Inter-Process Communication Lessons for Distributed Multimedia Conference Control: A Case Study

Jon Crowcroft

February 17, 1997

Abstract

The CAR Conferencing System permits the control of multiple programs transmitting media streams from different sites. It was built using the ANSA Remote Procedure Call (RPC), and has been ported to use Sun RPC. The lessons learned during implementation and porting center on the requirements for a wider variety of Inter-Process Communications mechanisms between the components that control the system, rather than on the protocols to deliver media between sites. In particular, choosing RPC as the basic building block limited the robustness of the system.

Keywords: Conferencing, Distributed Programming, RPC

1 Introduction

The CAR Conferencing System is a distributed multimedia conference control system. Its principle function is to provide control of programs that transmit one or more media from one or more sites[Handley and Wilbur]. It was built originally using the ANSA [Andrew Herbert] Remote Procedure Call (RPC) toolkit[Birrell and Nelson 84], and has been ported to use Sun RPC system[Sun 86]. Through the evolution of this system, we learned that a rich variety of Interprocess Communications (IPC) mechanisms is required to communicate between different components that control distributed conferencing systems.

We typically find unidirectional (possibly multicast) streams of data between the different application media sources and sinks. For the purposes of control, single messages usually suffice, but these can be directed in a number of different ways, including one-to-one, one-to-many, many-to-one and many-to-many. IPC that involves many end points for a message rarely has a response. Often, the response that might have been required is carried in a message from another source sending a one-to-many message.

RPC integrates communications as a programming language. To provide one-to-many or many-to-one with RPC, there are two possible approaches. A sequence of one-to-one calls and responses can be made. Alternatively, a call message could be multicast. In this case there would be multiple responses. Most programming languages used in this area do not support automatic type conversion of return values from procedures or functions to (variable length) arrays of that type. Hence handling many-to-many communication is not transparent.

2 The CAR Conferencing System Architecture

The CAR Conferencing System was designed to provide European automobile designers audio and video conferencing and the ability to share views of their designs. It was intended to work in the wide area initially using ISDN as the interconnection technology, but later adapted to using IP.

The system has both centralised and distributed elements: It is distributed in the sense that components (may) run on separate machines in a network; it is centralised in that there are several single points of failure. It does not provide failure tolerance, nor does it provide a performance enhancement over a single machine implementation.

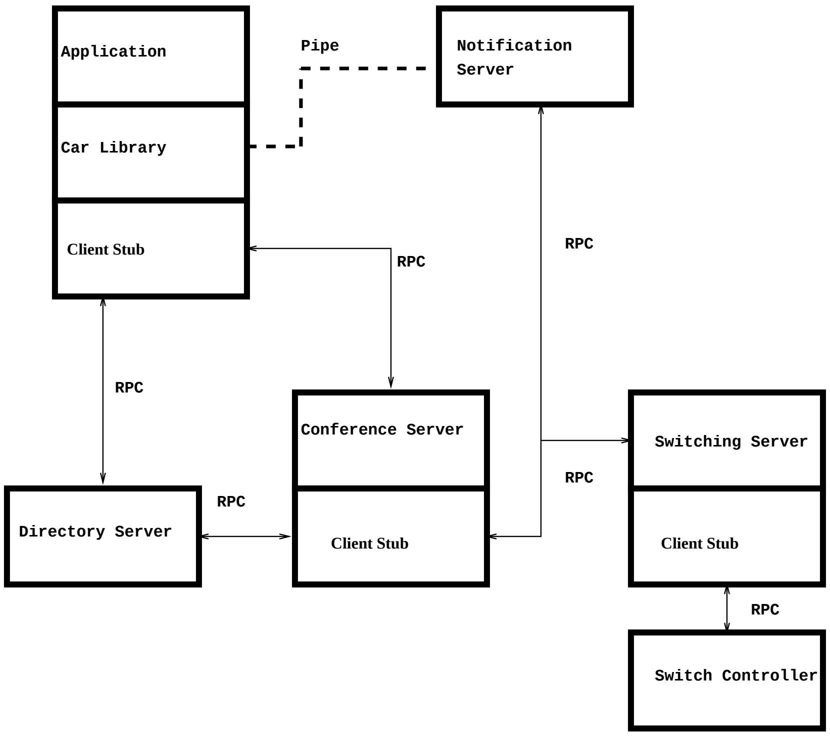

Figure 1: The CAR Conference System Architecture

2.1 CAR Conferencing System Components

The CAR Conferencing System has many components in common with the architectures described in[Nicolaou] and [Schooler]. We believe that the components of the CAR Conferencing System are necessary and sufficient. These components are shown in Figure 1.

The components, written in C, all communicate using Remote Procedure Call, with the exception of the channel between the Notification Service and Car Library, which uses a Unix Pipe.

The system's clients and servers have the following roles:

- The *Switch Controller* acts purely as a service that exports an interface only to the Switching Server. The former runs on a machine attached to any hardware specific to providing switched video or audio services. These include a digitally controlled analogue video switch and a CODEC hub, providing multiplexing and demultiplexing of video streams.

- The *Switching Server* is a client of the Switch Controller, can run on any host in the system. It exports an interface that is used by the Conference Server.

- The *Directory Server* is purely a server, and provides user and multimedia workstation location information for a site. This is used by the Conference Server.

- The *Conference Server* primarily provides a service for the applications. Applications interface via the CAR standard library. The Conference Server is also a client of the Directory Server, the Switching Server, and the Notification Server.

- Applications are clients of the Conference and Directory Services, but are linked with a standard CAR library to hide this communication and to hide their dependence on the Notification server described next.

- Every Application is run alongside a *Notification Server*, which is called by the Conference Server as a result of the actions of other applications. The interface between the Notification

Server and the application is via a pipe. This is implemented inside the Car Library, so that it is largely transparent to the application.

For a given CAR set of conferences, there is only one Switch Controller, one Switching Server, one Directory Server and one Conference Server. These are run as demons by the system (possibly on a variety of different machines, but quite possibly on a single machine). Due to the blocking synchronous nature of the version of ANSA RPC used, and the choice to import interfaces immediately at startup of each client, the servers must therefore be started in the correct order: Switch Controller, then Switching Server, in parallel with the Directory Server, then Conference Service; finally, conferencing applications.

There is another level of indirection and cause of unreliability, to paraphrase Dijkstra: there is a trader, ANSA's RPC Binder which is required by ANSA RPC clients to locate servers, as they do not make use of well-known ports: a restart of part of the system results in deadlock because new services' locations are no longer known to old clients. This is because there is state in the client which reflects the old locations.

2.2 Applications

The user may run a variety of applications. For example, the user usually runs a Conference Manager application, that lets the user control and know what is going on in a conference. A user can also run two other distinguishable types of application, called *conference aware* and *conference transparent*. Conference aware applications have been explicitly implemented as conferencing applications for the CAR system, such as a purpose-written shared editor. Conference transparent applications are ones that have been implemented either as stand-alone, or else as distributed, but unaware of possible interactions with other distributed applications. In both these cases, a so-called *Nest* program is used to encapsulate the application and map Car Conference Library calls and Notifications into meaningful application specific actions. A Nest program must be written for each new type of conference transparent application to implement these mappings. Examples of Conference Transparent Applications currently fall into 3 types:

1. X Windows based applications

   These are conferenced by using a CAR version of Shared X, and a Nest for Shared X to provide Application Inclusion/Termination or Exclusion, Floor Control and other CAR functions[Handley and Wilbur].

   Shared X is a mechanism for replicating the window of an X Windows client program on a number of servers' displays. Other shared window paradigms are feasible, as long as they allow single user applications to produce replicated bitmap display output, and have modest control mechanisms for the workstation providing input at any one time.

2. Conventional Interactive Applications.

   These include applications that are normally run on a workstation interactively by a single user. For instance, editors (whether text or graphics) fall into this category. Each of these requires a special purpose Nest.

3. Simple (mono-media) Conferencing Applications.

   These use a multicast network address as their means of identifying a conference as well as transmitting their media data. These include the LBL VAT [Jacobson] program and the INRIA IVS program [Turletti and Huitema]. They need to have multicast addresses allocated per conference. Floor control, restricting which source can send at any one time, can be applied through the Notification Service and a VAT or IVS specific Nest.

2.3 Inter-process Communication

The CAR Conferencing System used RPC because the system designers were familiar with RPC, and not so familiar with lower level network programming tools. The modularisation that we chose lent itself to separate implementation and testing. Different programmers were able to carry out these tasks, even though several of these programmers had no previous experience with networks. Testing could take place without the need for a network at all, since RPCs could be called in the same system. The benefit was that the complete system was produced in a matter of months.

It was expected that conferences would be small and tightly controlled so that a single, centralised Conference Server would be adequate. Networks were believed to be reliable. Our experience of running the system across European sites interconnected by IP routers across the narrow band ISDN, was that demand for wide area conferences amongst larger groups exceeded our expectations. We also found that network failures were common.

Points of failure will cause the entire system to fail:

- If any of the Switch Controller, Switching Server, Directory Server or Conference Server fail, the entire system fails.
- If the servers above are distributed across a network, then if any link between them fails, then the entire system blocks until that link recovers.
- Worse still, the nature of synchronous blocking RPC is such that if a link fails between the Conference Server acting as a Notification Client and a Notification Server, the entire system blocks until that link recovers.

Since the Conference Server is central to a conference, it is also a performance bottleneck for the system. Recent experience broadcasting the Internet Engineering Task Force Meeting over the Internet in the US [Casner and Deering] using a more loosely structured system show that it is not unreasonable to have 500 participants in a networked multimedia conference. However, there is little hope of the CAR Conferencing System Architecture scaling to 500. In the Internet, 500 sites rarely have full connectivity, and any single link outage will thus frequently block the progress of the entire conference. It may be that real-time multimedia chatlines will become as commonplace as Internet Relay Chat and Bulletin Board use - in this case, we may see many systems spread over the wide area, and inevitably have to deal with partial availability.

We believe that our instantiation of components as processes, and the subsequent communications architecture was a poor implementation strategy. To use Open Distributed Processing terminology, from the information viewpoint we have reflected the right model, but from the engineering viewpoint, we may not have.

Simple replication of the servers provides no solution to the lack of fault tolerance - an entire protocol must be developed to make any access to those replicated servers location transparent, and map failures into the right application exceptions. In other words, the decomposition into services has done little more than basic software engineering would, and does not address the real distributed computing aspects of the problem.

3 Weak RPC

During the task of porting the application to Sun RPC, we learned that there are ways in which an RPC implementation permits the programmer more flexibility than a strict remote emulation of local procedure call implies. We call this 'weaker' RPC:

- Call Back

  A client calls an RPC and normally blocks until there is a response. Some systems permit servers to make calls to the client process while it is blocked awaiting their response. This role reversal may continue, providing mutual recursion of RPCs. This is called *Call Back*.

Since Sun RPC supports Call Back, we can dispense with the CAR Notification Service as a separate process, and merge it in the CAR Library. We simply replace reading the pipe between separate processes, by polling on the RPC socket, followed by the dispatch table call if any data is present. This makes the system simpler, more robust and possibly more responsive.

- Broadcast

  Sun RPC has a form of Broadcast. At the moment several of the servers, the Conference Server, Directory Server, Switch Controller and Switching Servers, are bound to particular hosts, and are found through global knowledge of their host location. Broadcast [or better still, multicast] RPC could provide a discovery based approach, where no bootstrapping information at all is required.

- Streaming

  Sun RPC provides a streaming service, where RPCs do not block at the client, and a number of calls may be made successively without awaiting a response for each. This may be used to make the system robust in the face of link outages. For example, notifications can be streamed to applications, instead of the Conference Server blocking unnecessarily for responses.

ANSAWare 4.0 has many of the same mechanisms as Sun RPC, but is based on a system of Tokens that permit call back and streaming and so on. In addition, it provides a threads package, which permits lightweight concurrency within servers. These capabilities would permit a similar restructuring of the system, but would make the Conferencing System further dependent on a less commonly available RPC system. We would also be more dependent on its annotation approach to enhancing C. This is felt to be inappropriate because the maintenance of the system would be greatly complicated by having to track ANSA releases as well.

In a related piece of work, we are designing a compiler for the GDMO language (Guidelines for the Definition of Managed Objects, an ITU standard) to produce C++ code to provide easy access to new Managed Objects in a large system. Programmers reported that they were happier with a scripting approach than with an annotation approach to this type of problem. The script approach entails the programmer writing scripts that direct the actions of the stub compiler in a special purpose compiler directive language, as opposed to adding notes in the actual RPC code in a hybrid language. Thus the distributed application is kept one step removed from the communications software.

4 Event Driven Approaches

In the future, it is clear that a more object oriented approach will enhance the system. This will remove redundancy in the code, eradicate most of the (trivial) annoyances in the current environment by further removing the scripts or annotation, and permit clean integration of networked data types with ordinary programming data types.

However, the CAR communications architecture is still flawed: it is inherently designed around a model of distributed processes that perform specialised tasks on behalf of the whole system. Any attempt to provide robustness (e.g., the broadcast for directory or the streaming of Notification RPCs) only alleviates some of the problems. There still exist centralised points of failure. This is in fact inevitable for a system designed this way. The CAR Conference System was built initially in the original spirit of RPC, which is [Wilbur and Bacarisse] to provide an "exactly once" semantics for remote procedure call that is as close to local procedure call as possible[Xerox]. Consequently, systems built this way are sequential programs running in multiple address spaces - when these are instantiated as separate processes on separate machines networked, such systems are far less reliable than single process/machine. In the extreme, pure RPC is an anathema to distributed algorithms.

In the case of the system discussed here, the requirements for consistency are weak in many components, thus not a strong case for RPC. In more interesting distributed computations, whether for increased concurrency and performance, or fault tolerance, we often find that divide and conquer or other approaches to the distribution again have little requirement for a response to messages coupled with associated processing. That this should appear attractive for conferencing is a consequence of the peer-wise nature of the application. Communications between components in a multiparty conference is much more akin to that of a collection of distributed routers, than to a client/server model.

Re-examining each of the basic CAR Conference System Services in turn, we can see that very few of them need the stricter requirements of pure RPC.

- The Switch Controller is attached to a physical resource that keeps permanent state, and therefore cannot be distributed. However, access to it may be distributed. Thus RPC is appropriate here.
- The Switching Server is, on the other hand, a deliberate attempt to remove the service from the location, and thus should be distributed. Indeed, we can foresee a system with multiple distributed servers, and the Switching Server should be aware of all these.
- The Directory Service, like any other directory service, need not be a central system. DNS [Mockapetris], X.500 [Kille] or other mechanisms do not make use of RPC, nor should these, especially since a directory is a read only service which needs weaker semantics than the normal *exactly once* RPC. Some information held by the Directory Server may be completely distributed with the Application.
- The Notification Service is inherently asynchronous. Thus a message passing system is sufficient. Given notifications come from the Conference Server largely as a result of other application actions it is possible that these can be sourced from each application, and directly multicast to all other applications' Notification Interface, using reliable delivery (if large notification messages are sent), but not awaiting any response.
- Lastly, the conference service is merely a coordinator of the system. If the steps outlined above are taken, it withers away and becomes redundant.

In figure 2, we illustrate the components of a more distributed approach. Here, we have adopted a pure message passing/event driven model that incorporates the ideas above. Independent failure of a site is now feasible without delaying or halting an entire conference.

5 Related Work

Recent years have seen a great deal of research and development in the area of multimedia conferencing.

Crowley[Crowley] describes *MMConf*, a system for building shared multimedia applications. Here, the actual protocols that control and coordinate the applications are based around extensions to the Diamond/Slate Multimedia mail systems.

Arango[Arango et al] describes the *Touring Machine*, a system for moving the media around. The conference control architecture is not expounded in detail here.

Schooler[Schooler] describes *mmcc*, a system for controlling largely tightly bound multimedia conferences, based around RPC.

The ITU standard, H.320[H.320], is a classical design for the signaling of messages between components in a telephony style conference control system.

A different approach can be seen in the work at LBL on a Session Directory and Visual Audio Conferencing tool [Jacobson] make use of a distributed algorithm for conference identification and for session and conference announcements based on periodic multicasting of information in a similar way to beaconing in Cell-Based phone systems or station-identification protocols. This can replace a directory service, including session membership, and a fair portion of a conference management system, including floor control. The n-to-n use of multicast can lead to synchronized traffic unless care is taken. We have named this behaviour of distributed applications a "systolic tendency" from the original meaning of the word systolic, meaning heartbeat. It has been reported in distributed routing algorithms by Floyd. [Floyd and Jacobson]

If engineered properly, we believe that the approach using loosely coupled messages and events as espoused by these researchers forms a more robust scalable starting point. Work at UCL on the Conference Control Channel Protocol[Handley and Wakeman] is attempting to bring together all aspects of this approach.

The notion of *tightly coupled* conferences, with strict membership and floor control, has led some researchers to re-examine the more RPC-like approach. However, the CCCP work shows that the semantics of conferencing are orthogonal to the way they are constructed from IPC mechanisms.

6 Conclusions

Enhancing the Remote Procedure Call semantics to use call back makes structuring recursive programs in separate address spaces easier, and does not violate the original spirit of RPC. However, Broadcast RPC and Streaming RPC both conflict with normal procedural behaviour. The reason for using these extensions is that we are concerned with moving error handling from the system up into the application (i.e., we do not want protocol errors to map into system failure). This means that we now have to implement exception handling on an ad hoc basis for every procedure, in every program/interface in the system. However, in a conferencing system, there are almost certainly very simple patterns of failure that can be tolerated, others which lead to reconfiguration, and still others that lead to system failure.

The long term solution is to provide the CAR services in an inherently distributed way, so that any client program is not dependent on any single server process. These functions must move from the application into the infrastructure. This can be viewed as providing them as part of standard Distributed Operating System support.

It has been noted elsewhere[Needham] that the combination of Remote Procedure Call and a threads[Gentleman] package is dual in some sense to a message based approach to distributed systems. While this is true, it is true for the designer of a threads based RPC system. However, for the software engineer, it is the actual implementation of a tool that influences their design. The RPC used in the initial CAR design did not support threads, and so the option of using concurrency in servers to mimic message passing was not available. In any case, the application programmer is still left with the problem of eliminating orphan processes after failure of concurrent RPC. A system designed without the expectation of a reply where possible, seems more appropriate.

In some cases, there is no reply required in any cases. In others, the lack of reply is in itself the indication required by the application. Further, all servers are also clients and call each other in any case, and can carry the same information in the 'call' as they would have put in a response to an RPC, thus saving a great deal of traffic. In many cases, clients makes a call to a number of servers, and the use of multicast is indicated. However, the collection of multiple replies is not required since the outgoing message/call can carry the information needed as above. This multi-way rendezvous technique was suggested by Jacobson [Jacobson].

The direction that our work has led is to give more and more responsibility to the programmer for designing the protocol data, and semantics, especially with respect to failure handling. This strongly supports ideas expressed by Clark [JH Saltzer, DP Reed, DD Clark] and Tennenhouse [David Clark, David Tennenhouse].

These conclusions have led us to completely re-think our approach to control and coordination of conferences. The final design is reported elsewhere, but we felt it important to report the reasons we have abandoned the tools and approaches described in this paper to help other researchers avoid these pitfalls.

7 Future Work

Work is in hand to develop a more complete solution along the lines pictured in figure 2. We propose the notion of an Active Channel called the Conference Communications Channel Protocol, or CCCP. The starting point for this is the recognition that the main advantage of RPC mechanisms lies only in the networked data typing language. Beyond these functions, we must first discard the illusion of networked threads of control, and concentrate on the distributed algorithm design before determining when and where responses are required of a service, and at what stage any error notification is needed if at all.

As mentioned before, work in some communities on multimedia conference control is still based around models of reliable networks, and hence the use of RPC like control. Our work has led us away from this assumption and towards the systems we are now designing using CCCP[Handley and Wakeman].

8 Acknowledgements

Thanks are due to Mark Handley and Ian Wakeman of UCL for discussion of the CAR system. Thanks to anonymous reviewers of two earlier versions of this paper, and to Peter Danzig for very useful feedback. Remaining mistakes are the author's.

The CAR System was initially developed under the RACE CAR program at UCL, by Stefan Penter, Henrik Schwartzlose, Nermeen Ismail and Mark Handley at UCL. The Sun RPC port was by Jon Crowcroft for the RACE PREPARE project.

References

[Andrew Herbert] "An ANSA Overview"
IEEE Network, pp 18-23, January, 1994

[APM Ltd] "The ANSA Testbench Version 4.1 Manual"
Architecture Projects Management Ltd, Cambridge, U.K., May, 1992. 1988

[Arango et al] "Touring Machine: A software platform for distributed multimedia applications"
IFIP Upper Layer Protocols, Architectures and Applications Vancouver Canada, June 1992

[Birrell and Nelson 84] "Implementing Remote Procedure Calls"
in ACM Trans.Comp.Sys, 2,1,pp39-59, Feb 1984

[Casner and Deering] "The First IETF Audiocast"
in ACM CCR, Vol 22, 3, July 1992

[David Clark, David Tennenhouse] "Architectural Considerations for a new Generation of Protocols"
in ACM SIGCOMM, Philadelphia, September 1990

[Crowley] "MMConf: An infrastructure for building Shared Multimedia Applications"
in Proceedings of CSCW '90, Los Angeles, USA, October 1990

[Floyd and Jacobson] "On the Synchronisation of Periodic Routing Messages"
Proc ACM SIGCOMM, 93 San Francisco 1993.

[Gentleman] "Message Passing Between Sequential Processes: the Reply Primitive and the Administrator Concept"
Software – Practice and Experience, 11, 5 May 1991

[H.320] "NARROW-BAND VISUAL TELEPHONE SYSTEMS AND TERMINAL EQUIPMENT"
ITU H series recommendations, 1992

[Hamilton] “A Remote Procedure Call System”
University of Cambridge Computer Laboratory, Technical Report no 70 Dec 1984

[Handley and Wilbur] Mark J. Handley, Prof Steve R. Wilbur “Multimedia Conferencing: from prototype to national pilot”
Proc INET '91, Kobe, June 1992

[Huitema] “Measuring the Performance of an ASN.1 Compiler”
Upper Layer Protocols, Architectures and Applications, IFIP 6.5, 95-112 Vancouver, 1992

[Jacobson] “Visual Audio Tool”
Unix Manual Page, anonymous ftp from ftp.ee.lbl.gov:sun-vat.tar.Z 1991

[Jacobson] “Tutorial on Lightweight Sessions to AARNet Annual Conference”
http://www.it.kth.se/ klemets/vatplay.html

[Kille] “The QUIPU Directory Service”
in Proceedings, Fourth International Symposium on Computer Message Systems pp 173-185, Sep 1988

[Mockapetris] “Domain names - implementation and specification”
RFC 1035 SRI-NIC November 1985

[Needham] “On the duality of operating systems structures”
Operating Systems Review, 13, 2, April 1979

[Nicolaou] “An Architecture for Real-Time Multimedia Communication Systems”
IEEE Journal on Selected Areas in Communication, 8, 3, pp 391-400 April 1990

[JH Saltzer, DP Reed, DD Clark] “End-to-End Arguments in System Design”
ACM Transactions on Computer Systems, 2, 4, pp 277-288 November 1984

[Schooler] “Case Study: Multimedia Conference Control in a Packet switched Teleconferencing System”
Journal of Internetworking Research and Experience, V 2, No. 4 June 1993

[Sun 86] “Remote Procedure Call Protocol Specification”
Sun Microsystems Inc rfc1057/Part no. 800-1324-03, revision B February 1986

[Turletti and Huitema] “Inria Videoconferencing System”
Unix Manual Page, anonymous ftp from avahi.inria.fr:/pub/videoconference 1992

[Unix Man] “Make: maintain, update, and regenerate related programs and files”
Unix Manual Page 1990

[Handley and Wakeman] “The Conference Control Channel Protocol Architecture”
work in progress, UCL Computer Science Department May 1994.

[Wilbur and Bacarisse] “Building Distributed Systems with Remote Procedure Call”
in IEE Software Engineering Journal, 2, 5, pp 148-159 Sep 1987

[Xerox] “Courier: The Remote Procedure Call Protocol”
XSIS 038112, Stamford, Conn Dec 1981

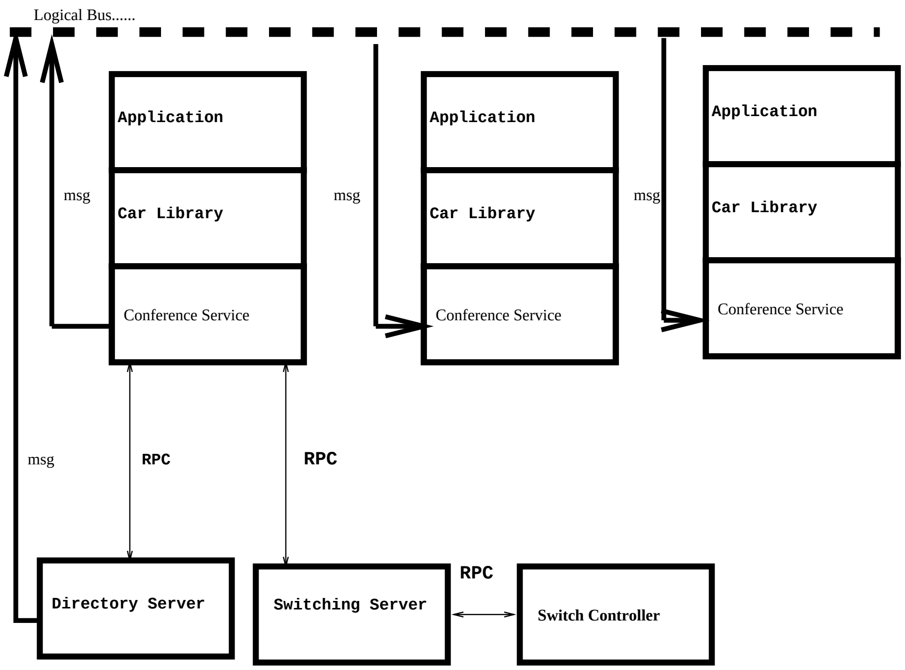

Figure 2: An Event Driven Conference System Architecture