

CORBA Based Infrastructures for Real Time Cooperation

Wolfgang Becker, Pete Bonham, Jeroen DeBorst, Stephan Kauss
Tandem Computers Europe Inc., Max-Planck-Str. 36, D-61381 Friedrichsdorf, Germany
Phone: +49 6172 734348, Fax: +49 6172 74655, becker_wolfgang@tandem.com, <http://www.tektonic.de>
and

University of Stuttgart, Institute of Parallel and Distributed High Performance Systems (IPVR)
Breitwiesenstr. 20 - 22, D-70565 Stuttgart, Germany, Phone +49 711 7816 411, Fax +49 711 7816 424
wbecker@informatik.uni-stuttgart.de, <http://www.informatik.uni-stuttgart.de>

Abstract

This paper introduces the *real time Matrix* (rM) being developed by Tektonic, a Tandem Computers business. rM is a framework of servers and client components yielding large scale, heterogeneous cooperative applications. rM defines an architecture for the components with function and interaction specification, featuring fast direct memory access communication, CORBA object interactions up to Java Beans event communication protocols for complex CSCW clients.

Besides proposing a suitable architecture, this paper elaborates possible extensions to current CORBA technology for serving real time cooperative environments.

1 The Real Time Matrix: A CORBA Based Infrastructure for Synchronous Cooperation

Real time cooperating applications like chat, games, audio-video conferencing, shared whiteboard / editing / publishing, calendars and remote presentations became feasible with the PC and workstation networks, and became especially interesting due to the establishment of fast networks and powerful graphical front-end hosts. While various specific solutions have been developed, the broad availability and acceptance for larger user communities has not yet been achieved. The economic reasons for this are the rigid and expensive pricing policies of the telecommunication companies and the necessary local hardware extensions like sound and video processing devices and direct fast network access for each client host. The technical problems are the still poor and unreliable bandwidth of the larger packet switching networks (LAN and WAN), the inflexible traffic sharing capabilities of fixed reserved bandwidth networks, and the establishment of efficient and flexible multicasting.

An important challenge however, to establish CSCW environments in a broader user community, is the development of higher level, modern interfaces and abstractions for the cooperation of such shared applications which define different abstraction levels and communication protocols,

and architectural slots for different CSCW components like text chat, audio chat, video, tele pointer and whiteboard to be arbitrarily composed into larger environments -adjusted for and manageable by- certain end users.

In order to achieve this, interfaces must allow to exploit very efficient communication (transport) technology, different object / state sharing and news / update distribution policies, but also convenient, portable and interoperable technology that allows for rapid application development and safe investment due to standards compliance.

The Tandem Tektonic *real time Matrix* enables rapid development of portable, efficient, extensible and robust synchronously cooperating applications and virtual environments, including cooperative work / workflow environments, teleconferencing / chat applications and multi player games (distributed interactive simulation).

The main design goals for rM are the following:

- Low latency, real time - but also reliable - exchange of information and updates between clients, including priority management and bandwidth control.
- Scalability to large sessions, including many clients and heavy interaction message traffic.
- Transparent geographical distribution with utilization of locality and local autonomy.
- Portability of application code and message formats to all relevant system platforms.
- Interoperability of cooperative applications between different programming languages and platforms.
- Extensibility of applications and protocols, and easy integration into other applications.
- Robustness, high availability and fault tolerance of central services and cooperation infrastructure.

Developed considering these design goals, the rM approach includes the following distinguishing features:

- In the center of rM, there is logically one central, general purpose message distribution switch, serving several sessions with different participating client

applications. It offers transport and application protocol independent real time or reliable message delivery. While current high end multimedia cooperative applications still require handcrafted system and network optimized development, rM is suitable for applications in the domains of chat, distributed simulation and gaming, audio / video conferencing, remote presentation, joint document editing and related areas.

- rM uses the standardized, portable, efficient and interoperable communication subsystem *Krypton*. *Krypton* is a CORBA [CORB95] object request broker (ORB), which allows objects to call each other by IDL defined interfaces.

This ORB is available for Windows NT, various Unix derivatives and Tandem's NonStop Kernel operating system, and objects can communicate across the different systems. Thus, it largely enhances code portability, run-time interoperability and facilitates software development.

Additionally to other ORB implementations, Tandem Tektonic's *Krypton* ORB allows for efficient asynchronous calls without requiring multithreaded programs.

A third differentiating feature of the *Krypton* ORB is that it allows arbitrary distribution of the communicating objects between hosts, within hosts and even within the same process address space without recompiling or relinking any code.

The transport medium which the *Krypton* ORB uses depends on where the objects reside. If they are configured to run within the same address space, then object calls are local calls, otherwise within NonStop Kernel systems the fast Guardian messages are used. In the general case, *Krypton* uses TCP/IP. On systems where Tandem's fast direct memory access (DMA) based ServerNet is available, *Krypton* can also use this (via Tektonic's *Titanium* transport) to get maximum performance for local area networks (LAN), because CORBA based communication uses the preferred request - response communication profile.

The *Krypton* ORB provides application programming interfaces for the C, C++, Cobol and Java languages.

Krypton is a distributed ORB, i.e. does not require any central broker instances. However, if explicit forwarding, inter-ORB transmission or trading for issues like load balancing is desired, *Krypton* supports tradable object references that communicate through central object request broker instances.

The high level object call interface allows for substantial performance optimization below it and plugin of new technology without changing the application soft-

ware. It therefore saves investment and can profit from future performance enhancements.

- The rM server reference implementation *TekRelay* is available for most currently important operating systems, including UNIX derivatives, Windows NT and Tandem's NonStop Kernel.
- rM servers, while representing one central server to their clients, actually build a distributed, scalable and fault tolerant network, allowing local administration and minimizing central bottleneck behavior.
- rM provides an architectural slot for cooperation control components. Cooperation control components facilitate the composition of integrated cooperative work (CSCW) environments.
- rM provides an architectural slot for conference control components. Conference control components allow for reuse and replacement of conference control functionality within integrated cooperative work environments.
- For platform independent, secure, installation free Java based development, rM offers Java Bean based intracient interfaces between CSCW components and cooperation control and conference control.
- The *Krypton* build and run-time environment is coupled with a generic, platform independent, object based graphical management & configuration environment.

2 The Real Time Matrix (rM) Architecture

rM is a framework consisting of servers (relays), services and clients. This framework, and the internal structure of servers and of front end clients build the rM architecture. Figure 1 outlines the overall rM architecture. In general it is client - server structured, but the servers build a tree structured framework and the front end clients have an internal structure that decouples central and application specific functionality.

The following presents the functionality of the components and the communication protocols between them. After that the server architecture is further detailed.

Usually, the client components are composed using an application builder like Delphi, Visual Basic or Java Studio, and communicate through the builder environment's event or message mechanism. This paper, however, concentrates on functionality and communication logic, and just mentions an implementation based on CORBA components and Java Beans [Hami97].

2.1 Functionality of the Main rM Components

The main rM components, outlined in figure , have following functionality:

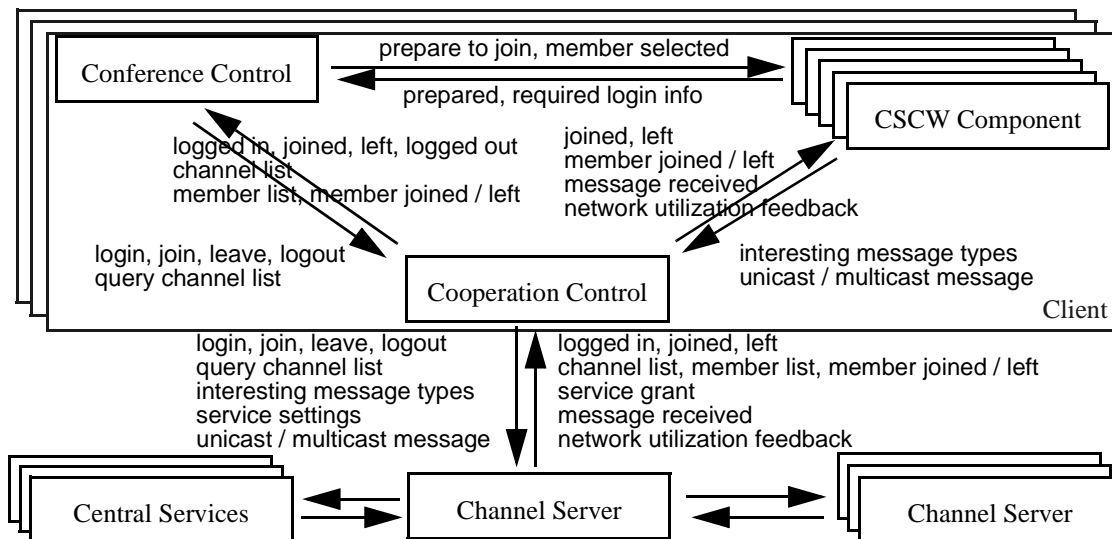


Figure 1: Overall architecture of the real time Matrix.

- **Conference Control.** Manages the overall application user's session participation and the CSCW components which the application consists of. It provides a user interface for application start-up, overall state control, channel list and selection. Each client should have exactly one conference control component.
- **Cooperation Control.** First, the cooperation control is a proxy for the CSCW components and the conference control, essentially providing channel server functionality. So, the client components can communicate locally to the server proxy and do not need to talk the full client - server protocol to a channel server, including login, session join and the like. Second, cooperation control represents a single client towards the Channel Server, so that a server does not have to manage each CSCW component as a separate client.
- **CSCW Component.** Provides an application which is used together with other users (e.g. chat tool, whiteboard, game, audio/video conference, remote presentation). Each user has an appropriate instance of a CSCW component on his local machine running. Usually, a cooperative work environment consists of several components which are collected at each client. A CSCW component provides the user interface and manages the information distribution and coherence of shared state (together with the Cooperation Control and Channel Servers). Its cooperation task is to integrate arriving information and state updates from the Cooperation Control into the local state, and to tell the Cooperation Control about new local informations or state changes.
- **Channel Server.** Manages users, sessions, session participants (clients) and message distribution between

clients. Each channel server pretends to be a global server, i.e. knows about and deals with all sessions and clients and messages, although actually a network of channel servers exists. An rM channel server deals with following entities:

- **Users.** A channel server maintains a list of currently registered users, including username, security credentials, capabilities, currently active clients, etc. The rM session space is logically tree structured to facilitate the search for certain sessions or topics.
- **Sessions.** A channel server maintains a list of currently active sessions, including session name, session moderating client, participating clients and their roles, etc.
- **Clients.** A channel server maintains a list of currently active clients, including name, login state, user capabilities, network address, session participation, etc.
- **Messages.** A channel server distributes messages with arbitrary content from clients to other clients and messages from clients to all participants (clients) of a session. The channel server can store or discard messages which cannot be delivered (not within the given time limit or not at all), and can deliver messages according to message priorities. Overall, a channel server maintains a list of partially delivered messages, including information like message id, message type, sender client, receiver client, priority, delivery state, etc.

Channel servers forward messages transparently through the server network. Clients do not have to perform server addressing and routing. However, to

enhance message delivery performance and allow for efficient session, client and message management, clients can also include routing information for session or client addressing to save broadcasts or global queries.

The servers view their neighbor servers much like clients, with the exception that neighbor servers participate in sessions with a special 'server' role, so that they are not displayed at clients as members and are not addressable like session members.

A server does not understand application message contents. It just looks at a short header information passed along with each message and distinguishes the applications and their message types by simple flat enumerations, for filtering and interoperability checking.

A server is able to manage several clients and several sessions concurrently. Servers must not be monolithic but operate in a distributed way, i.e. potentially consist of a set of cooperating servers.

Server functionality cannot be enhanced arbitrarily. As the servers are structured (see below) and built from clean components, each component can be replaced by another implementation, but the interfaces remain. Server extensions that do not fit into these interfaces can usually be realized as central services.

Administrative tasks, like changing user accounts, can be performed through management clients, using the usual rM client - server interface. Additionally, servers can be managed through separate tools. However, in rM the use of Tandem Tektonic's *Amethyst* object management is recommended for retrieving and setting server properties.

- **Central Services.** From an rM server point of view, a Central Service is just another client. However, it is well-known by some clients and is used by these clients. Usually, central services provide functions like persistent file storage, talk history or background scenery simulation. A central service can serve a session, a server or the rM universe globally.

Protocol converters and gateways to other, legacy applications, clients, relays or protocols, fit into the architecture by connecting a coupler component in the role of either an rM client, rM central service or rM server to a channel server. The coupler has to perform the required entity, protocol and message transformation.

2.2 Architecture and Concepts of rM Servers

The internal structure of a Channel Server is outlined in the figure below: rM server control flow is driven mainly by a certain number of outstanding message receive operations from connected clients and servers, as well as by

completion of message delivery operations to clients. Additionally, timers drive actions like timeouts on message delivery, cleanup procedures and feedback information distribution to clients and servers.

Asynchronous processing in CORBA objects is usually driven by the CORBA subsystem, so that the object functions are invoked appropriately. Note that real, heavily utilized servers need to dynamically control the amount of incoming calls which requires extensions to most of the available CORBA implementations (see section 2.4).

The main functionality of an rM server is message distribution. Messages are received from clients and servers, queued and distributed to other clients and servers. On transport level, rM servers can exploit unicast as well as multicast transport protocols. The recommended and most portable / interoperable transport however, are CORBA object calls which currently lack efficient multicast concepts. So, appropriate extensions to Tandem's Object Request Broker for efficient and flexible multicasting are being designed (see section 2.4).

Incoming messages are handed from the receiving transport to the respective transport adapters. Because asynchronous actions in rM servers are top down driven, i.e. transport adapters have initiated the message receive operations, no clumsy parsing or protocol adapter selection is needed but a context handle is passed with the operation, and the amount and frequency of incoming messages can be controlled to keep the server in the optimum level of concurrence and parallelism. Therefore, CORBA object calls have to be extended so that the server can control and limit the incoming connections and calls from clients in a suitable way. Usually, clients pass a message by one object call, which immediately is responded with an acknowledgement that the message is recognized and queued. The response to the call is usually not delayed until the message is completely delivered to the targets.

Protocol adapters pass the message on to the central dispatching component, some opaque content along with certain attributes like source and destination client identification, priority, urgency, required reliability as well as application and message type identifiers.

The message dispatcher sorts the message by priority (and ordering constraints - for real time streams) into a central queue. Priority can be specified in a time dependent fashion and grows linearly with the time. This is the unified rM concept for specifying both importance and urgency for message delivery; It has been developed and successfully evaluated in [Beck95]. If the queue cannot be further expanded to keep the message, the message with lowest priority in the queue is removed instead, and handed to the congestion handler. Note that an rM server may start only

as many concurrent receive operations as it can store the messages either in the central queue or at the congestion handler. A central queue is absolutely necessary to control message priorities and incoming / outgoing bandwidth utilization.

To deal with differing network bandwidth, responsiveness of clients and rise of messages to certain clients and neighbor servers, per-client message queues seem appropriate, or at least the individual delivery states to the target clients should be maintained. rM servers do not maintain per-receiver queues for following four reasons:

- First, a sole central queue enables efficient storage and management of the messages without replication.
- Second, it avoids to decide when to move messages from the central into the receiver queues, and so avoids a second, more complicated overflow handling policy.
- Third, it allows to maintain the state of a message just once, which is important e.g. for error reporting to the sender client, if a message could only be delivered to part of the targets, or partially expired. Without central management per message, situations like these can cause several error messages back to the sender per message, aggravating the congestion situation.
- Fourth, if the respective sending transport component allows for efficient multicasting, then a message can be delivered to a multicast group of clients as a whole. With per-client queuing, multicast group queues would be necessary, and exceptions, e.g. if a message is distributed to not a complete multicast group, would require per-client and per-group queues.

rM servers do not resynchronize messages within/between related streams (e.g. audio message streams of some application) or artificially delay them. Instead, they deliver as fast as possible within the constraints of bandwidth control, priorities and sequence order restrictions.

Passing on queued messages to a protocol adapter for delivery is triggered if a new message was enqueued, or if an outgoing transport reports successful delivery of messages, or in consequence of timeout cleanup actions. In the first case, the server checks whether the new message may be handed to protocol adapters to be sent to some of the target clients, i.e. if some outgoing transports can send further messages and if there are no other messages in the queue that must be sent before the new one (due to order or priority constraints). In the second case, the server checks whether there are enqueued messages that can be handed to the transport now without blocking.

The protocol adapters convert an outgoing message to the appropriate protocol and hand it over to the respective transport component. Which outgoing protocol adapter and transport is to be used, is specified per client (or multicast group) and is determined when the client connects to the server, so the selection of the adapter and transport is efficient. rM recommends CORBA object calls as outgoing transport components, for the same reasons like mentioned above for the message receiving transport components. Necessary extensions are elaborated in section 2.4. Usually, CORBA based clients issue object calls to the server requesting a message, and the transport components in the server reply to these calls.

The congestion handler is activated in different situations. First, if a new message arrived and the queue is full, it decides which message to discard, and is responsible for notifying the sender if necessary. Second, it observes the bandwidth usage of incoming and outgoing messages per client. If incoming messages exceed the granted bandwidth of a client, the congestion handler advises the respective protocol adapter to reduce the client's message traffic. The protocol adapter can reduce or delay receive operations (source quench, see section 2.4 for doing this in CORBA environments), or can send an appropriate

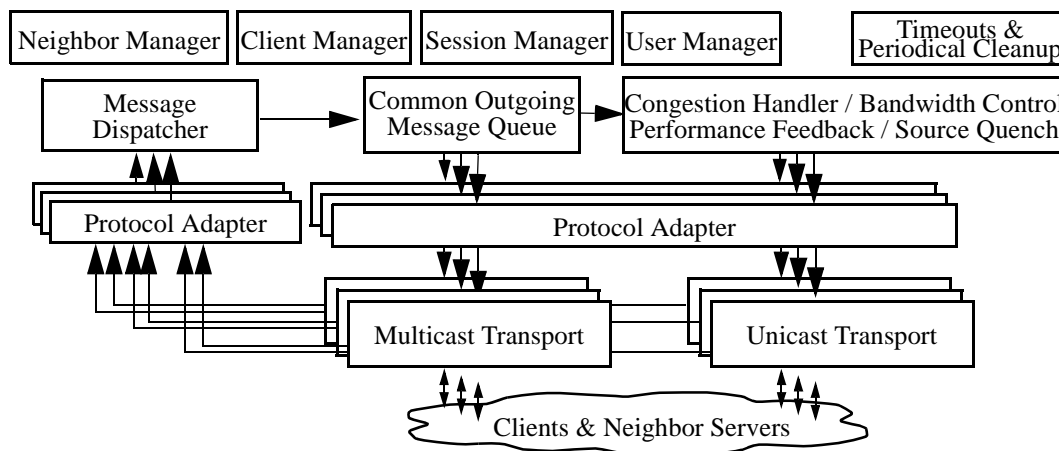


Figure 2: Architecture of a real time Matrix Channel Server.

feedback to this client so that the client can react. If outgoing message traffic exceeds a client's granted bandwidth, then the congestion handler reduces its message priorities in the queue. rM bandwidth control is just an addition to bandwidth control mechanisms in the transport components, if there exist any. So third, the congestion handler can ask the transports about their currently available bandwidth and can ask protocol adapters about the transport addresses which clients use, and can therefrom decide about how much bandwidth to grant to clients which ask for it (fourth service of the congestion handler).

Client, neighbor server, session and user management mainly maintains lists of their objects with appropriate properties. They can be called by all other components in an rM server.

2.3 Scalability and Fault Tolerance

Distributed cooperating servers are a key to increasing both scalability and fault tolerance of large, distributed real time cooperative environments, if designed properly.

In general, there are three design alternatives: A central server approach is easy to implement and maintain and very efficient for small, local sessions, but is a single point of failure and not scalable. A completely distributed approach without servers is complex to implement and to operate. Scalability and fault tolerance depends on the clients and their interaction protocols. rM employs the approach of logically centralized, physically distributed networks of servers.

The rM distributed server concept partitions all kind of state information and responsibility: Client connection management, message dispatching / distribution and session state management. Message traffic (latency and bandwidth) is optimized by tree-like message distribution, where the servers build an arbitrarily configured logical tree. This design allows to scale up a cooperative environment by adding more servers, it can be fit to the underlying network topology and capacity, and it avoids unnecessary central bottlenecks.

The main idea of load distribution between rM servers is to distribute it as much as possible according to the distribution of the clients. Hence, assuming that clients are distributed evenly among the servers guarantees a reasonable load distribution between the servers. However, the distribution of the clients' session participation and the distribution of the session coordination role among the servers, still has a significant influence on the server load distribution.

The main idea of proper network utilization by rM based cooperative environments is to optimally match the server topology to the underlying network topology. Provided

this, the propagation of multicast messages through the logical server tree yields a reasonable compromise between low bandwidth utilization and low latency.

The distribution of session management is designed as follows: For each session, one server is coordinator who knows about the session. The session name contains the routing information to the coordinator server. Therefore existence of sessions may not be known globally, session name uniqueness is easy to control, up-to-date session information is always reachable from everywhere, and no broadcasts are necessary for session management. The session coordinating server may not know of all clients that are currently in the session. But he must at least know about some neighbor servers who have additional information about the session, so that asking them or distributing messages to them is sufficient to collect the complete session state or to reach all clients of the session.

In rM, each server manages those clients who connected to it. It manages their session participation state, registers at the session's coordinator servers as someone who is involved into the session. This helps to distribute the server load and amount of state at the servers according to the client connection distribution.

These concepts realize fixed, partitioned state management. Replication / caching of state among the servers can increase performance and reduce communication in cases where informations are frequently read and rarely updated. Migration of state data among the servers can improve load distribution and increase access locality, but requires location independent state data addressing or appropriate forwarding.

Fault tolerance in general can be increased by replicating state data among distributed servers, and by leaving local autonomy and exploiting local state management (avoid central paths and central state management). The current rM design has no concepts for state replication and automatic failure detection or backup-takeover. It contains no single points of failure but requires availability of all servers in between interacting clients.

On client connection level, rM servers provide following fault tolerance concepts: Servers try to keep the messages for clients that are disconnected from the server or do not respond - due to any reason without explicit notice of leaving - as long as feasible, but then may discard them and forget about the client. The same semantics for handling unavailability and outages applies to inter-server connection. To save persistent state data of clients, central services must be employed. rM servers may or may not store their state data persistently.

2.4 Enhancing CORBA Technology for Real Time Cooperation

Exploiting CORBA [CORB95] object call technology for cooperative environments has several strong advantages over low level message passing protocols like UDP, TCP or proprietary protocols like window system events or certain RPCs. CORBA provides machine and programming language independence (code portability and runtime interoperability). CORBA supplies a high level abstraction for objects invoking other objects by defined interfaces, increasing code development speed, saving investments and containing a large potential for performance optimization and transport replacement by newer technology (in the CORBA run-time system without application changes).

The rM design and prototype implementations revealed that to fit for real time cooperating applications, current CORBA systems need the following enhancements:

- Lightweight one-way calls with certain guarantees are needed. The current state is that ORBs issue them with best-effort / no feedback semantics. What is required, is collective feedback and error reporting.
- The concurrence of incoming calls to an object must be limitable and controllable dynamically to avoid server congestion.
- Multicast support on object call level with specifying target object list or group identifiers is necessary for efficient multicasting.
- Notification and object information about newly calling objects and about objects that no longer exist or are disconnected, is necessary for proper cleanup.
- CORBA run-time systems should not be the only, central scheduling and concurrency control mechanism. Instead, they should interoperate with the applications' own asynchronous operation.
- More efficient transports must be used to realize the CORBA object calls: Within processes, function calls must be used. Between hosts, efficient, DMA based, request - response optimized fast network transports must be used, e.g. Tandem Tektonic's fast *Titanium* transport for Windows NT, Tandem NSK and Unix.

Part of the requirements are already realized in Tandem's *Krypton* ORB, some are faced in real time ORB research projects like TAO [Schm97], some are not yet considered in today's ORBs.

2.5 Components and Visual Development Environments

Time to market is an important success factor for software development. The rM environment therefore provides components and interfaces for rapid client application development that allow for maximum reuse, convenient high level interfaces and maximum platform independence: Java Beans [Hami97] components for CSCW Components, Cooperation Control and Conference control are provided and can be extended or replaced easily, and can be plugged together in a visual application builder. An rM demo application is composed in the Java Studio application builder from rM sample components, namely a Conference Control, a Cooperation Control and CSCW components including a shared whiteboard, shared Web viewer and audio conferencing. While Java Beans communicate via events with each other, the *Krypton* ORB provides language independent intra / inter process communication. Currently, however, Java systems still lack performance and functionality, especially if applets run in Web browsers, but Java and Web technology will have advanced properly until the CSCW applications that are being built today in rM must scale.

3 Related Work: Architectures and Environments

The motivation for building virtual collaborative environments is to overcome space and time limitations and realize fantasies and activities that were impossible previously, and to enable a new level of interaction and live experience in sharing fun, work, ideas and problems more frequently and intensively. Virtual, collaborative environments have existed in several forms for a long time, but their popularity is currently rising due to significant improvements in network connectivity and graphical user interfaces.

Distributed simulation is a large domain where significant work has been done. Distributed interactive flight, battle tank and driving simulation as well as distributed strategic war games and multiuser dungeons are commonly used for games as well as for training purposes. Typically there is a local simulation engine at each participant which simulates and displays the ongoing action of the whole virtual world, and changes to object behavior are distributed by messages to all session members. These applications are usually driven by central servers or by connecting independent, full functional clients. The DIS (distributed interactive simulation of combat) protocol standard was developed and used by the U.S. army for simulation and training [DIS93, USCo95, Stew95, Brut95, Reyn94], defining update message formats events and assuming

fully decentralized local simulations, exchanging updates periodically. With the distributed X-Windows system, several distributed graphical games came up, mainly architected as one server that drives multiple user interfaces. Three dimensional action adventure games like DOOM or flight simulation on PCs are mostly architected as full fledged, communicating clients. The Virtual Reality Modelling Language (VRML) [Bell95] is a proposed standard for exchanging three dimensional scenery descriptions which are displayed at the clients' browsers and in which the users can interactively navigate. It can be viewed by Web browsers if embedded as certain HTTP content type, and recently procedures can be attached to tree nodes which can interact and dynamically modify the scenery tree. All these protocols are defined at a low communication layer and hence their generality / extensibility and portability / interoperability is limited.

Virtual Meetings, like audio / video conferencing or text chat and virtual chatrooms begin to gain interest by real business users. Usually, participants choose their roles and images, walk around and meet and talk via text messages or even talk real audio and see each other's face. In games, participants then cooperate or attack one another in a virtual world, usually consisting of several rooms or scenarios with a server driven background story going on. Usually these applications are driven by central servers. Examples for multiuser dungeons are MOO [Curt93] or NannyMUD &LP, [Bart90] gives an extensive review of existing multiuser dungeons. Popular virtual chatrooms are e.g. Worlds Chat, providing pseudo three dimensional actors walking in a space station and talking by typing text lines, the Electric Minds Chatroom, Microsoft's virtual chatroom or Earthweb's chatroom, a Java based text-only chatroom with one session per server and no admission control. The most popular text based chatroom is the Internet Relay Chat (IRC) [Oika93] which supports text line based communication within sessions. IRC has fixed size queues per client and disconnects clients on queue overflow, it has no message priorities and no delivery guarantee. IRC has a distributed server structure like rM, but replicates information among the servers, and experience showed that the resulting broadcasts severely limit scalability. All those environments lack security concepts which limits their applicability for business purposes. Examples for video conferencing systems are CU-SeeMe, MS-Net Meeting and some MBone tools. The CU-SeeMe system uses a network of distributed relays.

Cooperative Work environments support interactive distributed engineering by joint viewing of complex technical objects, documents and presentations, usually coupled with on-line audio / video communication, in a virtual conference. Applications like shared document viewing,

shared desktop publishing, shared computer display / windows are emerging. [Burg94, Gemm97, Grud94, Palm94, Rein94] give overviews on Computer Supported Cooperative Work (CSCW) environments. Well-known implementations are Intel ProShare which connects two windows applications on MS-Windows event level, Habanero, Promondia [Gall97] and MBone tools [Schr95] which provide a set of CSCW components along with session management but are less general and flexible architected than rM, or Sony's virtual society project, providing on-line video conferencing, avatars in a 3D space and a shared whiteboard.

For tele conferencing, several rather low level standards exist. Higher level standards are e.g. H.323 for video conferencing, or T.120 from the International Telecommunication Union [Bouc97]. T.120 defines an essentially four level architecture: At the bottom there are transport protocols. Above them, a multipoint communication service is defined. On top of this there is a generic conference control. These two levels roughly capture rM Channel Server functionality. The highest level is a collection of domain specific application interaction protocols. Compared to T.120, rM does not define application protocol messages. The rM conference control is a per client coordinator for several media, as opposed to the T.120 central conference control service.

General architectures like rM have been proposed e.g. by [Bent94], who discusses scalability and responsibility distribution issues and focusses on shared viewing of central information.

In depth work has been done recently to support activation of real time operations through CORBA objects calls [Schm97]. The proposed quality of service parameters and ORB's real time call processing will be integrated into rM.

The current technology trends are directed towards more real life interaction, i.e. better user interfaces and animation and better on-line audio and video transmission. Besides that, concepts are being proposed to more easily convert, include and couple existing complex applications towards cooperative use.

4 Conclusion

This paper focused on real time cooperating applications, proposing a message delivery framework and architecture, examined the benefits and elaborated extensions when using CORBA object calls as infrastructure. This rM architecture is a base for building portable (same code for different platforms) and interoperable (components of different languages and systems communicate), scalable (large clients, many sessions and frequent interaction) and extensible (adding functionality and new components)

cooperative environments. However, today it cannot surpass hard wired, carefully low level optimized high volume high end multimedia conferencing tools. A first reference implementation with several small application examples has been achieved recently, but functionality and performance are not yet advanced and stable enough to provide measurement results for CORBA and Java Bean overhead, message traffic profile, network and processor utilization and scalability.

Acknowledgements

This work was partially sponsored by the Baden-Württemberg Research Ministry, Germany. The author would also like to thank his colleagues and managers in the Tandem Tektonic business unit for providing concepts, funding, ideas, implementation and support, especially Harald Sammer, Rob Nagler and Roland Zink.

References

- [Bart90] Bartle, Richard, *Interactive Multi-User Computer Games*, Muse Ltd., Colchester, UK, 12/1990
- [Beck95] W. Becker, Dynamische adaptive Lastbalancierung fuer grosse, heterogen konkurrierende Anwendungen, Ph.D. Thesis, Fakultät für Informatik, University of Stuttgart, 1995
- [Bell95] G. Bell, A. Parisi, M. Pesce, *The Virtual Reality Modeling Language Version 1.0 Specification*, <http://www.sdsc.edu/vrml/>, 5/1995
- [Bent94] R. Bentley, T. Rodden, P. Sawyer, I. Somerville, *Architectural Support for Cooperative Multi-user Interfaces*, IEEE Computer, 5/1994
- [Bouc97] J. Boucher, *T.120 Data Protocols for Multimedia Conferencing*, International Telecommunication Union, Standardisation Sector, Study Group 8, Geneva, 7/1997
- [Brut95] D. Brutzmann, M. Macedonia, M. Zyda, *Inter-network Infrastructural Requirements for Virtual Environments*, Virtual Reality Modeling Language Symposium, San Diego Supercomp. Center, 12/1995
- [Burg94] C. Burger, F. Sembach, *Ein Ueberblick ueber Verfahren zur rechnergestuetzten Kooperation*, Technical report 7/1994, Fakultät fuer Informatik, University of Stuttgart, 7/1994
- [CORB95] OMG, *CORBA Overview*, OMG document, 7/1995
- [Curt93] P. Curtis, D. Nichols, *MUDs Grow Up: Social Virtual Reality in the Real World*, Xerox Parc, Palo Alto, <ftp://parcftp.xerox.com/pub/MOO/papers/MUDsGrowUp.ps>, 1993
- [DIS93] Standard for Information Technology - *Protocols for Distributed Interactive Simulation (DIS) Applications*, version 2.0, Institute for Simulation and Training report IST-CR-93-15, University of Central Florida, Orlando Florida, 5/1993.
- [Gall97] U. Gall, F. Hauck, *Promondia: A Java-Based Framework for Real-Time Group Communication in the Web*, WWW6 Conference, 1997
- [Gemm97] J. Gemmell, G. Bell, *Noncollaborative Telepresentations Come of Age*, Communications of the ACM, Vol. 40, No. 4, 4/1997
- [Grud94] J. Grudin, *Computer-Supported Cooperative Work: History and Focus*, IEEE Computer, 5/1994
- [Hami97] G. Hamilton (Ed.), *JavaBeans*, API Specification, Sun Microsystems, 7/1997
- [Oika93] J. Oikarinen, D. Reed, *Internet Relay Chat Protocol*, Internet RFC #1459, 1993
- [Palm94] J. Palmer, N. Fields, *Computer-Supported Cooperative Work*, IEEE Computer, 5/1994
- [Rein94] W. Reinhard, J. Schweitzer, G. V. Voelksen, M. Weber, *CSCW Tools: Concepts and Architectures*, IEEE Computer, 5/1994
- [Reyn94] P. Reynolds, *DISorientation*, Elecsim Electronic Simulation Conference, Mystech Corp, Virginia, 5/1994
- [Schm97] D. Schmidt, D. Levine, S. Mungee, *The Design of the TAO Real-Time Object Request Broker*, Computer Communications Journal, 1997
- [Schr95] D. Schreiber, *Multicast Routing in the Internet and on the MBone: Past, Present, and Future*, Graduate Course 6.853 Computer Systems, MIT, Cambridge, MA, <http://www.nmis.org/AboutNMIS/Team/DougS/MBone.htm>, 1995
- [Stew95] J. Stewart, W. Smith, K. Wilson, *Raising the Level of Capability for DIS Live Entity Integration*, Proceedings from the 13th Workshop on Standards for the Interoperability of Distributed Simulations, Orlando, FL, 9/1995
- [USCo95] U.S. Congress, *Distributed Interactive Simulation of Combat*, U.S. Congress Office of Technology Assessment, OTA-BP-ISS-151, Washington, 9/1995