifiers are arbitrary strings that services can choose upon registering with a lookup service.

Jini uses Java remote method invocation (RMI) to implement the basic inter-services protocols but service proxies can employ other protocols to communicate with service entities.

Jini provides the model of leasing services: Resources are allocated using a renewable, duration-based scheme. Furthermore Jini offers an event and notification interface and a transaction interface that enables Jini application to coordinate state changes.

The main focus of the Jini architecture is on resource and service localization and not on the area of client-service-interaction. Due to the use of RMI Jini is a Java-only solution that cannot interoperate with non-Java components.

CORBA and DCOM

CORBA [12] and DCOM [2] provide mechanisms for transparent invocation and accessing of remote distributed objects. Both require interface definitions and provide the usual object oriented features such as encapsulation, inheritance, and polymorphism for remote objects. Different techniques are used to achieve programming language and platform independence: CORBA provides abstract class definitions that are mapped to representations in concrete programming languages while DCOM, relying on the Component Object Model (COM), provides a per-platform binary standard.

The applications that both CORBA and DCOM are targeted at usually require a tighter coupling of components and fall into a different category than the Mbus-based component system we have presented.

Both approaches offer elaborate solutions for the creation of type-safe distributed features that go beyond the services of a message-oriented coordination infrastructure. By requiring a significant amount of static information and/or further management components like naming services, interface repositories and Object Request Brokers (respectively Service Control Managers) both techniques do not appear adequate for simple, flexible ad-hoc cooperation. Besides, they are both not universally available and particularly easy to implement for small, limited platforms.

CONCLUSIONS

It has been shown that the Mbus is a simple lightweight message-oriented coordination protocol that is useful for decomposing applications into components that are distributed across a local network link. Although the Mbus does not provide complete support for distributed applications it can be used in systems, where components are relative independent and can cope with the message oriented coordination services.

The group communication mechanisms add a valuable degree of flexiblity that allows to plug-in components into a running system in a uncomplicated way. This result in tailorable systems that are open for future extensions.

The higher layer services allow for structured communication interactions while still preserving the simple and flexible message exchange paradigm.

Given these benefits and considering the platform-neutrality the Mbus is an interesting alternative to the presented existing solutions.

FURTHER INFORMATION

Detailed information and implementations for different platforms are available at http://www.mbus.org/.

REFERENCES

1. Braden, R., Zhang., L., Berson, S., Herzog, S., and Jamin, S., Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification, RFC 2205, September 1997
2. Brown, N., Kindel, C., Distributed Component Object Model Protocol -- DCOM/1.0, draft-brown-dcom-v1-spec-03.txt, January 1998
3. Handley, M., Crowcroft, J., Bormann, C., and Ott J. The Internet Multimedia Conferencing Architecture. draft-ietf-mmusic-confarch-03.txt. July 2000
4. Handley, M., Schulzrinne, H., Schooler, E., and Rosenberg, J. SIP: Session Initiation Protocol. RFC 2543. March 1999
5. Handley, M., and Jacobsen, V. SDP: Session Description Protocol. RFC 2327. April 1998
6. Handley, M., Perkins, C., and Whelan, E., Session Announcement Protocol, draft-ietf-mmusic-sap-v2-06.txt, March 2000
7. Handley, M., and Wakeman, J., CCCP: Conference Control Channel Protocol – A scalable base for building conference control applications, 1995
8. Jacobsen, V. Multimedia conferencing on the Internet. SIGCOMM'94 Tutorial. August 1994
9. Jacobsen, V., McCanne, S., vat – LBNL Audio Conferencing Tool, http://www-nrg.ee.lbl.gov/vat/
10. Kouvelas, I., Hardman, V. and Watson, A., Lip Synchronisation for use over the Internet: Analysis and Implementation, In GLOBECOMM'96, 1996
11. McCanne, S., and Jacobsen, V., Vic: A flexible Framework for Packet Video, In ACM Multimedia'95, pp. 511-522, November 1995
12. Object Management Group, A Discussion of the Object Management Architecture, http://www.omg.org/technology/documents/formal/object_management_architecture.htm, January 1997
13. Ott, J., Bormann, C., and Bormann, U., Service Interoperability in Teleconferences, published as Common Functional Specification L313 of the RACE Industrial Consortium, February 1994
14. Ott, J., Perkins, C., and Kutscher, D., A Message Bus for Local Coordination. draft-ietf-mmusic-mbus-transport-02.txt. July 2000
15. Ott, J., Perkins, C., and Kutscher, D., The Message Bus: Messages and Procedures, draft-ietf-mmusic-mbus-semantics-00.txt, 1999
16. Schulzrinne, H., Dynamic Configuration of Conferencing Applications using Pattern-Matching Multicast, In NOSSDAV'95 (5th International Workshop on Networks and Operating System Support for Digital Audio and Video), 1995
17. Schulzrinne, H., Casner, S., Frederick R., and Jacobsen, V. RTP: A Transport Protocol for Real-Time Applications. RFC 1889. January 1996
18. Schulzrinne, H., Rao, A., and Lanphier, R., Real Time Streaming Protocol (RTSP). RFC 2326. April 1998
19. Sisalem, D., and Schulzrinne, H., The Multimedia Internet Terminal, Journal of Telecommunication Systems, vol. 9, no. 3, pp. 423-444, September 1998
20. Stiemerling, O., Component-based Tailorability, Dissertation, Bonn, May 2000
21. Sun Microsystems, Jini Specification, http://www.sun.com/jini/specs/jini1_1spec.html, 2000
22. Szyperski, C., Component Software – Beyond Object-Oriented Programming, Addison-Wesley, 1999
23. ter Hofte, H., Working Apart Together – Foundations for Component Groupware, Telematica Instituut, Enschede, 1998

24. 3Com, 3Com SIP PDA Phone Control Application, http://www.3com.com/products/sip/sip_solutnpda.html, 2000

25. University College London, Robust Audio Tool, http://www-mice.cs.ucl.ac.uk/multimedia/software/rat/, 2000