Scalability and Mobility in a Network Middleware for Large Scale Mobile and Pervasive Augmented Reality Games

Pedro Ferreira, João Orvalho and Fernando Boavida
Centro de Informática e Sistemas, Departament of Informatics Engeneering
University of Coimbra
Polo II, Pinhal de Marrocos, 3030-290 Coimbra, Portugal
{pmferr,orvalho,boavida}@dei.uc.pt

Abstract - *Ubiquitous or pervasive computing is a new kind of computing, where specialized elements of hardware and software will have such a high level of deployment that their use will be fully integrated with the environment. Augmented reality extends reality with virtual elements but tries to place the computer in a relatively unobtrusive, assistive role. In this paper we examine the scalability of a innovative network middleware for large scale mobile and pervasive augmented reality games, analytically and by comparison of the architecture proposed with alternative possible architectures, with the aim of proving our architecture is the more scalable one. We also analyze mobility issues of the architecture and how they are handled.*

Keywords: *Pervasive computing, augmented reality, scalability, mobility.*

1 Introduction

A significant requirement of pervasive applications is fast service development and deployment [1], which implies the introduction of various service and application frameworks and platforms. For this, middleware is a common solution. The benefits of middleware utilization are the improved programming model, and the hiding of many implementation details, which make middleware based application development much faster. It is now becoming quite clear that entertainment, and more specifically mobile gaming, will be one of the killer applications of future wireless networks [2].

Augmented reality extends reality with virtual elements while keeping the computer in an assistive, unobtrusive role [3]. It is possible to create games that place the user in the physical world through geographically aware applications. Most of the latest mobile phones are equipped with cameras and some of the latest ones are coming with some form of 3D rendering technology [4] [5]. Bluetooth technology and increasing miniaturization will lead, in the near future, to low-cost, specialized pervasive equipment for augmented reality. In [6] we described the main objectives of our research concerning systems that satisfy the requirements of network middleware for large scale mobile and pervasive augmented reality games. In [7] we described a middleware system that is being developed for large scale mobile and pervasive augmented reality games that satisfies these objectives. The system targeted by the middleware is composed of 3 levels: the back-office central level, the large scale network level, and the personal area network level. This paper focuses on scalability and mobility issues of the middleware proposed. The paper is divided in Introduction (this section), Architecture, Scalability, Mobility, and Conclusions, aside from abstract, keywords, acknowledgments and references.

The main objective of this paper is to prove that our architecture is the most scalable from a number of alternate architectures that represent the main architectural trends in modeling distributed systems.

2 Architecture

The system targeted by the proposed middleware is composed of 3 levels: the back-office central level, the large scale network level, and the personal area network level.
The back-office central level consists of one or more of a series of parallel servers and serves as the main controlling station of the game administrator, the person responsible for starting, stopping and managing game performance and general maintenance tasks.
The large-scale network is the standard 3GPP network, where servers are distributed according to some logic of spatial distribution, typically corresponding to aggregations of cells of the mobile communications network.
The personal area network level consists of the network of pervasive devices dedicated to personal communications and to augmenting reality, which the person carries. These may be sensors, actuators, and other devices that can communicate using Bluetooth or other means of communication. All these communicate with the mobile host, probably just a cell phone or specialized device connected to the large-scale 3GPP network. In this way, the player is so enabled to play games of augmented reality irrespective of his/her location.
Targeting this architecture allows the study, evaluation and proposal of mechanisms to deal with issues of scalability, multimedia data heterogeneity, data distribution and replication, consistency, security, geospatial location and orientation, mobility, quality of service, management of

networks and services, discovery, ad-hoc networking and dynamic configuration.
We consider that building augmented reality applications using a network middleware (option B) is better that building them standalone (option A). This is because with option B many game applications may then use the same application programming interface (API) to leverage network resources, giving it much faster service development and deployment.
The middleware presented in this paper is being built according to the characteristics of agile pervasive middleware [8], such as application-awareness, mobility, integration, interoperability, scalability, portability, adaptability, robustness and simplicity of evolution.

2.1 Central level

At the central level, there is one server, which may be constituted by more than one parallel server, running Java Standard Edition 1.5.0. There will also be database servers, which may or may not be integrated with the same server.
This server or collection of servers will be connected to the HSS (Home Subscriber Server) of the 3GPP Network by the DIAMETER protocol SH application and are, together, an IMS (IP Multimedia Subsystem) application server.
All authentication, accounting, and authorization will happen through this interface. All management of the game servers will happen through this server.
Status Transmission Framework version 2.0 APIs for the server side include a DIAMETER [9] API which includes the base protocol, the CX and DX [10] applications and the SH applications [11] of 3GPP. This would communicate preferably through SCTP [12][13] (we also developed a java SCTP API that presently only works under Linux, but can be easily extended to other platforms, as soon as those platforms support SCTP natively) if available. If not, TCP will be chosen. The DIAMETER API implementation supports TLS [14] and works over IPSec.
The terminal (UE) from the personal area network will communicate with the central server through SIP [15] to initiate the session, authenticate itself and get the details for the session through SDP [16] negotiation (that's another API we have developed, the J2ME SDP API - in the server side we use JAIN SDP API based on JSR 141). The SIP and SDP exchanges include enough information to choose a distributed server to communicate with, according to the terminal's geographical location. The terminal geographical location is acquired through the use of the J2ME Location API (JSR 179).

2.2 The large scale distributed server level

At the distributed server level, there are multiple distributed servers, linked to geographical coverage areas which in the extreme may even be linked to the cells of the mobile network, which will distribute the load off the main server.

These servers run Java Standard Edition 1.5.0, also. They will have integrated database servers running on the same or different computers.

These servers will be interconnected by a reliable multicast protocol capable of working in an IPv6 network, without the support of network elements, capable of working in the many-to-many scenario, without the nak implosion problem but nak based, source ordered and avoiding duplicates: The Sixrm Protocol [17]. The Sixrm Protocol is integrated in a new version of ARMS – of which version 1 is published in [18] [19] – the Augmented Reliable Corba Multicast System, that is capable of running over Ipv6 networks.

2.3 The personal area network level

At the personal area network level we will find the most diversified types of devices. The main device will probably be a cell phone or a specialized device for game playing.

The required characteristics for this device is that it must support the Java language , more specifically, Java Micro Edition, in its Connected Limited Device Configuration (CLDC) version 1.1, and the MIDP – Mobile Information Device Profile - version 2.0..

This central device must support also the Java Bluetooth API (JSR-82), the Java SIP (Session Initiation Protocol) API for J2ME (JSR-180) and the location API for J2ME (JSR-189).

Other devices that are needed on the personal area network level are input and output devices. These devices must also support at least Java (same version and configuration) and the Bluetooth API.

Output devices are essentially video and audio output devices. Video and audio output devices should also support, besides Java (CLDC 1.1) and Bluetooth for Java Micro edition (JSR-82), the Mobile 3D graphics API (JSR-184), and the Mobile Media API for J2ME (JSR-135).

As for input devices, in the real world environment, the user is often used to using one or both hands to perform a task. Therefore, the input devices used with wearable computers need to be designed with this requirement in mind. Appropriate input devices need to be utilized to allow the user to efficiently manipulate and interact with objects. For data entry or text input, body mounted keyboards, speech recognition software, or hand held keyboards are often used. Devices such as IBM's Intellipoint, trackballs, data gloves, etc., are used to take the place of a mouse to move a cursor to select options or to manipulate data. One of the main advantages of using a wearable computer is that it allows the option of hands free use.

Common factors in the design of input devices are that they all must be unobtrusive, accurate, and easy to use on the job.

In order for any digital system to have an awareness of and be able to react to events in its environment, it must be able to sense the environment.

This can be accomplished by incorporating sensors, or arrays of various sensors (sensor fusion) into the system. Sensors are devices that are able to take an analogue stimulus from the environment and convert it into electrical signals that can be interpreted by a digital device with a microprocessor.

For a sensor or array of sensors to be supported by the Status Transmission Framework version 2.0, it must be accompanied by hardware that translates its electrical impulses to digital signals transmitted over Bluetooth communications over the personal area network to the central device.

The central device will coordinate all the augmented reality experience for the user, using all the multimedia capacities of the other devices and eventually, even own multimedia capacities of the central personal area network device.

3 Scalability

To analyse the scalability of our architecture, we are going to analyse the network traffic that is probably going to be generated in an analytical way. The network traffic generated also affects the processing time at the nodes so all aspects of scalability are affected in this way. To do this, we must analyse all levels of the system and the way they work together.

3.1 The personal area network level

The first level that is analysed is the personal area network level. Within this level are sensors, actuators and the main game device. All sensors and actuators communicate with the game device that communicates with the large scale distributed level servers of the system. The API for sensor and actuator communication with the central game device is the SENSACT API on the sensors and actuators and the API on the game device is the STF PAN API. The STF PAN API coordinates all sensors and actuators and sends and receives only one stream of data to the current distributed server.

If we note by A_i and S_i the messages (it's size in bytes) sent from a sensor i to the central game device on the PAN and the messages sent to actuator i from the central game device, and we suppose that set of actuators and sensors the central game device can process that data so that only the minimal messages M_k ,in bytes, get transmitted or received from the distributed servers, we get that Tm, the total maximum number of messages (it's size in bytes) handled by the central game device on a period of time between instants t_0 and t_1 is:

$$Tm = \sum_{i=1..N} A_i + \sum_{j=1..M} S_j + \sum_{k=1..L} M_k . \qquad (1)$$

Where N is the number of sensors active, M is the number of actuators active and L is the number of messages received from the distributed server on that period of time, which depends on the number of objects we are getting updates from, which is limited by partitioning. So we can consider L approximately equal to the number of objects in a period of time sufficiently small between t_0 and t_1 multiplied by 2, because we both send and receive.

We so have that:

1. The maximum number of messages from the sensors increases linearly number of sensors active;

2. The number of messages to the actuators increases linearly with the number of messages relevant received from the distributed servers (objects in view that trigger the activators) and the number of sensors active.

3. The number of messages to the distributed servers increases not with sensor number, but with the number of objects in view, and this is limited by partitioning the virtual world.

3.2 The large scale distributed server level

At the large scale distributed server level each distributed server, at the same time interval, will be responsible for the users in its area and only minimal communication will be maintained between the servers. We can denote that minimal communication by MC_i, in bytes. We can denote the user communication at each server by M_k, in bytes, as we did on the personal area network level. Note that this user communication depends on the number of objects in its view, but that largely is limited, due to partitioning that is made by the system of the virtual world. So the messages are even further minimized in that way in a location oriented dependent manner. As we communicate through ARMSV6 and Sixrm reliable multicast, the formula for the maximum number of messages TDm (it's size in bytes), which each distributed server will handle, will be approximated by:

$$TDm = \sum_{i=1..N} M_i + \sum_{j=1..M} MC_j \, . \qquad (2)$$

Where N is the number of users and M is the number of distributed servers. We have that the total number of messages handled by the distributed server will:

1. Increase linearly with the number of users on that distributed server.

2. Increase linearly but in a much slower rhythm with the number of distributed servers (because MC_j is really a small amount).

3.3 The backoffice central level

The back office central level will be handling management and session initiation and termination. In management, messages exchanged depend linearly on the number of distributed servers on the network. In session initiation and termination messages, as also mobility handling messages, the total number of messages also depends linearly on the total number of users on the system. But these kinds of messages happen infrequently, only when users join or leave the system, or when the manager wants to look, examine data or change things.

3.4 Analysing the scalability of possible alternate architectures

3.4.1 The totally centralized architecture

We assume that by the totally centralized architecture we mean that the central game device on the PAN will do no processing on sensors and activator messages and send all to a central server to do all processing related to the user and send the result back to this user sensors and activators.

This would be of course a situation where the server would be a major bottleneck, as the equation for the total number of messages on the server clearly shows, for the time between instants t_0 and t_1:

$$Tm = \left(\sum_{i=1..N} \left(\sum_{j=1..M} S_j + \sum_{k=1..L} A_k \right) \right) x2 \, . \qquad (3)$$

We use here the same notation used until now with Tm being the total maximum number of messages, in bytes, handled by the central server. We multiply the sum by 2 because we both receive and send messages. N is the number of users, M the number of sensors per user and L the number of actuators per user.

We now have the entire load on one component, and loose the benefits of distributing the load for more than one component.

Here we are not counting with session initiation, session termination, mobility handling, and management messages. These tend to happen infrequently.

3.4.2 The totally distributed architecture

In the total distributed architecture, we would have only the large scale distributed level of the system doing all the processing. There would be no processing on the PAN and no processing on the Central BackOffice Level.

This leads to problems of finding the correct server to connect to in the first place, we would have to build a list of servers in each user central device. And these lists must be maintained synchronized with the configuration of the network, witch would be no easy task.

Mobility and management will also be moved to the distributed level completely, complicating things a little more. On our architecture mobility and management have a distributed component, but are centrally coordinated, which does not happen in this scenario.

But, seeing things in number of messages transmitted and using the same notation we have in each distributed server, between instants t_0 and t_1, taking that the PAN central device does not do any processing, TDm the total maximum number pf messages (it's size in bytes) processed in one distributed server is:

$$TDm = \left(\sum_{i=1..N} \left(\sum_{j=1..M} S_j + \sum_{k=1..L} A_k \right) \right) x2 + \sum_{l=1..O} MC_l \quad (4)$$

Where N is the number of users on this distributed server, M is the number of sensors by user, L is the number of actuators per user and O is the number of distributed servers on the system.

We have that the total number of messages handled by the distributed server will:

1. Increase linearly with the number of distributed servers

2. Not increase linearly but in a faster rhythm with the number of users, sensors, and actuators on the users of this distributed server.

So we have that the solution is more scalable than the totally centralized one, but less scalable than our solution, because the load on the distributed servers is much more.

3.4.3 The pure peer-to-peer architecture

By the peer-to-peer architecture, we aim to analyse a situation where the central game device has all the work of the system, communicating only with other central game devices through the 3GPP network. It has processing capabilities and processes the sensor and actuator messages. It only sends and receives messages from all other nodes on the system. We denote by Si the sensor messages, Ai the actuator messages and Mi the node messages to and from other nodes. We have that TPm, the total maximum number of messages (it's size in bytes), on the main game device, the main device on the personal area network, is, on the time between t0 and t1:

$$TPm = \sum_{i=1..O} A_i + \sum_{j=1..N} S_j + \sum_{k=1..M-1} M_k \ . \qquad (5)$$

Where N is the number of sensors active on the PAN, M is the number of users on the system and O is the number of actuators active on the PAN. We see that this formula is very similar to our formula in our 3 levels system for the PAN level, but now the Mk factor depends on the number of users and not on the number of distributed servers, which clearly is worse than in our case. So, the totally peer-to-peer architecture is not as scalable as ours.

3.5 Graphical case study of the alternate architectures

For a more visually appealing comparison, we will do a graphical comparison between de various alternatives. For this, we will fix the number of sensors in 5 and the number of actuators in 3. We will start with 100000 users and work our way up to 3000000 users in steps of 100000. We will analyse a network, in our case, with 100 distributed servers uniformly distributed and with users uniformly distributed, on the three levels architecture and on the totally distributed architecture. We will analyse output variables. We will fix the size of messages in 50 bytes for the sensors, 300 bytes for the actuators (in reality this maybe more depending on the kind of actuator but for our study this will do), 300 bytes for MCi and Mk. We will fix the number of objects in view to 10. We will build a program to run the simulation and output the results to a CSV file that then gets analysed by Microsoft Excel to output the graphs shown here.

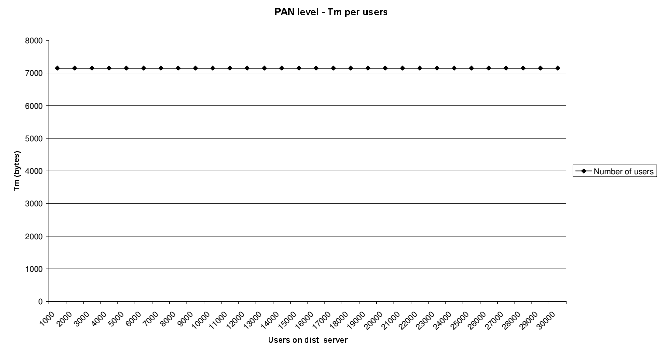

Figure 1 - Personal Area Network level

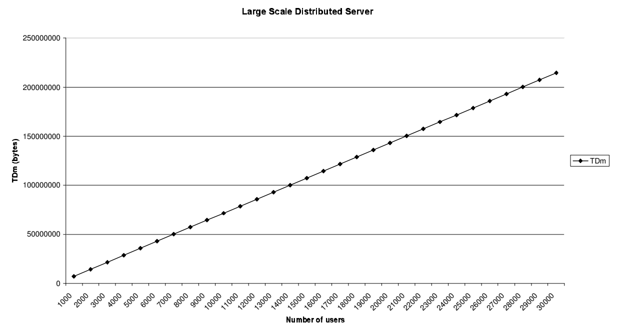

Figure 2 - Large Scale Distributed Server Level

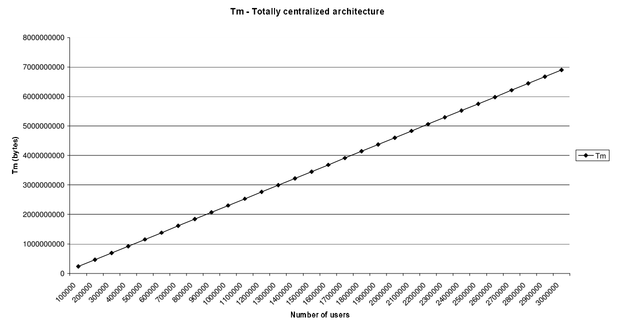

Figure 3 - Totally centralized architecture

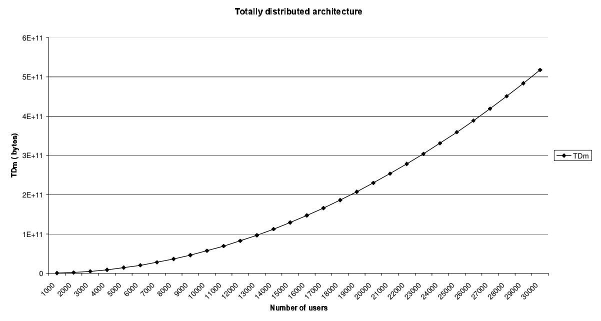

Figure 4 - Totally distributed architecture

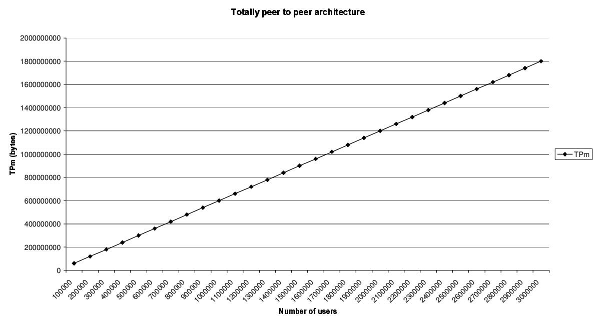

Figure 5 - Totally peer-to-peer architecture

Notice how the PAN level on our architecture does not depend on the number of users, contrary to what happens on the totally peer to peer architecture. Notice on how on the totally distributed architecture when the users go up to 3000000, we go exponentially to 6E+11 bytes in each server (corresponding to 30000 users on each server), contrary to only 214560000 bytes linearly in our solution in each of the distributed servers. In the fully centralized architecture, the maximum number of messages linearly increases to a maximum of 6900060000 bytes, when we have a maximum of 3000000 users, which is much more than any of our distributed servers on our large scale network.

We may have had better results yet if we had allowed our distributed servers to grow to accommodate the ever growing user base, but that was not the scenario envisioned. In reality, probably we will have more distributed servers with a growing user base. Even so, when comparing our solution to the totally distributed one, our solution has linear increase with the number of users and the totally distributed solution has exponential increase with the number of users. Our solution would be better none the less.

4 Mobility

Mobility in our architecture happens in two ways, both when the user changes cell on the 3GPP network, or routing areas, and then the mechanisms defined in 3GPP for these situations work as expected, and when the user changes from the area controlled by one distributed server to the area controlled by another distributed server. Is this last kind of mobility we discuss.

In our architecture, at the personal area network, the user is localized using the Java Location API (JSR-179), which, with correct hardware, can provide the user with 3D position and orientation.

When initiating a session, we, the central game device, provide the central back office server with our current position, in a SUBSCRIBE sip message with a special header defined by us with our 2D coordinates, and the server replies by inviting us to a session of augmented reality gaming in a distributed server adequate to our position by sending us a INVITE with SDP (Session Description Protocol) session information and a bounding box ,in a special header defined by us, that defines the region the distributed server controls.

If we have not changed position outside of that region in the meanwhile, we accept the invite, negotiate the SDP session protocol QoS and parameters, do what else is necessary to initiate the session and start the game session. If we stepped outside the bounds, we repeat the process by sending another SUBSCRIBE and not accepting the INVITE.

During the game, we update our position and orientation using JSR-179 and if we fall outside the 2D bounds of the region the current distributed server gived us, we close the session connections with this distributed server and SUBSCRIBE to another distributed server through the central back office server.

Each distributed server maintains a rectangular region for which it is responsible. Communication between distributed servers is only needed if an object falls in the frontier region for which it may be visible on the neighbourhood distributed servers.

5 Conclusions

We conclude that our solution is adequate for large scale mobile and pervasive augmented reality games. We have proven it is a more scalable architecture than alternative architectures and talked about its mobility aspects. Future work on this middleware platform will include optimization, further testing, and developments in the area of QoS – Quality of Service – , security, and management.

Aknowledgments. This work is being partially financed by the Portuguese Foundation for Science and Technology (FCT) and by the European FP6 CONTENT Network of Excelence (NoE).

6 References

[1] Kimmo Raatikainen, Henrik Bærbak Christensen, Tatsuo Nakajima, "Application Requirements for Middleware for Mobile and Pervasive Systems", Mobile Computing and Communications Review, Volume 6, Number 4, October 2002, pp. 16 – 24 , ACM Press

[2] Keith Mitchell, Duncan McCaffery, George Metaxas, Joe Finney, Stefan Schmid and Andrew Scott, "Six in the City: Introducing Real Tournament – A Mobile IPv6 Based Context-Aware Multiplayer Game", Proceedings of

NetGames'03, May 22-23, 2003, Redwood City, California, USA, pp. 91-100, ACM Press

[3] Hideyuki Tamura, Hiroyuki Yamamoto, and Akihiro Katayama, "Mixed Reality:Future Dreams Seen at the Border between Real and Virtual Worlds", Virtual Reality, November/December 2001, pp. 64 –70, IEEE

[4] Nokia – Developer resources (Forum Nokia), http://www.forum.nokia.com/, Accessed April 2004

[5] Sony Ericsson Developer World, http://developer.sonyericsson.com/, Accessed April 2004

[6] Pedro Ferreira, "Network Middleware for Large Scale Mobile and Pervasive Augmented Reality Games" in Proc. of the CoNext 2005 - ACM Conference on Emerging Network Experiment and Technology, pp. 242-243, CoNext 2005 - ACM Conference on Emerging Network Experiment and Technology, Toulouse, France, October-2005

[7] Pedro Ferreira, João Orvalho, Fernando Boavida, "Large Scale Mobile and Pervasive Augmented Reality Games", in Proc. of the EUROCON 2005 - The International Conference on "Computer as a Tool", pp. 1775-1778, Vol. 1, # 1, EUROCON 2005 - The International Conference on "Computer as a Tool", Belgrade, Serbia and Montenegro, November-2005

[8] Eila Niemelä, Teemu Vaskivuo, Agile Middleware of Pervasive Computing Environments, Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops (PERCOMW'04), 2004, IEEE

[9] P. Calhoun, J. Loughney, E. Guttman, G. Zorn, J. Arkko, "Diameter Base Protocol", RFC 3588, September 2003

[10] 3GPP TS 29.229 v7.0.0 , "3rd Generation Partnership Project; Technical Specification Group Core Networks and Terminals; Cx and Dx interfaces based on Diameter protocol; Protocol details (Release 7)", January 2006

[11] 3GPP TS 29.329 V7.0.0, "3RD generation Partnership Project; Technical Specification Group Core Network and Terminals; Sh interface based on the Diameter protocol; Protocol details (Release 7)", December 2005

[12] L.Ong., J. Yoakum, "An Introduction to the Stream Control Transmission Protocol (SCTP)", RFC 3286, May 2002

[13] Stewart, R., Xie, Q., Morneault, K. Sharp, C., Shwarzbauer, H., Taylor, T., Rytina, I., Kalla, M., Zhang, L. and V. Paxson, "Stream Control Transmission Protocol", RFC 2960, October 2008

[14] T.Dierks, E.Reskorla, "The Transport Layer Security (TLS) Protocol version 1.1", RFC 4346, IETF, April 2006

[15] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002

[16] M. Handley, V. Jacobson, "SDP: Session Description Protocol", RFC 2327, April 1998

[17] Pedro Ferreira, João Orvalho and Fernando Boavida , "Sixrm: Full Mesh Reliable Source Ordered Multicast", in Proc. of the SoftCom2006 - 14th International Conference on Software, Telecommunications & Computer Networks, SoftCom2006 - 14th International Conference on Software, Tellecommunications & Computer Networks, Split, Croatia, September 2006

[18] João Gilberto de Matos Orvalho, "ARMS – Uma plataforma para aplicações multimédia distribuídas, com qualidade de serviço", Phd Thesis, December 2000, DEI-FCTUC

[19] João Orvalho, Fernando Boavida, "Augmented Reliable Multicast CORBA Event Service (ARMS): a QoS-Adaptive Middleware", in Lecture Notes in Computer Science, Vol. 1905: Hans Scholten, Marten J. van Sinderen (editors), Interactive Distributed Multimedia Systems and Telecommunication Services, Springer-Verlag, Berlin Heidelberg, 2000, pp. 144-157. (Proceedings of IDMS 2000 – 7th International Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services, CTIT / University of Twente, Enschede, The Netherlands, October 17-20, 2000).