UNIVERSIDADE DE COIMBRA

Faculdade de Ciências e Tecnologia

Departamento de Engenharia Informática

NETWORK MIDDLEWARE FOR MOBILE AND PERVASIVE LARGE SCALE AUGMENTED REALITY GAMES

(MIDDLEWARE DE REDE PARA JOGOS DE REALIDADE AUMENTADA MÓVEIS E PERVASIVOS EM LARGA ESCALA)

By

Pedro Miguel da Fonseca Marques Ferreira

PhD thesis submitted for obtaining the degree of

Doutor em Engenharia Informática

May 2008

Thesis made under the supervision of

Prof. Doutor Fernando P. L. Boavida Fernandes

(Faculdade de Ciências e Tecnologia da Universidade de Coimbra)

And under the co-supervision of

Prof. Doutor João de Matos Orvalho

(Escola Superior de Educação de Coimbra, Instituto Politécnico de Coimbra)

This thesis was financed by FCT
(Fundação para a Ciência e Tecnologia)

Dedicated to my family and friends

ACKNOWLEDGMENTS

I specially acknowledge the help of my advisor Fernando Boavida and co-advisor João Orvalho for their precious input along the road that led to the writing of this thesis.

I also like to acknowledge the support of FCT – Portuguese Foundation for Science and Technology for their financial Support and the E-NEXT and CONTENT Networks of Excellence for their support when i needed to present papers on top conferences.

Last, but not least, I would like to acknowledge the support of my family and friends through these years of work, without which things would surely have been much more difficult.

TABLE OF CONTENTS

INDEX OF FIGURES

INDEX OF TABLES

ABBREVIATIONS

3D	-	3 (Three) Dimensions (in space)
3G	-	Third Generation mobile networks
3GPP	-	3rd Generation partnership Project
AAA	-	Authentication, Authorization, and Accounting
ACL	-	Asynchronous Connection-Less
AES	-	Advance Encryption Standard or Ryjandael
AH	-	Authenthication header
AKA	-	Authentication and Key Agreement
AN	-	Access Network
ANSI	-	American National Standards Association
API	-	Application Programming Interface
AR	-	Augmented Reality
ARMS	-	Augmented Reliable corba Multicast System
ARMS	-	Augmented Reliable corba Multicast System
AS	-	Anti spoofing
AS	-	Application Server
AuC	-	Authenthication Center
AVP	-	Attribute Value Pair
BD	-	Bluetooth device
BGCF	-	Breakout Gateway Control Function
BSC	-	Base Station Controller in BSS system
BSS	-	Base Station Subsystem
BTS	-	Base Transceiver Stations in BSS system
CBC	-	Cell Broadcast Center
CBC	-	Cipher Block Chaining mode
CC	-	Country Code
CDMA	-	Code Division Multiple Access
CGI	-	Cell Global Identity

CI	-	Cell Identity
CLDC	-	Common Limited Device Configuration
CN	-	Core Network
COPS	-	Common Open Policy Service Protocol
CORBA	-	Common Object Request Broker Architecture
CS	-	Circuit Switched
CSCF	-	Call Session Control Function
DA	-	Directory Agent
DCON	-	Dissemination Controls
DEA	-	Data Encryption Algorithm defined in DES (DEA=DES)
DES	-	Data Encryption Standard
DES-EDE2	-	Protocol variation of Triple DES
DES-EDE3	-	Protocol variation of Triple DES
DES-EEE2	-	Protocol variation of Triple DES
DES-EEE3	-	Protocol variation of Triple DES
DGPS	-	Differential GPS
DiffServ	-	Differentiated Services
DRM	-	Digital Rights Management
DSA	-	Digital Signature Algorithm
DS-CDMA	-	Direct Sequence CDMA
DSS	-	Digital Signature Standard
DSSS	-	Direct Sequence Spread Spectrum
ECB	-	Electronic Codebook mode
ECC	-	Elliptic curve cryptography
EDR	-	Enhanced Data Rate
EIR	-	Equipment Identity Register
ESP	-	Encapsulating Security Payload
FDD	-	Frequency Division Duplex
GALILEU	-	European positioning system contender to GPS

GGSN	-	Gateway GPRS Support Node
GLMC	-	Gateway Mobile Location Center
GPRS	-	General Packet Radio Service
GPS	-	Global Positioning System
GSM	-	Global System for Mobile Communications
GSN	-	GPRS Support node (GGSN or SGSN)
HLR	-	Home Location Register
HMD	-	Head Mounted Display
HSS	-	Home Subscriber Server
I-CSCF	-	Interrogating CSCF
IETF	-	Internet Engineering Task Force
IKE	-	Internet Key Exchange
IMEI	-	International Mobile Equipment Identity
IMS	-	IP Multimedia Subsystem
IMSI	-	International Mobile Subscriber identity
IM-SSF	-	IP Multimedia Service Switching Function
IMT-2000	-	International Mobile Telecommunications 2000
IntServ	-	Integrated Services
IP	-	Internet Protocol
IPSec	-	IP Security
Ipv4	-	Internet Protocol Version 4
Ipv6	-	Internet Protocol Version 6
IS-95	-	Interim Standard 95
ISIM	-	IP Multimedia Services Identity Module
IT	-	Information Technologies
J2ME	-	Java 2 Micro Edition
J2ME	-	Java 2 Micro Edition
J2SE	-	Java 2 Standard Edition
L2CAP	-	Logical Link Control and Adaptation Protocol

LA	-	Location Area
LAC	-	Location Area Code
LADGPS	-	Local Area DGPS
LAI	-	Location Area Identifier
LAN	-	Local Area Network
LAPI	-	J2ME Location API
LCS	-	Location Services
LMP	-	Link Manager Protocol
LMSI	-	Local Mobile Station Identifier
LSP	-	Locally Significant Part
M3UA	-	MTP3 User Adaptation Layer
MAC	-	Message authentication code
MAP	-	Message Application Part
MAPSec	-	MAP Security
MCC	-	Mobile Country Code
ME	-	Mobile Equipment
MGCF	-	Media Gateway Control Function
MGW	-	Media Gateway
MIDP	-	Mobile Information Device Profile
MMAPI	-	Mobile Media API
MNC	-	Mobile Network Code
MR	-	Mixed Reality
MS	-	Mobile Station
MS	-	Mobile Station
MSIN	-	Mobile Subscriber identification Number
MSISDN	-	Mobile Subscriber ISDN
MSRN	-	Mobile Station Roaming Number
MT	-	Mobile Termination
MTP	-	Message Transfer Part

NDC	-	National Destination Code
NDS/IP	-	Network domain security for IP based protocols
NE	-	Network Element
NIST	-	National Institute for Standards and Technology
NMSI	-	National Mobile Subscriber Identity
Node-B	-	Equivalent to BTS in RNS system
OAM	-	Operation, Administration and Management
OBEX	-	Object Exchange protocol
OSA-SCS	-	Open Services Architecture Service Capability Server
PAN	-	Personal Area Network
P-CSCF	-	Proxy-CSCF
PDN	-	Packet Data Network
PDP	-	Packet Data Protocol
PIN	-	Personal Identification Number
PKCS	-	Public Key Cryptography Standards
PLMN	-	Public Land Mobile Network
PN	-	Pseudo Noise
PPP	-	Point to Point Protocol
PPS	-	Precise Positioning Service
PS	-	Packet Switched
PSTN	-	Plain old Switching Telephone Network
P-TMSI	-	PS domain TMSI
QoS	-	Quality of Service
QoS	-	Quality Of Service
RA	-	Routing Area
RAC	-	Routing Area Code
RAI	-	Routing Area Identifier
RC2	-	Rivest’s Cipher 2 Encryption Standard
RC4	-	Rivest’s Cipher 4 Encryption Standard

RC5 -	Rivest’s Cipher 5 Encryption Standard
RFCOMM-	Radio Frequency Communications
RNC -	Radio Network Controller (Equivalent to BSC in the RNS system)
RNS -	Radio Network Subsystem (also known as UTRAN)
RNTI -	Radio Network Temporary Identity
RRC -	Radio Resource Control
RSA -	Rivest Shamir Adleman algorithm for asymmetric cryptography
RSVP -	Resource Reservation Protocol
RTP -	Real Time Protocol
S/MIME -	Secure Multipurpose Internet Mail Extensions
SA -	Select availability/ Service Agent
SCO -	Synchronous Connection Oriented
S-CSCF -	Serving CSCF
SCTP -	Stream Control Transmission Protocol
SDK -	Software Development Kit
SDP -	Service Discovery Protocol (Bluetooth) / Session Description Protocol
SEG -	Security gateway
SENSACT-	SENSors and ACTuators API
SGSN -	Serving GPRS Support Node
SIGTRAN-	Signaling and Transport Working Group of IETF
SIP AS -	SIP protocol Application Server
SIP -	Session Initiation Protocol
Sixrm -	The Sixrm reliable multicast protocol.
SLF -	Subscription Locator Function
SLM-API-	Salutation manager API
SLM-TI -	Salutation manager Transport Interface
SLP -	Service Location Protocol
SMS-GMSC-	SMS gateway MSC
SMS-IWMSC-	SMS Interworking MSC

SN	-	Subscriber Number
SNR	-	Serial Number
SPS	-	Standard Positioning Service
SS7	-	Signaling System 7
SSDP	-	Simple Service Discovery Protocol (SSDP)
SSL	-	Secure Sockets Layer
STF	-	Status Transmission Framework
STFPAN	-	STF Personal Area Network middleware
STFSERVER	-	STF Server middleware
SUA	-	SCCP User Adaptation Layer
TAC	-	Type Allocation Code
TCP	-	Transmission Control Protocol
TCS BIN	-	Telephony Control Specification Binary protocol
TDD	-	Time Division Duplex
TE	-	Terminal Equipment
TLS	-	Transport Layer Security
TMN	-	Telecommunications Management Network
TMSI	-	Temporary Mobile Subscriber Identity
TOA	-	Time of Arrival
Triple DES	-	Triple Data Encryption Standard
UA	-	User Agent
UDP	-	User Datagram Protocol
UE	-	User Equipment
UICC	-	UMTS Integrated Circuit Card
UMTS	-	Universal Mobile Telecommunications System
uPnP	-	Universal Plug and Play
URA	-	UTRAN Registration area
URL	-	Uniform Resource Locator
USIM	-	Universal Subscriber Identity Module

UTC	-	Coordinated Universal Time
UTRAN	-	Universal Terrestrial Radio Access Network
VLR	-	Visitor Location Register
VR	-	Virtual Reality
WAAS	-	Wide Area Augmentation System
WADGPS	-	Wide Area DGPS
WAP	-	Wireless Application protocol
WCDMA	-	Wideband Code Division Multiple Access

ABSTRACT

Mark Weiser theorized about a new kind of computing, called ubiquitous or pervasive computing, where specialized elements of hardware and software would be so ubiquitous no one would notice their presence. According to Mark Weiser the technology required for ubiquitous computing would come in three parts: cheap, low power computers including equally convenient displays, software for ubiquitous applications and a network that ties them all together.

In the current decade we will see the merger of telecommunications and IT worlds. The Internet Protocol (IP) is the network layer protocol in the 3GPP specifications, and the current trend in developing new telecommunications networks is to utilize internet protocols. So, the network that ties all things together is now possible. But there are many issues under study in the internet community. These are mobility, quality of service, security, management of networks and services, discovery, ad - hoc networking and dynamic configuration, and geospatial location.

A significant requirement of pervasive applications is fast service development and deployment, 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, however, mobile gaming applications face issues that are different from fixed network applications. These issues include fluctuating connectivity, quality of service and host mobility. Another issue is how to manage game state consistency with a dynamic mobile networked environment in which devices may be physically close but topologically distant. Further yet, there is the issue of how to manage multiple wireless network connections such as, for example, GPRS and IEEE 802.11 at the same time.

Augmented reality extends reality with virtual elements while keeping the computer in an assistive, unobtrusive role. It is possible to create games that place the user in the physical world through geographically aware applications. The latest mobile phones are being equipped with GPS receivers and there are software and hardware tendencies from the largest manufacturers to equip mobile phones with more advanced context - aware technology. All the latest mobile phones are equipped with cameras and some of the latest ones are coming with some form of 3D rendering technology. Bluetooth technology and increasing miniaturization makes possible, in the near future, specialized pervasive equipment for augmented reality. The opportunity for some cheap augmented reality is here.

To our knowledge, there was no specialized network middleware solution for large-scale mobile and pervasive augmented reality games.

The main objective of this PhD was the creation of a network middleware for mobile communications that will enable integrated large-scale augmented reality applications to be built around it.

The middleware that was created evolved from previous work from the candidate in the area of interactive distributed multimedia, more specifically in state transmission for a collaborative virtual environment middleware platform, the Status Transmission Framework (STF). This platform extended ARMS - Augmented Reliable corba Multicast System - with capabilities for the handling of state transmission for distributed collaborative virtual environments.

In this context mechanisms were studied, proposed and evaluated to deal with issues ranging from Internet issues to architectural issues.

The internet issues are: Mobility (such as fluctuating connectivity, host mobility and handling of multiple simultaneous network connections), Quality of Service (QoS - such as minimizing delay and jitter, and reliability), security (such as authentication and prevention of cheating), management of networks and services, discovery, ad-hoc networking and dynamic configuration, geospatial location and orientation.

The architectural issues are: scalability; consistency; multimedia data heterogeneity; data distribution and replication.

The main contribution of this thesis was the creation of a network middleware for mobile and pervasive large scale augmented reality games, based in Java.

As sub contributions, we have the development of the total architecture of the system, the development of the SENSACT API, a API for sensors and actuators in a PAN based on Bluetooth network and Java CLDC, in connection with the STF PAN API. We also have the development of the Sixrm Reliable Multicast protocol (part of the STF Server API). As for QoS we developed the QoS API, for the distributed servers (part of the STF Server API) and for the clients (part of the STF PAN API), and for security we developed the Security Architecture for the whole system. We also have proven system scalability.

RESUMO

Mark Weiser teorizou sobre um novo tipo de computação, chamada computação ubíqua ou computação pervasiva, onde elementos especializados de software e hardware seriam tão ubíquos que ninguém notaria a sua presença. De acordo com Mark Weiser a tecnologia requerida para a computação ubíqua viria em três partes: Computadores baratos e de baixo consumo energético incluindo ecrãs igualmente convenientes, software para aplicações ubíquas e uma rede que os ligasse todos.

Na década corrente iremos ver a junção dos mundos da telecomunicação e da tecnologia da informação (IT). O protocolo da Internet (IP) é o protocolo de camada de rede nas especificações 3GPP, e a tendência corrente ao desenvolver novas redes de telecomunicações é utilizar protocolos de Internet. Logo, a rede que liga todas as coisas juntas é agora possível. Mas existem muitos problemas em estudo na comunidade da Internet. Estes são por exemplo a mobilidade, a qualidade de serviço (QoS), a segurança, a gestão de redes e serviços, a descoberta de serviços, redes ad-hoc e configuração dinâmica, e localização e orientação geo-espacial.

Um requisito fundamental de aplicações pervasivas é um rápido desenvolvimento e instalação de serviços, o que implica a introdução de vários frameworks e plataformas de serviços. Para isto o middleware é uma solução comum. Os benefícios da utilização de middleware são o modelo melhorado de programação, e o esconder de muitos detalhes de implementação, que fazem o desenvolvimento de aplicações baseadas em middleware muito mais rápido.

Está agora a tornar-se muito claro que o entretenimento, e mais especificamente o jogo móvel, serão uma das aplicações mais utilizadas nas redes wireless do futuro. No entanto, as aplicações de jogos móveis enfrentam problemas diferentes das suas congéneres de rede fixa. Estes problemas incluem conectividade flutuante, qualidade de serviço, e mobilidade do host. Outro problema é como gerir a consistência do estado do jogo com um ambiente móvel dinâmico em que os aparelhos podem estar fisicamente perto mas topologicamente distantes. Mais ainda existe o problema de como gerir múltiplas ligações de rede como, por exemplo, GPRS e IEEE 802.11 ao mesmo tempo.

A realidade aumentada estende a realidade com elementos virtuais enquanto mantém o computador num papel de assistente, não obstrutivo. È possível criar jogos que colocam o utilizador no mundo físico através de aplicações que sejam sensíveis á localização. Os últimos aparelhos móveis estão a ser equipados com aparelhos GPS e existem tendências de software e hardware dos maiores fabricantes para equipar os telefones móveis com ainda mais avançada tecnologia sensível ao contexto. Todos os últimos telefones móveis são equipados com câmaras e alguns dos últimos vêm com alguma forma de tecnologia 3D. A tecnologia Bluetooth e a miniaturização que cada vez é maior tornarão possível, num futuro próximo, equipamento especializado para realidade aumentada. A oportunidade para realidade aumentada barata está aí.

Ao nosso conhecimento, não existe nenhuma solução de middleware de rede para jogos de realidade aumentada móvel e pervasiva em larga escala.

O principal objectivo deste doutoramento é a criação de um middleware de rede para comunicações móveis que permita a construção de aplicações de realidade aumentada de larga escala.

O middleware criado evoluiu de trabalho prévio do candidato na área de multimédia interactivo e distribuído (no seu mestrado), mais especificamente em transmissão de estados numa plataforma para ambientes virtuais colaborativos, a Status Transmission Framework Versão 1.0. (STF). Esta plataforma estendeu o ARMS – Augmented Reliable corba Multicast System – com capacidades para o manuseamento de transmissão de estados para ambientes virtuais distribuídos e colaborativos.

Neste contexto foram estudados, propostos e avaliados mecanismos para tratar de problemas desde problemas de Internet a problemas arquitecturais.

Os problemas de Internet são: Mobilidade (tais como a conectividade flutuante, mobilidade do host e a gestão de conexões múltiplas do host), qualidade de serviço (QoS – tais como minimizar atraso e jitter, e maximizar a confiabilidade), segurança (como autenticação e prevenção de batota), gestão de redes e serviços, descoberta de serviços, redes ad-hoc e configuração dinâmica, localização e orientação geo-espacial.

Os problemas arquitecturais são: escalabilidade; consistência; heterogeneidade dos dados multimédia; distribuição e replicação de dados.

A principal contribuição desta tese é a criação de um middleware de rede para a criação de jogos de realidade aumentada móvel e pervasiva em larga escala, baseado em Java.

Como subcontribuições, temos o desenvolvimento da arquitectura total do sistema, , e a arquitectura detalhada. Temos também o desenvolvimento da API SENSACT, uma API para sensores e actuadores numa PAN baseada numa rede Bluetooth e Java CLDC, em conexão com a API STF PAN. Temos também o desenvolvimento do protocolo de multicast confiável Sixrm (parte da API STF Server). Para a qualidade de serviço desenvolvemos a API QoS, para os servidores distribuídos (parte da STF Server API) e para os clientes (parte da STF PAN API) e para segurança desenvolvemos a arquitectura de segurança para o sistema de middleware completo. Provámos também a escalabilidade do sistema.

INTRODUCTION

FRAMEWORK

According to Mark Weiser in [1], “The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it.”

According yet to Mark Weiser in [1], and his colleagues at Palo Alto Research some time ago, the idea of The Personal Computer itself is misplaced and the and that the vision of laptop machines, “dynabooks” and “knowledge navigators” is only a transitional step towards achieving the real potential of information technology.

In the same article Weiser continues to argue that such machines can now truly make computing an integral, invisible part of people’s lives, and they were trying at Palo Alto Research to conceive a new way of thinking about computers, one that takes into account the human world and allows the computers themselves to vanish in the background.

Augmented reality tries to extend the real world with virtual objects while maintaining the computer in an assistive, unobtrusive role, thus trying to keep in line with the pervasive computing objective of invisibility.

Augmented Reality is a sub concept of Mixed Reality [4]. AR augments the real world with synthetic electronic data. Augmented Virtuality (AV) enhances the virtual world with data from the physical (real) world. Mixed reality is a term that covers a continuum from AR to AV.

An example project of Mixed Reality is exposed in [4] by the Mixed Reality Systems Laboratory.

Both Mixed Reality and Augmented Reality are different than Virtual Reality, which is not pervasive at all (as we discussed in the previous section), and just deprives the user from the real world, completely evolving it in a virtual world.

It is quickly becoming clear that entertainment will be one of the killer applications of future wireless networks [3]. More specifically mobile gaming was predicted to be worth more than 1.2 billions of American dollars in 2006 in the United States market alone. However mobile applications faces issues that are different from fixed network applications, including fluctuating connectivity, network Quality of Service and host mobility.

In the times before computers existed, the games people played were designed and played out in the physical world with the use of real world properties, such as physical objects, our sense of space, and spatial relations [30].

Pre-computer day’s game interactions consisted of two elements: human to human interaction and human to physical world interaction. In the current times, due to their higher level of attractiveness to game players, computer games have become the dominant form of entertainment.

Computer games motivate the players by bringing them more challenge, curiosity and fantasy, which are the three main elements contributing to fun in games. They provoke curiosity because they are usually designed with an optimal level of information complexity, they create fantasy by creating the illusion of the player being immersed in an imaginative virtual world with computer sound and graphics, and they create challenge by having goals typically more interactive than traditional games.

However, there is a also a new tendency in gaming called pervasive games,, where the real world is coming back to the computer entertainment field stressing the pervasive and ubiquitous nature of these games: Pervasive games are no longer confined to the domain of the computer (the virtual domain), but integrate the physical and social aspects of the real world.

Pervasive gaming is a relatively new field but we can already identify several unique types of pervasive games, each focusing on different aspects of the gaming experience.

To our knowledge, there is no specialized network middleware solution for large-scale mobile and pervasive augmented reality games, and we feel this is a field worth exploring. Not only pervasive technology is becoming more and more important and its use widespread, but also pervasive games are beginning to be built and used by gamers all around the world. Augmented reality is an interesting concept for games and pervasive computing, bringing together the real world with a virtual world that exists only in the game. Enabling it in large scale brings true mobility to the solution, enabling the players to play alone or against one another wherever they are, as long as they are covered by the system. Building a network middleware to help build this pervasive and mobile large scale augmented reality game applications is important because in this way we hope to contribute with a solution that solves most of the problems these applications face in terms of software, excluding the ones related to graphic presentation.

OBJECTIVES

The main objective of this PhD is the creation of a network middleware for mobile communications that enables integrated large-scale mobile and pervasive augmented reality applications to be built around it. It should be a highly scalable middleware that should address issues related to network programming of large scale augmented reality game applications for mobile and pervasive environments.

The middleware created evolves from previous work from the candidate in the area of interactive distributed multimedia, more specifically in state transmission for a collaborative virtual environment middleware platform, the Status Transmission Framework (STF)[7][8]. This platform extended ARMS - Augmented Reliable CORBA Multicast System [9] -with capabilities for the handling of state transmission for distributed collaborative virtual environments.

In this context mechanisms were studied, proposed and evaluated to deal with issues ranging from Internet issues to architectural issues.

The internet issues are:

- Mobility, such as fluctuating connectivity, host mobility and handling of multiple simultaneous network connections;
- Quality of Service (QoS), such as minimizing delay and jitter ,and reliability;
- security, such as authentication and prevention of cheating;
- management of networks and services;
- discovery;
- ad-hoc networking and dynamic configuration;
- geospatial location and orientation.

The architectural issues are

- scalability;

- consistency;
- multimedia data heterogeneity;
- data distribution and replication.

CONTRIBUTIONS

The main contribution of this thesis was the creation of a network middleware for mobile and pervasive large scale augmented reality games, based in Java.

As sub contributions, we had:

- The development of the total architecture of the system. The general PhD vision was published in [10] (see Annex 1), with the general architecture being published in [11] (see Annex 2), and the detailed architecture with results published in [12] (see Annex5).

- The development of the SENSACT API, a API for sensors and actuators in a PAN based on Bluetooth network and Java CLDC, in connection with the STF PAN API, published in [13] (see Annex 3).

- The development of the Sixrm Reliable Multicast protocol (part of the STF Server API), published in [14] (See Annex 4).

- The proof that the architecture chosen is the most scalable architecture from the set of possible alternative architectures that could have been chosen to develop the middleware system, published in [17] (See Annex 6).

- The development of the QoS API, for the distributed servers (part of the STF Server API) and for the clients (part of the STF PAN API), published in [15] (See Annex 7).

- The development of the Security Architecture for the whole system, published in [16] (See Annex8).

STRUCTURE OF THIS THESIS

This PhD thesis is structured as follows: First, we have an abstract (Resumo, in portuguese), then, the introduction (this section), then we continue with the state of the art and related work chapter, and go on with the proposal and experimental development chapter, the evaluation chapter, followed finally by the conclusions, by the references, and, to close, the various annexes (which are the papers published related to the thesis work, and the classes and interfaces of the various APIs of the middleware).

2. STATE OF THE ART AND RELATED WORK

2.1. PERVASIVE COMPUTING

2.1.1. INTRODUCTION

According to Mark Weiser, Pervasive computing is different from Ubiquitous Computing in many ways. “Ubiquitous computing” in this context does not mean just computers that can be carried to the beach, jungle or airport. Even the most powerful notebook computer, with access to a worldwide information network, still focuses attention on a single box. By analogy with writing, carrying a super laptop is like owning just one very important book. Customizing this book, even writing millions of other books, does not begin to capture the real power of literacy, and although ubiquitous computers may use sound and video in addition to text and graphics, that does not make them “multimedia computers.” Today’s multimedia machine makes the computer screen into a demanding focus of attention rather than allowing it to fade into the background.

Diametrically opposed to Mark Weiser’s vision of Pervasive Computing is the notion of virtual reality, which attempts to make a world inside the computer. Users normally wear special goggles that project an artificial scene onto their eyes; they wear gloves or even bodysuits that sense their motions and gestures so that they can move about and manipulate virtual objects. Although it may have its purpose in allowing people to explore realms otherwise inaccessible— the insides of cells, the surfaces of distant planets, the information web of data bases—virtual reality is only a map, not a territory. It excludes desks, offices, other people not wearing goggles and bodysuits, weather, trees, walks, chance encounters and, in general, the infinite richness of the universe. Virtual reality focuses an enormous apparatus on simulating the world rather than on invisibly enhancing the world that already exists. Indeed, the opposition between the notion of virtual reality and ubiquitous, invisible computing is so strong that many people use the term “embodied virtuality” to refer to the process of drawing computers out of their electronic shells. The “virtuality” of computer-readable data—all the different ways in which they can be altered, processed and analyzed—is brought into the physical world.

Still according to Mark Weiser, the technology for this will come in three parts: cheap, low-power computers that include equally convenient displays, software for ubiquitous applications and a network that ties them all together.

It’s clear to see that currently, we already have the cheap low power computers needed for pervasive computing and can even easily build new ones for specialized purposes. Examples are the many PDAs and mobile phones on the market. There is also software for ubiquitous applications that can be built. The network that ties all things together is the internet or the various wireless networks that support the IP protocol. One of these networks is the 3G (Third Generation) Mobile Network standardized by 3GPP.

Pervasive computing is all about making computers disappear in the background, while keeping or increasing their usefulness. The implications in terms of application potential and in terms of security, for example, are enormous.

But most importantly, while using pervasive computers, people will continue to place importance more on other people than in computers, as it should be. Computers should be a tool, nothing more.

2.1.2. ESSENTIAL ISSUES

Essential issues for pervasive computing are location and scale [1]. Pervasive computers must know where they are, to be able to adequately provide its service. Their scale (size) must also be adequate for the service provided. There is no question that the scale of a mobile phone is adequate for its function, or that the scale of an electronic board is adequate for its function.

Since Weiser's vision, many pervasive computing projects have emerged at major universities and industry thought the world.

Examples of these projects include Project Aura at Carnegie Mellon, Oxygen at MIT, Portolano at University of Washington and Endeavour at the UC Berkley, besides many more other projects, examples of some we include in the section of related work.

The evolution that led to pervasive computing started in the 1970's. Two related and previous distinct earlier steps in that evolution are distributed systems and mobile computing [18].

Some technical problems faced by pervasive computing are similar to the ones found previously and solved successfully by these earlier steps, others are new and new solutions have to be found.

In the case of distributed systems, this field arose of the intersection between personal computers and local networks [18].

Through the 1970's to the 1990's a conceptual framework and algorithmical base was created for all work involving two or more computers interconnected by a network. This includes remote communication, fault tolerance, availability, remote information access and security.

Mobile computing made its appearance in the early 1990's as full function laptop computers joined wireless LANs and led researchers to face the problems that arise when building a distributed system with mobile clients.

Results achieved in mobile computing can be grouped in mobile networking, mobile information access, support for adaptative applications, system level energy saving techniques, and location sensitivity broad areas.

Pervasive computing includes the issues of distributed computing and mobile computing, but goes further, including four more research issues [18]. These issues are smart spaces, uneven conditioning, invisibility, and localized scalability.

A space is any physical space that can be defined and can be acted upon such as a room, corridor, or any given area. By embedding computers in the physical space, one builds a smart space.

Invisibility means the complete disappearance of pervasive technology from user's consciousness and it's the ideal for pervasive computing expressed by Weiser's vision [1].

The smart space's growth in terms of sophistication has implications in terms of bandwidth, energy and distraction for the wireless mobile user that the presence of multiple users will only complicate further.

Scalability, and localized scalability, are thus important and critical problems in pervasive computing.

Uneven conditioning means developing methods of masking the different computing environments available. Many levels of integration of technology will probably exist in different smart spaces (pervasive computing environments) and ways of masking these differences must be developed at the penalty of loosing invisibility.

Other issues are pro-activity, the ability to actively look-ahead on a task before a user needs the result of it, and self tuning, automatically adjusting the behaviour to fit the current circumstances.

Pro-activity must be balanced with transparency, to maintain the objective of invisibility of the whole pervasive system.

It may be also necessary to use adaptation, when a resource (such as network bandwidth) is necessary in a higher quantity than the one that it's available. How the adaptation strategy is chosen and how the client and user manage adaptation are questions that must be answered [18].

To manage energy use efficiently, it's important for the hardware's operating system to use high level energy management since sophisticated capabilities such as proactivity and self-tuning increase the energy demand of software. By itself, advances in battery technology and low-power circuit design cannot be enough to compensate the needed energy.

For this, the operating system could use energy aware memory management and energy aware adaptation. One has to take into account how much the energy management techniques influence the invisibility of the system, for example.

Another question is how powerful has a mobile client to be for it to be useful in pervasive computing. It must be a thick or a thin client? A thick client is a powerful client, and a thin client is a minimal client. Many applications will require one or another. For a given application, the minimal acceptable thickness of a client is determined by the worst case environmental conditions under which the application must run satisfactorily [18]. A thin client may be enough if one can always count with a sufficient bandwidth, low-latency wireless communication to a near computing infrastructure, and if batteries could be recharged or replaced

Also, for a pervasive system to strive to be minimally intrusive, and therefore invisible, it must be context aware. It must be able to modify its behaviour based on information about the user's environment and current attributes, such as its location, for example. And many other possible attributes. Things that can be sensed by adequate hardware like sensors make part of the context.

Security is an important issue in pervasive computing, and it's largely more important in pervasive computing than in distributed and mobile computing.

The amount of information the pervasive system, because of its context awareness, has on its users it's enormous. The need for privacy and trust is as such greatly increased.

This decade we will see the merger of telecommunication and IT worlds, being that the current trend in developing new telecommunication networks is to utilize the internet protocol (IP) [12]. An example of this is the 3GPP specifications.

Issues under study on the internet community and in other standardization bodies include mobility, Quality of Service (QoS), management of networks and services, discovery, ad-hoc networking and dynamic configuration, and geospatial location.

An important implication of the introduction of various service application frameworks and platforms is the requirement of fast service development and deployment, which can be achieved through the use of specialized middleware.

2.1.3. MIDDLEWARE

The properties of pervasive middleware can be subdivided in properties referent to the pervasive environment where the middleware runs and quality attributes of the middleware [14].

As for properties that refer to the pervasive environment, we have application aware adaptation (a part of context awareness, it means the collaboration between system infrastructure and individual applications), mobility (divided into actual, physical and virtual mobility [14]), integration architecture (the way the system is integrated).

As for quality attributes of the middleware, we have interoperability, portability and adaptability, robustness and agility, and fidelity.

2.1.4. CONTEXT AWARENESS

There are various research challenges and opportunities of instrumenting the physical world with pervasive networks of sensor rich, embedded computation, fulfilling two of the objectives of pervasive computing according to Weiser: Ubiquity (injecting computation into the physical world with high spatial density), and Invisibility (By Having the nodes and sets of nodes operate in an autonomous fashion).

To make pervasive computing itself pervasive, we need reusable building blocks that can help us moving away from the specialized instrumentation of each of particular service environment and build toward building reusable techniques for sensing, computing and manipulating the physical world [19].

The physical world presents an incredibly rich set of input modalities, including acoustics, image, motion, vibration, heat, light, humidity, pressure, ultrasound, radio, magnetic and many more exotic modes.

Traditionally, the engineering required to collect and analyze all this data was complex.

The opportunity lies ahead in the ability to easily deploy flexible sensing, computation, and actuation capabilities into our physical environments such that the devices themselves are general purpose and can support a variety of application types.

2.1.5. SYSTEM CHALLENGES

There are of course, a number of system challenges in pervasive computing, which must be taken into account.

The first one is the immense scale of the system. A vast number of small devices will make part of those systems and to achieve dense instrumentation of these complex physical systems, these devices must scale down to extremely small volume. Applications should be formulated in terms of immense numbers of them.

It is expected that in 5 to 10 years, complete systems with computing, storage, communication, sensing, and energy storage could be as small as a cubic millimeter [19]. Of course each aspects capacity will be limited. Fidelity and availability will come from the quantity of partially redundant measurements and their correlations, not the individual's components quality and precision.

The second one is limited access. Many devices will be placed in environments where it is difficult to connect them with wires or are plainly inaccessible.

These systems individual elements become unattended, and resource constrained. Much of the communication used will be wireless and the nodes will have to use the energy they have on-board and harvested (batteries or solar cells). If they are inaccessible, they must operate without human intervention.

The third one is extreme dynamics. By virtue of the nodes and the system as a whole being closely tied to the ever changing physical word, these systems will experience Extreme Dynamics. They are greatly affected by the environment and, as a rule, when something happens, many changes happen at the same time.

2.1.6. TAXONOMIES

It is possible to define taxonomy of pervasive systems and applications [19]. Taxonomies are one step toward identifying common building blocks so that we can identify and foster reusable and parametrizable features.

One can divide the space of physically embedded systems along the critical dimensions of space and time. Within these dimensions, we are interested in scale, variability, and autonomy. One addresses these dimensions with respect to the environmental stimulus being captured and the system elements.

Starting with scale, the scale of special and temporal sampling and its extent are important factors in determining system emphasis. The finer grain the sampling more important are innovative collaborative signal processing techniques. Also, as the system extent grows, the ability to configure and control the environment becomes impossible. So systems intended to operate over extended time frames and regions of space must emphasize techniques for self-organization. With high density come many challenges and opportunities for self-configuration, not encountered by low-density systems.

Continuing with sampling, sampling scale is ultimaly dictated by the physical phenomena measured. The sampling scale is determined both by the phenomena and by the application.

As for extent, the spatial and temporal extent of systems also varies widely. At the top, there are the environmental modelling systems, which can span on the order of tens of thousands of meters. Most existing and planned pervasive computing systems are an order of magnitude, or smaller, such as what are needed to cover a building or a room. There are also elements of pervasive computing systems that extend to lower end of the spectrum, such as reconfigurable fabric, that a user can wear or that we can deploy to monitor structure or machinery surfaces.

As for density, it is a measure of sensor nodes per footprint of input stimuli. Higher density systems provide greater opportunities for exploiting redundancy to eliminate noise and extend system lifetime. In a similar way, where density is enough, such as to allow oversampling to occur, nodes can go to sleep for very long periods of time and thereby extend coverage over time.

As for variability, this is a differentiating characteristic of many systems and associated designs. It can apply to system elements or the phenomena being sensed.

Relatively static systems put the emphasis on design time optimization while more variable systems must use runtime self-organization and may be fundamentally limited since they are both varied and long-lived.

As for structure, a structure may be ad-hoc or engineered. The difference lies in the variability of the system composition. At one extreme (engineered) is a system that models things like buildings, bridges, etc…, and on another, (ad-hoc), a system that models sensor networks deployed in remote regions to study bio-diversity, for example.

As for task, the variability in a system's Task determines how much we can optimize a system for a single mode of operation.

As for space, variability in space (or mobility), applies both to system nodes and phenomena. A system designed to manage the physical environment of a stationary human user faces very different challenges that a system designed to manage the physical environment of a human moving quickly.

As for autonomy, the higher the overall systems autonomy, the less the human involvement and the greater the need for extensive and sophisticated processing in the system. We can have systems that just process the sensorial stimulus and deliver them to the applications and to the user, systems that sensor the stimulus and then activate a complete set of events in an event system for the applications to consume and alert the user, up to full autonomy.

Perhaps more than any other dimension, autonomy is most significant in moving us from embedding instruments to embedding computation in our physical world.

As for modalities, truly autonomous systems depend on multiple sensor modalities. Different ones provide noise resilience to one another and can combine to eliminate noise and identify anomalous measurements.

And then there is systems complexity, since greater system Autonomy also causes greater complexity in the computational model.

2.1.7. DEVICES

A major theme in Weiser's work [1] was the creation of small devices used in a larger intelligent environment (Smart Space). The Xerox ParcTab [1] provided limited display, user input, and infrared communication in a palm-sized unit.

Currently, we have many more capable gadgets for pervasive computing and can build specialized new ones for specialized applications.

Small, attachable RF identification (RFID) tags helped determine the approximate position and identity of the tagged object in a space equipped with specialized readers.

We see many consumer devices in the palm form factor possessing roughly the processing and storage capabilities of the early 1990's, some more. Personal Digital Assistants (PDAs), and pocket PCs have gained wireless connectivity either as variations of pager networks or 802.11 wireless LANs. There is now an increased tendency to incorporate Bluetooth short-range wireless technology in handheld devices. Numerous other attractive radio technologies are on the horizon.

In recent times, these devices have gained a rich set of input modes besides user buttons and knobs [19]. Most have acoustic input and output, and some incorporate accelerometers to detect gestures, orientations or video input.

At the same time, many cell phones have gained internet browsing capabilities, and PDAs have gained cell phone capabilities for data and voice access.

Soon all cell phones will know where they are thanks to a GPS, which will bring a degree of location awareness into various personal computing devices.

Increased miniaturization raises the possibility that every tiny sensor and controller having its own processing and communicating capabilities, such that the aggregate performs sophisticated functions.

Sensors and actuators have undergone a revolution with the advent of the MEMS – Microelectricomechanical Systems technology that allows for accelerometers, barometers, and movable mirrors and many more such devices to be constructed at minute size on a silicon ship using lithographic processes similar to those for integrated circuits.

Improved equipment and technology results in smaller MEMS devices, lower power consumption, and improved device performance. A strong correlation exists between the improvements and feature size.

The node subsystems size and performance improvements reduce power consumption and allow a corresponding decrease on the size and cost of the power supply as well.

Substantial improvements in battery technology, with improved form factor, storage density and recharging, as well as the emergence of alternative storage devices, such as fuel cells and energy-harvesting mechanisms, have also been happening.

Several research groups are exploring how to build intelligent, information-rich physical environments by deploying many small, deeply integrated nodes.

Core challenges involve designing systems that operate at low power consumption, storing and obtaining that power, organizing nodes to form larger networks, and deploying applications over fine-grain networks.

At the far extreme, researchers have conducted design studies on the feasibility of building an entire system, including power storage, processing, sensing, and communication, in a cubic millimeter [19].

In practice, this scale should be achievable five to ten years from now.

There have been some noticeable improvements in low-power CMOS radios at several performance points. The research community now uses a one-inch scale platform which provides an opportunity to explore the node system architecture. It must be extremely energy efficient and conveniently handle event bursts. It must meet hard real time constraints, such as sampling the radio signal within bit windows. At the same time, it must also handle asynchronous sensor events and support localized data processing algorithms. It also must be robust and reprogrammable in the field.

2.1.8. OPERATING SYSTEMS

The growth in capabilities and complexity of these devices is so great that different operating systems have emerged to make application design more manageable.

Examples of these operating systems are the real-time operating systems Vxworks [20], Geoworks [21], and Chorus [22]. These three have scaled down their memory footprints and added TCP/IP capabilities.

Windows Mobile [23] has strived to provide a familiar windows PC environment in a subset of the windows interface.

PalmOS [24] focused successfully on data element exchange with infrastructure machines, but it provided little support for the concurrency needed for interactive communication.

UNIX variants (and more particularly, Linux variants) have come into the picture as soon as the devices reached a processing and storage capability beyond the early workstations, and has gained substantial popularity because it provides real-time support in a multi-tasking environment with well developed networking.

Another specialized tiny operating system for sensors was developed, the TinyOS [25], that provides a framework for dealing with extensive concurrency and fine-grain power management while providing substantial modularity for robustness and application specific optimization. The TinyOS framework establishes rules for the construction of reusable components that can support extensive concurrency on limited processing resources.

2.1.9. SENSING AND ACTUATING

Interfacing to the physical world involves exchanging energy between embedded nodes and their environments. This takes two forms: Sensing and actuating. [19].

Sensing is transforming a particular form of energy (heat, light, for example) into information. Actuating is converting information into an action.

Together, sensing and actuating are the means of physical interaction between the nodes and the world around them.

A common problem in both sensing and actuating is uncertainty, since the physical world is a partially observable, dynamic system and the sensors and actuators are physical devices with inherent accuracy and precision limitations. There is also the problem of latency both in sensing and actuating.

To increase accuracy, we can use filtering (at the nodes) or redundancy (increasing accuracy and fault tolerance of the system).

As for localization, for a system to operate on input from the physical world, nodes must know their location in 3D space to provide context aware services to other system elements.

A static or mobile node must be able to answer the question “Where am I?” It might do so relative to a map, relative to other nodes, or in a global coordinate system.

For a sensor network, this is particularly relevant, because we often tag the queries to which it will provide answers based on some sort of location information.

Localization is the main part of registration between the physical and the virtual worlds.

To support an architecture where the sensor network does more than just output measurements, in high-bandwidth bit streams, but instead output semantically higher meaningful attributes, already processed, important system-oriented trends are emerging.

Two of those trends are self-configuring networks and data-centric systems. The objective of self-configuring networks is to let the systems exploit node redundancy (derived from node density), in order to achieve longer unattended lifetimes.

There are also emerging techniques for coordinating and adapting node sleep schedules to support a range of trade-offs between fidelity, latency and efficiency.

Ad-hoc routing techniques were an even earlier example of self configuration in the presence of wireless links and node mobility, but they still support the traditional IP model of shipping data from one edge of the network to another [19].

A second trend is the increased reliance on tiered architectures in which fewer higher end elements complement the somewhat limited capabilities of widely and densely dispersed nodes.

And now, where are we headed? The tremendous progress through miniaturization means that we can put instruments into the experiment, rather than conducting the experiment with an instrument.

The technical problems, the challenges and opportunities in pervasive computing are enormous.

2.1.10. PERVASIVE SOFTWARE ARCHITECTURE AND FUTURE EVOLUTION

We now focus on the software architecture of pervasive systems, and discuss future evolution.

2.1.10.1. BASE CHARACTERISTICS

Two base characteristics of ubiquitous systems are physical integration and spontaneous interoperation.

As for physical integration, a ubiquitous system involves some integration between computing nodes and the physical world.

As for spontaneous interoperation, in an environment or ambient, there are components - units of software that implement abstractions such as services, clients, resources or applications [26], that must interoperate between each other.

Physical integration and spontaneous interoperation have major implications for software infrastructure. These ubiquitous computing challenges include a new level of component interoperability and extensibility, and new dependability guarantees, including adaptation to changing environments, tolerance of routine failures or failure like conditions, and security despite a shrunken basis of trust [26].

Ubiquitous software should have a clear notion of what will be decided by the software and what will be decided by the user.

2.1.10.2. SOFTWARE CHALLENGES

Among the challenges for software for ubiquitous applications, there are discovery, adaptation, and integration, programming frameworks, robustness and security.

Discovery concerns to how mutual discovery takes place between a device and other available services or devices, and the determination of among which ones is interaction adequate.

Adaptation concerns to how we can use the device to display or manipulate data in other heterogeneous devices near to it in the environment.

Integration concerns to what connects the device’s software and the physical environment and what affects that connection.

As for the programming framework, how can we program the devices of the ubiquitous environment for a specific application? Must we address adaptation, discovery and integration at the application level, middleware or operating system level, language level, or a combination of these?

Robustness concerns to how we can shield a device and its user from transient faults and similar failures.

Security concerns thing like what operations can the user execute, and what operations can the device execute (if in deed they are different), Whom the device or user trust and how does authentication takes place, and how are threats to privacy minimized, have to be answered.

Considering a component that enters an environment, to satisfy spontaneous interoperation, the component faces three issues: Bootstrapping, Service Discovery, and Interaction.

Bootstrapping is a largely solved problem. The component requires a priori knowledge of addresses and any other parameters needed for network integration and service discovery. Protocols exist for joining underlying networks, such as IEEE 802.11. Addresses can be dynamically assigned state fully using DHCP or statelessly using IPv6.

A service discovery system dynamically allocates a service instance that matches a component's requirements, solving the association problem.

As for interaction, components must agree to a common interoperation model to be able to interact with each other.

Another challenge is adaptation to the various situations that may exist in the various places where the pervasive application or device may be used, with multiple conditions in terms of bandwidth available, for example.

The Content challenge is also important since embedding computation in or linking general purpose computing devices with the physical world implies heterogeneity across the devices, including devices embedded in and those brought to the environment by the users. Content adaptation is important here.

The challenge of the human interface is also of extreme importance. To achieve the common scenario of pervasive computing – a user walking into a smart space and being able to access and control various aspects of its environment through his PDA – requires separating user interfaces from their applications, which is a well understood problem in desktop systems. In ubiquitous (and in pervasive) computing the goal may be to move the interface to a physically different device chosen on the fly.

To integrate the computing environments (virtual by definition) with the physical world, we need low level application programming interfaces that let software deal with physical sensors and actuators, and a high level software framework that lets applications sense and interact with their environment, including physical sensors and actuators [26].

As for programming frameworks for pervasive computing it is important also to remember legacy applications and operating systems. This is also a research challenge. It is important to leverage existing applications because of the large investments in the applications, their data, and the knowledge on how to use them. As for operating systems is important to assure legacy systems support in new operating systems, since the volatility principle [26] tells us that ubiquitous environments will always change incrementally and continuously over time. Today's new systems are tomorrow's legacy systems. Given innovation's current pace, obsolescence is happening faster than ever.

Robustness and routine failures are a challenge for pervasive system since their frequency increases compared to a wired distributed system. Some of these failures are not literal failures but events that are similarly difficult to recover from (Battery exhaustion, Interference, for example).

2.1.10.3. WIRELESS NETWORKING PROBLEMS

Wireless networks are much more exposed to interference and communication failures that it's wired counterparts. In spontaneous interoperations, associations are sometimes gained and lost in a way that cannot be predicted.

Although the literature on distributed systems has many techniques for fault-tolerant computing, they are often based on resource redundancy while what we have here is resource scarcity.

And, what's more, under the current distributed system's assumption failures are relatively rare and could lead to expensive or ungraceful recovery, contrary to the expectation of calm behavior of pervasive systems.

In ubiquitous computing scenarios, various transient failures may actually characterize steady state operation. These scenarios have been characterized by the Internet Protocol and Distributed Systems community. [26].

We take a look now to too techniques for these scenarios, whose common tie is the underlying assumption that recovering from frequent transient failures basically means being always prepared to reacquire lost resources.

These two techniques are: Expiration based schemes and soft state, and Separating failure free and failure prone operations.

As for Expiration based schemes and soft state, when failure is a common case, identify what critical static or dynamic state you must persistently store and consider reconstruct able state for everything else.

As for separating failure free and failure prone operations, since not all operations are equally likely to fail, clearly indicate which ones are more likely to fail because of dynamism and required spontaneous interoperation, so developers can provide more effective error handling.

We can use group communication to help rediscover lost resources through the level of indirection it provides. Although the approaches relying on group communication have been criticized by their poor scalability, in ubicomp, we might be able to explore the Boundary principle.

2.1.10.4. SCALABILITY

Although ubicomp projects should certainly address scalability, the Boundary Principle can confound the attempt. Sometimes observing the Boundary principle can help, by enabling solutions that favor robustness over scalability. Other times, observing the principle can introduce complications, as in the case when a physically range-limited networking technology still fails to capture other (cultural or administrative) important boundaries. [26]

2.1.10.5. SECURITY

As for Security, how can we protect components from one another? Devices should have authorization methods. Users require security for their resources and ultimately, privacy for themselves.

Mobile computing has exacerbated these needs, leading to an understanding of vulnerabilities such as the openness of wireless networking. New challenges are also raised by physical integration and spontaneous interoperation, requiring a new model of trust and authentication as well as new technologies.

As for trust, this is an issue in ubiquitous systems because of spontaneous interoperation. It's difficult to establish basis of trust between components that can associate spontaneously, since these components may belong to disparate individuals or organizations and have no a priori knowledge of one another or of any trusted third party.

The key here is trust human judgment. Humans can make judgments about the trustworthiness of a particular environment they're in, and the physical world offers mechanisms for boot-strapping security based on that trust. For example users may exchange cryptographic keys with their environment or each other over a physically constrained channel such as short-range infrared, once they have established trust [26].

In resource poor devices, security protocols are influenced by physical integration. Even when using elliptic Curve cryptograph, resource poor devices do not have the processing capabilities to do asymmetric (public key) cryptography. And to worsen things its protocols must minimize battery consumption.

One of the best known attacks on wireless networks of sensors powered by batteries it's the "sleep deprivation torture attack", where an attacker can always deny service by jamming the wireless network with a signal that rapidly causes the devices to run down their batteries.

Another problem in pervasive systems is access control, that is, how to control access to resources. It would be interesting to exchange minimum or no information about the user when interacting between devices.

2.1.10.6. LOCALIZATION

As for location, this has important security implications, which have to be taken into account when designing security and privacy aspects of a pervasive system.

Of course, there are also challenges of location aware computing [27], a new class of computing created mainly by the evolution of mobile computing, location sensing, and wireless networking.

2.1.10.7. MOBILE COMPUTING

Mobile computing is normally associated with small form factor devices such as PDAs, GSM phones, etc…, not necessarily context aware. If context-aware it may be location-aware, and respond with a user's location either spontaneously or when active.

Four fundamental issues complicate the design and implementation of mobile computing hardware:

- Mobile elements are relatively resource-poor relative to static elements because of limitations of power, size and weight.
- Mobile communications are inherently vulnerable to security violations because data is transmitted through open airspaces
- Wireless connectivity is highly variable in reliability and performance
- Mobile elements must rely on limited energy resources

These issues are intrinsic to mobility.

We give more information about location technology on the technology section of this chapter.

Underlying the goal of “geospatial everywhere” is the ability to obtain information on demand, wherever the user is. For this we will need to develop methods to accommodate user mobility. It makes sense to invest in developing portable , lightweight display technologies such as electronic paper, foldable displays, handheld projectors (that can be pointed at any convenient surface), and augmented reality glasses. We also must invest in appropriate interaction paradigms. [26].

2.2. AUGMENTED REALITY

Augmented reality tries to extend the real world with virtual objects while maintaining the computer in an assistive, unobtrusive role, thus trying to keep in line with the pervasive computing objective of invisibility.

Augmented Reality is a sub concept of Mixed Reality [4]. AR augments the real world with synthetic electronic data. Augmented Virtuality (AV) enhances the virtual world with data from the physical (real) world. Mixed reality is a term that covers a continuum from AR to AV.

An example project of Mixed Reality is exposed in [4] by the Mixed Reality Systems Laboratory.

Both Mixed Reality and Augmented Reality are different than Virtual Reality, which is not pervasive at all (as we discussed in the previous section), and just deprives the user from the real world, completely evolving it in a virtual world.

In many ways, the design of wearable computers and augmented reality systems has been motivated by two primary goals: the first goal is driven by the need for people to access information, especially as they move around the environment; the second goal is motivated by the need for people to better manage information [28]. Until just recently, if a user needed to access computational resources, the user had to go to where the computational resources were located, typically a desktop PC or a mainframe computer. Once the user left the terminal, the flow of information stopped. Now networked wearable computers along with other digital devices allow the user to access information at anytime and at any location. However, the ability to access large amounts of information might not always be beneficial – too much information presented too fast may result in information overload. For this reason, the issue of information management is also important. In this regard, wearable computers and augmented reality systems along with software acting as an intelligent agent can act as a filter between the user and the information. Intelligent agents will allow only the relevant information for a given situation to be projected on a head-mounted display, a hand-held computer, or an auditory display.

Wearable computer systems will likely be a component of other advanced information technology initiatives as well. For example, in the area of “smart spaces” by embedding sensors and microprocessors into everyday things, wearable computer and augmented reality systems will be able to respond to and communicate with objects in the environment. In addition more and more wearable computing medical devices will be implanted under the skin to regulate physiological parameters, or to serve as cognitive or sensory prosthesis. One prototype wearable device in this area consists of an electrode that is implanted in the motor cortex of the brain to allow speech incapable patients to communicate via a computer. Gold recording wires pick up electrical signals from the brain and transmit signals trough the skin to a receiver and amplifier outside the scalp. The system is powered by an inductive coil placed over the scalp so that wires for powering the device do not have to pass through the skull. Signal processors are used to separate individual signals from the multiple signals that are recorded from inside the conical electrode tip.

Over the past several thousand years, nature has provided humans the sensory systems that allow them to detect and respond to visual, auditory, olfactory, haptic, and gustatory information. Nature has also provided humans well developed cognitive abilities that allow decisions to be made under conditions of uncertainty, patterns to be detected that are embedded in noise, and common sense judgments to be made. However, even thought we can sense a broad range of stimuli, often over a wide range of energy values, our senses are still limited in many ways. For example, we see images that represent only part of the electromagnetic spectrum, detect tactile and force feedback sensations across a limited range of energy values, and have marked decrements in spatializing images when sound is the primary cue.

Due to these limitations, several types of prosthesis have been developed to extend our sensory, motor and information processing abilities. To extend the visual modality, we invented glasses, microscopes, and telescopes. To extend the auditory modality we invented microphones, hearing aids, telephones, etc... To extend the haptic modality we have invented sensors that detect forces, which then transmit these forces back to the human.

We have also invented other types of sensory and motor prosthesis as well. Some of these include artificial hearts and kidneys and artificial arms and legs. However, until only recently, the prosthesis that have been developed were designed primarily to enhance the detection capability of our sensory systems or to assist our motor capabilities, and to a far lesser extent, to enhance our cognitive capabilities. With the invention of the computer and developments in digital technology, microelectronics, and wireless networking, this is beginning to change. We are now able to wear digital devices that contain considerable computational resources. These devices clearly enhance our decision making capabilities. These devices may be worn on the skin (or body), as in the case of a wearable computer, or under the skin, as in the case of medical devices. Digital devices worn on the skin have led to exciting developments in the area of computational clothing and digital accessories. In this area of research, computer scientists and interface design specialists are working with clothing designers as well as experts in textiles and fabrics to build wearable computers that look more like clothing and less like computers.

Advances in computers under the skin have also led to exciting developments in information technology. In fact, in the near future we may be able to integrate computer chips directly into the nervous system. Along these lines, researchers at Johns Hopkins University and North Carolina State University have developed a computer microchip that is connected to the retina. The chip is designed to send light impulses to the brain. Thus far the device has allowed fifteen test subjects with blindness resulting from retinal damage to see shapes and detect movement.

We describe a wearable computer as a fully functional, self powered, self contained computer that is were on the body.

Augmented reality can be thought of as an advanced human – computer interface technology that tries to blend or fuse computer generated information with our sensations of the natural world. For example, using a see-trough head-mounted display (HMD), one may project computer-generated graphics into the environment surrounding the user to enhance the visual aspects of the environment.

Augmented reality is a technology to augment our senses and wearable computers are in general far more mobile than augmented reality systems.

One of the primary display or output devices for wearable computer systems is a head mounted display. Currently, there are three main application areas for HMDs, virtual reality, augmented reality and wearable computers.

In virtual reality, a participant uses an HMD to experiment an immersive representation of a computer-generated simulation of a virtual world. In this case, the user does not view the real world and is connected to the computer rendering the scene with a cable allowing about 3-4 meters of movement.

With augmented reality, a participant wears a see-trough display (or views video of the real world with an opaque HMD) that allows graphics or text to be superimposed (projected) in the real world. As with the virtual reality experience, the user is connected to the computer rendering the graphics or text with a cable, again allowing about 3-4 meters of movement.

With wearable computers, the user actually wears the computer, and, as in virtual or augmented reality, wears the visual display (hand-held or head-mounted). The wearable computer may be wirelessly connected to a LAN or WAN, thus allowing information to be accessed whenever and wherever the user is in the environment. Wearable computers allow free-hand manipulation of real objects as does augmented reality displays.

There are several dimensions by which wearable computers and augmented reality systems can be evaluated, two of which include the level of mobility, provided by the computing system, and the level of scene fidelity afforded by the rendering platform. The level of scene fidelity refers to the quality of the image provided either by the virtual reality simulation, real scene, or augmented world. One of the primary differences between virtual reality and augmented reality is in the complexity of the projected graphical objects. In basic augmented reality systems, only simple wire frames, template outlines, designators and text are displayed and animated. An immediate result of this difference is that augmented reality systems can be driven by microprocessors either worn on the body or integrated into the workplace. Today's processors have the computational power to transform and plot complex graphics in real time. Unlike full virtual reality systems, augmented reality systems are generally not attempting to generate a complete world or realistic scene. Instead, augmented reality systems tend to rely on reality to simulate reality, only superimposing the necessary graphical objects necessary to perform the task at hand. By doing so, many of the human factors issues found in full virtual reality systems such as vertigo and simulation sickness are avoided. The person continues to receive all the orientation cues from the physical visual scene. Display technology, input and output devices, computer architectures, network communication, power supplies, and image registration and calibration techniques are all important aspects of augmented reality systems.

Humans carry theirs sensors with them as they move around the environment, and they experience the world with the resolution provided by these biological sensors. Wearable computers also allow a high degree of mobility, but not quite that associated with our biological sensors. For example, currently we cannot swim with wearable computers although our biological sensors easily allow this degree of mobility. The weight of wearable computers and augmented reality systems further adds to their lack of mobility. Wearable computers allow a high degree of scene fidelity because the real world is viewed either directly with see-through optics or via live video. However, the level of scene fidelity may be less than that associated with augmented reality because wearable computers currently do not have the rendering capability of workstations that are often used to render graphics for augmented reality environments.

Finally virtual reality in its present form is often low on scene fidelity and very low on mobility within the real world.

There are different types of wearable visual display technologies, which may be used to combine real world objects with computer generated imagery to form an augmented scene. The two main types of visual display systems supporting wearable computers and augmented reality are optical based systems and video based systems.

Optical based systems allow the observer to view the real world directly with one or both eyes with computer graphics or text overlaid onto the real world. Optical see-trough HMDs are worn like glasses with an optical system attached at a location that does not interfere with visibility.

Video based systems can be used to view live video of real world scenes, combined with overlaid computer graphics or text. Furthermore, monocular (one eye) or binocular (two eyes) displays can be used. Video based see-trough displays are opaque displays that use cameras mounted near the user eyes to present live video on the display. Using chrome or luminance keying techniques, the computer then fuses the video with the virtual images to create a video based augmented reality environment.

When creating a wearable computer system that can be used to augment the environment with text or graphics, an important visual requirement is that the computer generated imagery register at some level of accuracy with the surroundings of the real world.

In terms of developing scenes for wearable computer systems, the problem of image registration or positioning of the synthetic objects within the scene in relation to the real objects is both a difficult and important technical problem to solve.

Image registration is an important issue regardless of whether one is using a see-trough HMD or a video-based HMD to view the augmented reality environment. With applications that require accurate registration, depth information has to be retrieved from the real world in order to carry out the necessary calibration of the real and synthetic environments. Without an accurate knowledge of the geometry of both the real world and computer generated scene, exact registration is not possible. To properly align video and computer generated images with respect to each other, several frames of reference must be considered. There are various techniques to achieve this. Although these methodologies work reasonability well for systems that have a limited range of mobility, much more research is needed before accurate registration of images using wearable computer systems can occur.

Other important issues for wearable computer systems using a video based HMD relates to optics of the HMD and input / output devices.

Camera lenses that may be applied to augmented reality HMD may vary in field of view. The use of a zoom lens that has the capability to change from wide angle to telephoto views of the scene is desirable for video based wearable computers as this allows the viewer to match the field of view of the real world scene to the field of view of the computer generated scene. In addition, depth of field, defined as the distance from the nearest to the farthest parts of a scene that are rendered sharp at a given focusing setting, is another important variable to consider for augmented reality displays. Depth of field increases as the lens is stepped down (smaller lens aperture), when it is focused for distant objects, or when the lens has a short focal length. Other important variables to consider when designing a video based system that displays stereoscopic images include the horizontal disparity (horizontal offset) of the two video cameras and its correspondent convergence angle.

If the video of the real world is to appear as if viewed through the user eyes, the two CCD cameras must act as if they were at the same physical location as the user eyes. The interpupilary nodal distance is an important parameter to consider for applications that require close to medium viewing distances for depth judgements.

Placing the cameras in front of the user eyes may result in accurately registered video and computer generated images, but the image of the outside world seen through the HMD will appear magnified and objects will appear closer than they actually are.

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.

Table 1 shows the possible types of sensors and its use in intelligent environments [28].

Sensor type	Stimulus	Use
Mechanical	Position, acceleration, force, shape, mass, displacement	Detecting people's/object's position, weight, movements
Biological	Heart rate, body temperature, neural activity, respiration rate	Measuring people's mood, mental state, physical state
Acoustic	Volume, pitch, frequency, phase, changes	Detecting sounds, interpreting speech
Optical	Emissivity, refraction, light wave frequency, brightness, luminance	Computer vision detection, IR motion/presence detection
Environmental	Temperature, humidity	Monitoring the conditions of the environment that people are in

Table 1 - Types of sensors

In general, there are two kinds of sensors: active and passive. Their characteristics impact their use in wearable computer systems.

Active sensors require an external power source or excitation signal in order to generate their own signal to operate. The excitation signal is then modified by the sensor to produce the output signal. Therefore, active sensors consist of both a transmitting and a receiving system.

Passive sensors, in contrast, directly convert stimulus energy from the environment into an electrical output signal without an external power source. Passive sensors consist only of a receiver.

When building and employing sensors in an intelligent environment like a wearable computer for augmented reality, there are a number of variables inherent to the particular sensors that need to be addressed. These are [28]:

- Accuracy (Resolution): The smallest change in magnitude a sensor can detect
- Range or field of view: The amount of area covered by the sensor
- Calibration error: The maximum amount of inaccuracy permitted by the sensor
- Power consumption: The amount of energy required by the sensor to operate
- Size: The physical dimensions of the sensor
- Saturation: The maximum amount of stimulus the sensor can respond to
- Repeatability: Ability to accurately recreate responses under identical stimuli
- Sample rate: The frequency at which the sensor samples the stimulus.
- Noise filtering: The ability of the filter to filter out unwanted environmental noise

Networks for wearable computing may be divided into two broad classes: Off body and On- or Near-Body [29].

Off Body networks are networks that connect the wearable computer to other systems not worn or carried by the wearable computer user.

On- or Near-Body networks are networks that connect the computers, peripherals, sensors, and other devices that comprise a single user's wearable computer system.

Connecting a wearable computer to an off body network demands a solution different from that used for connecting desktop machines. The wearable user cannot be wired with an Ethernet or telephone/modem cable and still maintain all the benefits expected of wearable computing systems.

Wireless solutions are needed to support the freedom of movement in the environment that motivates wearable computing. Other factors, like minimum bandwidth needs, operational or mobility range, tolerable infrastructure and usage costs, and the usual wearable constraints of power, size, and weight, tend to vary with the specific application.

On body or near body networks are also important to wearable computing. These networks tie together the computers, peripherals, sensors and other devices that are carried/worn on or relatively near the body, and they contribute significantly to the ergonomics of the wearable system.

An on-body network can be thought of as a new level in the traditional WAN-MAN-LAN network hierarchy: The personal area network, or PAN.

The concept of a PAN is more general than wearable computing but is especially important to wearable systems because PANs enable the use of devices (CPU and peripherals) that are as small as possible and conveniently placed, worn, and carried on the body.

Advances in mobile personal communications and wireless LAN technology are the primary technology sources for off-body networking in a wearable computing system. The mobile personal communications industry brings forward analogue and digital cellular technology, the paging infrastructure, and commercial wide-area messaging services. Wireless LAN technology can also be exploited for off-body communications in many environments, depending upon the needs of the wearable application.

The current state of the art does not present a single ideal off-body communications solution for wearable computing. Options differ in respects that demand careful consideration to system level requirements. Data rates, availability, operational range, infrastructure requirements, cost/pricing plans, and other issues must be considered when selecting from the technology options available today.

The future will offer improved off-body communications for the wearable computing user.

As for the IP architecture, there are three basic options for integrating an on-body network into the global internet or private IP intranet. A number of other approaches are also possible, but most are essentially just variations or combinations of these options.

The first option is that no device on the body has an IP address. All communications across the body and across the off body link as well are carried in a non-IP format. These are translated to IP at a proxy gateway computer which then acts as the application level go-between that glues the body network into the Internet. This approach was the benefit of being simple, requiring little in the way of IP address administration, avoiding the expense of sending IP headers across scarce radio resources, and avoiding the power and computational resources needed to run an IP stack on the body.

It has the drawback of requiring a specialized proxy gateway, which tends to act as a bottleneck to the introduction of a few features. All in all, however, this is the option most commonly adopted.

The second option is assign to each body a common IP address. IP datagrams can flow directly between the internet and the body hub over the off body network, but communications across the body network are not IP. This has the advantage of eliminating the proxy gateway and merging its functionality into the body hub in favour of providing a full fledged Internet connection to the body, but still not requiring every device to be able to run an IP stack. It retains the basic drawbacks of a proxy gateway, however.

The third option is that every device that is wearable has its own IP address, and IP datagrams can flow directly between any body device and the internet. In general, this solution requires the body network to have a distinct IP network address, and the body hub to have at least minimal IP router functionality. This has the advantage of fully integrating the body network, and its devices, into the global internet. The prime disadvantages, thought, are that every device must have enough power and computational resources to be able to run an IP stack, that IP headers must be conveyed across perhaps thin radio links, and that it requires a fairly large number of IP addresses.

A personal area network (PAN) is used to interconnect the computers and devices (peripherals, sensors, actuators) used by an individual within their immediate proximity. These devices may be carried or worn on the person or in some cases placed nearby.

There is an obvious need for PAN technology on wearable computer systems. A basic wearable computing system comprises a main central processing module, a separate keypad or pointing device, a display device, a voice input device, and a speaker for audio output. In addition to this basic set of devices, application specific devices commonly added to the wearable system include bar-code scanners, wearable printers, geo-location devices, off-body network connections, health monitoring sensors, etc.

To effectively support wearable computing, a number of requirements must be satisfied by a personal area networking technology [29]:

- It must be wireless without line of sight restrictions, since this kind of restrictions would greatly complicate the placement of devices on the body.
- Have extremely low power consumption requirements (every device must be self powered, so the PAN must consume minimal amounts of power)
- Small and lightweight: The PAN component must be a small contributor to the size of a necessarily small and lightweight wearable device.
- It must be a solution supporting multiple devices and bidirectional communication
- It must provide cross-network interference tolerance, since every wearable user will have his or her PAN, which must continue to operate when in proximity from one another
- Broad generality, having the capacity to integrate a broad range of devices

- Low cost, since some of the devices that are to be included in a wearable computing system PAN are very inexpensive (like sensors, pointing devices, etc.) so the PAN must be low cost in order to avoid becoming the principal cost component.

2.3. GAMES

It is quickly becoming clear that entertainment will be one of the killer applications of future wireless networks [3]. More specifically mobile gaming was predicted to be worth more than 1.2 billions of American dollars in 2006 in the United States market alone. However mobile applications faces issues that are different from fixed network applications, including fluctuating connectivity, network Quality of Service and host mobility.

In the times before computers existed, the games people played were designed and played out in the physical world with the use of real world properties, such as physical objects, our sense of space, and spatial relations [30].

Pre-computer day's game interactions consisted of two elements: human to human interaction and human to physical world interaction. In the current times, due to their higher level of attractiveness to game players, computer games have become the dominant form of entertainment.

Computer games motivate the players by bringing them more challenge, curiosity and fantasy, which are the three main elements contributing to fun in games. They provoke curiosity because they are usually designed with an optimal level of information complexity, they create fantasy by creating the illusion of the player being immersed in an imaginative virtual world with computer sound and graphics, and they create challenge by having goals typically more interactive than traditional games.

However, there is a also a new tendency in gaming called pervasive games, where the real world is coming back to the computer entertainment field stressing the pervasive and ubiquitous nature of these games: Pervasive games are no longer confined to the domain of the computer (the virtual domain), but integrate the physical and social aspects of the real world.

Pervasive gaming is a relatively new field but we can already identify several unique types of pervasive games, each focusing on different aspects of the gaming experience.

One of the types is smart toys. Augmenting traditional toys with pervasive computing technology is a first and simple step towards the realization of pervasive games.

Most realizations include traditional toys equipped with simple sensing technology linked to computer logic, which reacts to changes in the toy's physical state by displaying graphical information or playing sounds. One common use is digital storytelling by small computers, encouraging use of computers by small children.

Other type is affective gaming which goal is the capturing of how the player is feeling at any given moment and integrating this very personal representation of context into the game. This falls into one of the goals of pervasive computing which is creating context-aware applications that adapt their behaviour to the information they collect from the environment.

This is different from previous games which goal was the where and who of the player context (location aware pervasive games) or the what and the where (common elements of most traditional games).

Affective computing is described as “computing that relates to, arises from, or deliberately influences emotions” [30]. Affective gaming aims to create a magical gaming experience by integrating the player’s emotions into the game so the game environment can adapt to them.

While sensing a player’s emotional state is a complex research problem, sensing some aspects of human emotions while it’s engaging in an entertainment experience is a more common scenario.

The most common method of doing this is using sensors to capture affective state measuring the user’s physical activity. Once a player affective state has been determined, some personality can be made into the game.

Augmented tabletop games are another type of game. Traditional tabletop games such as chess have been popular for ages, and they are still popular, even with the arrival of computers. The success of tabletop games can be explained by the directed communication and interaction of the players between each other.

Augmented tabletop games introduce computing into the scenario using sensing technology and at the same time social concepts.

A common type of context aware game is the location-aware game which uses the location of the user to improve the gaming experience.

When an entire building, a block, or city become the game playing field and we need to track the physical location of the human players a host of technical and conceptual challenges arise.

The player’s position can be determined technically by GPS signals (satellite), Wi-Fi, or GSM signal strength and/or cell ID, short rage proximity sensing using RFID, infrared beacons, or ultrasonic emitters.

Due to the necessity for infrastructure, short-range proximity sensors are not ideal for implementing pervasive games; thus GPS and Wi-Fi are used in most recent location aware games. Given the massive market penetration of GSM phones, cell based games will probably become more common case in the future.

GPS signal reception does not work very well indoors, but it allows us to cover a large playing field (up to the entire planet).

Location determination using Wi-Fi triangulation requires an appropriate infrastructure and a careful layout of stationary access points.

Finally, there is the augmented reality game type, the most technically advanced pervasive games. We have already discussed augmented reality.

There are several characteristics augmented reality games have that differentiate them from other types of games.

In augmented reality games, the social space through which users communicate is aligned with the task space where they interact. The task space is further enhanced by a visualization space where information may be presented. This allows users to communicate face to face, view content and work with objects all the same time, with reduced context switching.

Augmented reality games (and all augmented reality systems) try to capture the user’s full range of motion, using a wide range of physical interactions, including location, gestures, and posture. Physical objects can also be tracked, allowing the use of props and sporting equipment.

A common limitation of augmented reality is that is easy to affect virtual objects based on physical interactions, but is difficult to affect physical objects based on virtual ones.

Augmented reality gaming is a new field of research and the first games are just now becoming to appear (the first prototypes).

We present some of these prototype games in the section on related work.

One of the challenges of making pervasive games, especially location oriented ones, is dealing with uncertainty [31].

Causes of uncertainty can be the location technology used itself (like GPS for example), which is only accurate to a certain extent and so introduces a degree of uncertainty in the game, or the network transmission technology used. Wireless communication can fail at unpredictable moments in time and that causes uncertainty too.

It's a job for the game designers and for the game programmers to take uncertainty into account when making a pervasive game.

2.4. METHODS, PROTOCOLS AND TECHNOLOGIES

We now describe in more detail the specific methods, protocols and technologies that can be used in mobile pervasive augmented reality gaming, and we divide these methods, protocols and technologies in the following sections: security, geospatial location and orientation, service discovery, third generation mobile networks, ad-hoc networking and dynamic configuration, scalability, consistency, multimedia data heterogeneity, data distribution and replication, quality of service and management of networks and services.

2.4.1. SECURITY

Pervasive computing promises a rich and seamless interaction with surrounding computing environment, but it will be difficult to reconcile pervasive systems with social and legal trends that are simultaneously underway [32]. There are regulatory trends that will only be exacerbated in the context of pervasive computing.

To trust pervasive systems, one must be able to manage the privacy, confidentially, availability, and controlled access to digital information as it flows through the pervasive system.

Data can be exchanged seamlessly trough the pervasive system. However, that does not mean it will necessarily be.

The term security is normally used to describe techniques that control whom may use or modify the computer system or the information contained in it, whereas the term privacy is normally used to mean the ability of an certain entity to determine whether, when and to whom information a certain piece of information is to be released.

The term trust normally means the grounds for confidence that a system will meet its security objectives.

There are five big problems related to trust in pervasive computing, these are:

- Knowing to whom one is talking too. The transient, ad-hoc nature of device interaction turns difficult to one device to establish with certainty with physical device, among many, is it really interacting with, which is especially true if interaction occurs over the wireless network.
- The way one's privacy will be safeguarded, since pervasive systems can cause widespread erosion of personal privacy, which places these systems at odds with legal and public-policy safeguards that seek to protect privacy. Privacy-sensitive information in the context of pervasive computing can be classified in two categories each requiring different approaches to deal with the privacy problem:
  1. Privacy of personal data that originates in digital form and its explicitly exchanged, a category that applies to sensitive personal information that the user owns and willingly exchanges with the pervasive infrastructure based on informed consent and mutually agreed upon privacy policy. It may be possible for the infrastructure to selectively apply privacy by, for example, applying privacy tags to the exchanged information.
  2. Privacy of sensed information, that is, sensitive personal information that is sensed by the infrastructure, without the control of the user. Preserving the privacy of this kind of information requires the infrastructure to have special means of protecting privacy.
- The question of trusting the device we are communicating with. Dealing with this problem requires the development of trust negotiating protocols where the trustworthiness of devices can be assured by third parties.
- The question of recourse. Because pervasive systems will mediate everyday physical activity, technical mechanisms to facilitate recourse must be built in from the start. Recourse is often provided by risk-management techniques like insurance, liability and institutional guarantees. It is more difficult to manage risk in pervasive systems than in traditional systems because of their decentralized administrative ownership model
- The question of denial of service attacks. Will the pervasive system be reliably available ? The immersive computing infrastructure of pervasive systems exposes a larger attack surface with many points of failure if we compare it with traditional computing environments. Due to the high degree of interdependency (connections) between components, the need for decentralized administrative authority, and the wide variety of computing and communications technologies involved, pervasive systems will be extremely vulnerable to deny of service (DoS) attacks.

Having defined five fundamental problems of security on pervasive systems there are also six fundamental challenges for secure pervasive computing [33]. These are:

- The need to integrate the socio-technical perspective. Issues such as usability and trust in security technologies, and how these technologies relate to the larger sociological, economic, cognitive and legal aspects of our lives, should be important considerations. Different security profiles should be considered for the different roles we play in our daily routines.
- The need for models that break out of the classical concept of perimeter security and the need to support dynamic trust relationships. In some pervasive environments, user identity may not be known and the bounds of the system may also not be known, making identification and authorization based on user identities and the use of network boundary firewalls useless.
- The challenge of balancing in the correct way security and non –intrusiveness. We should step away from intrusive schemes such as asking the user a login and a password, and requiring explicit user input. The required information should be able to be sensed or otherwise obtained from the environment or context securely and automatically and exchanged with the

communicating devices and user in a seamless manner. For this to happen we need context awareness, which was discussed in the section on pervasive computing.

- Security for pervasive computing has to be able to react according to the characteristics of mobility and dynamism of pervasive systems. A user may be mobile and interact with multiple devices and access multiple applications. Applications and data may migrate on behalf of roaming users. The user may be frequently disconnected from networks.
- Pervasive computing introduces the need for resource constrained operations, severely limiting the type of cryptographic operations, security protocols and security mechanisms that can be supported.
- There is the need to balance security and other services, since pervasive computing includes a series of other applications, usage scenarios, and data handling demands. To be successful, each area of application of pervasive computing should offer the right set of tradeoffs between a variety of service attributes which include privacy, security, usability, quality of service, and cost.

Having discussed the challenges of pervasive computing security, there is the question of the models of pervasive computing security, before we get into the details of the protocols, architectures and mechanisms.

Models for authentication in pervasive computing should be able to handle user mobility across networks, applications and devices, application migration across multiple execution platforms and devices, and disconnected network operations requiring localized trust establishment.

Also, models for access and usage control must go beyond system centric subject-object models, where the main abstractions are subjects, objects and rights. In pervasive computing, there is the need to include the socio technical aspect. Attributes of users transfer in part to principals and further to subjects, but principals and subjects also gain additional context attributes. There is a clear need to formulate access control and usage models that exploit context information. Some preliminary work using well known context information such as time of access and location of the user has been done by a number of research projects.

Context information opens up the possibility of using proximity based access control or encounter based access control schemes.

There is also the question of the models for privacy. For pervasive computing, these should include well known privacy principles, legal requirements for specific application areas, as well as general society expectations. Some of the principals that should be included are notice, choice, consent, anonymity and pseudonimity, adequate security and access and recourse.

And finally there is the question of the models for dissemination control, which deals with how digital objects dissemination will be controlled in a pervasive computing system ? Thought the digital world makes it easy to copy and distribute multiple electronic copies, access control models typically do not deal with copy protection and distribution. This has been study under the topic of Digital Rights Management (DRM). A research challenge is how to understand how DRM models can be interfaced and adapted to work with pervasive computing environments to provide dissemination controls (DCON).

About cryptographic algorithms, these can be subdivided in symmetric cryptographic algorithms and asymmetric cryptographic algorithms, being that the first ones can be characterized by the fact that both the sender and receiver use the same key to encrypt and decrypt data, while the second ones use different keys to encrypt and decrypt data, while related [34].

The symmetric cryptographic algorithms can be characterized further by the way the data is processed in block cipher and stream cipher algorithms.

In block cipher algorithms we include: DES (Data Encryption Standard), Triple DES, Advanced Encryption Standard (AES) or Ryjandael, RC2, RC5.

In stream-cipher algorithms we may include for example RC4.

Since only block cipher algorithms are standardized in the industry, they are the ones used in most situations.

The data encryption standard (DES) defines the data encryption algorithm (DEA). DEA and DES mean exactly the same thing. The DES algorithm was developed by IBM in 1974, and was declared an official standard by the USA government in 1977. DES is defined by the ANSI standard X9.32. Many experts have studied DES in search of ways to break it, but no practical and instantaneous way was found until now. However, a brute force attack is capable of decoding the data in no longer than a few hours today, due to the computing power advances over the 1970's [34].

DES uses a key length of 64 bits with 8 parity bits, so that the actual key size is 56 bits. The DES block length is 64 bits. The algorithm operates in two modes: ECB (Electronic Codebook Mode) and CBC (Cipher Block Chaining mode).

ECB encrypts each block separately, which means that the blocks can be encrypted or decrypted in parallel, leading to a better performance.

In the CBC mode, each block is combined with the encrypted previous block using a XOR operation, before the next block is encrypted. This has the advantage that the same plain text block results in another encrypted block, depending on the text that was encrypted before. The first block is combined with the Initiation Chaining Vector to start the encryption.

The Triple DES makes encryption with the DES algorithm even more secure by applying three DES operations to the same data. There are four variations of triple DES: DES-EEE3, DES-EEE2, DES-EDE3 and DES-EDE2

In the DES-EEE3 variation the input data is encrypted three times using the same key every time.

In the DES-EEE2 variation all is similar to the DES-EEE3 variation, but in the second encryption a different key is used.

In the DES-EDE3 variation the data is encrypted, than decrypted, and then encrypted again using different keys every time.

In the DES-EDE2 variation all is similar to the DES-EDE3 variation but now the keys for the first and third encryption are equal.

The Advanced Encryption Standard (AES) or Ryjandael was published as FIPS (Federal Information Processing Standards) Publication 197 in October 2000. It was created by two Belgium cryptographers named John Damen and Vincent Rijmen as a block cipher with variable block and key length. Currently, AES is standardized with key lengths of 128, 192, or 256 bits and block length of 128, 192 or 256 bits. All combinations of key lengths and block lengths are possible.

The strengths of AES are good performance and low memory requirements, making it suitable for pervasive environments.

Currently it will take about a year to break the algorithm with a cluster of computers with a brute force attack.

The algorithms RC2, RC4 and RC5 (RC stands for Rivest's Cipher) were developed by Dr. Ronald L. Rivest for RSA Data Security.

RC2 is a block-cipher algorithm working with a variable key length. So the strength of the encryption and its performance can vary with the key length that is used. RC2 has better performance than DES.

RC4 is a stream cipher algorithm with variable key length needing 8 to 16 operations for each output bit.

RC5 is, as RC2, also a block cipher algorithm with variable key length and block length. It is better than DES and better than RC2.

As for Asymmetric cryptographic algorithms, they were developed starting in 1976, when Withfield Diffie and Martin Wellman developed the Diffie Hellman algorithm which is the base for today's public key systems, and its objective was to solve the key distribution problem that users of symmetric cryptography have.

In asymmetric cryptography, and based on the concept of Diffie-Hellman, everybody has two keys, a public and a private one. The public key is accessible for the public and can be requested from a known and trusted third party, which guarantees that that specific key belongs to that person in fact. The private key always stays with the owner of the key and should be kept as a secret, for example, in a smartcard.

In a public key system, the information that was encrypted using a person's or devices public key can only be decrypted using the corresponding secret key.

The RSA (Rivest Shamir Adleman) algorithm is the most widely used asymmetric cryptographic algorithm today. It was developed in 1977 by Ron Rivest, Adi Shamir and Leonard Adleman and its patent was acquired by RSA Data Security, protecting the implementation until the 20th of September of the year 2000.

Today a key length of 1024 bits is used in most systems. RSA is the one of the possible cryptographic algorithms in many security standards, like S/MIME, IPSec, TLS (Successor of SSL) and PKCS.

The Digital Signature Algorithm (DSA) was published as part of the Digital Signature Standard (DSS) by the National Institute for Standards and Technology (NIST). It has a variable key length ranging from 512 to 1024 bits.

Using DSA, validation of signatures is slower than its generation. With RSA it's the other way around.

Digital Signatures enable the recipient of electronic messages to verify the identity of the sender and the origin as well as the integrity of the message, and are always based on asymmetric cryptography algorithms. The messages are signed with the private key of the sender and verified with its public key, which must be available to the receiver through a trusted third party.

There are two types of digital signatures used today: The ones that encrypt the whole message and the ones based on a MAC (Message Authentication Code) that is attached at the end of the document, which is transmitted unencrypted.

Elliptic Curve Cryptography (ECC) provides the same security as any other public key algorithm using a shorter key length. It was first proposed by Victor Miller and Neal Koblitz in the mid-80s.

ANSI is currently focusing on standardizing ECC as ANSI X9.62 for digital signatures and ANSI X9.63 for key agreement, thought some of the most efficient implementations of ECC are patented by a company called Certicom, which makes widespread adoption more difficult.

The MD2, MD4 and MD5 algorithms were developed by Rivest for RSA Data Security. While MD2 is optimized for 8 bit platforms, MD4 and MD5 are optimized for 32 bit platforms. The MD5 algorithm is a more secure version of MD4, and a little bit slower. It first splits the message in 512 bit blocks and generates a hash of 128 bits in three steps.

The Secure Hash Algorithm (SHA) was standardized in the Secure Hash Standard (SHS), published by the USA government as a “federal information processing standard”, and SHA-1 is an improved version of this algorithm.

The SHA-1 algorithm is used to generate a 160-bit MAC (Message Authentication Code) from a message that shouldn’t be longer than 2^{64} bit. The algorithm is a little slower than MD5, but it is more secure from brute force attacks due to its longer MAC.

A MAC is an authentication tag or checksum that is computed by applying a secret key to a message. The MAC is always verified using the same key. The MAC is generated using as a base a hash function, a stream-cipher or block-cipher algorithm. It's very common to generate hashes on the Internet using the MD5 algorithm.

A hash function is a one-way function that generates a fixed-length string out of a given input. That string is called the hash. A one way function is a function that is hard to invert.

A certificate is a document distributed by a trusted third party that binds a public key to a specific person, and which contents are guaranteed to be valid and correct by the trusted third party.

Certificates are standardized by X.509, and should at least contain the digital signature of the trusted third party, the name of the person owning the public key and the public key itself.

The Secure Sockets Layer was developed by Netscape in 1995 was recently superseded by TLS (Transport Level Security). It sits on top of TCP or other transport protocols and below the application layer. The protocol consists of two parts: The handshake protocol and the record protocol. The handshake protocol is needed to setup a session, and the record protocol is used to transfer the actual data between the communication partners. During the handshake the two partners authenticate each other and negotiate the security features for the data transfer that happens during the record protocol phase.

GEOSPATIAL LOCATION AND ORIENTATION

2.4.2.1. LOCATION

Location systems are increasing their accuracy, coverage, and availability, and maintaining moderate cost, which will rapidly make them pervasive. A number of candidate technologies are already in the market place, and new ones are continuously being developed. In the future, the development of new location systems is likely to be influenced by market trends in the hardware and software platforms of mobile devices – most particularly mobile phones – as much as the technology options with location systems itself.

Privacy is clearly a major factor in location systems development and deployment. Networked applications will appear to request location information from the user, suggesting that new techniques must be developed that extends the user control beyond his device.

Developers of applications will have to accept variation in the accuracy and availability of location data for some time.

Attempts have been made to alleviate many of today's wide area location solutions disadvantages and hurdles. There a number of variants of the basic GPS strategy that improve the accuracy and time to lock for GPS handsets. Both ground based and satellite based versions of differential GPS improve accuracy from 8m-10m to 5m for a non-enhanced unit. GPS chipsets are being integrated into mainstream mobile phones and PDAs, lowering cost and decreasing the user's barrier to entry. To help meet the E911/E112 requirements cell phone manufacturers are now producing handsets that use a mix of location technologies. When GPS is not available, the locations of nearby cell towers are used to produce a course location estimate. Based on this course location, the phone can download the expected position of the satellites, allowing the handset to lock on to the GPS much more quickly (on the order of the seconds) instead of one minute or more, when GPS becomes available. By 2009 the European Union will deploy Galileo, a next-generation GPS system that promises greater accuracy and operation covering both indoors and out, due to stronger radio signals that should penetrate most buildings.

2.4.2.1.1. GPS

Already, GPS is fully operational. The GPS system provides continuous worldwide three dimensional positions and velocity information to users with the appropriate receiving equipment.

GPS also disseminates a form of Coordinated Universal Time (UTC). The Satellite constellation consists of 24 satellites arranged in 6 orbital planes with 4 satellites per plane. A worldwide ground control/monitoring network monitors the health and status of the satellites. GPS can provide service to an unlimited number of users since the receivers operate passively (they only receive).

The system utilizes the concept of one way Time of Arrival (TOA) ranging. Satellite transmissions are referenced to highly accurate atomic frequency standards on board the satellites, which are in synchronism with an internal GPS system time base.

The satellites broadcast ranging codes and navigation data on two frequencies using code division multiple access (CDMA). Those frequencies are L1 (1575,42 MHz) and L2(1227,6 MHz). Each satellite transmits on these frequencies but uses different ranging codes than those employed by other satellites. Those ranging codes were selected because they have low cross-correlation properties with respect to one another [35].

The navigation data provides the means for the receiver to determine the location of the satellite at the time of the signal transmission, while the ranging code enables the receiver to determine the transit (i.e. propagation) time of the signal and thereby determine the satellite to user range. This technique also requires that the user receiver also contains a clock.

Utilizing this technique to measure the user three dimensional positions requires that TOA measurements be made to four satellites. If the receiver clock was synchronized with the satellite clocks, only three satellites would be needed. However, a crystal clock is usually employed in navigation receivers to minimize the cost, complexity, and size of the receiver.

GPS provides two services: The Standard Positioning Service (SPS) and the Precise Positioning Service (PPS). The SPS is designated for the civil community while the PPS is designated for the U.S. Authorized military and select government agencies. Access to the GPS PPS is controlled trough cryptography. Civilian use is only permitted trough special U.S. Department of Defence Approval.

The PPS is specified to provide a predictable accuracy of at least 22m (2 drms, 95%) in the horizontal plane and 27.7 m in the vertical plane. The distance root mean square is a common measure used in navigation. Twice the drms value, or 2 drms, is the radius of a circle that contains at least 95% of all possible fixes that can be obtained with a PPS system at any one place. The PPS provides a UTC time transfer accuracy of 200 nsec (95%) referenced in the time kept at the U.S. Naval Observatory and is denoted as UTC (USNO). Velocity Measurement accuracy is specified as 0.2 m/sec (95%).

Access to the aforementioned PPS positions accuracies is controlled trough two cryptographic features denoted as ant spoofing (AS) and select availability (SA). AS is a mechanism intended to defeat deception jamming, a technique in which an adversary would replicate one or more of the satellite ranging codes, navigation data signal(s), and carrier frequency Doppler effects with the intent of deceiving the victim receiver.

AS Dithers the system clock, thereby corrupting TOA measurement accuracy to non AS capable users (SPS users).

The PPS reached full operational capability in spring 1995, when the entire 24 production satellite constellation was in place and extensive testing of the ground control segment and its interactions with the constellation was complete.

The SPS is available to all users worldwide. There are no restrictions on SPS usage. This service provides predictable accuracies of 100m (2 drms,95%) in the horizontal plane and 156m (95%) in the vertical plane. UTC(USNO) time dissemination accuracy is within 340 nsec (95%). The accuracy of the service is both US. DOD and DOT established based on US security interests. AS is usually a primary error source in a SPS-derived position fix.

SPS initial operating capability was attained in December 1993, when a combination of 24 prototype and production satellites were available and position determination/timing services complied with the associated specified predictable accuracies.

The use of differential GPS (DGPS) enhances standalone GPS accuracy and removes common (correlated) errors from two or more receivers viewing the same satellites.

In the basic form of DGPS, one of these receivers is called the monitoring or reference receiver and is precise location is known. The other receivers are users and are denoted as “rovers” and are in line of sight of the reference station. The reference station makes code-based GPS pseudo range measurements, just as any standard GPS receiver, but because the monitoring station knows its precise location, it can determine the biases in the measurements.

For each satellite in view of the monitoring station, these biases are computed by differencing the pseudo range measurements and the satellite to earth station geometric range. These biases contain errors incurred in the pseudo range measurement process, like ionospheric and troposheric delay, receiver noise, and receiver clock offset from GPS system time. For real time applications, the reference station transmits these biases, which are called differential corrections, to all users in the coverage area.

The users incorporate these corrections to improve the accuracy of their position solution. DGPS achieves enhanced accuracy since the reference and user receivers experience common errors that can be removed by the user. Position errors less than 10m are typically realized. To provide even greater, sub meter accuracy, DGPS techniques that utilize phase information of the GPS satellite signal carrier frequencies have been developed and are being refined. These techniques are based on interferometric measurements of the satellite carrier frequencies and are occasionally referred to as carrier phase tracking. Extremely high accuracies can routinely be achieved by processing the received signal Doppler frequencies. We are talking about 20 cm for dynamic applications and millimetre level for static applications.

The constant motion of the satellite constellation demands that the receiver, in general, be capable of accounting for the changing Doppler frequency shift on L1. Where dual-frequency receivers are used, both the L1 and L2 are tracked. The shift in frequency arises due to the relative motion between the satellites and the receivers.

Typical satellite motion with respect to an earth fixed observer can result in Doppler frequencies between +-5000Hz with respect to the L1 and L2 carriers. Integration of the Doppler frequency offset results in an extremely accurate measurement of the advance in signal carrier phase between time epochs. Interferometric techniques can be applied to take advantage of these precise phase measurements, and, assuming sources of error can be mitigated, real-time positional accuracies in the centimetre range can be achieved for L1 and L2. Most error sources can be eliminated but one exception remains, and that is multipath – it can be mitigated, but not eliminated.

Several code based techniques have been proposed to enhance standalone GPS accuracy.

In local area DGPS, a LADGPS station typically serves receivers within close vicinity.

LADGPS accuracy depends on the fact that some of the pseudo range error components are common to all the receivers within the local geographical area. If the receiver is close to the reference station, the error components attributed to the space and control segments may be entirely removed while the overall error induced by the user segment can be significantly reduced.

LADGPS is limited by spatial decorrelation of the errors, when the distance is large.

Local Area DGPS can improve system accuracy from 33.3 (rms) to 3.3 (rms).

Conceptually, the simplest way to implement LADGPS is to place the reference GPS receiver at a surveyed location, compute the coordinate differences in latitude, longitude and height between the GPS-derived position and the surveyed location, and transmit those differences to the users. The user receivers employ these coordinate differences to correct their own GPS position solutions.

This technique requires that all receivers must coordinate their choice of satellites with the reference station, or the reference station must transmit the position corrections for all combinations of satellites in view. Better yet, it can transmit only the pseudo range corrections for all satellites in view, reducing in such a way the number of combinations it has to treat.

As the user moves away from the reference station, those common errors that are spatially correlated become increasingly decorrelated. So expressions were developed to deal with this issue.

For tropospheric and ionospheric delays, models are used to determine delays that depend on various parameters that change over space and time.

For example, on the ionospheric model, the delay depends directly on the total electronic content (TEC), which varies more than two orders of magnitude, so that we have to calculate either bound on the delays of typical delays.

Nevertheless, these expressions give us an idea of the magnitudes of the various error components that we may expect as the user moves away from the reference station.

The equations themselves and a detailed discussion about them are available in [35].

The corrections needed for satellite perturbations, ephemeris prediction error, and the satellite position errors all change with the viewing angle of the receiver. True many interactions, with the formulas available in [35] we arrive at the following conclusion that the error increases directly with the separation between the reference station measuring the error and the user receiver employing the correction.

The speed of electromagnetic radiation varies, depending on temperature, pressure, and relative humidity, as it passes through the troposphere. Because these factors depend on local conditions such as cloudiness and precipitation, the correlations in the delays at two receivers due to the troposphere usually decreases more rapidly than for delays caused by the ionosphere. At a receiver separation of 100 Km, the surface refractivity's are uncorrelated, and consequently, the difference in troposheric delays are uncorrelated.

Up until 100Km, again explained by the formulas in [35], the error in the correction of tropospheric delays is proportional to the separation between the user and reference station.

Formulas derived in [35] also tell us that the difference in delays experienced by the reference station and the user 100 km away due to errors from ionospheric delay is typically 0.03m.

We have that the difference in delays experienced by the user and reference stations owing to different viewing angles and to tropospheric and ionospheric phenomena varies directly with the horizontal distance between them. The rate at which the distances increase with distance depends on factors that vary with time. Typical differences for separations of 100 km are found to be 1m, except for the effect of the troposphere on altitude differences.

These differences in the delays translate to differences in pseudo range measurement errors with the effect on the position solution also being a function of the dilution of precision.

To extend the range of accurate DGPS, reference stations may be distributed (three or more), along the perimeter of the region of coverage. The user receiver then obtains a more accurate correction estimate by a weighted average of the corrections from the stations.

Because the error in the estimated corrections varies with distance from the station, the weights may be determined by geometric considerations alone to give the largest weight to the closest station, such as by choosing those weights that describe the user position as the weighted sum of the station positions.

A common two step approach to using multiple monitoring stations to improve the accuracy of the user position estimate is a two step process like this. In the first step, the pseudo range corrections from each monitor are used to determine the position of the user individually. The second step entails computing a weighted average of the individual position estimates to provide a more accurate estimate. Each weight is formed from the inverse of the product of the distance of the monitor from the user and the standard deviation from the average of the estimates from that station, normalized by the sum of the weights. The error introduced by each monitor receiver is then diluted by its weight, so that if, for example, the weights were all equal, then each monitor receiver error would be diluted by a factor of 1/n. But since the errors are uncorrelated, the standard deviation of their sum is $1/\sqrt{n}$ .Thus, the standard deviation of the total error due to the monitors is decreased by a factor of $\sqrt{n}$ from that of one monitor.

In addition, the components of the total pseudo range error due to tropospheric delay and ionospheric delay can be calculated using tropospheric and ionospheric delay models.

These delays are then removed from the range residuals and the variance of the range residuals are compared to a threshold to check for excessive magnitude. If the variance for any station is too large, its measurements are not used in forming the weighted average of the position estimate. This test provides a means of testing and isolating a faulty monitoring station.

WADGPS attempts to attain meter level accuracy over a wide region by using a fraction of the number of reference stations that would be required by LADGPS to attain the same accuracy within the same coverage region.

The general approach is to decompose the total pseudo range error into its components and to estimate each component for the entire region, rather than just at the station positions. The accuracy, then, does not depend on the closeness of the user to a single reference station.

The WADGPS concept is a network of reference stations that aid in the accurate determination of satellite ephemerides, atmospheric delay, and discrepancies between GPS system time and satellite system time tags.

Time synchronization amongst elements of the network of reference stations is essential to ensure correction and measurement time-tagging.

There are three basic types of errors that must be dealt with: errors in the satellite ephemeris estimates, errors in the reported satellite clock times (i.e. error between time reported by the satellite and GPS system time), and errors due to atmospheric delays (both ionospheric and tropospheric).

Three sources of errors in the satellite ephemerides (satellite perturbations, satellite ephemeris prediction errors, and SA epsilon) are included in the SPS error budget. These errors must be avoided by having a network of reference stations predict and transmit their own more accurate determinations of the satellite ephemerides, based on their own measurements. The reference stations then transmit the three dimensional error in the reported ephemeris, or the ephemeris itself, so that the user receiver can accurately determine the resulting pseudo range error at its own position.

One proposed approach to measuring satellite positions is to reverse the basic GPS algorithm. Here, four or more ground stations whose positions are accurately known each calculate the pseudo range to a given satellite after estimating and removing the atmospheric delays. If the ground stations have their clocks closely synchronized, they can accurately determine both the range to the satellite and the difference between the satellite time reports and their own system time. However, this technique encounters some of the same error sources of standalone GPS. There are techniques that mitigate errors in the ground station measurement process.

The double differencing technique removes errors arising in both the satellite and reference station to enhance the accuracy of satellite position estimates. Measurements are made in sets, with each requiring two satellites and two reference stations. By differencing the pseudo range measurements made by each reference station to the same satellite, common satellite errors in the pseudo ranges are eliminated.

To obtain even greater accuracy with the differencing technique, carrier phase tracking is used instead of code tracking. As with the standard use of GPS, satellites may be tracked with more than the minimum necessary for increased accuracy.

Tropospheric and ionospheric propagation delays affect both the satellite position measurements made by the ground monitoring network and the user pseudo range measurements. Compensating for them is necessary if the greatest accuracy possible in the user position estimate is to be attained. The dependency of tropospheric delay on the pressure, temperature, and water vapour pressure is reflected in the various tropospheric delay models; thus, for the most accurate estimates of tropospheric delay, estimates of these quantities need to be made at the user positions. These measurements decorrelate quickly and are essentially uncorrelated at 100km.

Similarly, ionospheric delay depends on electron density, which may vary significantly over space and time. Here, too, measurements are needed to complete the model, which must take into account both the spatial and temporal variations of delay. Ionospheric delays are much larger than tropospheric delays, so that in virtually all WADGPS systems a large effort is put into obtaining accurate ionospheric delay corrections.

As an example of an ionospheric delay model, the ionospheric corrections in the FAA's WAAS are made from data that gives the vertical delays at a subset of a predefined set of 929 grid points, distributed somewhat evenly in latitude and longitude. The delays at some subset of the grid points are transmitted to the users. Each user receiver calculates the latitude and longitude of an ionospheric pierce point (IPP) for each satellite and calculates its delay by interpolating the delays from the four nearest grid points.

The troposheric propagation delay is a function of temperature, pressure, and relative humidity. Measurements of these quantities at widely spaced monitoring stations would be ineffective owing to the short spatial correlations of these quantities.

Instead, as an example, the WAAS user receiver determines an average signal delay as a function of the season (day of the year) that is valid at its position for elevation angles greater than 5º. WAAS does not determine the positions of satellites with elevation angles less than 5º.

With accurate satellite ephemerides and an accurate atmospheric delay model, the reference stations can accurately determine GPS system time or an internal monitoring network system time.

The pseudo range measurement contains errors in the satellite clock time, reference station clock time, tropospheric delay, and ionospheric delay.

If the atmospheric delays are accurately modelled and the reference station clock carefully monitored, the errors they introduce into the pseudo range measurement can be removed. And if the satellite ephemeris is accurate, the true range is also known accurately.

Then, the discrepancy between the time reported by the satellite and the GPS system time can be determined and reported by the reference station to the user.

GPS receivers can be thought of as discrete time position and velocity sensors with sampling intervals of approximately 1 second. The need to provide continuous navigation between the update periods of the GPS receiver, during periods of shading of the GPS receiver's antenna, and trough periods of interference, is the impetus of integrating GPS with various additional sensors. The most popular are inertial sensors, but the list also includes dopplometters (Doppler velocity, altimeters), altimeters, speedometers, and odometers to name a few.

The method most used for this integration is the kalman filter. The kalman filter is an estimator. It estimates the instantaneous state of a linear system perturbed by Gaussian white noise. One of the key attributes of the Kalman filter is that it provides a means of inferring information by the use of indirect measurements. It does not have to read control variables directly, but it can read an indirect measurement (including associated noise) and estimate the control variables. In the use of GPS, the control variables are position, velocity and possible altitude errors. The indirect measurements are GPS measurements.

2.4.2.1.2. GALILEO

Galileo, the European contender to GPS, is expected to be into full operation around 2009, with the first satellites having been launched in 2006. In contrast to GPS system, Galileo is a civil operation, operated under public control. The program is managed and financed by the European Commission and the European Space Agency under a mandate from their member states. The basic features of Galileo are worldwide coverage, four positioning services for different user groups, integrity reporting and increased accuracy compared to GPS (at least to present GPS) [36].

Integrity refers to the capability of the system to recognize errors in terms of deviations of position fixes from the true position and to report these errors immediately to the users, an important feature especially for life-critical services like navigation or aviation.

Galileo works with 10 pilot signals that are emitted on three carrier frequencies , named E5a-E5b (12,164-1,214 MHz), Eg (1,260-1,300 MHz) and E2-L1-E1(1,559-1,591Mhz), which can be used in various combinations for offering positioning services tailored to special requirements of the different user groups.

As we can see, some of the carriers overlay GPS carriers, and so it is possible to use Galileo and GPS in conjunction in order to increase accuracy or availability of positioning services.

2.4.2.2. ORIENTATION

Orientation is normally achieved through the use of a digital compass. The Java Location API (discussed next) can be used to obtain both location and orientation information on J2ME compliant devices.

However, the orientation given by a digital compass is only in one plane, while we need orientation in all three degrees of freedom (roll, pitch, and yaw). For this, orientation sensors can be used (which may involve the use of accelerometers, gyros, and magnetometers).

2.4.2.3. JAVA LOCATION API (FOR J2ME)

J2ME JSR 179 or LAPI (Java Location API) [37] is a Java API for use with Java for micro devices such as java phones that provides a programming interface for getting information about current location and orientation in devices with such capabilities. It is an ideal API for use in a middleware solution which uses Java 2 Micro Edition as platform. It is independent of the technology used for location and orientation information.

Basically, the API consists in two interfaces, nine classes and two exceptions. The two interfaces are respectively LocationListener and ProximetyListener.

The LocationListener interface represents a listener that receives events associated with a particular location provider, which represents a source of location information., which contains information about latitude, longitude, altitude and precision (horizontal and vertical).

The ProximityListener interface represents a listener that receives events associated with detecting proximity to some previously registered coordinates.

The AddressInfo class holds textual address information about a location, like street address, postal code, number of door. If the value of a field is not available, it is set to null. The names of the fields use terms and definitions that are commonly used e.g. in the United States. Addresses for other countries should map these to the closest corresponding entities used in that country. This class is only a container for the information. The `getField` method returns the value set for the defined field using the `setField` method. When the platform implementation returns `AddressInfo` objects, it must ensure that it only returns objects where the parameters have values set as described for their semantics in this class.

The `Coordinates` class represents coordinates as latitude-longitude-altitude values. The latitude and longitude values are expressed in degrees using floating point values. The degrees are in decimal values (rather than minutes/seconds). The coordinates are given using WGS84.

The Criteria class defines the criteria for the selection of the location provider. It is up for the implementation to choose the best location provider available. This takes into account parameters such as cost, energy consumption, response time, accuracy and others.

The Landmark class represents a landmark, i.e. a know location with a name.

The `LandmarkStore` class provides methods to store, delete and retrieve landmarks from a persistent landmark store.

The `Location` class represents the standard set of basic location information. This includes the time stamped coordinates, accuracy, speed, course, and information about the positioning method used for the location, plus an optional textual address. The `Location` gives the accuracy of the coordinates as the radius of a circular area indicating the 1-sigma confidence level. The 1-sigma confidence refers to the standard deviation of the distribution. Assuming a normal distribution (which is not necessarily the case), this implies that the actual location is within the circle defined by the returned point and radius at a probability of approximately 68%. The actual location may thus be also outside of the circle, which has to be taken into account when using the location information in applications.

The LocationProvider class is the starting point for applications using this API and represents a source of location information. This may be implemented using any possible location methods, for example, satellite based methods like GPS, cellular network based methods, short-range positioning methods like Bluetooth Local Positioning, etc. The implementation may also combine the methods in various ways to get the optimal result.

The Orientation class represents the physical orientation of the terminal. Orientation is described by azimuth to north (the horizontal pointing direction), pitch (the vertical elevation angle) and roll (the rotation of the terminal around its own longitudinal axis).

It is not expected that all terminals will support all of these parameters. If a terminal supports getting the `Orientation`, it must provide the compass azimuth information. Providing the pitch and roll is optional. Most commonly, this class will be used to obtain the current compass direction. It is up to the terminal to define its own axes, but it is generally recommended that the longitudinal axis is aligned with the bottom-to-top direction of the screen. This means that the pitch is positive when the top of the screen is up and the bottom of the screen down (when roll is zero). The roll is positive when the device is tilted clockwise looking from the direction of the bottom of the screen, i.e. when the left side of the screen is up and the right side of the screen is down (when pitch is zero). No accuracy data is given for `Orientation`.

The QualifiedCoordinates class represents coordinates as latitude-longitude-altitude values that are associated with an accuracy value.

The LandmarkException is thrown when an error related to handling landmarks has occurred.

The LocationException is thrown when a location API specific error has occurred.

2.4.3. SERVICE DISCOVERY

For the past several years, competing industries and standard developers have been pursuing automatic configuration, now called by the broader term service discovery. Jini, Universal Plug and Play (UPnP), Salutation, Bluetooth Service Discovery Protocol, and Service Location Protocol are among the most important protocols for this [38].

Service discovery has a lot of potential in mobile and pervasive computing environments.

Service discovery in general means that a subset of the following capabilities is supported by a device [34]:

- The capability to make other devices aware of its own existence and presence in the network
- The possibility to make other devices in the network aware of the services offered by the device and to describe these services to them.
- The ability to search for a service in the network
- Zero-administration
- Interaction with other devices in the network to fulfil a function.

2.4.3.1. JINI

Jini was introduced by Sun Microsystems in 1998 and is based on Java technology. Jini is based on three protocols: Discovery, join and lookup [38].

Discovery and join occur when you plug a device into the network. Discovery when a service looks for a lookup service with which it can register, join when a service locates a lookup service and wants to join it.

Lookup occurs when a client or user locates and invokes a service described by its interface type and possible other attributes.

An interface type is written in Java Language.

For a client in a Jini community to use a service, first, the service provider must locate a lookup service by multicasting its request on the local network or locating a remote lookup service known to it a priori.

Secondly, the service provider must register a service object and its service attributes with the lookup service. This service object contains the Java Programming language interface for the service, including the methods that users and applications will call to execute the service, along with any other descriptive attributes.

Third, a client request a service by Java type and maybe other service attributes. The lookup server sends a copy of the service object over the network to the client, which will then use it to talk to the service.

Fourth, the client interacts directly with the service via the service object.

Jini consists of an infrastructure and a programming model that addresses how devices connect with each other to form an ad-hoc community, using the Java Remote Method Invocation (RMI) protocol to move code around the network.

We can view the Jini lookup service as a directory service or as a broker. When a service first becomes active, the multicast request protocol finds lookup services in the vicinity.

Lookup services announce their presence to services in the community that might be interested by using the multicast announcement protocol.

The unicast discovery protocol is used finally do establish communications with a specific lookup service known a priori over a wide-area network.

A Jini lookup service also maps the interfaces that clients see to service proxy objects, and maintains service attributes and processes match queries.

The clients normally download a service proxy, which is usually an RMI stub that can communicate back with the server, and which lets clients use the service without knowing anything about it. So, there is no need for device drivers in case the service provided is one for a device.

Jini only grants access to its services on a lease basis, that is, for limited time. A client request a service for a desired time period, and Jini negotiates a lease for that period, which it will grant for the client. This lease must be renewed before it expires. If not renewed Jini releases the resources associated with the service when the lease expires.

Jini also supports remote events and transactions that help programmers write distributed programs. Remote events notify an object when desired changes occur in the system. Recently published services or state changes in registered services can trigger these events.

2.4.3.2. UPNP

Universal Plug and Play (uPnP) is a Microsoft initiated Standard that extends the Microsoft Plug-and-Play peripheral model, aiming to enable advertisement, discovery and control of network devices, services and consumer electronics.

The uPnP forum [39] oversees the standard's developments, headed by Microsoft, in a similar process to the Java Community Process.

In uPnP a device can dynamically join a network, obtain its IP address, describe its capabilities as requested, and learn about the presence and capabilities of other devices. A device can leave a network automatically without leaving any state behind.

UPnP uses the simple service discovery protocol (SSDP) for service discovery. This protocol announces a device's presence to others and discovers other devices or services, being analogous to the three protocols of Jini.

A joining device sends an advertisement multicast message to advertise its services to control points, which function similar to Jini's lookup services.

A control point, if present, can record the advertisement, or other devices might also directly see this multicast message. UPnP can work with or without the control points, where's Jini cannot live without its lookup service.

A search multicast message is sent when a new control point is added to a network. Any device that listens and hears this multicast responds with a unicast response message.

The advertisement message contains a URL pointing to a XML document in the network describing the uPnP device's capability. Trough this XML document, other devices can inspect the advertised device's features and device whether they are relevant to them.

XML is a more powerful description of a service compared to the attributes Jini uses.

After a control point has discovered a device, it learns more about how to use, control, and coordinate with it by retrieving its XML description document. Control is described as a collection of SOAP (Simple Object Access Protocol) objects and their URLs in the XML document.

To use a specific control, a SOAP message is sent to the SOAP control object at the specified URL, and the device then returns action-specific values.

An uPnP description of a service includes a list of actions to which the service responds and a list of variables that model the service state at the runtime. The service publishes updates when these variables changes. A control point can subscribe to receive information about the changes of these variables. These updates are published by sending event messages that contain the names and values of one or more state variables. These messages are XML documents and are formatted using the General Event Notification Architecture.

UPnP also includes an additional high level description of services in the form of a user interface. This makes it possible for a user to directly control the service. If a service or device has a presentation URL, then the control point can retrieve a page from the URL, load the page into a browser and let a user control the device or view the status of the device, depending on the page capabilities.

AutoIP is an important feature o uPnP, and lets a device join the network without any explicit administration. When a device connects to the network, it first tries to acquire an IP address from a Dynamic Host Configuration Protocol server. If a DHCP server is not available an IP address is claimed automatically from a reserved range for local network use. This is done by the device by randomly choosing the IP from the reserved range and then making an ARP request to see if anyone else has already claimed that address.

2.4.3.3. SALUTATION

Salutation is another standard for service discovery and use, of special importance for devices and services of different capabilities. The architecture of Salutation provides a method for services and applications as well as devices to describe and advertise their capabilities to other services, applications and devices, enabling also search and discovery based on their capabilities.

The Salutation standard is managed by the Salutation consortium [40].

The Salutation manager and the transport manager are the two major components of the Salutation architecture. Similar to the Jini lookup service or the uPnP control point, the Salutation manager is the core of the Salutation architecture. It is a service broker. The service providers all register their capabilities with the salutation managers. When a client asks the local Salutation manager for a service, performing a service search, all the Salutation managers coordinate to perform the search. The client can then use the resulting service that was returned. A salutation manager sits above the of the transport managers that provide reliable communication channels, regardless of the underlying network transports.

A transport independent interface to server and client applications is provided by the Salutation Manager. This interface is called SLM-API (Salutation Manager API), and includes service registration, discovery, and access functions.

Another interface, called SLM-TI (Salutation Manager Transport Interface), sits between the Salutation Manager and the Transport Manager and works to achieve transport independence in Salutation.

The transport manager is an entity which is dependent on the network transport it supports. More than one transport manager, attached to multiple physical networks, may be handled by the Salutation Manager.

The Salutation manager sees the transport layer of the architecture trough SLM-TI performing the service registration, service discovery, service session management and service availability functions.

In discovery, the Salutation manager discovers other Salutation managers and its registered services. In doing that, it uses matching types and sets of attributes specified by the local Salutation Manager. This feature is called capability exchange and is needed because services are registered with the local Salutation Manager running in the same equipment.

While conceptually similar to Jini, this service works as a distributed lookup service while Jini's lookup service does not.

In registration, all registrations are done with the local Salutation manager or a near one connected to the client. This Salutation manager contains a registry of services that are registered.

About service availability, the local Salutation manager can be asked by a local application to periodically check the availability of services, a procedure that is done between the local manager and the corresponding manager.

About service session management, the Salutation manager might not be involved at all in message exchanges in the service session, depending on the mode of operation. There are three modes of operation of the service session: native, emulated or salutation.

The native mode of operation exchanges messages trough a native protocol, so the Salutation manager is never involved.

In the emulated mode of operation messages are exchanged by the Salutation manager protocol between the client and service but their content's are never inspected.

In the salutation mode, messages are exchanged by the Salutation manager protocol between the client and service and their format is defined by the Salutation manager.

2.4.3.3.1. SALUTATION LITE

Salutation lite is nothing more than a scaled down version of salutation for use in devices with small footprints.

It is more adequate to devices like handheld computers or palm-size computers and networks like IR and Bluetooth where bandwidth is low.

2.4.3.4. BLUETOOTH'S SERVICE DISCOVERY PROTOCOL

Bluetooth SDP (Service Discovery Protocol) is the service discovery protocol specific to Bluetooth networks.

Contrary to Jini, uPnP, Salutation, or SLP (Service Location Protocol), it is specific only to Bluetooth devices and networks. It primarily addresses the service discovery problem. It does not provide service brokering, service access, service advertisement, or service registration, and there is no event notification when services become unavailable.

Bluetooth SDP supports searching by service attributes or service class. It also supports service browsing. Service browsing is used when the client does not have prior knowledge of the services that are available in its vicinity.

Bluetooth SDP is structured as a Bluetooth profile and runs on a predefined connection oriented channel of the L2CAP Logical Link Layer.

2.4.3.5. SERVICE LOCATION PROTOCOL

SLP (Service Location Protocol) [41] provides a scalable framework for the discovery and selection of network services. Using SLP, computers using the Internet need little or no static configuration of network services for network based applications.

SLP uses service URLs, which define the service type and address for a particular service. Based on the service URL, users or applications can browse available services in their domain and select and use the one they require.

There are three types of agents on SLP: The UA (User Agent), the DA (Directory Agent) and the SA (Service Agent).

The UA sends service discovery requests on a user's applications behalf. The SA broadcast advertisements on behalf of a service. The DA is a centralized service information repository. It caches advertisements from SAs and processes discovery queries from UAs. An SA makes itself known by registering with a DA. The registration message contains the URL for the service and the service lifetime, and a set of descriptive attributes of the service. This registration is periodically renewed and the DA caches it, sending a acknowledge message back to the SA.

To discover a service, the UA sends a service request message to the DA in order to get the service's location. The DA then responds with a service reply message that includes the URLs of all services that mach the request of the UA. The UA then access one of the services the DA has returned.

In SLP, the existence of the DA is optional. In case the DA does not exist, as is common in small networks, the service request messages are sent directly to the SAs.

SLP supports searching for attributes based on string queries and also service based browsing.

It is possible to use search query operators like AND, OR, comparators and substring matching, which is more powerful than Jini or uPnP attribute searching, which can only search against equality.

2.4.4. THIRD GENERATION MOBILE NETWORKS

The 3rd Generation Partnership Project (3GPP) [42] is working on standards for the 3rd generation mobile cellular networks based on the WCDMA radio access scheme and the UTRA (Universal Terrestrial Radio Access) radio interface. 3GPP come into existence as the need for a certain degree of coordination was evident between the different standardization bodies working on 3G systems, which was particularly important because one of the goals of IMT-2000 was global roaming.

The term IMT-2000 refers to a set of radio interface standards that fulfill the requirements for 3G networks.

The term 3G (3rd Generation) is defined as "a term coined by the global cellular community to indicate the next generation of mobile service capabilities in terms of bandwidth and network functions. These service capabilities in turn allow advanced services and applications, including multimedia".

2.4.4.1. WCDMA

WCDMA (Wideband Code division Multiple Access) evolved from the code division multiple access scheme (CDMA) and offers high bandwidth (up to 2Mbps) to users of the mobile network.

WCDMA is a direct sequence CDMA (DS-CDMA) system [43]. It uses spreading codes to transform the user signal into a spread spectrum coded signal. These spreading codes are used to provide access to multiple users simultaneously. WCDMA uses Direct Sequence Spread Spectrum (DSSS), which is a spreading technique that uses a carrier that remains fixed to a specific frequency band. The data signal is then spread into a much larger range of frequencies using a specific encoding scheme, rather than being transmitted into a narrow band. This encoding scheme is known as pseudo-noise sequence (PN sequence).

WCDMA uses a carrier signal of approximately 5 MHz, while original DS-CDMA systems, like the IS-95, used a carrier of about 1 MHz, being referred now as narrowband CDMA.

Third generation systems based on the 3GPP specifications use the asynchronous network based scheme of WCDMA, where synchronization between base stations is not required.

WCDMA supports two basic modes of operation: one for paired spectrum and one for unpaired spectrum. Here pairing refers to the frequency bands available for communication. The FDD mode (Frequency Division Duplex) is used for the paired spectrum, while TDD (Time Division Duplex) is used for the unpaired spectrum.

In the FDD mode, separate 5 MHz carriers are used in the uplink and downlink directions. Information transfer is then symmetric. Data can be exchanged in both directions simultaneously. This is the traditional GSM mode of operation.

In the TDD mode, only one 5 MHz carrier is used for downlink and uplink. The benefit of the TDD mode is that the uplink and downlink Bandwidths can be different. For example, the downlink bandwidth can be substantially higher than the uplink bandwidth. The information transfer is said to be asymmetric.

2.4.4.2. ARCHITECTURE OF A UMTS NETWORK

A typical UMTS network can be modelled in three sub-systems: UE (User Equipment), AN (Access Network) and CN (Core Network) [43].

The UE is a device used by a subscriber (user) to access network services. The UE is divided in two logical parts: The ME (Mobile Equipment) and the USIM (Universal Subscriber Identity Module). The ME is further divided into two distinct functional groups: the MT (Mobile Termination) and the TE (Terminal Equipment). The MT performs functions radio transmission termination, authentication, and mobility management. The TE manages the hardware and hosts user applications. The USIM contains the logic required to securely identify the user. It contains the IMSI (International Mobile Subscriber Identity – the permanent identity of the user), the shared secret key (used for authentication), phone book and some other information. The USIM resides on a smart card that can be inserted or removed from the ME. The smart card is called the UICC (UMTS Integrated Circuit Card). The USIM on the UICC card is provided by the service provider.

The AN resides between the UE and the CN. It performs the functions specific to the access technique. In case of the UMTS, the AN performs the functions specific to the WCDMA air interface. The CN, however, may be used with any access technique.

The AN in UMTS allows two different types of AN systems to interface with the CN network: The BSS (Base Station Subsystem) and the RNS (Radio Network Subsystem). BSS is the legacy of the GSM era, and the RNS is the newly standardized access network for UMTS networks for release 99 and above of 3GPP specifications.

Both BSS and RNS have similar structures. They are composed of a Base Station Controller (BSC) and one or more Base Transceiver Stations (BTS). BTS and BSC are the names of the components in the BSS system. In the RNS system they are named Node-B and RNC (Radio Network Controller) respectively. RNS is also known as UTRAN (Universal Terrestrial Radio Access Network).

The CN consists of the entities that provide support for mobility management, call control, switching, session management, routing, authentication and equipment identification.

There are two types of traffic handled by the CN: voice traffic and data traffic. While 2G networks where designed primarily for voice traffic and GPRS introduced data traffic, the UMTS CN is an evolved GSM/GPRS core network that is divided in two domains: the CS (Circuit Switch) domain for voice traffic and the PS (Packet Switched) domain for data traffic.

We refrain here from discussing the CS domain in great detail since is the data traffic we are interested about.

The PS domain uses packet-switched connections for communication between the UE and the destination. To route packets in the PS domain, a routing entity is required, so the PS domain has a SGSN (Serving GPRS Support Node). The SGSN performs both functions of the database and the switching. The PS domain also has the GGSN (Gateway GPRS support node) that provides connectivity to external packet switched networks.

There are also entities common to both PS and CS domains. These are the HLR (Home Location Register) located in the home network of the subscriber, the AuC (Authentication Center) which holds authentication information, the EIR (Equipment Identity Register) which monitors the legitimacy of a UE used in the network, the SMS-GMSC (SMS gateway MSC) and SMS-IWMSC (SMS Interworking MSC), specific to SMS functions.

There are also some service specific entities like the GLMC (Gateway Mobile Location Center), some Camel entities and the CBC (Cell Broadcast Center).

Apart from the CS and PS domain, the IMS (IP Multimedia Subsystem) subsystem is part of UMTS in Release 5 and above of the 3GPP specifications. The IMS uses the services of PS domain to provide IP based multimedia services.

The highest level of the hierarchy of the UMTS network is the PLMN (Public Land Mobile Network), uniquely identified by the PLMN identifier. The PLMN identifier is composed of the MCC (Mobile Country Code) and the MNC (Mobile Network Code).

The second level is the LA (Location Area), an area in which an MS (Mobile Station) can move freely without updating its current location in the VLR (Visitor Location Register, a CS domain entity that holds the database of subscribers). In case the MS moves outside the LA it informs the VLR of its new position through location update procedures. Each LA is uniquely identified by a LAI (Location Area Identifier), which includes, besides MCC and MNC, the LAC (Location Area Code).

The third level (inside a LA) is the cell. A LA as one or more cells and is the last level of the hierarchy. The cell is identified by the CI (Cell Identity), unique within a LA. To identify a cell uniquely across PLMNs, the CGI (Cell Global Identity) is defined, which is composed of the LA and the CI.

Mobility handling happens in the CS domain through the use of Location Areas, Cells, and the VLR. Since the MS updates its location when changing LAs, the VLR always has accurate information on the LA of the MS. Whenever the MS receives a call, the VLR pages the MS too seek the exact location of the MS (its cell), which is then used to establish a connection.

The RA (Routing Area) is the correspondent entity in the PS domain to the LA in the CS domain, and it too may include one or more cells. It is defined as an area in which an MS may move freely without updating its current location at the SGSN. In case the MS moves outside of the current RA, it informs the SGSN through the routing area update procedure. A major consideration here is that an RA is always inside an LA, so an LA may contain one or more RAs, each one uniquely identified by the RAI (Routing Area Identifier), which is the LAI plus a RAC (Routing Area Code).

A URA (UTRAN Registration Area) is defined by an area covered by a number of cells. It is nothing more than an abstraction used between cells and the RA. A URA contains one or more cells and the RA contains one or more URAs. The URA is used to track the location of the MS within the UTRAN, and is uniquely identified by the URA identity.

In UMTS, a number of identifiers are used for the purpose of addressing and identification. A subscriber is uniquely identified by the IMSI (International Mobile Subscriber Identity), composed by the MCC, the

MNC and the MSIN (Mobile Subscriber Identification Number). The combination of MNC and MSIN is called NMSI (National Mobile Subscriber Identity).

The MSISDN (Mobile Subscriber ISDN) is the mobile number used to contact a person. A subscriber can have multiple MSSIDNs, and so multiple services. MSISDN numbers are based on E.164 ITU-T specification, and are composed of a CC (country code), a NDC (National Destination Code) and a SN (Subscriber Number).

The Packet Data Protocol (PDP) address of the UE is the address used for communication between the UE and entities of a PDN (Packet Data Network). This address is most commonly an IP address (ipv4 or ipv6), since the most common PDN is based in IP (Internet Protocol). This address is attributed either statically or dynamically trough the GGSN. A static PDP Address is allocated permanently by the network operator of the home PLMN. Dynamic assignment is more used since IP address are a scarce resource.

A dynamic PDP Context address is allocated during the activation of PDP Context. A PDP Context can be viewed as a set of information maintained by the UE, SGSN and GGSN. It contains a PDP type, identifying the type of the PDN, like for example Ipv4 or Ipv6, QoS Information, and other session information. Activating the PDP context refers to creating the PDP Context at the UE, SGSN and GGSN so that the UE can communicate with an entity in the PDN using the PDP address that was attributed.

An MS (Mobile Station) is identified by the IMEI (International Mobile Equipment Identity), composed by the TAC (Type Allocation Code) and the Serial Number (SNR).

A location number is also needed to provide location services (LCS). The location number is composed of the CC, the NDC and a LSP (Locally Significant Part), which exact structure is a matter of agreement between the PLMN operator and the national numbering authority in the PLMN's country.

The core network entities, including MSC, GMSC, SGSN, GGSN, EIR, HLR and VLR, are identified using E.164 numbers. Besides the E.164 number, the SGSN and GGSN also require a GSN address (an IP address), because they communicate with each other using the IP protocol. In the UTRAN, the RNC is globally uniquely identified by the global RNC identifier, composed by the PLMN identifier and a RNC identifier.

There are some temporary identities like the TMSI, LMSI, MSRN and RNTI.

The TMSI (Temporary Mobile Subscriber Identity) is used to hide the IMSI for questions of security. The TMSI is temporarily allocated by the VLR or by the SGSN. It is possible that two temporary identities be used, one for CS domain and other for PS domain, in which case the one for PS domain is called P-TMSI. The TMSI (or P-TMSI) has only local significance within the area controlled by the VLR (or SGSN).

The LMSI (Local Mobile Station Identifier) is a logical identifier allocated and used by the VLR to speed up database search. The VLR sends the LMSI with IMSI/MSISDN to the HLR during message exchange and the HLR stores it and uses it in further communications with the VLR, but does not use it further than this.

The MSRN (Mobile Station Roaming Number) is a roaming number allocated by the VLR to facilitate roaming. The MSRN has the same format as the MSISDN but is not the same. The MSRN is allocated by the visited network according to the numbering plan of the visited PLMN. The MSRN may be equal to the MSISDN when the MS is on the home network. It is used to route calls to the MS. When a mobile terminated call is received by the GMSC, it queries the VLR (via HLR) for a number through which it can route the call and the VLR allocates a MSRN for the MS and passes it to the HLR which forwards it to the GMSC. The GMSC then uses the MSRN to route the call to the MS via MSC/VLR.

The RNTI (Radio Network Temporary Identity) is allocated by the UTRAN, while TMSI,LMSI and MSRN are allocated by the CN. The RNTI is used as a UE identifier to exchange signaling messages between UE and UTRAN. There are various types of RNTI, one of which is the s-RNTI, allocated by the Serving RNC, in charge of the radio connections between the UE and the UTRAN, for all UE that have a RRC (Radio Resource Control) connection. There are also the d-RNTI, c-RNTI and u-RNTI and a few other identifiers [43].

Traditionally, the signaling traffic generated by UMTS was carried by SS7 networks using protocols like MTP (Message Transfer Part). The use of SS7 implies that a network entity has to support two interfaces: one for signaling (SS7) and one for data (IP). If it were possible to join the two interfaces in two one unique interface based on the IP protocol by developing a reliable transport protocol that could remove some of the drawbacks of TCP (like support for multiple IP addresses on the same connection), then the two networks could be converged. For this the SIGTRAN (Signaling and Transport) Working Group of IETF has developed SCTP (Stream Control Transmission Protocol). It has also developed adaptation protocols, designed in such a way that their users are unaware of the fact that SS7 protocols have been replaced with IP protocols. In between the many adaptation layers introduced, the two important protocols developed were M3UA (MTP3 User Adaptation Layer) and SUA (SCCP User Adaptation Layer). 3GPP has chosen M3UA for IP signaling transport in the AN as well as in the CN.

2.4.4.2.1. THE SCTP PROTOCOL

The SCTP protocol [44] is a transport level datagram transfer protocol that operates on top of an unreliable layer (IP). Like TCP, SCTP provides reliable transport service, ensuring all the data is transported across the network without errors and in sequence.

SCTP works on the notions of associations and streams. An SCTP association is similar to a TCP connection, but it can support multiple IP addresses at either receiving or transmitting ends. The SCTP association is composed of multiple streams, and the sequenced delivery is guaranteed within a single stream.

SCTP retransmits lost messages, like TCP, to establish its reliable service. However, unlike TCP, the retransmission of lost messages in one stream does not block the sending of new messages in another stream, thus solving the issue of head-of-the-line blocking of TCP.

2.4.4.2.2. THE IP MULTIMEDIA SYSTEM OF 3GPP

The IP Multimedia System was introduced in Release 5 of the 3GPP specifications [43][45][46]. The standardization of IMS services, unlike the PS and CS domain, is deliberately kept outside the IMS specifications. Only basic services are provided by IMS specifications. The IMS framework enables the network operator to deploy IP multimedia applications in a network independent manner. It does not have to depend on 3GPP to standardize services.

It is assumed that the standard services available on the internet will be reused and the PS domain is used to provide underlying transport for both signaling and bearer channels. The AN (UTRAN) and PS collectively provide the bearer for user and control plane.

3GPP has chosen SIP (Session Initiation Protocol) [47] and related protocols for session establishment and management. It is used for signaling between the UE and the IMS and between entities in the IMS. It is also used by the IMS to terminate sessions in the Internet (voice and multimedia).

The IMS requires SIP components, a subscriber database, a service platform and interworking entities to function.

In between SIP components, there is the UA (User Agent), which is the UE, the proxy or registrar which is the CSCF (Call Session Control Function), which is of three types: P-CSCF (Proxy-CSCF), I-CSCF (Interrogating CSCF) and S-CSCF (Serving CSCF). The S-CSCF acts as the registrar, to which the UE has first to register.

The subscriber database is provided by the HSS (Home Subscriber Server), which is an evolved HLR, including interfaces for CS and PS domains, and also for IMS database functions. The I-CSCF or S-CSCF communicate with HSS to obtain subscriber data over the Cx interface (a DIAMETER [48] protocol interface application cxdx [51]), and the AS (Application Server) communicate with it via the Sh interface (another DIAMETER [48] protocol interface application sh [50]) to obtain data.

The service platform is where the service execution logic resides and is also called the AS (Application Server). In IMS there are various AS: The SIP AS, the OSA SCS (Open Services Architecture Service Capability Server), and the IM-SSF (IP Multimedia Service Switching Function). SIP AS hosts SIP based services, OSA SCS hosts services based on OSA APIs, and IM-SSF hosts services based on CAMEL network features.

Interworking entities include the BGCF (Breakout Gateway Control Function), MGW(Media Gateway) and MGCF(Media Gateway Control Function), which are required for interworking with PSTN (Plain old Switching Telephone Network) and other networks.

In the IMS the USIM is replaced by the ISIM, which holds the UE’s identity and other necessary information, thought it is possible for a UE with an USIM to access the IMS.

The P-CSCF is the first contact point in the IMS. If the user is roaming, the P-CSCF is in the visited network, while if the user is on the home network, the P-CSCF is also on the home network. The S-CSCF in association with the AS provides the IMS services.

Once the SIP sessions are established, the transport bearer does not follow the signaling path. The bearer path extends from the UE to the destination network via SGSN/GGSN, and this destination network may be another GGSN, an IP network, or an MGW. Typically the RTP (Real Time Protocol) [49] protocol is used on the user plane.

The I-CSCF is the first contact point within an operator’s network. It provides a single point of contact for entry into the network hiding its configuration from other network operators.

There is another optional component called the SLF(Subscription Locator Function), which function is to find the address of the HSS, if more than one HSS is present on the network. It is accessed by the I-CSCF or S-CSCF by the DX interface (trough a DIAMETER[48] protocol interface application cxdx[51]).

The SGW (Signaling Gateway Function) carries out the signaling conversion between SS7 based transport (at the transport level) used in pre-Release 4 networks and the IP based transport signaling, possibility used in post-release 99 networks (between SIGTRAN SCTP/IP and SS7 MTP).

2.4.4.2.3. THE DIAMETER PROTOCOL

The DIAMETER [48] protocol is a protocol, like RADIUS, used for AAA (Authentication, Authorization, and Accounting). DIAMETER is more comprehensive than RADIUS.

The DIAMETER protocol is unique in the sense it cannot be used alone. The protocol specified in [48] is called the DIAMETER base protocol, which must be complemented by another protocol to be used. This other protocol is referred to as the Diameter Application protocol.

The Diameter base protocol provides basic functionality while the application protocol provides specific functionality. The generic functionality of the DIAMETER base protocol is the following:

- Exchange of AVP (Attribute Value Pairs)
- Exchange and negotiation of protocol capabilities so that only supported messages are exchanged between peers.
- Establishing and terminating user sessions.
- Monitoring the state of peer connections.
- Exchanging accounting information.

The DIAMETER base protocol is developed by IETF. There is an application of DIAMETER for Cx/Dx interfaces [51] and another for Sh interface[50].

Only a subset of the DIAMETER base protocol is needed for the Cx/Dx and Sh applications. Namely, support only for the SCTP protocol, no use of sessions, and no support for accounting features.

2.4.4.2.4. SECURITY IN UMTS NETWORKS

We can subdivide the security in UMTS networks in user domain security, network access security and network domain security [43].

The user domain security function ensures that only authenticated users can gain access to mobile stations. This is assured by restricting access to USIM until the user has been authenticated by a PIN shared between the user and the USIM.

Network access principles are applied to ensure that only legitimate users are allowed to use the network. In this case, we have mutual authentication, the user authenticates the network and the network authenticates the user.

The used scheme is called UMTS AKA (Authentication and Key Agreement). This scheme, besides mutual authentication, also generates a cipher key and an integrity key. The integrity key is not used in the user plane for performance reasons, but the cipher key is used to encrypt communications.

We also already have talked about how TMSI and P-TMSI are applied in hiding IMSI from the airwaves, achieving identity confidentiality.

Included in network access security is also the mobile equipment identification function by its IMEI, getting it from the MS and checking it in the EIR.

All the information needed for the security protocols for mutual authentication is in the USIM and the AuC of the home environment.

Network domain security can be obtained by using MAPSec or by using IPSec. MAPSec is the protocol used to encrypt MAP (Message Application Part) protocol messages, and IPSec is the protocol used to encrypt IP (Internet Protocol).

The CS and PS domains are protected using MAPSec. The IMS domain is protected using IPSec. We discuss IMS security in the following section.

2.4.4.2.5. SECURITY IN IMS

The big difference between general UMTS security and IMS security is in the network domain security, which uses IPSec.

NDS/IP(Network domain security for IP based protocols) refers to how the protocol suite specified by IPSec is applicable to UMTS.

While IPSec defines the use of two security protocols, AH(Authenthication Header) and ESP(Encapsulating Security Payload), only ESP is adequate according to 3GPP specifications for the requirements of NDS/IP.

The architecture of NDS/IP uses the concepts of security domain and security gateway (SEG). A security domain is a collection of networks managed by the same administrative authority. A SEG is responsible for enforcing the security policy of a security domain towards other SEGs in the destination security domain.

A security domain may have multiple SEGs to avoid single point of failure or to improve performance. In that case the SEG may provide connectivity to all reachable security domain destinations or only to a subset of them.

The interface between two SEGs uses the IKE (Internet Key Exchange) protocol to negotiate, establish and maintain a secure tunnel between them. After the tunnel is established ESP is used to provide authentication and encryption.

The interface between network elements (NE) and SEGs is optional but if it exists it uses ESP and IKE.

2.4.5. AD-HOC NETWORKING AND DYNAMIC CONFIGURATION

Ad-hoc networks are envisioned as infrastructure-less networks where each node is a mobile router, equipped with a wireless transceiver. A message transfer in a ad-hoc network environment would either take place between two nodes that are within the transmission range of one another or between nodes that are indirectly connected via multiple hops trough some intermediate nodes [52].

Ad-hoc networks have dynamic topologies, asymmetric link characteristics, multi-hop communication, decentralized operation, bandwidth constrained, variable capacity links and energy constrained operation.

2.4.5.1. BLUETOOTH

One of the ad-hoc networking technologies more used in the market today is Bluetooth [53][54][34], a technology for short range wireless connections between devices.

Major Bluetooth design points are low power, low cost, and the ability to support high speed ad hoc networking, while providing for high levels of security.

Starting with a basic data rate of 1Mbps for versions up to 1.2, today the standard can support rates of about 3Mbps for version 2.0 + EDR (Enhanced Data Rate).

Bluetooth enabled devices can connect and communicate in a wireless fashion with up to seven devices in one piconet (a short range ad-hoc network), and can belong to several piconets simultaneously. These piconets are established automatically as devices enter or leave radio proximity.

Bluetooth can handle voice and data traffic simultaneously and operates at a unlicensed frequency band between 2.4 to 2.485 GHz using a spread spectrum, frequency hopping full duplex signal at 1600 hops/sec. Bluetooth achieves two-way communication trough TDD operation.

Within a Piconet, one device acts a master and the others as the slaves. The first device to initiate communication will act as a master. The slave devices will synchronize their internal clocks and frequency hopping sequences with those of the master.

A scatternet is formed by two or more overlapping piconets. It is possible for a master in a piconet to be a slave in another piconet. Scatternets are not synchronized.

Devices within a piconet setup links for communication depending on need. Two types of communication links have been defined: The SCO (Synchronous Connection-Oriented) link and the ACL (Asynchronous Connection-Less) link.

The protocol stack of Bluetooth is composed on the top by the Application layer, then by the adopted industry protocols (OBEX, WAP, AT commands, TCS BIN, UDP, TCP, IP, PPP, RFCOMM, SDP, Audio), then by the Bluetooth specific protocols (L2CAP,LMP) and then by the Bluetooth baseband and the Bluetooth radio layers, in order.

The LMP(Link Manager Protocol) defines messages that are exchanged between participating devices to setup and maintain links. These messages control authentication and encryption also, and are transported in the payload data field of a Bluetooth data packet.

The L2CAP (Logical Link Control and Adaptation Protocol) is used by upper layer protocols for data transport, and can be defined as an adapter between lower and higher layers. L2CAP messages are transported in the payload data field of Bluetooth data packets just as LMP messages are. L2CAP is responsible for upper-level protocol data segmentation and reassembly as well as transport of QoS information. L2CAP is a data link protocol that provides both connection oriented and connectionless services.

2.4.5.1.1. SECURITY IN BLUETOOTH

In Bluetooth, security is implemented in the link layer.

Bluetooth provides device authentication as well as message confidentiality. In order to achieve this, two kinds of keys are defined: The link key and the encryption key [55].

The link key is a 128 bit key, and is also known as the authentication key. It is used for device authentication as well as for generating the encryption key. The link key can be one of four types:

- Unit key: This is a semi permanent key that is generated within a Bluetooth device (BD) without interacting with other BDs. A unit key may be used as the link key between a pair of BDs if the memory available in the BD does not allow creation/storage of a combination key.
- Initialization key: This is a temporary key that has significance for a pair of BDs. The initialization key is deleted after creation of the combination key.
- Combination key: This is a semi permanent key that is created for every pair of interacting BDs.
- Master key: This is a key that is shared among several BDs in a piconet, and is useful in a multicast configuration.

The encryption key is derived from the current link key and is used for message encryption. It can be between 8 and 128 (in octet multiples) bits long.

A semi permanent key is one that is stored in non-volatile memory and is valid even after session termination. A session is defined as the duration that a BD is connected to a piconet. A piconet is the set of BDs that use the same channel. A temporary key, on the other hand, is only valid for the session duration. Unit and combination keys are semi permanent, while initialization and master keys are temporary.

The unit key is generated when the Bluetooth device is being operated for the first time. The BD generates a 128 bit random number RAND, which is input to an algorithm E2 along with the 48 bit BD_ADDR of the device to produce the 128 bit unique key.

The initialization procedure is needed when a pair of BDs (Say, BD_A and BD_B) do not share a link key or when the link key is lost, and communication between the two is needed. Before initialization, a PIN is entered securely into each of the BDs, either manually or by automated means.

The PIN (of length L octets) is augmented with BD_ADDR_B (assuming that BD_B is the claimant) to produce PIN' (of length 16 octets).

The verifier BD_A generates a 128 bit random number RAND, which along with PIN' is input to algorithm E2, producing the 128 bit initialization key as output. RAND is transmitted by the verifier to the claimant, which then computes the key as well.

The combination key is produced after the initialization key when two a pair of BDs need to communicate. Once the combination key has been generated, the initialization key is no longer needed and is discarded. The combination key can also be regenerated between a pair of BDs at a later time. In order to generate a combination key, a pair of BDs, BD_A and BD_B first generate a temporary key (say, KEY_A and KEY_B) using a procedure identical to that used to generate a unit key. $RAND_A$ and BD_ADDR_A are input to algorithm E2 by BD_A to produce KEY_A , while $RAND_B$ and BD_ADDR_B are input to algorithm E2 by BD_B to produce KEY_B. BD_A then XORs $RAND_A$ with the current link key (which is the initialization key soon after initialization) and transmits the result to BD_A. Each BD regenerates the RAND used by the other BD, and using the BD_ADDR of the other BD computes the KEY generated by the other BD. Knowing KEY_A as well as KEY_B the two are XORed by each BD to compute the combination key.

The master key is generated when the same key needs to be used for several BDs (one master and two or more slaves). The master generates two random numbers $RAND_1$ and $RAND_2$, and passes them trough algorithm E2 to produce the master key K_{master}. A third random number $RAND_3$ is generated by the master and passed on to each of the slaves. (Different slaves may receive different random numbers, if so desired, but this is not necessary). The slave inputs $RAND_3$ and the current link key with the master through algorithm E2 to produce a temporary key K_{temp}. The master does the same to produce K_{temp} as well. The master XORs K_{temp} with K_{master} and transmits the results to the slave. The slave computes K_{master} by XORing the received value with K_{temp}.

The encryption key (8-128 bits) is generated using the current link key (128 bits), the RAND (a random number with 128 bits), and the COF (a 96 bit ciphering offset number) which is input to algorithm E3.

Device authentication in Bluetooth is based on a challenge response mechanism. The verifier BD_A generates a random number RAND, and sends it to the claimant BD_B. The verifier and the claimant generate the authenticator SRES based on the E_1 algorithm. The claimant transmits the authenticator (SRES) to the verifier. If this matches the authenticator generated at the verifier, then the claimant has been authenticated.

2.4.6. SCALABILITY

Taking into account we are talking about large scale games, the scalability problem is one to be taken into account. The large number of users connected to the game network should not hinder its performance significantly.

To that effect, distribution of the workload by a number of servers should prove to be effective.

A distributed system is scalable, if it works with large number of clients and objects. Ideally, the same amount of work should be distributed by the different clients and servers and there should not be points of congestion in the communication architecture.

In terms of communication protocols, it is customary to use multicast to increase system scalability. However, common multicast is not reliable. Messages may be lost.

So, reliable multicast is a requirement of requirement of large scale distributed entertainment applications. However, it has been shown that current solutions for that effect have significant problems [56], examples of those being the nak or ack explosion problems. The reliable multicast transport working group of IETF [57] has been addressing the problem of reliable multicast transport, but only in the one-to-many approach, not in the many-to-many area. The experimental protocol PGM [58], the only many-to-many protocol that has reached RFC status in IETF, requires support by network elements. This is hard to get implemented in reality.

2.4.7. CONSISTENCY

The problem of guaranteeing consistency in synchronous applications is the focus of attention of many research groups [59]. The biggest part of this work has been done in the area of distributed discrete interactive media, media that changes its state only in response to operations made by the user. There is also much research done in the area of distributed interactive continuous media. This type of media changes not only through operations of the user, but through time too. Martin Mauve shows that approximations utilized on the discrete case are not usable on the continuous case.

The mechanisms that exist to prevent problems of inconsistency divide them self's in two great classes: Optimistic and Pessimistic approaches.

Optimistic approaches prevent inconsistencies trough the use of floor control or locking to prevent the occurrence of simultaneous operations in conflict.

Optimistic approaches, make possible the existence of inconsistencies repairing them as soon as they are detected in the most quick and efficient way.

Each of these types of mechanisms has its own advantages and disadvantages. In the case of optimistic approximations, these have the advantage of supporting real collaboration between users, but also the disadvantages of allowing for inconsistencies of short term, being rather complex in its implementation and quite dependent of application. As for the pessimistic applications, these have the advantage of not permitting real collaboration, but on the other side has the advantage of being real efficient and simple of implement.

2.4.7.1. THE DEAD-RECKONING ALGORITHM

The algorithm of dead reckoning is normally utilized to guarantee the consistency in environments of distributed virtual reality, with a combination of previewing of state and transmission of state.

The algorithm is based on the fact of the application knowing how the various objects in the virtual world should behave themselves in the absence of perturbations caused by the user. Therefore, in the absence of messages referring to a determined object, the application can preview its state, even if it cannot confirm it.

Each object over which dead reckoning is applied possesses one only controlling application, which is responsible for notifying the remaining applications when the state of the object is not what it should be in determined point in time. If an application who visualizes the object receives such a message, it uses the newly received state as the base for future previsions. To guarantee the correct functioning of the algorithm when using mechanisms of transmission which are unreliable, a heartbeat message is transmitted with the actual state of the object in pre-defined intervals.

This approximation to guarantee consistency possesses various advantages, of which the most important ones are robustness and scalability. In the meanwhile, it actually has some disadvantages, namely:

- It possesses a maximum number of short term inconsistencies, which implies its normal the existence of visual illusions and actions based on inconsistent states.
- Requires that each object be transmitted for each operation in its entirety, which may lead to great necessities in terms of bandwidth, especially when then size of the state exceeds the dozens of bytes.
- It is just possible to exist one only controller per object, which makes it impossible collaborative actions in which two or more users move one only object.

This approximation is adequate for distributed interactive simulations and for distributed virtual environments, so it is valid for augmented reality, but it becomes inadequate when the media possesses one or more of the flowing restrictions:

- The state of the media is complex
- The artefacts (or illusions) cannot be hided
- Its primordial true collaboration
- The actions which are based in inconsistent states are critical

2.4.7.2. THE LOCAL LAG

The local lag is a concept and mechanism used to attempt to reduce the number of short term inconsistencies. It is based on a simple principle. Instead of executing immediately one operation emitted by the user, this is delayed by determined period of time before its execution. This implies that the time mark of the message is bigger that the point of time in which the message is emitted by the user. The total local delay in this way introduced is called local lag. If this value is significantly higher, it may reduce significantly the number of inconsistencies of short term.

The big problem with this approach is the determination of the correct value of the local lag to apply. A too low value is ineffective and a too high value will have a too high impact on response time, which has to be small.

To determine the correct value of local lag to apply, obviously dependent of the application, normally three steps are followed:

- Step 1: A minimum value for the local lag is chosen, normally correspondent to the medium delay which is estimated for the application. If it were this value the chosen one there will only be short term inconsistencies when packets were lost on the net or were received with a delay considerably superior to the average.
- Step 2: We chose a maximum acceptable value for the time of response. This is the maximum interval time a user is prepared to lose between the operation it executes and its consequences. This value is normally obtained through tests to the psychological reactions of the users.
- Step 3: Finally, we choose a value for the local lag, somewhere between the obtained value in step 1 and the obtained value in step 2, whichever is considered most important for the application in consideration, being the response time or the number of short term inconsistencies.

2.4.7.3. REPAIRING SHORT TERM INCONSISTENCIES

The existence of short term inconsistencies is unavoidable, even when we utilize algorithms like dead reckoning or local lag, which significantly reduce, but not completely eliminate, this type of inconsistencies. It becomes important to find mechanisms to correct this kind of situations. Three approximations are known to solve this problem: Previewing and transmitting states, requesting states, and the time warp algorithm.

2.4.7.3.1. PREVIEWING AND TRANSMITTING STATES

An obvious solution is to combine the mechanism of dead reckoning with the mechanism of local lag. Instead of immediately executing an operation on the local object, the operation receives a time mark equal to the time at which the operation was emitted by the user plus the value of the local lag. The new state calculated for the object that is the result of the execution of the operation is transmitted to every other participant, with the same time mark of the operation.

When the physical time corresponds to the time mark of the operation, this operation is executed locally. In the meanwhile, the other participants already would have received the new state, immediately after it has been transmitted and will utilize it, without finding any type of short term inconsistencies. Only participants, who receive this state late, as an effect of loosing packets on the network or network delay above normal, will experiment a short term inconsistency while the state of the object is not repaired.

This approximation has two of the disadvantages of the traditional approach without local lag (only with dead reckoning):

- Is possible only one controller per object
- The state of the object must be small, because it is transmitted in its entirety in each operation

2.4.7.3.2. STATE REQUISITION

State requisition is a mechanism which objective is to avoid the need to transmit always the state in each operation. This is to make possible using states that are more complex, of size significantly larger of those permitted on the DIS (Distributed Interactive Simulation) protocol, which uses the combined mechanism of dead reckoning and local lag.

To solve this problem, it becomes imperative to transmit the state only when it is necessary to repair a short term inconsistency. For this, the mechanism has to:

- Identify which participants possess the correct state of the object in question
- Guarantee that at least a participant possesses the correct state of the object in question.

If those two rules are applied, whenever it happens to be a short term inconsistency on an object, the participant asks the object controller the retransmission of the object state, to correct its local state.

These two rules are valid if only one controller exists for each object in determined point in time. If the controller changes, it will have to be guaranteed the execution of all operations and the reparation of short term inconsistencies provoqued from the last controller, before the new one can control the object.

The main disadvantages of this approach are:

- True collaboration is not possible, because the existence of more than one controller for each object at the same time is not possible.
- The process of repairing short term inconsistencies is more delayed, because of the roundtrip time from the participant to the object controller, when the state is requested and received.

This approximation is adequate, in consequence, in cases where true collaboration is not required and that the frequency of short term inconsistencies is low.

2.4.7.3.3. THE TIME WARP ALGORITHM

This method is adapted from the area of parallel simulation of discrete events, and does not need every object to have its state transmitted, even in the case of an inconsistency of short term. The algorithm allows more than one user to work with an object at the same time and uses only local information to repair a state.

To use this mechanism, each participant records the state of the interactive media in determined point in time, and holds a file where it records all operations executed since them. When a short term inconsistency happens, the state of the interactive media is effectively turned back until the point in time immediately before the operation that provoked the inconsistency. For that, the participant restores the recorded state and applies again each of the operations recorded on the log file, until the last operation before the operation that caused the inconsistency.

The great disadvantage of this approximation is the fact that it needs complex application logic and spends a lot of memory and/or space on the disk. When the time warp algorithm is executed is also put on the processor a great processing charge and significantly reduced the response time. The response time diminishes also whenever the current state of the object is recorded in memory or disk.

2.4.8. MULTIMEDIA DATA HETEROGENEITY

Typically there are many different types of data exchanged in Collaborative Virtual Environments, and the same happens in augmented reality environments. Real time audio and video data, scene description data, typical 2D data, control data and state or update data. In addition to dealing with various types of data, continuous distributed interactive media can change their state in response to user operations as well as to the passage of time.

In response to this, is common to use or develop protocols to allow for multiple types of data or to use multiple protocols at the same time for each kind of data.

An example of a protocol that handles multiple kinds of data is MPEG (Motion Picture Expert Group) [60].

2.4.9. DATA DISTRIBUTION AND REPLICATION

Is essential in mobile computing to have the ability to replicate the data objects to increase availability and performance. It is common for shared data items to have different synchronization constraints depending on their semantics and particular use.

These constraints should be enforced on an individual basis. Replicated systems need to provide support for disconnected mode, data divergence, application defined reconciliation procedures, optimistic concurrency control, etc. Replication is a way in which the system ensures transparency for mobile users by replicating data at various locations or at various computers. Mobility of users and services and its impact on data replication and migration will be one of the main technical problems to be resolved.

There are many issues raised by the relocated data and mobility of users and services, such as:

- How to manage data replication, providing the levels of consistency, durability and availability needed;
- How to locate objects of interest. Should information about location be also replicated and to what extent;

- How the user moves affects the replication scheme. How should the copy follow the user? In general, data should move closer to the user;
- Is a mobile environment requiring dynamic replication schemes?
- Do we need new replication algorithms or the proposed replication schemes for distributed environment can be modified?

CORBA (Common Object Request Broker Architecture) is a specification of a distributed system. The specifications of this distributed system has been drawn up by the OMG (Object Management Group), which is a non profit organization with over 800 members, primarily from the industry. The most important goal of the OMG with respect to CORBA was to define a distributed system that could overcome many of the interoperability problems that occurs when integrating networked applications. The first CORBA specifications become available in the beginning of the 1990s. Currently, implementations of CORBA version 2.4 are widely deployed. The first first CORBA version system 3 systems are becoming available.

The specifications of corba are available at [61]. More information about implementations is in [62].

2.4.10. QUALITY OF SERVICE

Applications running over a pervasive network must support end-to-end Quality of Service (QoS) for users and developers, for example, in terms of performance, availability, maintainability, and survivability [52].

A pervasive network includes high speed networks, on one end of the spectrum, and limited bandwidth wireless connections on the other end of the spectrum. The wireless portion that interconnects mobile hosts / routers can change rapidly in unpredictable ways or remain relatively static over long periods of time.

These bandwidth constrained wireless networks typically support best effort voice and data communications, where the achieved “good put” is often lower than the maximum radio transmission rate after encountering the effects of multiple access, fading, noise, and interference, etc...

In addition to being bandwidth constrained, components in pervasive networks may be power constrained too because pervasive devices mostly rely on power (battery power) for energy. These environments are also characterized by continuous topology changes that demand stable and responsive system level adaptation.

Providing suitable QoS for the delivery of real-time audio, video and data in pervasive networks represents a number of significant technical challenges. It needs a suitable QoS framework paying particular attention to the performance of the signalling system under a variety of network, mobility and traffic conditions in support of fast reservation, restoration and adaptation. The system will manage resources according to the user specified QoS declarations and compete for these resources.

As many real time applications have been developed in the internet, the best effort delivery models become inadequate for these new applications. Two different models are proposed to guarantee QoS in the Internet by the Internet Engineering Task Force (IETF): The integrated services (IntServ) and differentiated services (DiffServ) models. In the IntServ model, network resources are explicitly identified and reserved. Network nodes classify incoming packets and use reservations to provide QoS. In de DiffServ model, resources are not explicitly reserved. Instead, traffic is differentiated into a set of classes, and a network of the most successful protocol, IPv4, allows DiffServ-style QoS to be applied.

However, applications such as streaming audio and video would be much better served under the IntServ model since they have a relatively constant bandwidth requirement for a known period of time.

TCP, used widely in the current internet, is not well suited to real time applications. Instead, Real time transport protocol (RTP) is usually implemented on top of UDP, which is better adapted to real time applications. This protocol mechanism is not enough to guarantee a specific quality of service (QoS) for a session between a sender and a receiver. Resource Reservation Protocol (RSVP) [63] is an attempt to provide real-time service through the use of virtual circuits. It is a resource reservation setup protocol designed for the IntServ model. RSVP is not a routing protocol but a control protocol, which allows Internet real time applications to reserve resources before they start transmitting data. That is, when an application invokes RSVP to request specific end-to-end QoS for a data stream, RSVP selects a data path relying on underlying routing protocols, and then reserves resources along the path. Since RSVP is also receiver-oriented, each receiver is responsible for reserving resources to guarantee requested QoS along its data path. The receiver sends a message to reserve resources along all the nodes on the delivery path to the sender.

The DiffServ architecture aims at providing simple and scalable service differentiation by recognizing that most data flows generated by most applications can ultimately be classified into a few general categories (traffic classes). It does this by discriminating and treating the data flows according to their traffic class, thus providing a logic separation of the traffic into the different classes.

2.4.10.1. QUALITY OF SERVICE IN 3GPP

UMTS (as also GPRS) defines four QoS classes. These are the Conversational, the Streaming, the Interactive and the Background classes [68].

The conversational class is the class to where conversations belong, and applies to any application that involves person-to-person communication in real-time such as video-conferencing and interactive video games, besides speech.

The basic qualities of the conversational class as it refers to speech are low delay, low jitter (delay variation), reasonable clarity (codec quality), and absence of echo. For multimedia applications it is also necessary to maintain correct relative timing of the different media streams.

This class is tolerant of some errors, since the dropping or the corruption of a voice packet, for example, lasting for a typical 20 ms is unlikely to be detected by the user.

The streaming class is to be used by real time applications that send information to a viewer or listener, but without having a user response (stream the information).

Because of the absence of interaction, there is no need for low delay, but the need for low jitter and media synchronization remains. The error tolerance remains, but higher quality is required.

Removal of the low delay criterion makes possible the use of buffering techniques in the end user equipment to even out the delay variation. Because of this, the acceptable level of jitter is higher than that for the conversational class.

The interactive class is to be used by devices and users that request data from another device. Examples of its use include some games, network management systems polling for statistics, Web browsing or searching databases.

There is a requirement for reasonable delay for an application timeout or reasonable prompt for human activities, but not as low as the conversational class. There is also a requirement for data integrity.

The background class is used by all applications that request data without any immediate need to process it or that receive data passively. Examples are file transfers, email and SMS (Short Message Service). The only requirement is data integrity.

In UMTS, QoS is established end-to-end depending on the policy of the network operator and the capabilities of the UE, and the specified needs of the applications.

The precise services offered over UMTS will depend on both the policy of the network operator and the capabilities of the ME. UMTS expects that an application used by the UE should be able to specify its QoS needs but the mechanism for this is outside the UMTS specifications. A possibility is the use of SIP [47] and SDP (Session Description Protocol) [81].

UMTS offers different scenarios for establishing end-to-end QoS and distinguishes between the mechanism used for the original ATM-based CS and PS domains core on one hand and the IMS core on the other.

The original method is the UMTS QoS mechanism where the QoS profile contained in the PDP context activation method is translated into QoS parameters that are included in the initial address message and ATM connection request, while IMS introduces the IP bearer service manager to handle QoS for IP. Some possible scenarios are:

- The UE has no IP bearer service manager, although it may support UMTS QoS and include a QoS profile in the PDP context activation request. In this case, the GGSN controls the QoS towards the remote terminal, while the remote terminal (or its own GGSN) controls QoS towards the UE. The GGSN will have DiffServ support and COPS support and can so interact with the P-CSCF to authorize a flow based on a SIP SDP media description, user's subscription rights, and resources actually available in the UTRAN.
- The UE supports UMTS QoS mechanisms and has an IP bearer service manager but does not support RSVP. In this case, the application QoS needs specified by SIP SDP parameters are mapped to IP QoS needs and then to the QoS profile to put in the PDP context activation. If the UE has received an authorization token from the CSCF, then this also goes into the activation request for the session.
- The UE supports UMTS QoS mechanism, has an IP bearer service and supports RSVP. Here the application needs are mapped to RSVP flowspec parameters and are included in the QoS profile for the PDP context activation. If the UE has received an authorization token, then it includes it in both the RVSP messages and the PDP context activation request.

2.4.11. MANAGEMENT OF NETWORKS AND SERVICES

Management of the network and management of its services are also important both for correcting possible working errors and to collect statistics of system execution.

For this, there are standards commonly used in every network, like SNMP (Simple Network Management Protocol) [69]. But, what changes when the network is UMTS (3GPP specified)?

What is defined and what is not defined in terms of network management?

In UMTS, the PLMN management objectives are the integrated operation of the elements of the PLMN using standardized functionality, the inter-PLMN operation of the mobile system, and the information of the QoS provided [70].

For this, there is a standardized OAM (Operations, Administration and Management) in the PLMN, so there is operational standardization, multivendor environment standardization, PLMN management facilities standardization.

Specific PLMN management facilities by operator and specific PLMN management or national requirements are considered out of the scope of standardization.

The PLMN interworks with other networks (PSTN, other PLMNs, etc…).

In the service and business areas of the PLMN concern: the administration of subscribers, their billing, the collection of revenue from other operators, maximization of revenue from network resources and provision of support for subscribers.

Subscriber administration consists of connecting the service to a subscriber, test such service and features, upgrade services and discontinue service.

The OAM in the 3GPP standards follow the architecture of a TMN (Telecommunication Management Network).

The services implemented on top of, for example, IMS should have its own management system for things not included in the basic system in the OAM (Subscribing a user is not needed to be programmed since the basic OAM does that).

2.5. RELATED WORK

Through our research for mobile and pervasive large scale augmented reality network middleware, we find none. So, to our knowledge, there is no network middleware specialized for mobile and pervasive large scale augmented reality applications. However, we did find some interesting augmented (and mixed) reality applications and platforms, and some other pervasive systems, that are related to the general area of research and that we talk about here, in this section on related work.

The MR platform (Mixed Reality platform) build by the MR systems laboratory of Canon, Inc. [71] is an example of a mixed reality platform (and so augmented reality). The platform includes a parallax-less stereo video see-trough HMD (Head Mounted Display) and a SDK (Software Development Kit) for a LINUX PC environment. The SDK is composed of a C++ class library for making runtime MR applications and related utilities such camera calibration tools. The SDK makes available, by using it, the following functions: Capturing video, handling a 6DOF(6 Degrees of Freedom) sensor, image processing such as color detection, estimating head position and orientation, displaying the real world image, and calibrating sensor placement and camera parameters of two cameras mounted on the HMD.

In [72], the authors introduce what they characterize as a scalable architecture for supporting interactive games on the Internet. But the architecture was not for pervasive games, not for augmented reality and was only for the internet, and was not really mobile. In their approach, the world is subdivided in partitions, with each partition each being assigned to a server. A client will join a server according to the position of the avatar it controls. Compared to a centralized architecture, this architecture is more scalable. Comparable to a fully distributed peer to peer architecture, it provides the means for detecting cheating in online games. Since interaction and accounting information must be sent to one of the servers for qualification and for verification, cheating between distributed game players will be minimized.

To speedup periodic updates and secure communication for interactions and accounting information, a hybrid scheme of communication is used between clients and the associated server using both TCP and IP multicast. Communication among servers is achieved using RTI (Runtime Infrastructure) services. The HLA (High Level Architecture) DDM (Data Distribution Management) is also used to limit the amount of communication between servers. The OM (Ownership Management) is also used to implement the transferring of avatars between the servers.

In [73] communication architecture for MMPGs (Massive Multiplayer Games) is introduced. Massive Multiplayer Games are multiplayer games that have a higher need for scalability, since they manage massive numbers of users.

The introduced communication architecture is based on the publisher/subscriber model, and the key issues addressed in the paper were scalability, ease of programming and dynamic system evolution.

This is also an internet based solution that does not take into account pervasiveness or mobility, not to mention augmented reality.

The division of the communication load into distinctive channels allows for the decoupling of the domains of the game enabling the overall system to scale flexibly with the underlying network infrastructure, map size or number of players simultaneously online. The underlying publisher/subscriber communication layer provides a high level API dramatically simplifying network programming in the game engine.

Communication is content based and there is no fixed correlation between the game players and the game backend servers, which permits the dynamic evolution of the system by adding or removing players, game servers, or game content on the fly.

In [74] the authors write about a middleware for large scale pervasive computing environments, however, it does not apply to augmented reality games.

They have built a distributed middleware to enable the construction and management of pervasive environments. These environments are called “Active Spaces” and are the basic elements of the large scale pervasive computing environment. In the paper the authors introduce the concept of “Super spaces” that should be available in the future which are a collection of reflective and recursive spaces (Active Spaces or other Super Spaces) that enable large scale management, operation and maintenance of pervasive computing environments.

An example augmented reality game application (thought not very pervasive) is ARQuake [75], a first person, indoor/outdoor augmented reality application. The authors of ARQuake presented architecture for a low cost, moderately accurate, 6DOF tracking system based on GPS, digital compass, and fiducial vision based tracking.

In [76] , the authors introduce the concept of distributing the load of the system in a massive multiplayer online game (MMOG) according to regions both to increase scalability and decrease latency. The middleware the authors talk about implements this through a distributed name service. This is done in C++ and an internet oriented architecture. While it has the advantages of increasing scalability and decreasing latency of internet online massive multiplayer games, and to an extent supporting internet mobility, it does not support the kind of pervasive mobility we are used to when we use a UMTS network with cellular phones or PDAs. It is also not a middleware specifically oriented to augmented reality.

From our analysis of the state of the art and related work, we can conclude that there is a need for a network middleware that supports the creation of large scale mobile and pervasive augmented reality game applications, and this is the proposal we talk about in the next section.

3. PROPOSAL AND EXPERIMENTAL DEVELOPMENT

3.1. PROPOSAL

3.1.1. INTRODUCTION

This work focuses on the creation of network middleware for mobile and pervasive entertainment, applied to the area of large scale augmented reality games. There is a tremendous opportunity for research and development in the area of massive multiplayer pervasive games applied to augmented reality, from the point of view of multimedia and communications. The installed infrastructure of mobile operators makes it possible to install distributed solutions directly linked to geographical locations bounded by its transmission cells. Solutions extending the work envisioned in [77] and applying map subdivision like in [73] may be useful. The need for killer applications to justify the overwhelming investments made is another important factor to consider. But the difficulties exist, because of the current characteristics of the mobile networks [78]. Bandwidth on mobile networks, though increasing, is a scarce resource when compared to fixed networks. Adding to this, both transient and persistent storage spaces on the mobile host are very limited. There are also the problems of mobility handling and disconnected operation.

The middleware that we propose to be created will incorporate experiences and results obtained from previous work from the candidate in the area of interactive distributed multimedia, more specifically in state transmission for a collaborative virtual environment middleware platform, the Status Transmission Framework (STF) [7][59]. This platform extended another platform, called ARMS – Augmented Reliable CORBA Multicast System [8], for event distribution. Knowledge will be used from areas as diversified as peer–to-peer computing, mobile and wireless networks, pervasive computing, embedded systems, multimedia protocols and systems, interactive distributed multimedia, and network gaming theory and protocols.

Figure 1 - Central level of the system

We may consider that distributed collaborative virtual environments have network requirements that will in its majority be common to augmented reality environments. The fundamental problem for collaborative virtual environments is how to maintain a consistent shared state of the virtual reality world [7] [59]. Another research topic is, from a quality of service point of view, how to efficiently transmit update messages so as to provide scalability, minimized delay, consistency and reliability.

Collaborative virtual environments also have the requirement of being able to handle multiple types of data, which may be multimedia data, state update and control data.

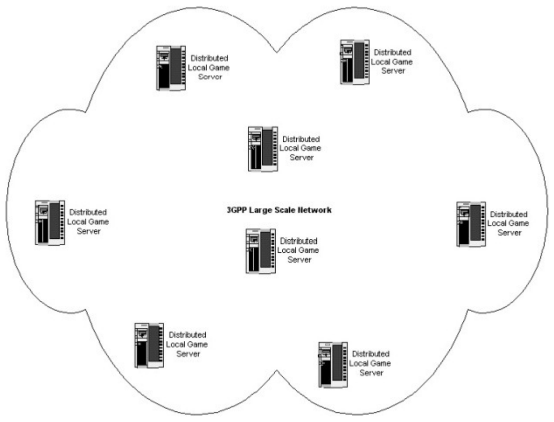

Distributed Local Game Server
Distributed Local Game Server
Distributed Local Game Server
Distributed Local Game Server
3GPP Large Scale Network
Distributed Local Game Server
Distributed Local Game Server
Distributed Local Game Server
Distributed Local Game Server

Figure 2 - Large scale network level of the system

The system that will be targeted by the middleware will be composed of 3 levels: the back-office central level, depicted in Figure 1, the large scale network level, depicted in general in Figure 2, and the personal area network level, depicted in general in Figure 3.

The back-office central will consist of one or more of a series of parallel servers and will serve as the main controlling station of the game administrator: The person responsible for starting, stopping and managing game performance and general maintenance tasks. This may be done for every specific game running on the system.

The large-scale network is the 3GPP network as it is being built by the specifications, where servers will be distributed according to some logic of spatial distribution and linkage of its location with specific geographic locations. These may be, in the extreme, the cells of the mobile communications network.

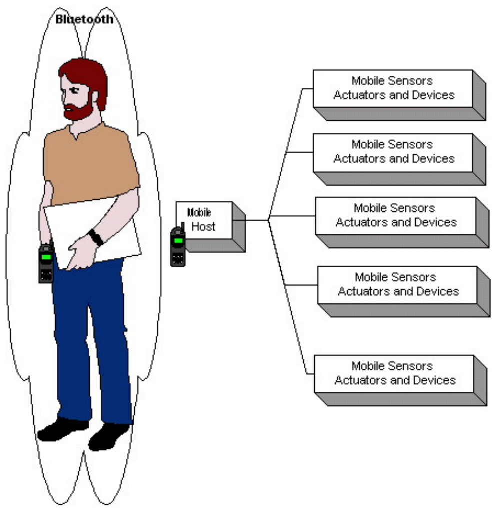

Bluetooth
Mobile Host
Mobile Sensors Actuators and Devices
Mobile Sensors Actuators and Devices
Mobile Sensors Actuators and Devices
Mobile Sensors Actuators and Devices
Mobile Sensors Actuators and Devices

Figure 3 - The personal area network level of the system

The personal area network level will consist of the network of pervasive devices dedicated to personal communications and to augmenting reality that the person carries with it. These may be sensors, actuators, and other devices that can communicate under Bluetooth or other means of communication. All those communicate to the mobile host, probably just a cell phone or specialized device connected to the large scale network, the 3GPP network. The player is so enabled to play games of augmented reality wherever it is.

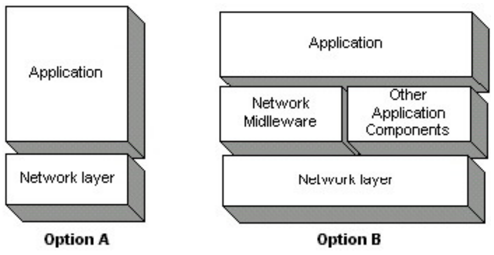


Figure 4 - Opting for network middleware architecture

Targeting this architecture will allow 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 can consider that building the augmented reality applications using the network middleware (option B of Figure 4) is better that building them standalone (option A of Figure 4). This is because with option B many games applications may then use the same application programming interface (API) to leverage network resources, giving it much faster service development and deployment.

The middleware produced will be build according to the characteristics of agile pervasive middleware [75], such as application – awareness, mobility, integration, interoperability, scalability, portability, adaptability, robustness and simplicity of evolution.

3.1.2. REQUIREMENTS SPECIFICATION

Is important to establish first the requirements needed to run our system. We are proposing to build a middleware for mobile and pervasive large scale augmented reality gaming, targeting the three levels described on the previous section.

The first thing we need to do is to establish the kind of devices, servers, programming languages and platforms the system will run on.

Today, mobile devices and gadgets come with more than just a cool and cutting-edge design. They come equipped with small brains and a common language, which makes them smarter and more powerful than ever before [79]. That language is Java.

Since Java can now run in servers, mobile devices and even small devices it’s clearly a language of choice for the programming of our middleware and its corresponding applications.

On the servers we will use the J2SE edition and on the small devices and mobile devices of the personal area network we will use the J2ME edition.

The middleware will use as localization technology any localization technology that provides enough precision and it’s available on the spot, using the Java Location API to abstract from location technology.

To provide truly ubiquitous positioning with precision, outdoors GPS would be used with carrier based techniques (which are already available commercially, thought at a price) and differential GPS techniques (such as Assisted GPS), while indoors some other solution would be needed, like acoustic tracking devices that employ ultrasonic sound waves that measure the position and orientation of the entity being tracked (This can be done through time-of-flight tracking and phase-coherence tracking). Galileu, because of the abstraction provided by the Java Location API, will be automatically used if it provides the needed precision when it becomes available. Orientation (see state of the art chapter) can also be obtained through the use of a digital compass and/or orientation sensors. Through the use of the Java Location API is possible to the middleware to obtain both location and orientation, if available. We require its availability in the main game device. So, we require both J2ME and Java LAPI and the existence of both an orientation and location technologies of sufficient precision on the main game device.

On the main game device, we feel we must demand an existing J2ME profile, and that profile is the MIDP 2.0, that runs on the CLDC 1.1 configuration of J2ME, which provides us with the capability to do floating point arithmetic. The main game device will be, preferably, a specialized device for the management of a personal area network of devices dedicated to augmented reality gaming. However, due to these specifications, there is nothing stopping it from being a common advanced mobile phone.

For the sensors and actuators, the requirements must not be so demanding, so we only require the CLDC 1.1 configuration, and we do not require any profile at all. We just require the possibility to use JSR-82 APIs (Bluetooth for Java) [84]. This for a sensor. For an actuator, optionally, it may have also the JSR-184 API (Mobile Media API) [87] and JSR-135 API (Mobile 3D API) [86]. This for it to be possible to build a multimedia actuator such as an augmented reality glasses device. An actuator that does not need these APIs may live without them.

For the servers, central and distributed, we will use J2SE, and demand simply enough processing power, memory capacity and disk capacity for the databases and the middleware and application that will run the game, for the probable load of the system. Obviously, there should be enough distributed servers for the dimension of the game area and the expected load of the game.

A basic requirement is the existence of a 3GPP (UMTS) network supporting IMS (IP Multimedia Service) where to run the central and distributed servers) and where the main game device connects wirelessly, that will be the main game network for the whole system.

3.1.3. PROPOSED GENERAL 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 is better that building them standalone. This is because then many games 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 thesis is being built according to the characteristics of agile pervasive middleware [80], such as application-awareness, mobility, integration, interoperability, scalability, portability, adaptability, robustness and simplicity of evolution.

3.1.3.1. 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 [48] API which includes the base protocol, the CX and DX [51] applications and the SH application [51] of 3GPP. This would communicate preferably through SCTP [44] (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 and works over IPSec.

The terminal (UE) from the personal area network will communicate with the central server through SIP [47] to initiate the session, authenticate itself and get the details for the session through SDP [74] negotiation.

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) [84] on the mobile terminal.

The Status Transmission Framework Middleware on this level is called the STFSERVER API, which includes ARMS communications APIs, works above ARMSV6 itself and Sixrm multicast.

The schematics of the middleware architecture the central level are represented in Figure 5.

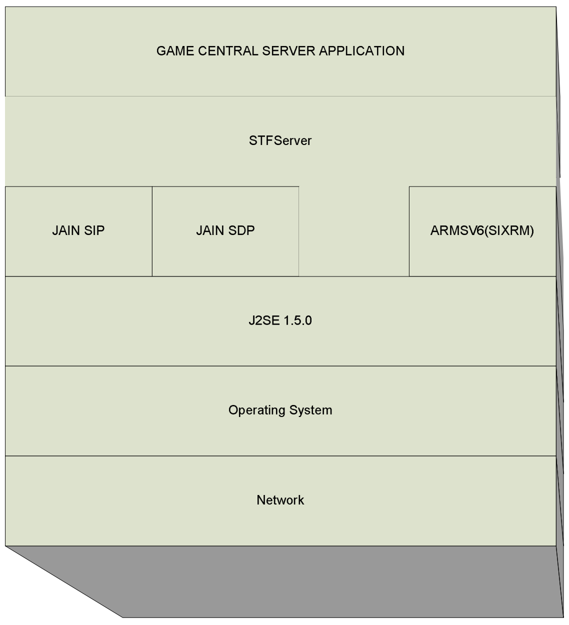


Figure 5 - Central level architecture

3.1.3.2. LARGE SCALE DISTRIBUTED 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 [14]. This protocol will also connect these servers with the central server.

The Status Transmission Framework Middleware on this level is the STFSERVER API, which includes ARMS communications APIs, works above ARMSV6 itself and Sixrm multicast.

The schematic of each of the distributed servers on the large scale distributed level is presented in Figure 6.

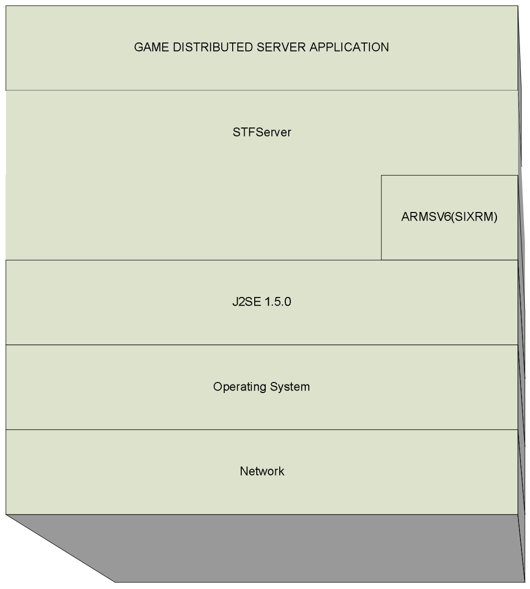


Figure 6 - Large Scale Distributed Level Architecture

3.1.3.3. 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 [82], and the MIDP – Mobile Information Device Profile - version 2.0 [83].

This central device must support also the Java Bluetooth API (JSR-82) [84], the Java SIP (Session Initiation Protocol) API for J2ME (JSR-180) [85] and the location API for J2ME (JSR-179) [37].

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 [84].

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)[86], and the Mobile Media API for J2ME (JSR-135) [87].

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.

Here, we have developed the Status Transmission Framework version 2.0 PAN API and the SENSACT API [13].

Figure 7 shows the minimum required architecture for a sensor, Figure 8 shows the minimum required architecture for an actuator, and Figure 9 shows the minimum required architecture for the central game playing device.

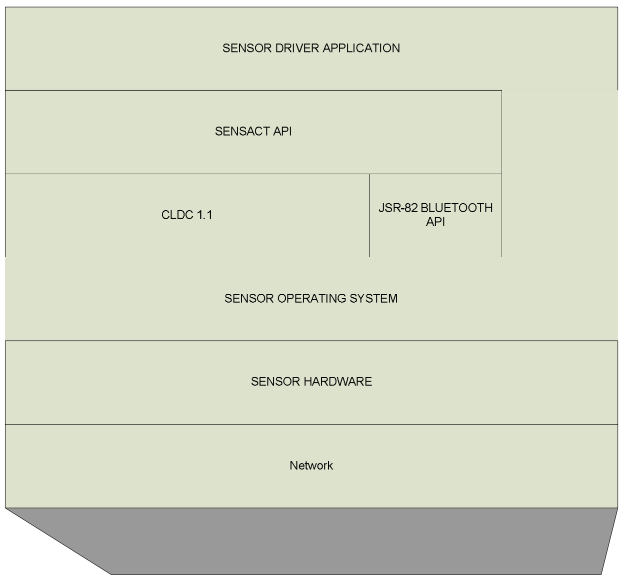


Figure 7 - Sensor architecture (Personal Area Network level)

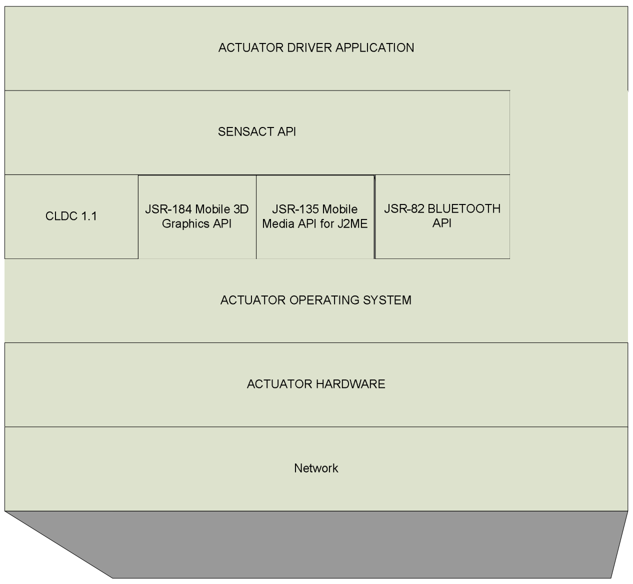

ACTUATOR DRIVER APPLICATION
SENSACT API
CLDC 1.1
JSR-184 Mobile 3D Graphics API
JSR-135 Mobile Media API for J2ME
JSR-82 BLUETOOTH API
ACTUATOR OPERATING SYSTEM
ACTUATOR HARDWARE
Network

Figure 8 - Actuator architecture (Personal Area Network level)

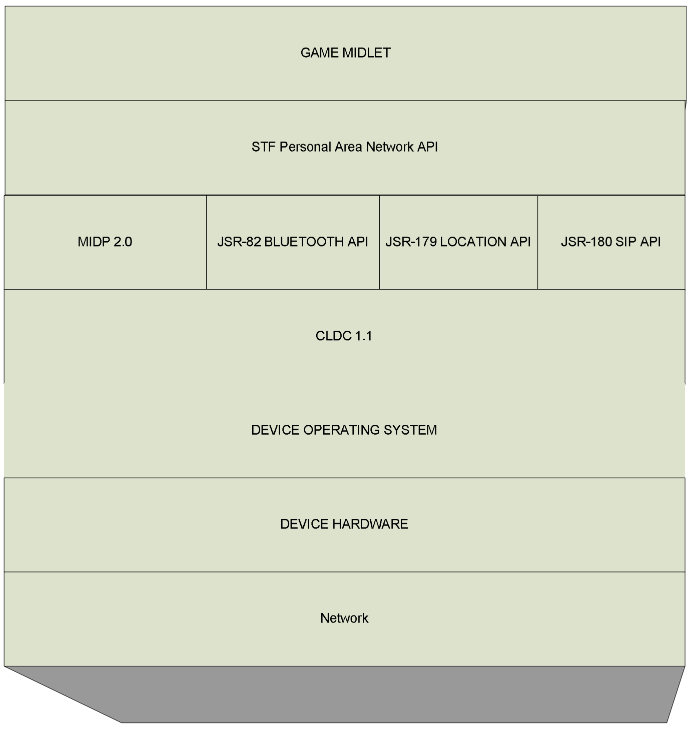


Figure 9 - Central Game Device Architecture (Personal Area Network level)

3.2. EXPERIMENTAL DEVELOPMENT

3.2.1.. THE SENSACT API

The SENSACT API, which was introduced in [13], is the part of the system which enables the sensors and actuators to be deployed with the help of java CLDC 1.1 and communicate by Bluetooth with the help of JSR-82. This API and correspondent protocols make part of the personal area network level of the system.

The SENSACT API is a small footprint set of classes occupying less than 60KBytes, constituted by 5 java packages, with the functions shown in Table 2.

In order to deploy one sensor/actuator, one has to have the respective hardware, a CLDC 1.1 implementation that runs on the hardware, a Bluetooth API (JSR-82) implementation for that hardware and platform, and the SENSACT API.

pt.uc.dei.lcst.stf.sensact.main
Main package of the SENSACT API. Contains classes that can and should be used to create new sensors or new actuators.
pt.uc.dei.lcst.stf.sensact.pack
Contains a pack of discovery messages, sensor and actuator classes already implemented that greatly simplify building the most common types of sensors and actuators.
pt.uc.dei.lcst.stf.sensact.net
Creates a general channel architecture that can be used to abstract from the communication system, be it Bluetooth or other (in the future, for example, ZigBee could be supported by simply extending classes).
pt.uc.dei.lcst.stf.sensact.link
Implements communication channels and helper classes (Bluetooth)
pt.uc.dei.lcst.stf.sensact.persist
Extends persistence to the world of J2ME CLDC with a low bandwidth alternative to the standard Java 2 SE persistence classes.

Table 2- SENSACT API Packages

The SENSACT API on the sensors and actuators and the STF PAN API (see below) on the main game device work as a whole to organize an ad-hoc network of sensors and actuators around a main game device in a way that the main game device works as a coordinator for the sensors and actuators.

The SENSACT API contains classes that allow us to build sensors and actuators, to control a network of sensors and actuators and interact with these sensors and actuators, all through a Bluetooth network.

To create a sensor, we may create a class that implements the Sensor interface, from the package pt.uc.dei.lcst.stf.sensact.main. This interface has three methods, with the functions shown in show in Table 3.

Method	Function
activate(GenericChannel c)	Activates the sensor
discover()	Discovers all sensors in this device
stopSensor()	Stops the sensor

Table 3 - Methods of the SENSACT Sensor interface

We must also implement for the same sensor, in order to implement the discover method of the Sensor interface correctly, a class that extends the SensorDiscoveryMessage class, from the same package. To implement the value read from the sensor, we must create a class that implements the SensorValue interface, again from the same package.

To create an actuator, we may create a class that implements the Actuator interface, from the package pt.uc.dei.lcst.stf.sensact.main. This interface has three methods, with the functions shown in Table 4.

Method	Function
actuate(ActuatorValue s)	Actuate based on the value given
discover()	Discovers all actuators on this device
startActuator()	Starts the actuator
stopActuator()	Stops the actuator

Table 4 – Methods of the SENSACT Actuator interface

We must also implement for the same actuator, in order to implement the discover method of the Actuator interface correctly, a class that extends the ActuatorDiscoveryMessage class, from the same package To implement the value written to the actuator, we must create a class that implements the ActuatorValue interface, again from the same package.

If our sensor or actuator is one of the predefined types of sensors and actuators supported by the SENSACT API, then we can use the sensor and actuator and correspondent SensorValue, ActuatorValue implementing classes and SensorDiscoveryMessage and ActuatorDiscoveryMessage derived classes from the package pt.uc.dei.lcst.stf.sensact.pack.

To implement a sensor, we must manage it. For this, we use a class that implements the interface SensactManager, from the package pt.uc.dei.lcst.stf.sensact.main. Typically, there will be one such class for each supported network technology. For Bluetooth, this class is the class BTSensactManager, from the same package. In the future, if other technologies are to be supported, different such classes need to be created and used. Methods of SensactManager are shown in Table 5.

Method	Function
Send(SensorValue s,GenericChannel c)	Sends a sensor value on a generic channel
Send(SensorValue s,GenericChannel c,String receiver)	Sends a sensor value on a generic channel for a specific receiver

Table 5 - Methods of SENSACT interface SensactManager

It is important to note that for the sensor or actuator to work on the STF PAN API, the sensor and actuator also need to be implemented on this API, this for new sensors or actuators. For supported actuators and sensors, it is enough to use the supplied classes. See the STF PAN API section for details.

3.2.2. THE STF PAN API

The STF – Status Transmission Framework – version 2.0 PAN – Personal Area Network level API is the correspondent API on the central game device of the user that communicates with the SENSACT API on the sensors and actuators and that communicates with the distributed and central servers.

We describe here the packages responsible for the personal area network communications, which are responsible for communicating with the SENSACT API on the sensors and activators, and which functions are listed in Table 6.

Session Description Protocol [81] is used in SIP [47] messaging to the central server and to the IMS – IP Multimedia Subsystem of 3GPP to negotiate session parameters and QoS. For this, we make use of Java SIP-API (JSR-180) [85] and the help of our developed SDP helper classes in pt.uc.dei.lcst.stf.pan.sdp.

The QoS API classes are excluded from this table because we discussed them in detail in a specialized section more ahead. But they also pertain to STF PAN.

pt.uc.dei.lcst.stf.pan
Main package of the STF PAN API. Main services like session control, late join, state receiving and transmitting, state representation, time warp support, checkpoint and distributed checkpoint support. Location and orientation support.
pt.uc.dei.lcst.stf.pan.comms.area
Contains a pack of discovery messages, sensor and actuator classes already implemented that greatly simplify building the most common types of sensors and actuators, and classes to create more sensors and actuators, and managers for the Personal Area Network. Is

effectively, the Personal Area Network API for small devices.
pt.uc.dei.lcst.stf.pan.comms.link
Implemented channels of communication and helper classes (including Bluetooth, but also ways of communicating with the servers)
pt.uc.dei.lcst.stf.pan.comms.sdp
Contains an API for SDP (Session Description Protocol [24]) representation in Java.
pt.uc.dei.lcst.stf.pan.net
Creates a general channel architecture that can be used to abstract any communication mean, be it Bluetooth or other (in the future, for example, ZigBee could be supported by simply extending classes). It also contains implementations of communication means to reach the distributed servers (TCP).
pt.uc.dei.lcst.stf.pan.persist
Extends persistence to the world of J2ME CLDC with a low bandwidth alternative to the standard Java 2 SE serialization and externalization mechanisms. Package exactly equal to the SENSACT of persistence.

Table 6 - Packages of the STF PAN API

To develop an interface for a sensor, to communicate with a sensor not supported by default by the API and already developed in SENSACT API, it is necessary to develop a class that implements the Sensor interface, implementing the methods shown in Table 7.

Method	Function
SensorValue sense()	Sense a value
void setChannel(GenericChannel c)	Sets the channel for communication
void startSensing()	Starts sensing
void stopSensing()	Stops sensing

Table 7 - STF PAN Sensor interface of package pt.uc.dei.lcst.sft.pan.comms.area

For this, it is also necessary to develop a class that extends the SensorDiscoveryMessage class for the sensor to be discoverable by the STF PAN API. It is also necessary to develop a class that implements the interface SensorValue to describe the value transmitted by the sensor.

To develop an interface for an actuator, to communicate with an actuator not supported by default by the API and already developed in SENSACT API, it is necessary to develop a class that implements the Actuator interface, implementing the methods shown in Table 8.

Method	Function
void actuate(ActuatorValue value)	Actuate on a value
void setChannel(GenericChannel s)	Sets the channel for communication
void startActuator()	Starts the actuator
void stopActuator()	Stops the actuator

Table 8 - STF PAN Actuator interface from package pt.uc.dei.lcst.sft.pan.comms.area

For this, it is also necessary to develop a class that extends the ActuatorDiscoveryMessage class for the actuator to be discoverable by the STF PAN API. It is also necessary to develop a class that implements the interface ActuatorValue to describe the value transmitted to the actuator.

The package pt.uc.dei.lcst.stf.pan contains the main STF PAN API. This includes main services like session control, late join, state receiving and transmitting, state representation, time warp support, checkpoint and distributed checkpoint support, and also location and orientation support.

To manage the session on the side of the main game device, the class to use is STFSessionManager, from which we can use the methods which are shown in Table 9, with corresponding functions.

Method	Function
void addSessionListener(STFSessionListener s)	Adds a new STFSessionListener to the session
void endSession()	Ends the session
float getLastKnownAzimuth()	Get last known value of orientation’s Azimuth
boolean getLastKnownIsMagnetic()	Get last know value of flag of magnetic Azimuth
float getLastKnownPitch()	Get last known value of orientation’s pitch
float getLastKnownRoll()	Get last known value of orientation’s roll
float getLastKnownX()	Get last known value of position’s X coordinate
float getLastKnownY()	Get last known value of position’s Y coordinate
float getLastKnownZ()	Get last known value of position’s Z coordinate
void startSession()	Starts the session

Table 9 - STF PAN STFSessionManager class of package pt.uc.dei.lcst.stf.pan

To manage the communication channel open after the session is initialized, the class to use is STFChannelManager, which implements the interfaces STFMessager, STFPartitioner, STFLatejoin and STFStatistics. These interface’s methods are shown in Table 10, Table 11, Table 12 and Table 13.

Method	Function
boolean removeAllFromKeySinceMessage(STFStateMessage m)	Removes all messages since this one from the transmitting queue of the entity, if possible
boolean removeAllSinceMessage(STFStateMessage m)	Removes all messages sice this one from the transmitting queues, if possible
boolean removeMessage(STFStateMessage m)	Removes a message from the transmitting queues, if possible
void sendMessage(STFStateMessage m)	Sends a new state transmission message through the framework

Table 10 - STF PAN STFMessager interface of package pt.uc.dei.lcst.stf.pan

Method	Function
void disablePartitioning()	Disables partitioning
void enablePartitioning(double radious, STFSessionManager session, int updateInterval)	Turns on partitioning for a spherical space around the UE.
boolean isPartitioningActive()	Checks to see if partitioning is active

Table 11 - STF PAN STFPartitioner interface of package pt.uc.dei.lcst.stf.pan

Method	Function
void disableStats()	Disables statistics collecting
void enableStats()	Enables statistics collecting
Enumeration getAllStreamKeys()	Gets a enumeration of all stream entity keys with statistics collected
STFStatisticsData getGeneralStats()	Get the general statistics data object
STFStatisticsData getStats(STFKey key)	Gets the stream specific statistics data object
void resetStats()	Resets the statistics collected
void resetStats(STFKey key)	Resets the statistics collected for the specified entity key
void setStats(boolean enabled)	Enables or disables statistics collecting

Table 12 - STF PAN STFStatistics interface of package pt.uc.dei.lcst.stf.pan

The main game device application class has to implement the STFApplication interface, which has the methods shown in Table 13.

Method	Function
void endLateReceiving()	Signals the end of late join message receiving
void receive(STFStateMessage)	Receives a state transmission framework message

void receiveLate(STFStateMessage)	Receives a late join protocol state transmission framework message
void startAllGameActivity()	Starts all game activity
void startLateReceiving()	Signals the start of late join message receiving
void stopAllGameGameActivity()	Stops all game activity
void timeWarpReceive(STFStateMessage)	Receives a time warped state transmission framework message

Table 13 - STF PAN STFApplication interface of package pt.uc.dei.lcst.stf.pan

3.2.3. THE STFSERVER API

The middleware that runs on the central level and on the large scale distributed server level is the STFServer API. This API makes use of ARMSV6 (and so, Sixrm).

It is an API that runs on top of J2SE 1.5.0 and its objective is to run on the IMS – IP Multimedia Subsystem core – as an Application Server.

The central server is an application server, the distributed game servers are also application servers.

Table 14 shows the STFServer API packages and corresponding functions.

pt.uc.dei.lcst.stf
Main package of the Status Transmission Framework version 2.0 Server Side. It contains the logic that complements the functions that the Status Transmission Framework version 2.0 provides to the mobile part of applications and also the logic that STF v2.0 provides to the server side of the application. That includes state representing, late joining, state receiving and transmitting, time warp support, checkpoint and distributed checkpoint support. Location and orientation support.
Pt.uc.dei.lcst.stf.applications
Package that implements some applications. Namely, the central game server, the distributed game server, and some partially simulated components of 3GPP IMS for testing.
pt.uc.dei.lcst.stf.applications.manager
Package that implements the manager application, which manages and presents execution statistics for all levels of the system.
pt.uc.dei.lcst.stf.applications.messages
Messages specifically used by the applications of

pt.uc.dei.lcst.stf.applications.
pt.uc.dei.lcst.stf.arms
Interface package for working with ARMSV6.
pt.uc.dei.lcst.stf.diameter
Package that implements the base diameter protocol [15] in java.
pt.uc.dei.lcst.stf.diameter.cxdx
Package that implements the 3GPP CXDX Diameter Application [16] protocol in java.
Pt.uc.dei.lcst.stf.diameter.sh
Package that implements the 3GPP SH Diameter Application [17] protocol in java.
pt.uc.dei.lcst.stf.net
Creates a general channel architecture that can be used to abstract any communication mean.
Pt.uc.dei.lcst.stf.persist
A low bandwidth alternative to the standard Java 2 SE serialization and externalization mechanisms.
pt.uc.dei.lcst.stf.sctp
Implements support for the SCTP [18][19] protocol in java in platforms that support it (java.net style). SCTP is the default protocol for communications in the DIAMETER[15] AAA protocol. Currently, Linux is supported.
Pt.uc.dei.lcst.stf.uelink
Implements specific channels of communication with the UE (mobile terminal – the personal area network central game machine).

Table 14 - STFServer API Packages

3.2.4. CHANGES IN STF FROM VERSION 1 TO VERSION 2

Status Transmission Framework version 2 (STF2), our middleware for large scale mobile and pervasive augmented reality games, is an evolution of a previous middleware for distributed collaborative virtual worlds (DCVE), the Status Transmission Framework Version 1 (STF), that was the subject of a Master Thesis publication by the same author [59].

Besides the division into three parts (SENSACT, STFPAN and STFSERVER), and the other developments we describe in this phd thesis (Sixrm, security, management, QoS, ArmsV6 and the SENSACT and STFPAN sensor and actuator network architecture), there are some other changes we need to describe. We do this on this section.

First, there are changes in the representation of the very basic states that are transmitted by the framework. This is done to enable location and orientation support by the middleware. Now, the interface STFState (that represents a State) includes methods to return the location and orientation of an object. There are no changes to the STFKey interface, which represents a key for identifying an object.

The Table 15 represents the methods of the STFState interface, part of packages pt.uc.dei.lcst.stf.pan (on STFPAN) and pt.uc.dei.lcst.stf (on STFSERVER).

Methods	Functions
Int getRedundancy();	Redundancy information. A state may be REDUNDANT or ESSENTIAL.
Int getVolatility();	Volatility information. A state may be VOLATILE or NON_VOLATILE.
Int getIndependency();	Independency information. A state may be INDEPENDENT or CUMULATIVE with respect to latejoining.
double getPositionX();	Get X coordinate of object's position.
double getPositionY();	Get Y coordinate of object's position.
Double getPositionZ();	Get Z coordinate of object's position.
Float getOrientationPitch();	Get the pitch orientation value of the object.
Float getOrientationRoll();	Get the roll orientation value of the object.
Float getOrientationAzimuth();	Get the azimuth orientation value of the object.

Table 15 - STFState methods and correspondent functions

Message transmission works in a similar way in STF2 as it worked in STF1. The State classification scheme is the same, as also bandwidth adaptation. Now we have two possible transports. The STF protocol can now be transported over TCP or over ARMSV6 (Sixrm multicast). If transmitting to or from a UE, then TCP is used, while if transmitting between distributed servers, ARMSV6 (Sixrm multicast) is used.

That means that while doing SDP negotiation, the media protocol negotiated is STF (our own media protocol), which must be appropriately authorized by the network. Of course any other media can be used additionally to this, such as media for audio and video, additionally to state and virtual 3D information.

The partitioning method also has changed. While in STF version 1 we had to specify the STFKey set we defined as the partition that was active for each node, now that is not necessary. This has various consequences. The first is that a particular partitioning method is always choosed for us, and the second is that this method is always available and optimized by the middleware.

The partitioning method is spatial partitioning. We choose only to enable partitioning and choose a particular partitioning radius, beyond which no state messages will be considered or transmitted. This works for STFPAN or STFSERVER, and the adequate changes are done on the methods of the STFPartitioner interface.

The latejoining method has changed a little to improve scalability. The change was to include the consideration of spatial partitioning. Now, if spatial partitioning is active, latejoining takes that into account and only includes states coming from users and servers in the spatial partitioning area.

The optional time synchronization method (which is active by default in STFPAN and STFSERVER), has some changes to accommodate the architectural differences between STF and STF2. When active on STFPAN, the method to vote for a new main time server is never activated. This is because in this case the main time server is always considered to be the current distributed server. So, a voting process is never required. All the other mechanisms function the same way as STF.

When working in STFSERVER, the time synchronization mechanism, if active, works the same way as STF, a main time server is automatically choosed. If the main time server fails, a new one is choosed by distributed voting. The system maintains a common virtual time synchronized by all servers. This virtual time is also sent to the UEs running STFPAN. The system takes into account network delay the same way as STF1 would.

3.2.5. THE SIXRM RELIABLE MULTICAST PROTOCOL

We felt it was necessary to create a protocol capable of working in the many-to-many scenario, without the nak implosion problem but nak based, source ordered and that avoided duplicates. We also needed a protocol that could work in ipv6, and that does not require the support of network elements. So, we created Sixrm.

The Sixrm protocol consists of six types of packets. These are: Data packets (TYPE_DATA), Open packets (TYPE_OPEN), Close packets (TYPE_CLOSE), nak packets (TYPE_NAK), information packets (TYPE_INFO), and error packets (TYPE_ERROR). It is a nak based protocol, in which the destinations, when they detect that they did not received a packet from certain source, transmit a nak for the group that only that source will listen to and repeat the transmission of the lost packet. If that is not possible, an error packet will be transmitted for the destination. A source first opens the channel for communication sending a packet for all destinations in the multicast group, an open packet (TYPE_OPEN), that all currently listening destinations will receive. This corresponds to the Open operation on the Sixrm node. The packet contains the source identity and source ip address. All currently open destinations (those that already have executed the Open operation), when receiving the open packet, perform initialization for that source, and respond to the open packet with an information packet containing the destination (and possible source) identity and ip address. The source (or destination) identities are Java long data types. When receiving the information packets, the source performs the same initialization that the destinations did when they received the open packet.

For the close operation, the source transmits the close packet and removes itself from the group. All destinations, when they receive the close packet, perform cleanup operations for that source.

As for data transmission, data is transmitted sequentially. Each new byte buffer that is handled to the Sixrm protocol for transmission is buffered for transmission and subsequently transmitted, in the next available time slot. When the data is transmitted, it is given a monotonically increasing sequence number, packed in a packet of type data (TYPE_DATA) and send to the group for all destinations to listen.

It must be mentioned by now that every packet transmitted by Sixrm contains the source transmit memory buffer state (send window), which includes all messages since the last one memorized until the newest one that was sent (the biggest sequence number). The newest and oldest sequence numbers are transmitted with each message.

Each destination maintains a receiving window, for each source, that is smaller than the source transmit window and which has is greatest sequence number always equal to the source transmit window newest sequence number and its oldest sequence number adjusted to be that number minus some minimum value.

Each time a source transmits a value, it puts that value in its sent buffer (transmit window), and if necessary, if the buffer cannot grow anymore, discards the oldest message that was in the buffer. The value is transmitted and stored in the buffer in a packet of type data (TYPE_DATA).

Each time a destination receives a packet, it verifies witch kind of packet it is. Depending on the kind of packet, the destination acts differently.

3.2.5.1. PACKETS OF TYPE NAK

When a destination (which is also a source) receives a Nak packet, it means someone did not receive a packet. The Nak packet contains the source address, the source identity, the sequence number, the target address, and the target identity.

We can tell if the nak packet is for us by comparing the target address and target identity with our ip address and identity. If equal, it was our packet that was not received, if not equal, it was someone else's packet that was not received, but we still need to memorize the nak. Our answer will depend on the fact that the nak was direct to us or not.

3.2.5.1.1. NAK DIRECTED TO US

If the Nak was directed to us, then we must try to resend the packet that was lost. Unfortunately, that may not always be possible. Given the sequence number of the nak, we access the sent memory (transmit window), and verify if we have the packet.

If we have the packet, then, we resend the packet to the group. We do not memorize the nak.

If we do not have the packet, then we cannot recover from the error and must send an error packet to the destination informing the destination that we do not have the packet with that sequence number. We also send an error packet to the application telling the application that a data packet with a determined sequence number did not reach all destinations.

3.2.5.1.2. NAK DIRECTED TO SOMEONE ELSE

If the nak was directed to some other destination (and source) then we must memorize the nak to prevent us from, in the near future, transmitting a nak for the same packet. For this effect, we memorize the nak and a time value that is equal to the current time plus a random back off time value during which we cannot transmit a nak for this packet. We do not prevent the transmission forever because even naks can be lost. In this way, probably we will receive the packet related to the nak if it is missing and clear the nak before the need to transmit it again arises.

3.2.5.2. PACKETS OF TYPE ERROR

When a destination (which is also a source) receives an error packet, it means a source could not retransmit a packet for a destination. The error packet contains the source address, the source identity, the sequence number of the packet that generated the error, and the target address and target identity.

We can tell if the error packet is for us by comparing the target address and target identity with our ip address and identity. If equal, it was our packet that wasn't available, if not equal, it was someone else's packet that wasn't available. Only if the error is for us do we deliver it to the application.

3.2.5.3. PACKETS OF TYPE DATA

When receiving a packet of type data, the destination first verifies if it did already initialize the memory structures for that source.

Then, if the packet sequence number is within the receiving window, it buffers the data packet. It also removes from memory any naks for that data packet that may exist.

A data packet contains the source address, the source identity, the sequence number, the data buffer, the transmit window newest sequence and the transmit window oldest sequence.

Taking for granted that the memory structures were already initialized, they are altered by the data packet, namely the receiving window. The newest element in the receiving window is always the transmit window newest sequence number.

After buffering the data packet, the destination verifies that the receiving window oldest sequence number is below the minimum difference with the newest and a packet with that sequence number exists. In that case, the packet is handed to the application. The process repeats while possible until the receiving window oldest sequence number aligns with the minimum configured difference from the newest sequence number.

After this, the receiving buffer is verified for missing packets in between the receiving window. Naks are transmitted for each missing packet if is not forbidden to do so. It is forbidden to do so if the nak is memorized and its associated time value is greater than the current system time.

3.2.5.4. PACKETS OF TYPE OPEN

Packets of type open means a source has executed the open operation and expects us to execute initialization of internal memory structures and respond with information packets with our information (source address and identity).

Open packets contain the source address and the source identity. When receiving an open packet, the destination (and source) will initialize its memory structures for that source, and send an information packet containing its information (that would be sent on an open packet) on an information packet.

3.2.5.5. PACKETS OF TYPE CLOSE

Packets of type close means a source has executed the close operation and will not be sending or receiving more packets with sequence numbers bigger than the latest.

3.2.5.6. PACKETS OF TYPE INFORMATION

Packets of type information means a destination (and source) is responding to a request from a source open packet. It expects us to execute initialization of internal memory structures, if we did not that already for it.

Information packets contain the source address and identity. When receiving an information packet, the destination (and source) will initialize its memory structures for that source, if it did not already did so.

3.2.6. THE SIXRM API

The Sixrm API is a simple API with only 4 classes, all on the package pt.uc.dei.lcst.stf.sixrm .These classes are the following: SixrmEntity, SixrmFileKey, SixrmListener and SixrmPacket. We now describe in more detail each of these classes.

3.2.6.1.. CLASS SIXRMENTITY

This class SixrmEntity takes care of communication for a Sixrm node (entity). A Sixrm node acts simultaneously as a source and destination of Sixrm packets.

A Sixrm entity is constructed from the identifier (long), the address (ip address – Java InetAddress), the group (ip address – Java InetAddress), the port (int), the ttl (time to live – int), the profile (Java Properties), and the listener (SixrmListener).

The profile contains configuration parameters summarized in Table 16.

Configuration	Meaning
maxOutBufferSize	Maximum size of an output packet
maxInBufferSize	Maximum size of an input packet
maxSentBufferSize	Maximum size of the sent buffer (transmit window)
maxQueuedBufferSize	Maximum size of the queue of messages to transmit
maxRecvBufferSize	Maximum size of the receive buffer
minRecvWindowSize	Minimum size of the receive window
maxRecvWindowSize	Maximum size of the receive window
minRandomBackoff	Minimum random back off time of nak packets
maxRandomBackoff	Maximum random back off time of nak packets
minIntervalTime	Minimum interval time between transmissions (adaptive throughput)
maxIntervalTime	Maximum interval time between transmissions (adaptive throughput)
intervalSteppingDown	Interval stepping down the interval time when there is no error or nak.
intervalSteppingUp	Interval stepping up the interval times when there is an error for us (2 xs) or a nak for us (1x).

Table 16 - Sixrm profile settings

When the SixrmEntity receives data packets or error packets, methods of SixrmListener are called.

Important methods of the SixrmEntity class are summarized on Table 17.

Method	Operation
void setListener(SixrmListener listen)	Sets the listener for received packets.
void sendMessage(byte[] b)	Sends a message (a data packet)
void Open()	Open operation
void Close()	Close operation

Table 17 - Methods of class Sixrm Entity

3.2.6.2. CLASS SIXRMPACKET

The class SixrmPacket represents a Sixrm packet. A Sixrm packet may contain a data buffer, a source ip address, a sequence number, a packet type, a source identity, a target identity, a target ip address, a source transmit window newest sequence number and a source transmit window oldest sequence number.

Basically, this class has constructors for the various types of packets that exist (TYPE_DATA, TYPE_OPEN, TYPE_INFO, TYPE_CLOSE, TYPE_NAK and TYPE_ERROR), and getter and setter methods for the data fields it contains.

3.2.6.3. CLASS SIXRMLISTENER

The class SixrmListener contains only one method, summarized in Table 18.

Method	Operation
Void receive(SixrmPacket pack)	Receives a packet (error or data).

Table 18 - Methods of class SixrmListener

This is the interface that applications must implement.

3.2.6.4. CLASS SIXRMFILEKEY

The Class SixrmFileKey is a helper class used to uniquely identify a source by its address and identity. A SixrmFileKey contains an ip address and an identity (long).

A SixrmFileKey instance identifies a source or destination of Sixrm packets (a Sixrm node).

3.2.7. THE ARMSV6 SYSTEM

ARMS – The Augmented Reliable corba Multicast System [8] [9] – was extended to work in IPV6 in large scale networks by substituting, in its new version ARMSV6, the reliable multicast protocol it used by the Sixrm protocol.

Now, the distributed servers communicate using ARMSV6, and so, the Sixrm reliable multicast protocol.

ARMS [8][9] is a improvement for the corba event service that, maintaining compatibility with the standard corba event service, adds reliable multicast communication to it. ARMSV6 does the same thing but now supporting ipv6 through Sixrm reliable multicast protocol.

3.2.8. THE QOS APIS

We now discuss the general architecture of the Status Transmission Framework version 2 QoS API on J2ME (that is, on the UE, the central coordinating device of the personal area network).

3.2.8.1. GENERAL ARCHITECTURE

The general architecture is based on two APIs for QOS: The PDP Context Handler API and the RSVP API. Both are used at the same time to guarantee quality of service on a 3GPP network with RSVP support (witch is optional) and that may or may not use uses Service Based Local Policy between the PDF (Policy Decision Function) and the GGSN as specified on [88].

3.2.8.2. THE PDP CONTEXT HANDLER API

The PDP Context handler API is an Application Programming Interface that allows us to activate and deactivate PDP contexts with all its characteristics including QoS characteristics.

Table 19 shows the classes of package pt.uc.dei.lcst.stf.pan.qos, the package of the PDP Context handling API and corresponding functions.

3.2.8.3. THE RSVP API

The RSVP API on the UE (J2ME) is based on the RSVP specifications [63][64][65][66][67] and is used to alter the way the GGSN in particular (if it supports RSVP) and other routers on the way to the distributed servers (which are located on the IP Multimedia Subsystem) allocate resources to the connection in question.

The GGSN affects, if supporting RSVP, the PDP context traffic handling.

Table 20 shows the classes of package pt.uc.dei.lcst.stf.pan.qos.rsvp, the package of the RSVP API and corresponding functions. This package belongs to the bigger STFPAN API.

Class	Function
DestinationPortRangeType	Defines a port range
FlowLabelType	Defines a Ipv6 flow label
IPV4SourceAddress	An Ipv4 Source Address
IPV6SourceAddress	An Ipv6 Source Address
PDPContext	A PDP Context per se
PDPContextListener	Listener for event related to PDP Contexts
PDPContextManager	The manager of PDPContexts
PDPPacketFilter	Base class of all Packet filters
PDPTrafficFlowTemplate	A PDPContext traffic flow template
QoSException	An exception occurred
QoSProfileIE	The QoS Specification
ProtocolIDNextHeaderType	A Protocol ID NextHeader
SecurityParameterIndexType	A Security Parameter Index

SingleDestinationPortType	A single destination port.
SingleSourcePortType	A single source port.
SourcePortRangeType	A source port range.
TypeOfServiceTrafficClassType	A type of service traffic class.

Table 19 - Classes of pt.uc.dei.lcst.stf.pan.qos

3.2.8.4. GENERAL ARCHITECTURE OF THE STF QOS API ON THE DISTRIBUTED SERVERS

On the distributed servers, we use RSVP (Resource Reservation Protocol) and related standards [63][64][65][66][67].

3.2.8.4.1. THE RSVP ARCHITECTURE

The RSVP architecture on the distributed servers is very similar to the architecture on the UE clients, with differences in implementation and in configuration of course. But the list of classes is the same as in Table 20, except that the package is now pt.uc.dei.lcst.stf.qos.rsvp that is part of the bigger STFServer API.

Class	Function
RSVPMessage	Base message
RsvpException	A RsvpException
RsvpExceptionNoService	RsvpException: No service available
RsvpListener	Listener of messages
RsvpManager	Rsvp manager
RsvpObject	Base object
RsvpObjectForwarded	Forwarded object
RsvpObjectIgnored	Ignored object

RsvpObjectIntServAdSpec	AdSpec object
RsvpObjectIntServFlowSpecObject	FlowSpec object
RsvpObjectIntServerSenderTSpec	TSpec object
RsvpObjectIntegrity	Integrity object
RsvpObjectIpv4ErrorSpec	Ipv4 Error Spec object
RsvpObjectIpv4FilterSpec	Ipv4 Filter Spec object
RsvpObjectIpv4ResvConfirm	Ipv4 ResvConfirm object
RsvpObjectIpv4RsvpHop	Ipv4 RsvpHop object
RsvpObjectIpv4ScopeList	Ipv4 ScopeList object
RsvpObjectIpv4SenderTemplate	Ipv4 SenderTemplate object
RsvpObjectIpv4UDP	Ipv4 session object
RsvpObjectIpv6ErrorSpec	Ipv6 ErrorSpec object
RsvpObjectIpv6FilterSpec	Ipv6 FilterSpec object
RsvpObjectIpv6ResvConfirm	Ipv6 ResvConfirm object
RsvpObjectIpv6RsvpHop	Ipv6 RsvpHop object
RsvpObjectIpv6ScopeList	Ipv6 ScopeList object
RsvpObjectIpv6SenderTemplate	Ipv6 SenderTemplate object
RsvpObjectIpv6UDP	Ipv6 Session object
RsvpObjectList	List of objects
RsvpObjectNull	Null object
RsvpObjectPolicyData	PolicyData object

RsvpObjectStyle	Style object
RsvpObjectTimeValues	TimeValues object
RsvpPATH	PATH message
RsvpPathErr	PATHERR message
RsvpPathTear	PATHTEAR message
RsvpRESV	RESV message
RsvpResvErr	RESVERR message
RsvpResvTear	RESVTEAR message
RsvpResvConf	RESVCONF message

Table 20 - classes of package pt.uc.dei.lcst.stf.pan.qos.rsvp

3.2.9. THE SECURITY ARCHITECTURE

Security and privacy are issues of significant importance on our middleware architecture for mobile and pervasive large scale augmented reality games.

Many times the information that is passed on the system should be kept secret for any other purpose other than gaming, and for anyone else than the appropriate other players.

A striking example of this information is the location and orientation context information of the central device on the personal area network.

Of course this information needs to circulate on the system, but needs to be kept secure from outside attack.

So, we need encryption on the system. We also need authorization and authentication, to know which players and which devices can be connected.

It's all that architecture we be discussing in this section, for all the levels of the system.

We have built architecture of security on our system that goes a step beyond and effectively extends the 3GPP security architecture (it is meant to work alongside with it).

3.2.9.1. PERSONAL AREA NETWORK LEVEL SECURITY

In the personal area network level of the system, we have a network of sensors and actuators that is connected to the central device through Bluetooth and a central device that is connected to a large scale distributed level server through the use of TLS/SSL over TCP.

So, the security we apply here is the following, we demand that all the Bluetooth connections be authenticated and encrypted. We apply security certificates do ensure authentication and authorization in TLS/SSL over TCP and the encryption itself (witch is RSA).

This is all possible without using nothing more than Java capacities in J2ME and J2SE, including the capacity to install security certificates.

In doing so, we secure communications in the personal area network and in the communications between the personal area network and the large scale distributed server level.

3.2.9.2. LARGE SCALE DISTRIBUTED LEVEL SECURITY

On the large scale distributed server level, we communicate between servers using Sixrm reliable multicast using ipv6, trough the use of our updated ARMSV6 corba event system using multicast, that evolved from previous work in ARMS – Augmented reliable corba Multicast System [8][9].

To be secure, we now symmetrically encrypt all communications that go through ARMSV6 (and Sixrm), in RCA5.

Communications are symmetrically encrypted (and not asymmetrically) because Sixrm is an any-to-any multicast protocol and so there is no direct correspondence between the sender and the receiver.

The key of encryption is distributed by the central server to the distributed servers in the authentication process.

3.2.9.3. CENTRAL LEVEL SECURITY

The central level is responsible for distributed server authentication and authorization.

Every distributed server must know a login and password to the distributed server so that it can access this server trough TLS/SLL over TCP so it can receive the encryption key to use on symmetric encryption.

The key is passed encrypted over the secure channel. Certificates for this are pre-installed on the Java Virtual Machines the servers run on (J2SE).

3.2.10. MANAGING THE SYSTEM

3.2.10.1. OBJECTIVES OF MANAGING THE SYSTEM

The main objective of building a managing subsystem and management application for our middleware for mobile and pervasive large scale augmented reality gaming is, primarily, a system that is able to work in the most autonomous way possible and at the same time able to get its management orders from outside the system.

In this way, in our system, we want the system to generate and report statistics automatically, but only report the statistics to the outside application when it requests them.

In the same way, the system must be able to do all that is necessary to authorize users and revoke user's authorization, including stopping users running in a clean way, automatically, but only when requested by the management application.

The management application must be able to enable, disable statistics, and to reset statistics, and have the middleware do that on the complete system automatically.

In this way, we have the maximum of self-management combined with the most controllability.

3.2.10.2. THE MANAGEMENT ARCHITECTURE

The management architecture is based on a distributed way of passing messages and on a direct way of passing messages. The distributed way is used for communication between the central server and the distributed servers and between these and the UE, and the direct way is used for communication between the central or distributed server and the management application.

All communication is message based and uses on the direct way, SSL over TCP, and on the distributed way, ARMSV6 (which uses Sixrm reliable multicast) between central server and distributed servers, and SSL over TCP between these and UE.

The management architecture supports the collecting of statistics of the UE, the distributed server, and the central server, and also supports aggregating the statistics of all the UEs of one distributed server, aggregating the statistics of all distributed servers, aggregating the statistics of all UEs of all distributed servers, and aggregating the statistics of all UEs and all distributed servers (global aggregation). Aggregation of central server statistics with those is not supported because they are fundamentally different.

Because of maintaining the scalability, it is not possible to get a specific UE statistics. Only aggregated statistics of all UE of a distributed server are possible to get.

This kind of statistics generation is done in the UEs by the middleware in its various tasks, and communicated periodically in a message (STFStatisticsReportedMessage) to the currently connected to distributed server.

The distributed server middleware generates its own statistics as it operates in its various tasks, and communicates it periodically to the central server in a message (STFStatisticsReportedMessage). The distributed server middleware also aggregates the received UE statistics into one report, and also periodically sends this report in a message (STFStatisticsReportUEMessage) to the central server.

The central server middleware generates its own statistics and stores it, and aggregates the received distributed server statistics into one report. It also stores the received aggregated UE statistics per distributed server.

The central server, the distributed server and the UE middleware has ways to enable and disable statistics generating, and to reset the values of statistics that are being generated.

When the central server receives a message DisableStatistics on the direct way channel (from the management application) it disables statistics generating itself and sends a DisableStatistics message to all distributed servers.

The distributed servers, when they receive the DisableStatistics message from the central server, disable statistics generation and reporting, and send a DisableStatsUE message to all UEs connected to the distributed server. When the UEs receive this message they stop generating statistics and reporting them.

When the central server receives a message EnableStatistics on the direct way channel (from the management application) it enables statistics generating itself and sends a EnableStatistics message to all distributed servers.

The distributed servers, when they receive the EnableStatistics message from the central server, enable statistics generation and reporting, and send a EnableStatsUE message to all UEs connected to the distributed server. When the UEs receive this message they start generating statistics and reporting them.

When the central server receives a message ResetStatistics on the direct way channel (from the management application) it resets the statistics generated itself and sends a ResetStatistics message to all distributed servers.

The distributed servers, when they receive the ResetStatistics message from the central server, reset statistics generated, and send a ResetStatsUE message to all UEs connected to the distributed server. When the UEs receive this message they reset generated statistics.

The central server also has the capability to stop and start the game. Every UE and distributed server implements the STFApplication interface, witch, between other methods, include stopGame() and startGame() methods, that indicate when to start and when to stop interacting to the application.

When the central server receives a StartGameActivity message in the direct way channel (emitted by the management application), the central server emits a StartGameActivity message to all connected distributed servers that in turn send a StartUEGameActivity message to all connected UEs. This will cause the method startGame() to be called in all UEs and distributed servers.

When the central server receives a StopGameActivity message in the direct way channel (emitted by the management application), the central server emits a StopGameActivity message to all connected distributed servers that in turn send a StopUEGameActivity message to all connected UEs. This will cause the method stopGame() to be called in all UEs and distributed servers.

When a new UE connects itself to the system, or a new distributed server connects itself to the system, startGame() or stopGame() will be called, depending on the last known status of the game, started or stopped. On initial central server starting condition, the game is started. As part of initial communications of the distributed server a new step is added to ask the central server in witch status the game is on. This is done after the initial setup of the communication channels and before any control is handled to the application. Basically the distributed servers just send to the central server on the ARMSV6(Sixrm) multicast channel a AskForGameStatusMessage and the server respond with GameStatusStartedMessage or GameStatusStoppedMessage if the game is started or stopped, respectively. Based on the response, the distributed server knows the function to call, startGame() or stopGame().

As for authorization of user and revoking the authorization of users, this can also be handled by the central servers and distributed servers in a coordinated fashion.

This requires not only the central server and distributed server coordination, but communication with the UEs, and that the central application server communicates with the HSS (Home Subscriber Server) of the 3GPP network to see if it can authorize the user.

So, to authorize a user to use the system, the application tries to do so by sending a message UserAuth to the central server containing the SIP identity to authorize. If the server, for any reason, cannot authorize this user, it returns an error code in the resulting response message UserAuthError, else it returns a code in that message indicating that all is right.

To revoke a user authorization it is not needed for the central server to consult the HSS, but may be needed to the system to disconnect a running user of the system (emitting a SIP BYE message and terminating the session).

To revoke a user authorization, the manager application tries to do so by sending a message UserRevokeAuth to the central server, containing the SIP identity to revoke authorization to, to which the server verifies if it exists and if it is running. If it does not exist, it returns an error in the message of response, UserRevokeAuthError, else it returns a code in that message that says all is ok. If the user was running a BYE SIP message is emitted to the UE, and the session is terminated, along with all open communication channels and QoS reservations.

Table 21 shows the messages and corresponding functions between the manager application and the central server, Table 22 shows the messages and corresponding functions between the manager application and the distributed server, Table 23 shows the messages and corresponding functions between central and distributed servers and Table 24 shows the messages and corresponding functions between distributed servers and UEs. The messages belong to the packages pt.uc.dei.lcst.stf.pan on the STFPAN UE and pt.uc.dei.lcst.stf and pt.uc.dei.lcst.stf.applications.messages on the STFSERVER central and distributed servers.

For the communication between central server and HSS, the 3GPP SH DIAMETER[19] application protocol is used.

Message class	Function
LocateServer	Locates a server given managed coordinates
LocatedServer	Returns the located server
StartGameActivity	Starts all game activity
StopGameActivity	Stops all game activity
AskForAggregateGlobalStatsMessage	Asks for aggregate global statistics excluding central server
AskForAggregateDistServerUEStatsMessage	Asks for UE aggregate statistics from

	all distributed servers
AskForAggregateDistServerStatsMessage	Asks for aggregate distributed servers statistics
AskForCentralServerStats	Asks for central server statistics
DisableStatistics	Disables statistics
EnableStatistics	Enables statistics
ResetStatistics	Resets statistics
CheckRunning	Verifies if user is running
CheckAuthorized	Verifies if user is authorized
UserRevokeAuth	Revokes user authorization
TerminateManagerCommunications	Terminates the manager communications
AllServerNames	Contains all distributed server names
AggregateGlobalStatsMessage	Aggregate global statistics excluding central server
AggregateDSTUEStatsMessage	Aggregated UE stats from all distributed servers
AggregateDSTStatsMessage	Aggregate distributed server statistics
CentralServerStats	Central server statistics
CheckAuthorizedResult	Result from check authorized
CheckRunningResult	Result from check running
AskForAllServerNames	Asks for all distributed server names
UserAuthError	Result from UserAuth
UserRevokeAuthError	Result from UserRevokeAuth

Table 21 - Messages between central server and management application

Message class	Function
STFStatisticsReportMessage	Asks for statistics for the server
STFStatisticsReportedMessage	Returns statistics (UE aggregate or server)
STFUEStatisticsReportMessage	Asks for statistics for UE (aggregate for the server)

CheckRunning	Check if user is running using server
CheckRunningResult	Result of checking if user is running
TerminateManagerCommunications	Terminates the manager communications

Table 22 - Messages between distributed server and management application

Message class	Function
EnableStatistics	Enables statistics
ResetStatistics	Resets statistics
DisableStatistics	Disables statistics
STFStatisticsReported	Reports statistics for a distributed server
RemoveUserRunning	Removes a user running. Stops it running.
StopGameActivity	Stops game activity.
StartGameActivity	Starts game activity.
STFStatisticsReportedUEMessage	Reports aggregate UE statistics for a distributed server

Table 23 - Messages between central and distributed servers

Message class	Function
DisableStatsUE	Disable statistics on a UE
EnableStatsUE	Enable statistics on a UE
ResetStatsUE	Reset statistics on a UE
StartUEGameActivity	Start a UE game activity
StopUEGameActivity	Stop a UE game activity
STFStatisticsReportedMessage	Reports UE statistics

Table 24 - Messages between distributed servers and UEs

3.2.10.3. THE MANAGEMENT APPLICATION

The management application is a Java 2 Standard Edition Application that runs with the help of the STFSERVER middleware and that connects by SSL over TCP to a known port of the central server, and may also connect to a known port again by SSL over TCP, to any of the distributed servers, the addresses and ports of witch it gets from the central server.

The interface of the application is based on the Swing framework of J2SE and has a system of menus that we can choose to do the manager work.

First, we must connect to the central server, for witch there is a menu option. When we do this, we can then obtain statistics for the central server, statistics for all distributed servers, statistics for the UEs of all distributed servers, statistics for all distributed servers and UEs (Global Statistics), and statistics for all distributed servers. We can only do that if statistics generating and reporting is enabled of course. For that, there are menu options to enable statistics, to disable statistics and to reset statistics.

When connected to the central server there are also menu options enabled that allow us to authorize users, revoke user's authorization, check if a user is authorized and check if a user is running.

There is another menu option to connect to a distributed server, which gets a list of distributed servers which can be connected to from the central server and allow us to choose one. When we choose one and connect to it, by SSL over TCP, other menu options are available, the distributed server statistics and the distributed server UE statistics. Menu options are described schematically in Figure 10.

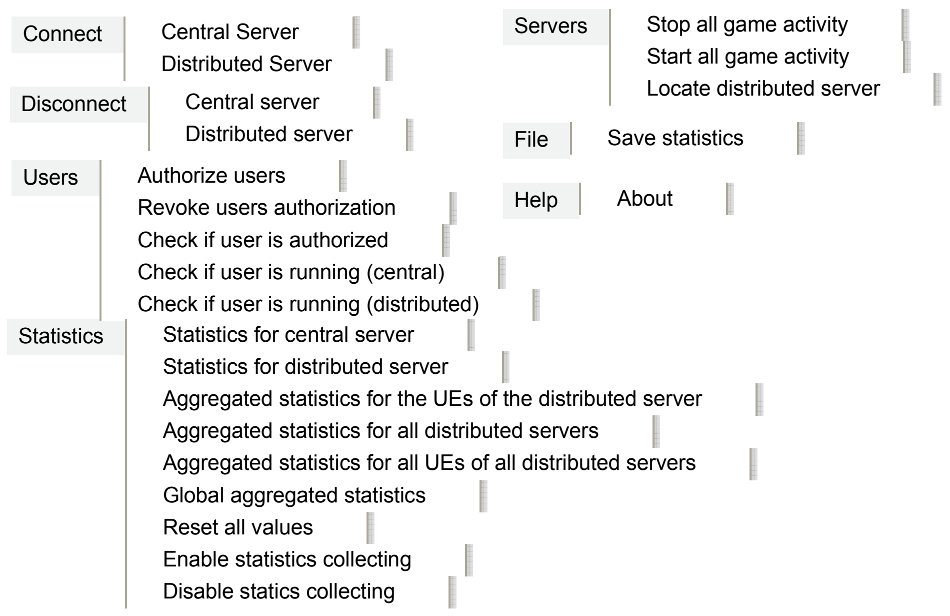

Connect
Central Server
Distributed Server
Disconnect
Central server
Distributed server
Users
Authorize users
Revoke users authorization
Check if user is authorized
Check if user is running (central)
Check if user is running (distributed)
Statistics
Statistics for central server
Statistics for distributed server
Aggregated statistics for the UEs of the distributed server
Aggregated statistics for all distributed servers
Aggregated statistics for all UEs of all distributed servers
Global aggregated statistics
Reset all values
Enable statistics collecting
Disable statics collecting
Servers
Stop all game activity
Start all game activity
Locate distributed server
File
Save statistics
Help
About

Figure 10 - Management application menu structure and available menu options

4. EVALUATION

To evaluate our network middleware for mobile and pervasive large scale augmented reality games, we first evaluate its scalability analytically considering all levels of its architecture, and then evaluate first the SENSACT and STFPAN technologies (personal area network), the SIXRM protocol (on the distributed and central server level), the QoS APIs (both on the personal area and on the distributed server level), the management architecture (present on all levels of the system) and the security architecture (present on all levels of the system) and also evaluate a common API (persistence). Finally to evaluate the middleware in general we demonstrate building and executing a concept application.

4.1. EVALUATING 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.

4.1.1. 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 (its 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 (its size in bytes) handled by the central game device on a period of time between instants t_0 and t_1 is:

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

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.

So we have that:

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

4.1.2. 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 (its size in bytes), which each distributed server will handle, will be approximated by:

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

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).

4.1.3. THE BACK OFFICE 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.

4.1.4. ANALYSING THE SCALABILITY OF POSSIBLE ALTERNATE ARCHITECTURES

4.1.4.1. 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:

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

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.

4.1.4.2. 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, which 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 of messages (its size in bytes) processed in one distributed server is:

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

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.

4.1.4.3. 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:

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

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.

4.1.5. GRAPHICAL CASE STUDY OF THE ALTERNATIVE 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.

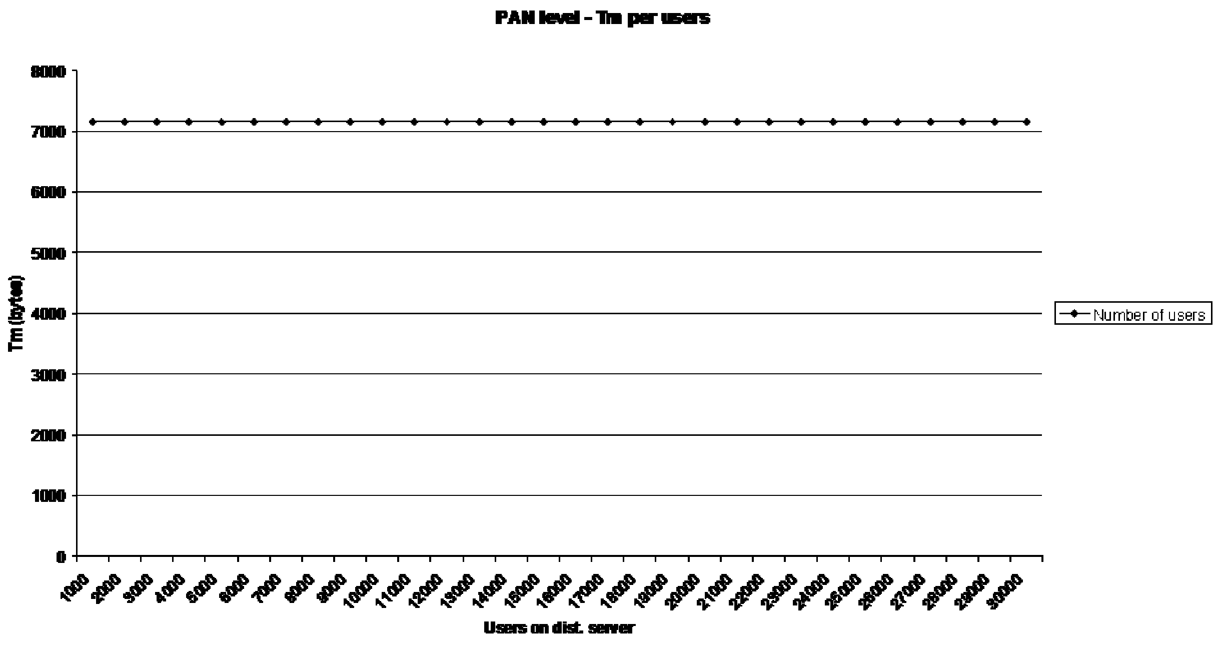


Figure 11 - Personal Area Network level

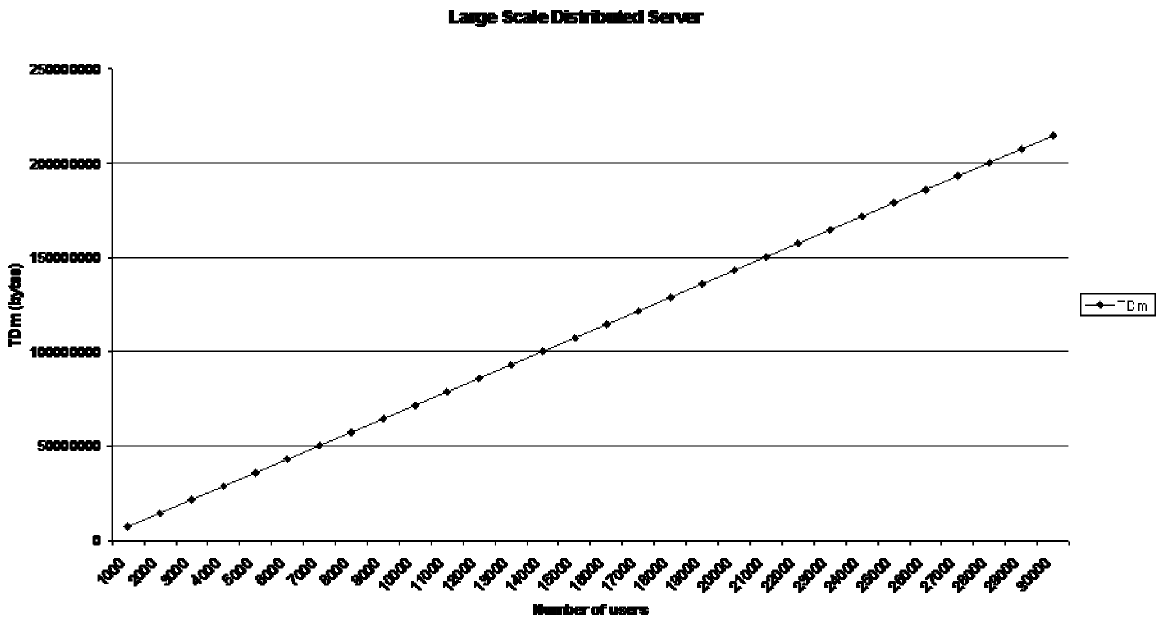


Figure 12 - Large Scale Distributed Server Level

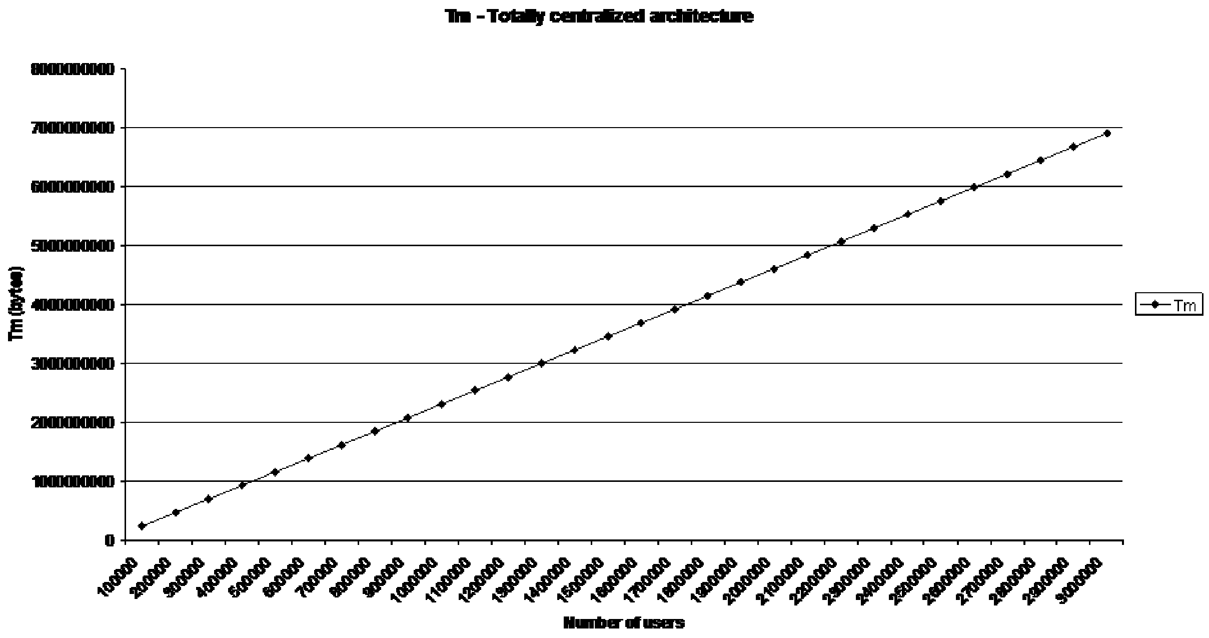


Figure 13 - Totally centralized architecture

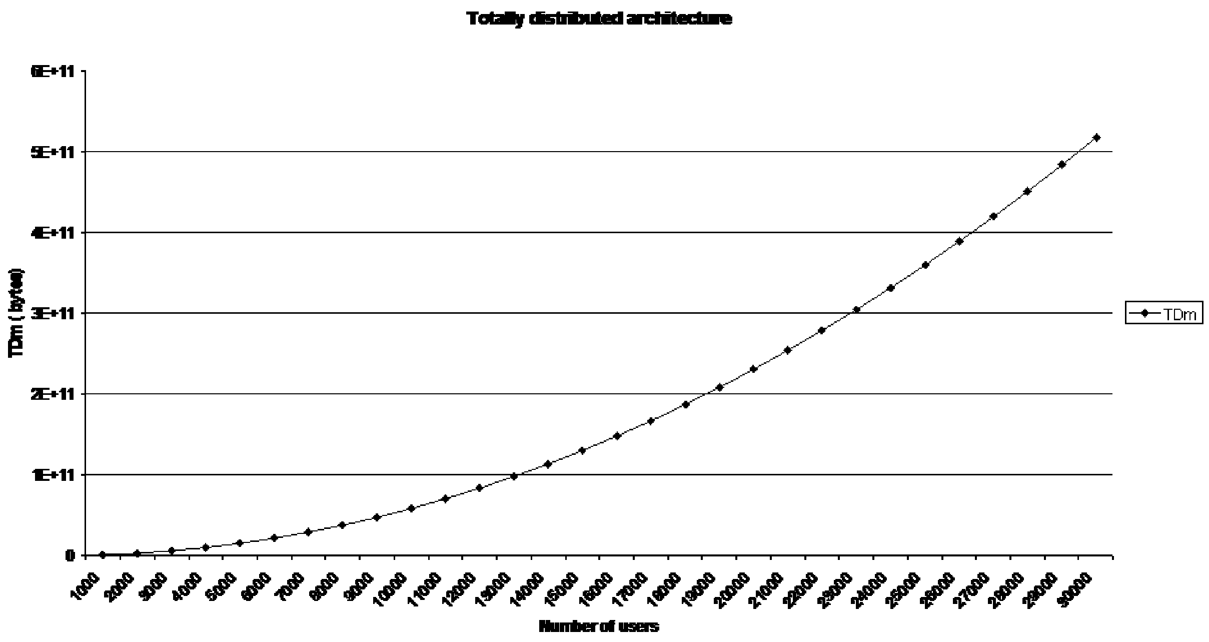


Figure 14 - Totally distributed architecture

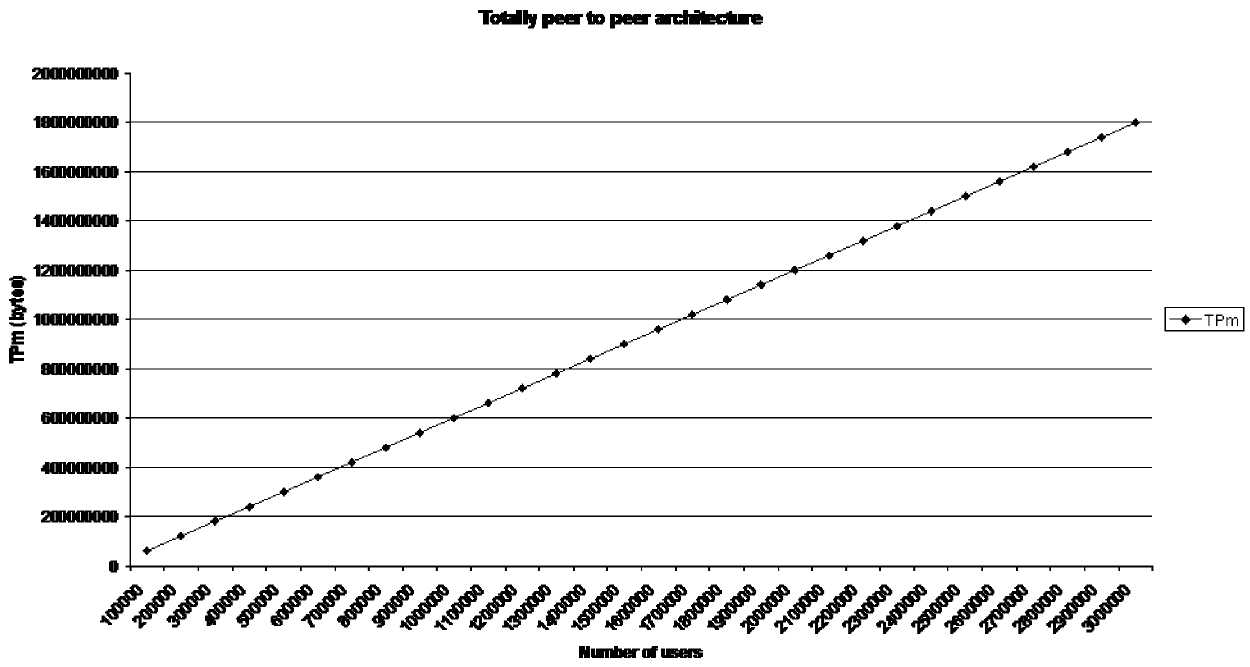


Figure 15 - Totally peer-to-peer architecture

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.

4.2. EVALUATING SENSACT AND STF PAN

The SENSACT API combined with the STF API were subject to extensive functional and performance tests, with various kinds of simulated sensors and actuators and a simulated reading and actuating application using Java Wireless Toolkit from Sun Microsystems running in a series of emulators in a Pentium 4 3.6 GHz System with 1 Gb Memory.

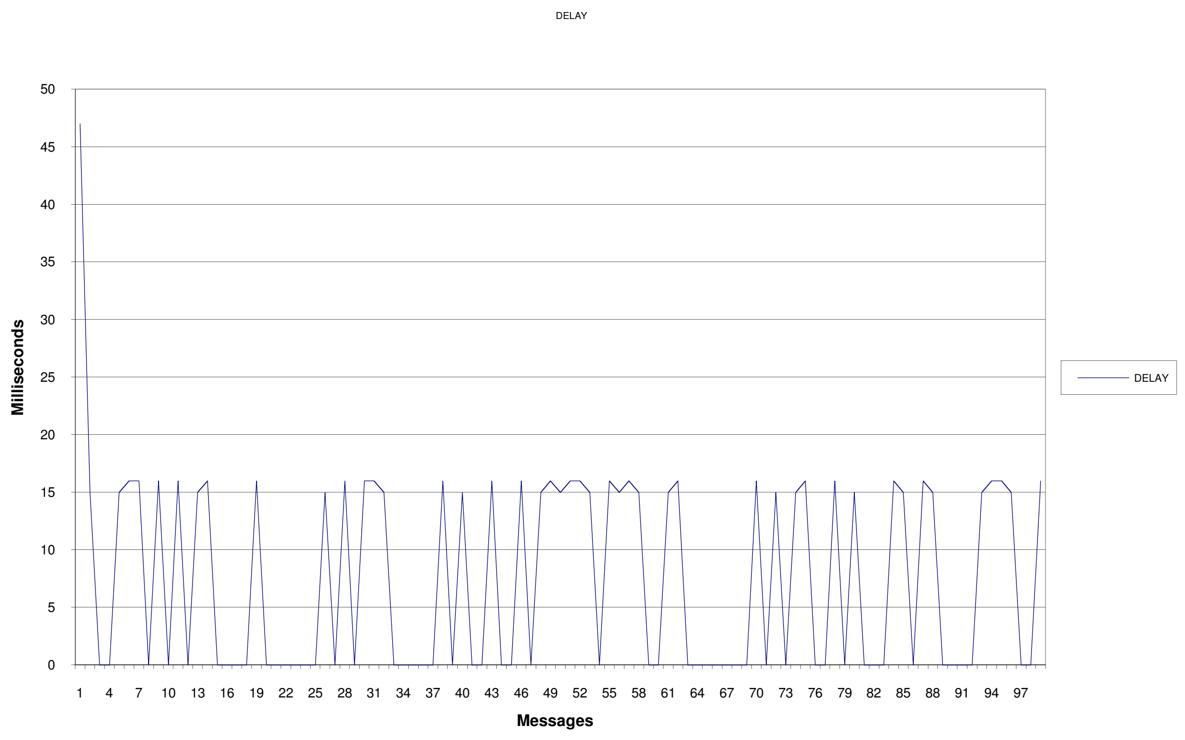


Figure 16 - Delay in milliseconds from messages received at the game controller device from a sensor

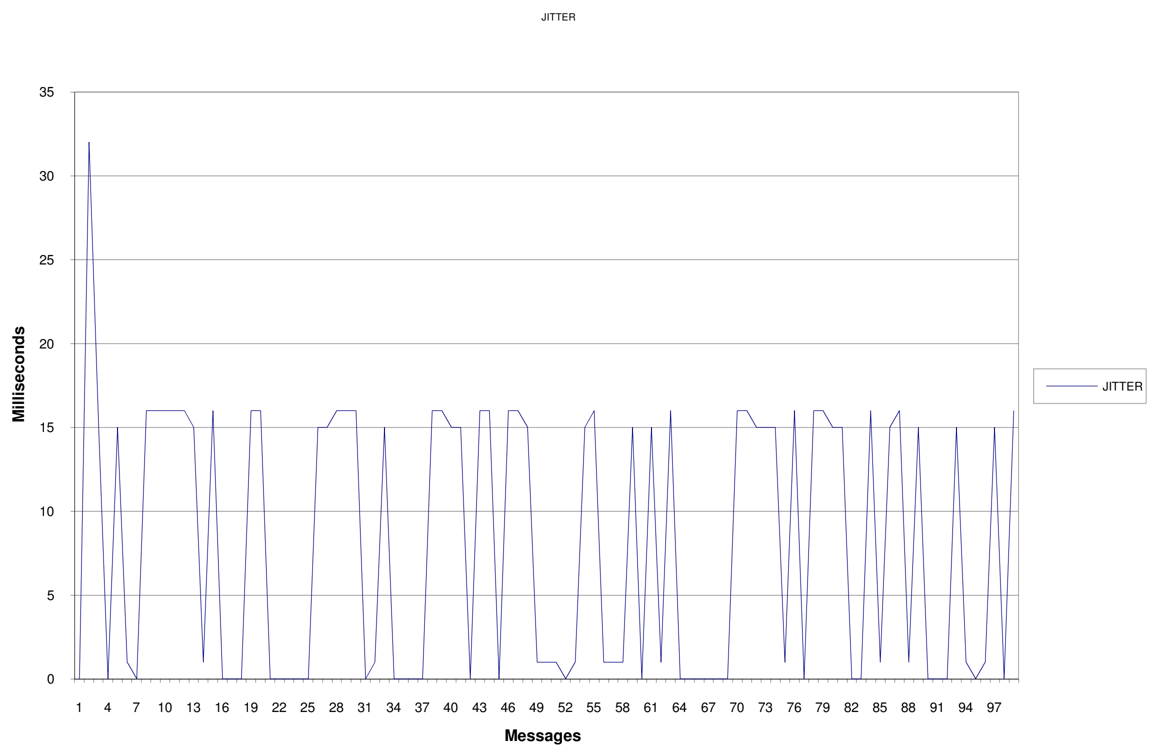


Figure 17 - Jitter in milliseconds from messages received at the game controller device from a sensor

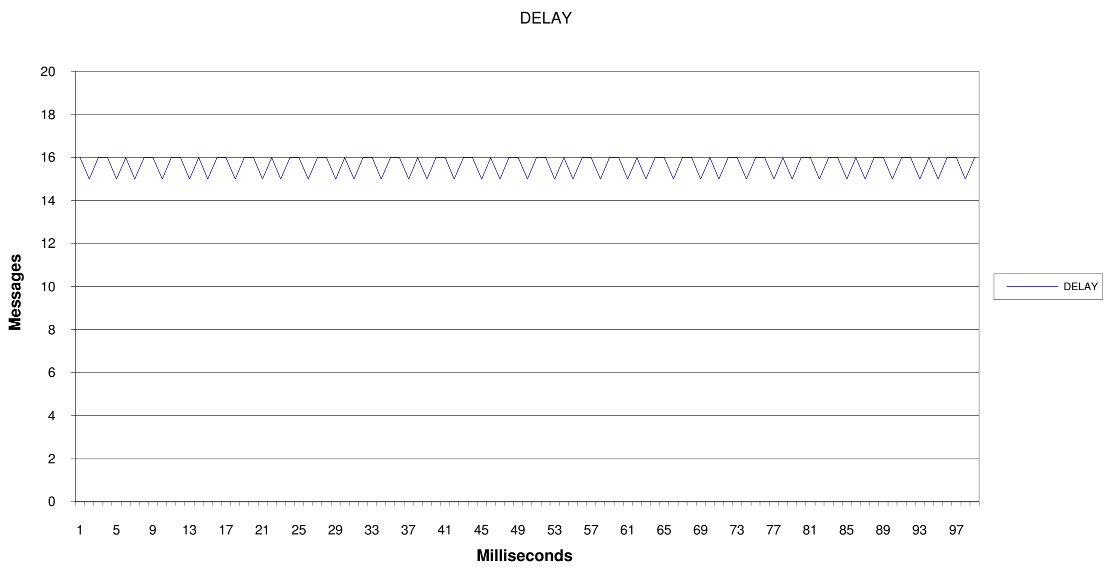


Figure 18 - Delay in milliseconds from messages received at the actuator from the main game controller device

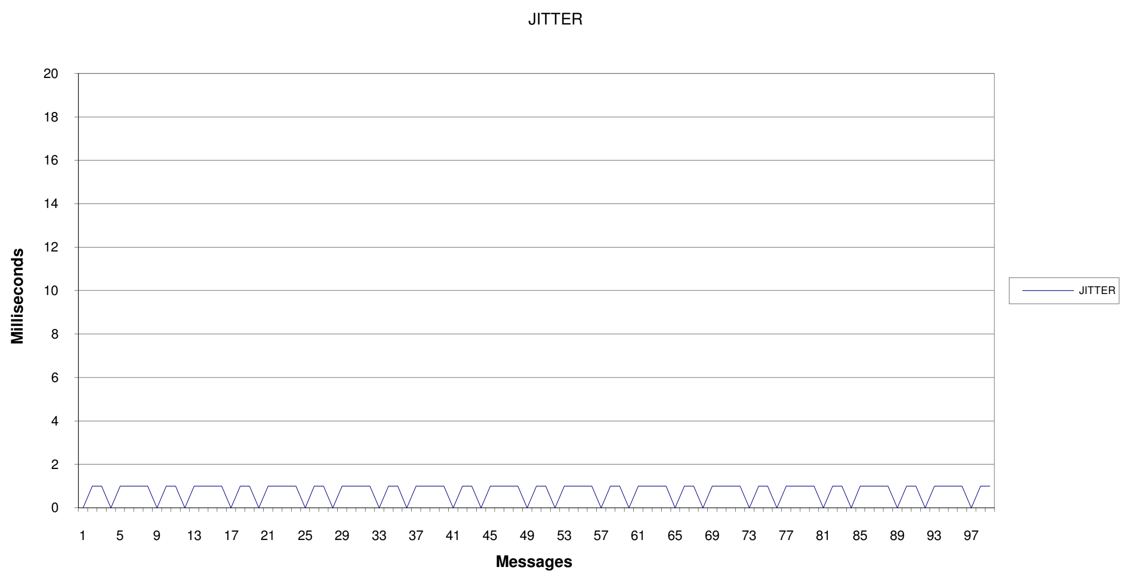


Figure 19 - Jitter in milliseconds from messages received at the actuator from the main controller game device

We present here extracts from the results for delay and jitter for two of those tests, one for a simple simulated position sensor, and another for a simple force actuator.

The tests show that the combined sensor-middleware delay varies from 0 to 16 milliseconds with jitter from 0 to 16 milliseconds, while the combined actuator-middleware delay varies from 15 to 16 milliseconds with jitter from 0 to 1 millisecond. These values prove that the solution is adequate for small sensor and actuator networks around a central coordinating device, such as the ones found in mobile and pervasive augmented reality scenarios.

4.3. EVALUATING THE SIXRM PROTOCOL

The Sixrm protocol API as it is implemented was subject to various functional and stress tests, with several profile settings. It was demonstrated that, according to the profile settings, one can have many instances of Sixrm running on the same computer (over 10 instances, if the settings are right). It was demonstrated also, that according to the settings, many more instances could be run on different computers. This is because they will not consume all CPU time, like we did on these stress tests, but only a minimum, and the sources will try to adapt to the slowest computer and weakest link (because of nak throughput adaptation, and error throughput adaptation, within the limits set forth in the profile).

We did tests on two computers connected by a 100 Mbit full duplex switch configured in a IPv6 network, and tests in one only computer configured to run in a ipv6 network.

Those were real tests, not simulated tests. The test results presented here are extracts from the tests with one computer, because they show the delay and jitter introduced solely by the responsibility of the Sixrm protocol.

We present here graphics for delay and jitter for one of the nodes (received at the first node) of a Sixrm network witch achieved twelve nodes in the same computer without errors. When the thirteen’s node was added, there were briefly some errors that were handed to the test application by the Sixrm protocol, in a period when the computer's processor was at a peak load of 100% and availability was arriving at its physical limits, which we believed were the causes of the inability of the node to handle processing data.

The profile settings for these tests are described in Table 25.

The Figure 20 represents delay for the second node on the system as received on the first node. The Figure 21 represents jitter for the same situation.

Profile value	Setting
maxOutBufferSize	3000 messages
maxInBufferSize	3000 messages
maxSentBufferSize	1000 messages
maxQueuedBufferSize	5 messages
maxRecvBufferSize	1000 messages
minRecvWindowSize	10 messages
maxRecvWindowSize	250 messages
minRandomBackoff	10 millisecond
maxRandomBackoff	50 millisecond
minIntervalTime	25 milliseconds
maxIntervalTime	250 milliseconds
intervalSteppingDown	1 millisecond
intervalSteppingUp	16 millisecond

Table 25 - Profile settings for the tests

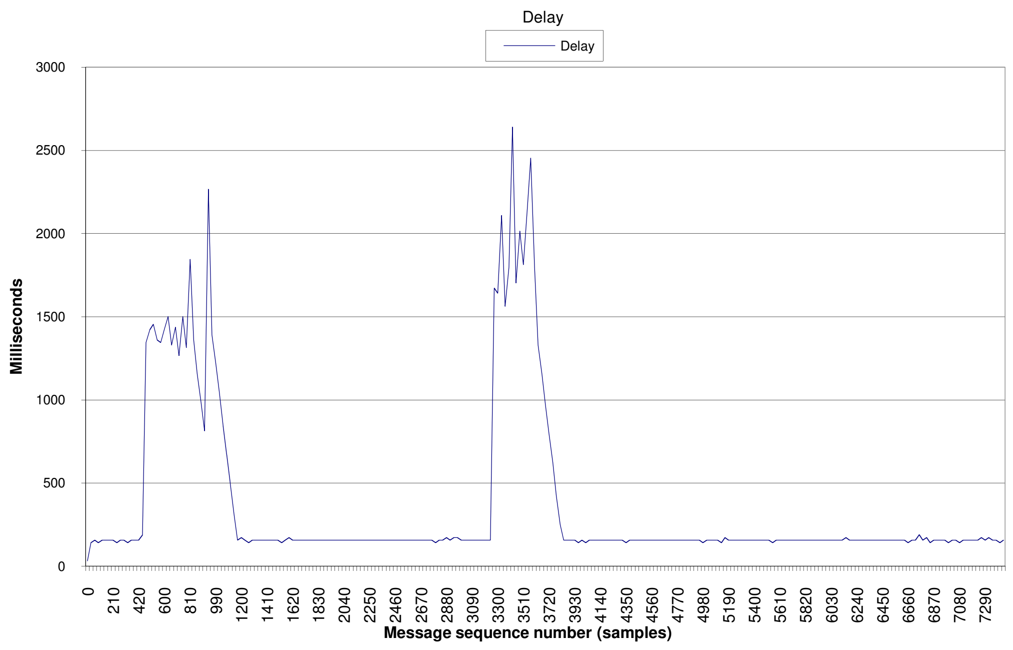


Figure 20 - Delay from second node on first node

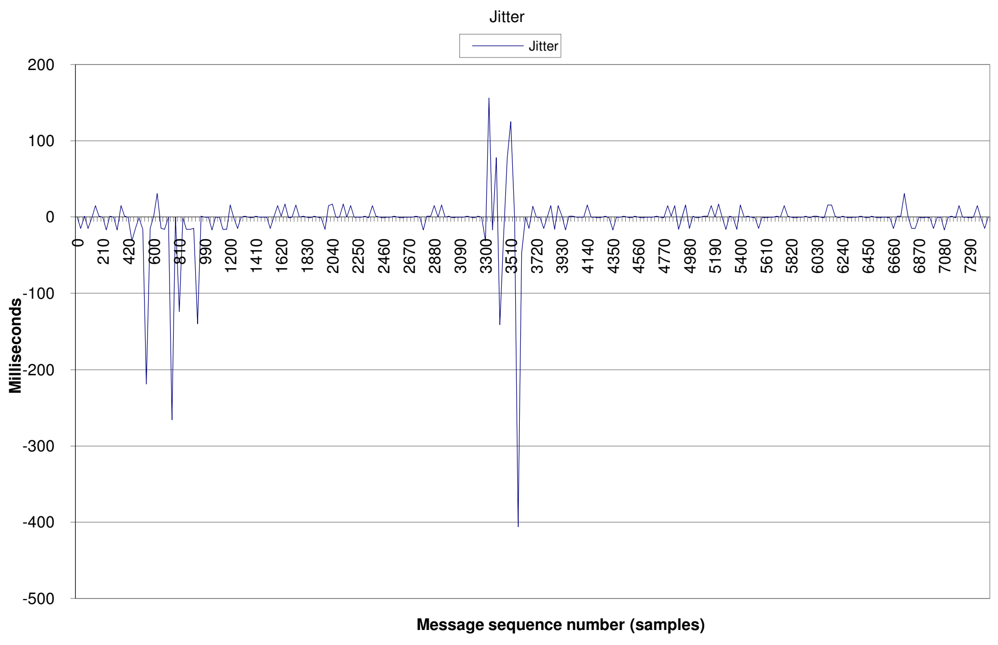


Figure 21 - Jitter from second node on first node

We do not show here results from more nodes because the results are in all cases similar to these.

From these results we can conclude that the delay is normally always approximately 150 milliseconds and that the jitter normally varies between 0 and approximately 17 milliseconds. This happens consistently except some exceptions, which occur exactly on the moments where we were adding nodes to the Sixrm network and the network was adjusting itself. We can see that in these moments the delay and jitter briefly increase and then stabilise in acceptable values again. In fact, they stabilize in the same values as the number of nodes increases.

These numbers for delay and jitter (the stable ones) are adequate for most interactive delay sensitive applications.

As the server network will mostly be a stable one, as probably all nodes will be functioning since the first few minutes when integrated into the large scale network, the Sixrm protocol is, we think, adequate for the purpose we create it for.

4.4. EVALUATING THE QOS APIS

To test our architecture we have made some functional tests, on a simulated implementation of the PDP context Handler architecture and on a protocol implementation of RSVP.

4.4.1. EMULATION OF THE PDP CONTEXT HANDLER ARCHITECTURE

We did not have access to a platform where we could implement real PDP context activation and deactivation, in a way that we could test it, in Java. So, we emulated the API, implementing all its functionality internally in such a way that applications can be made in the emulator using this API and in the future, a real implementation (not emulated), probably using a native interface to the native features of real UEs, can really allocate and deallocate PDP contexts.

Using this emulation environment, functional tests were made to the proposed API, enabling us to confirm its operational capabilities.

4.4.2. RSVP IMPLEMENTED BY UDP ENCAPSULATION WITH A SIMULATED ROUTER IN BETWEEN

As for RSVP, we implemented RSVP, both on the emulated UE and on the distributed server using UDP encapsulation, which is a feature of the protocol [63].

Our implementation does not support the features of integrity checking (optional in all messages), and policy data (stated in the RFC as further study item).

A future alternative implementation could implement these items.

For testing, we built a program that simulated a router with UDP encapsulation support and tested in an isolated fashion the communications of the RSVP protocol between a program that sent and received data (so it needed reservations), an RSVP simulated router, and a similar program (actually the same program) running on another machine.

The functional tests consisted of, considering both computers as senders and receivers at the same time, setting up reservations in RSVP having a router in between the two (actually, the simulated router).

The tests only targeted the protocol, no real reservations at the network layer were made.

The tests were successful and we proceeded with the integration of the API with the rest of the Status Transmission Framework Middleware.

4.5. EVALUATING THE SECURITY ARCHITECTURE

To test our security architecture, we are going to test its usability for our target middleware objectives, which are mobile and pervasive large scale augmented reality games.

These objectives put scalability constraints on security architecture and also QoS constraints such as delay and jitter.

We can test the security architecture for delay and jitter targets, and we can analyse it to derive conclusions about its scalability.

We can divide these tests into the various parts of the system, namely in the personal area network, on the link between the personal area network on the large scale network level, and in the large scale network level itself.

This is because the central level works essentially as a key distribution level and so does not get itself involved in any real-time communications.

4.5.1. PERSONAL AREA NETWORK LEVEL SECURITY TESTING

The personal area network part of the system, both the STF PAN API and the STF SENSACT API, was subject to extensive functional and performance tests, with various kinds of simulated sensors and actuators and a simulated reading and actuating application using Java Wireless Toolkit 2.5 Beta from Sun Microsystems running in a series of emulators in a Pentium 4 3.6 GHz System with 1 Gb Memory. These tests give the same results as the tests run in [13] as nothing has changed, we still use Bluetooth encryption and authentication. So, we do not present graphics here due to the fact that the results are the same as presented before. So, we did not add any delay or jitter to the previous architecture.

4.5.1.1. TESTING THE COMMUNICATIONS BETWEEN THE PAN AND THE LARGE SCALE DISTRIBUTED LEVEL SERVER

Between the personal area network central device, that runs Java 2 Micro edition Mobile Information Device Profile 2.0 over the Connected Limited Device Configuration 1.1 (MIDP 2.0 and CLDC 1.1), and the distributed server where it happens to be connected there is a TLS connection over TCP where STF messages are exchanged according to our protocol.

To be able to test for delay and jitter on this connection, which is handled by specialized classes of STFPAN (the library of classes for the central device of the personal area network) and STFServer (the library of classes for the distributed servers and central server), we implemented time stamping of messages with the current time when sending the message, and automatically calculating the delay based on that timestamp and the current time on the receiving machine when receiving the messages. For the test to be meaningful, both machines must be synchronized through NTP, preferably to a common timeserver.

We also implemented logging to a file both on STFPAN and STFServer of the received delay values so that we can calculate the delay and jitter values for the various UE (mobile terminals or central devices of the personal area network), and elaborate graphics.

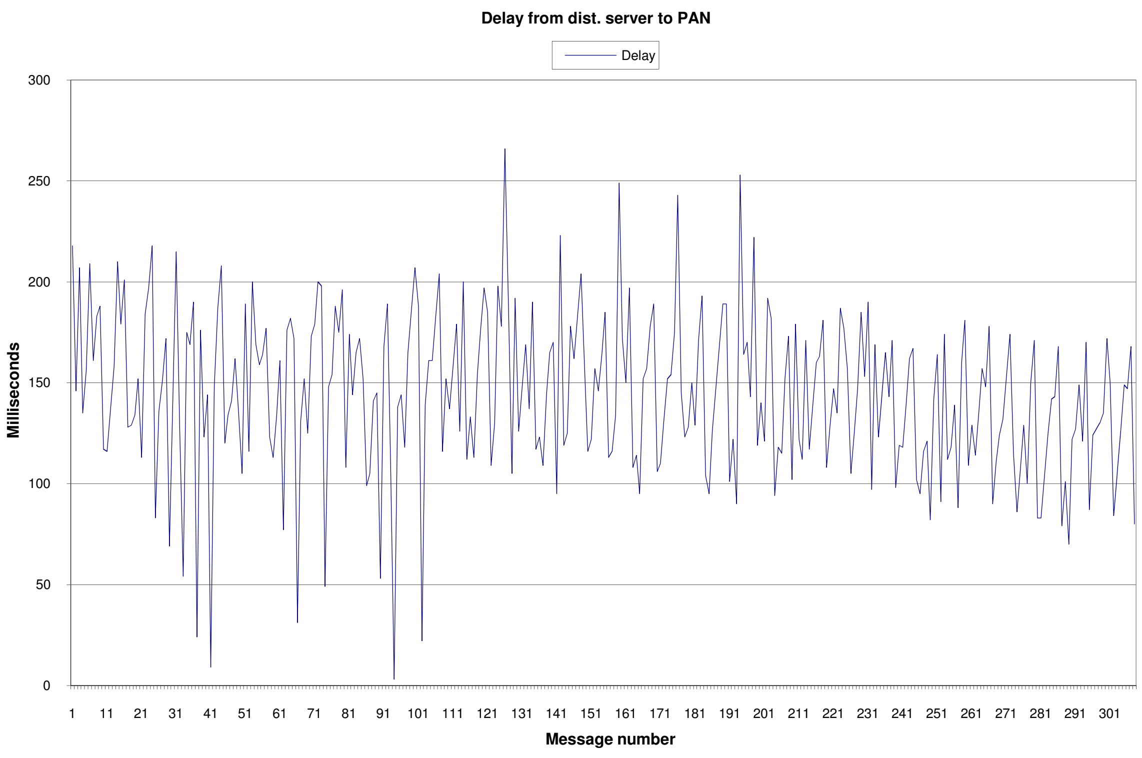


Figure 22 - Delay received at PAN from distributed server

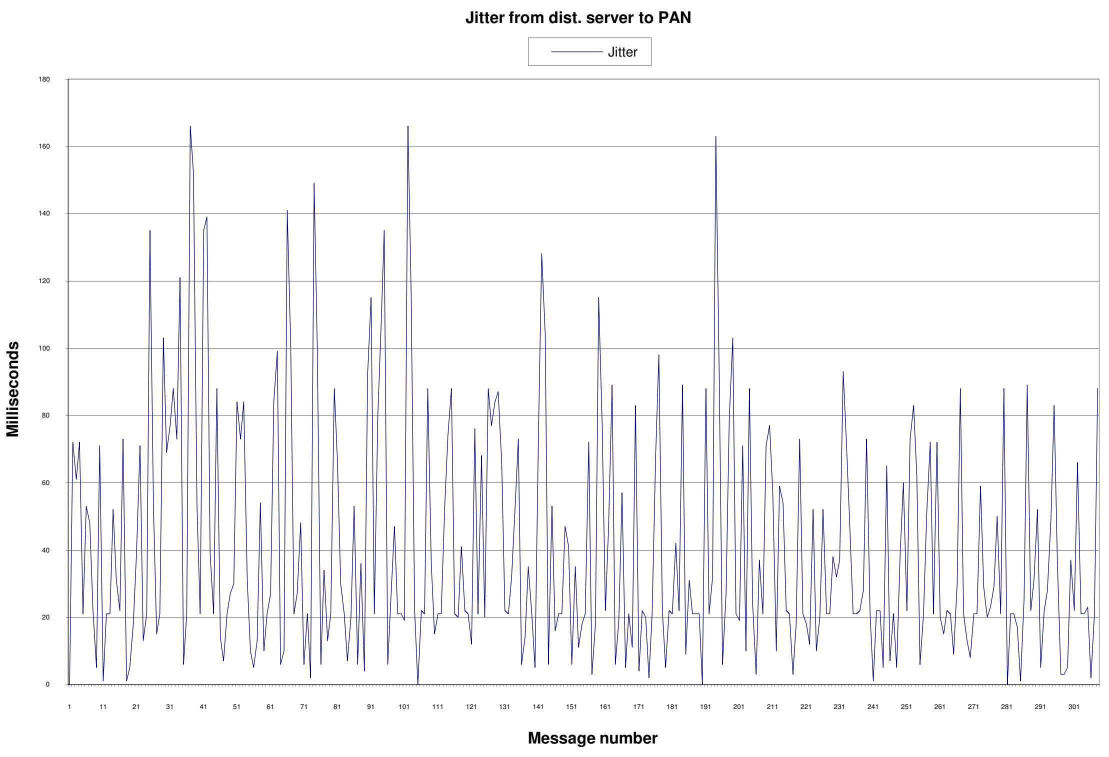


Figure 23 - Jitter received at PAN from distributed server

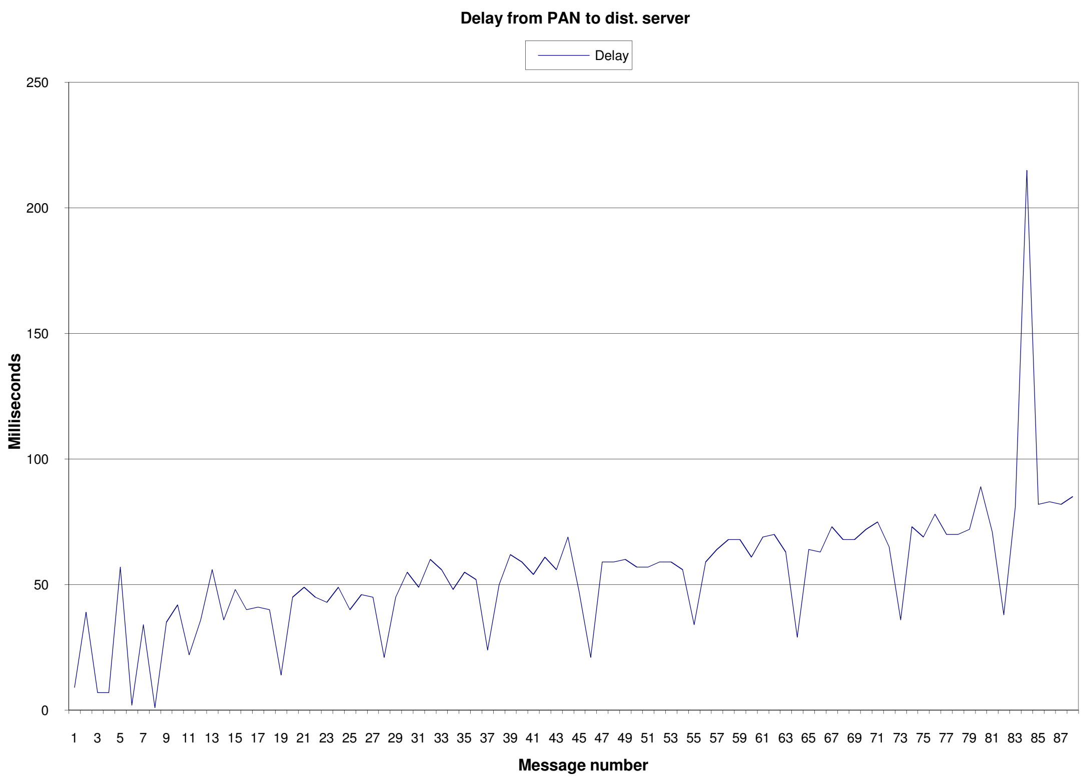


Figure 24 - Delay received at the distributed server from the PAN

We then made the tests with only one UE connected to one distributed server working in connection to one central server. In this situation, even if we do not have a working application to test messaging, there are messages exchanged between the distributed server and the central device of the personal area network, between the central device of the personal area network and the central server and between the distributed server and the central server.

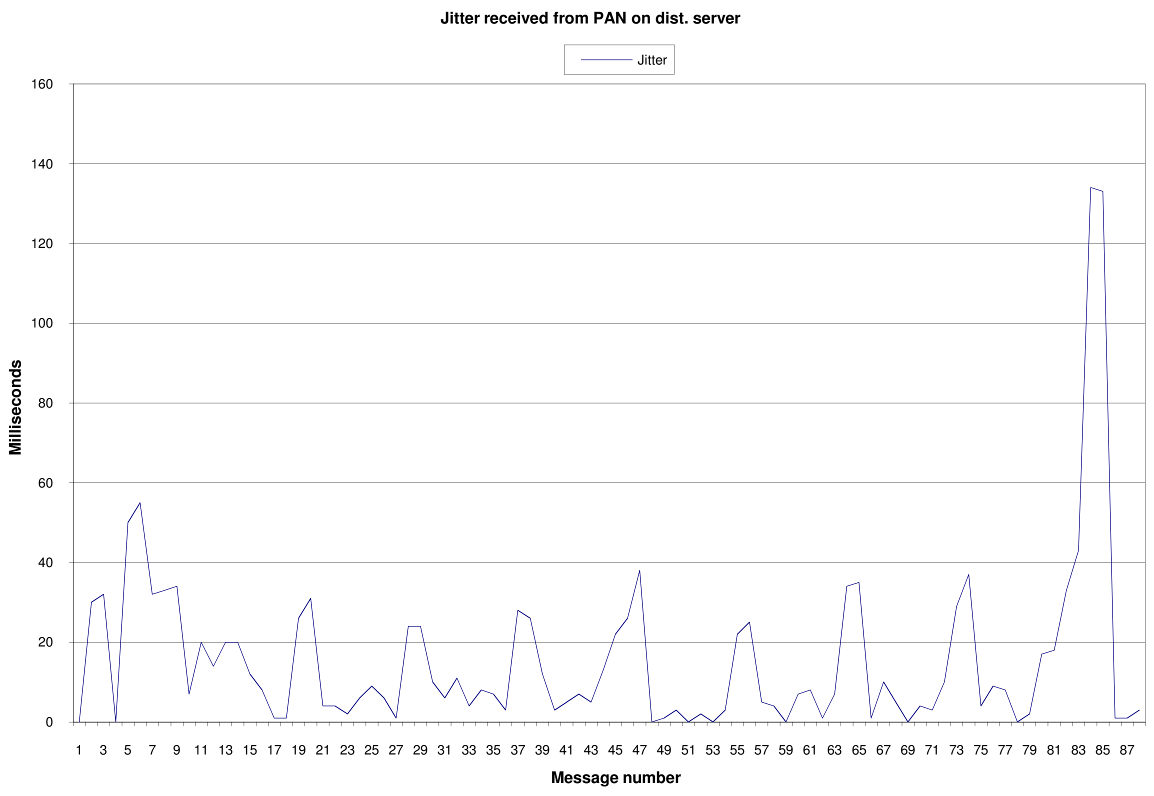


Figure 25 - Jitter received on the distributed server from the PAN

The messages we are interested in are the messages between the central device of the personal area network and the distributed server. At this stage, those messages are only messages of virtual time synchronization of STF's internal virtual time synchronization mechanism, which keeps a virtual clock synchronized between the central device of the personal area network and the distributed server. Is the delay and jitter of those messages that is evaluated in the graphics shown on this section.

Figure 16 shows the delays for all the messages received at the central personal area network device that were sent from the distributed server, and Figure 17 shows the jitters for the same messages, calculated as the difference between current and previous delay.

Figure 18 shows the delays for all the messages received at the distributed server that were sent from the central personal area device, and Figure 19 shows the jitters from the same messages.

From these figures we can see that the delay from the messages received at the central device of the PAN is always between 9 and 218 milliseconds. The Jitter from the same messages is always between 0 and 160 milliseconds.

We can also see that the delay from the messages received at the distributed server from the central device of the PAN is always between 7 and 215 milliseconds and the jitter varies from 0 to 134 milliseconds.

4.5.2. TESTING COMMUNICATION BETWEEN LARGE SCALE DISTRIBUTED LEVEL SERVERS

For testing communications between the large scale distributed level servers, we also implemented time stamping of messages the same way we implemented on the connection between the central device of the personal area network and a distributed server.

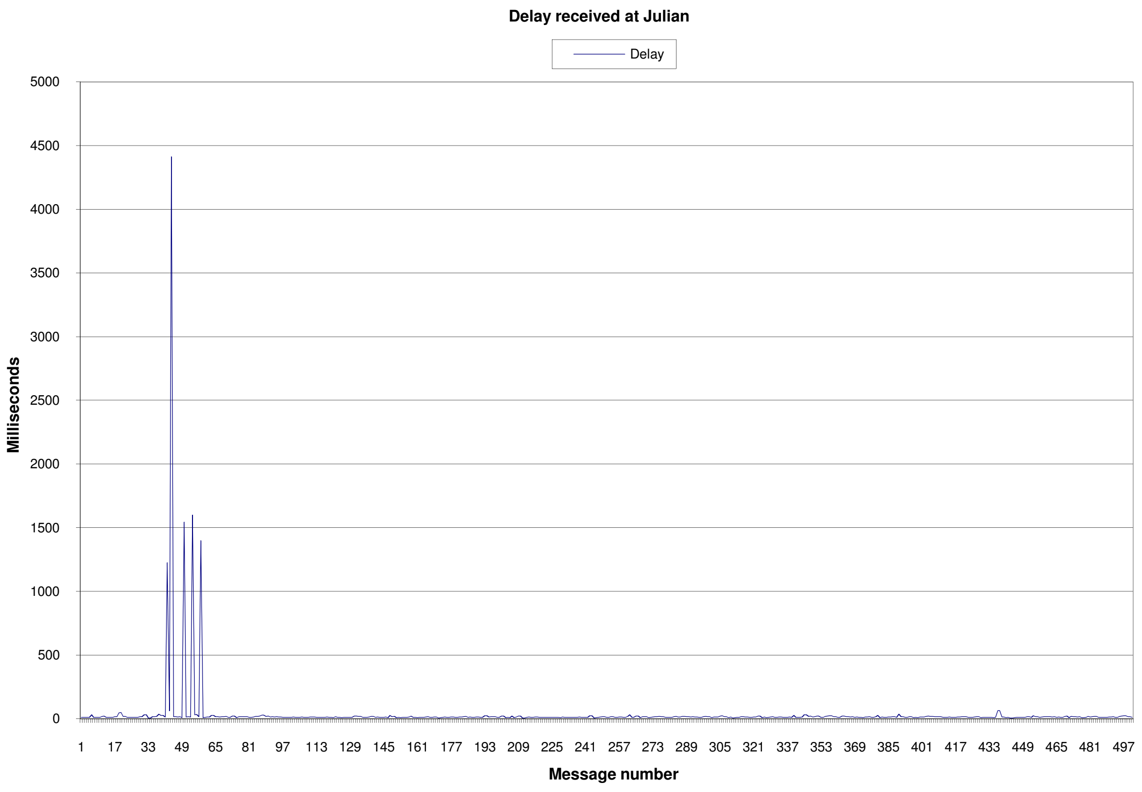


Figure 26 - Delays received from the distributed server Gillian on distributed server Julian

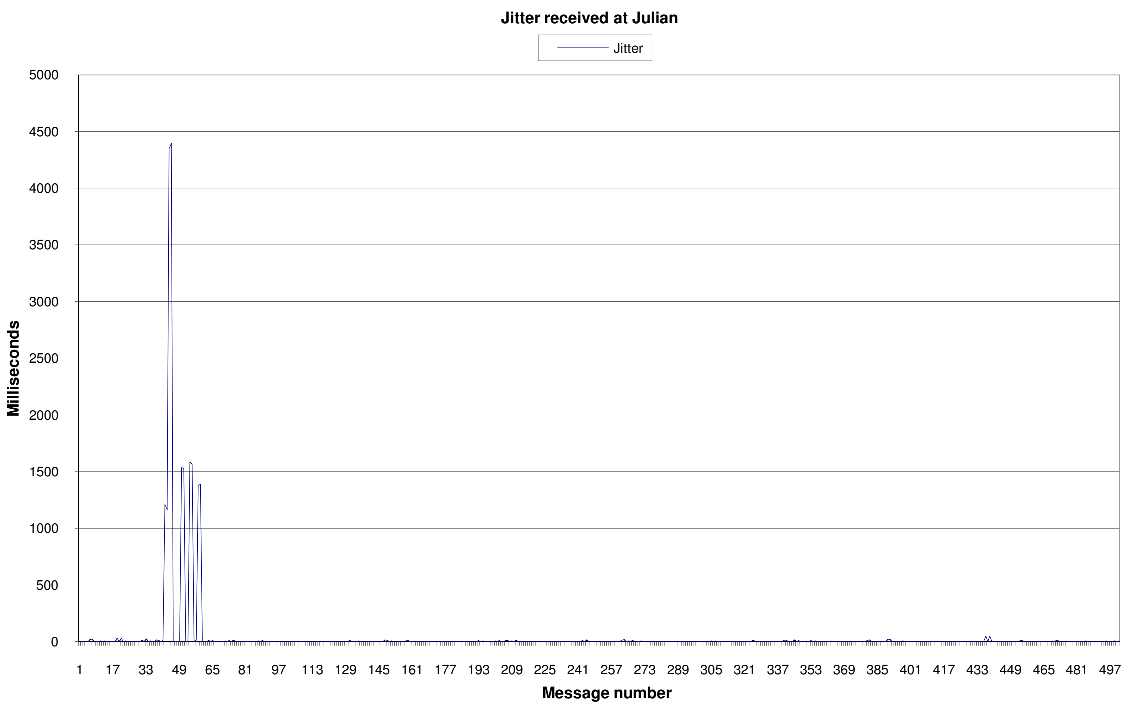


Figure 27 - Jitter received from the distributed server Gillian on the distributed server Julian

The message is marked with a timestamp equal to current time when sending and is marked again with the delay when receiving the message at the receiving end.

We also implemented logging of the received message delays (from which we can also derive the jitter), on the distributed servers.

Because we still do not have a concept application to run on the architecture, the only messages running on the distributed servers are the virtual time synchronization messages. And is the delay and jitter of these messages that's shown on the graphics below.

So, Figure 26 represents the delays of all the messages transmitted from distributed server Gillian to distributed server Julian, while Figure 27 represents the jitters for the same messages.

Figure 28 represents the delays for messages received at distributed server Gillian transmitted from distributed server Julian, while Figure 29 represents the jitters for the same messages.

We only tested with two distributed servers, because that were sufficient for testing the effects of cryptography on communication and we are not full of resources.

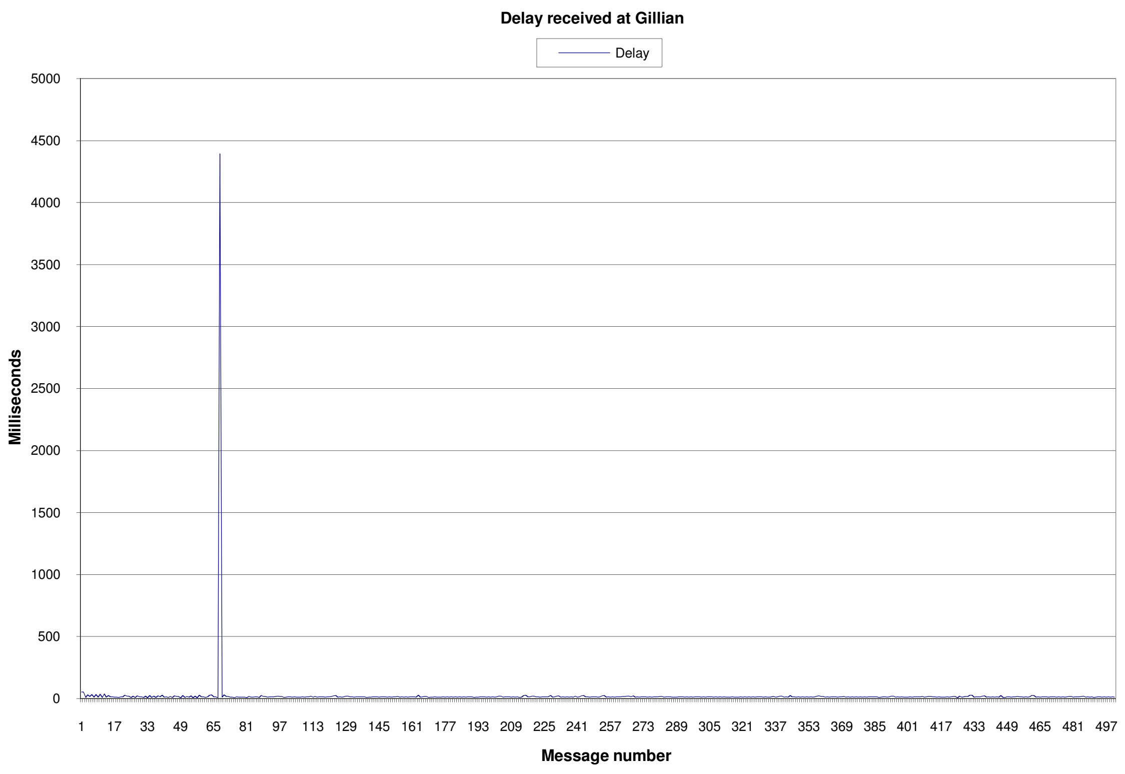


Figure 28 - Delay received from the distributed server Julian on the distributed server Gillian

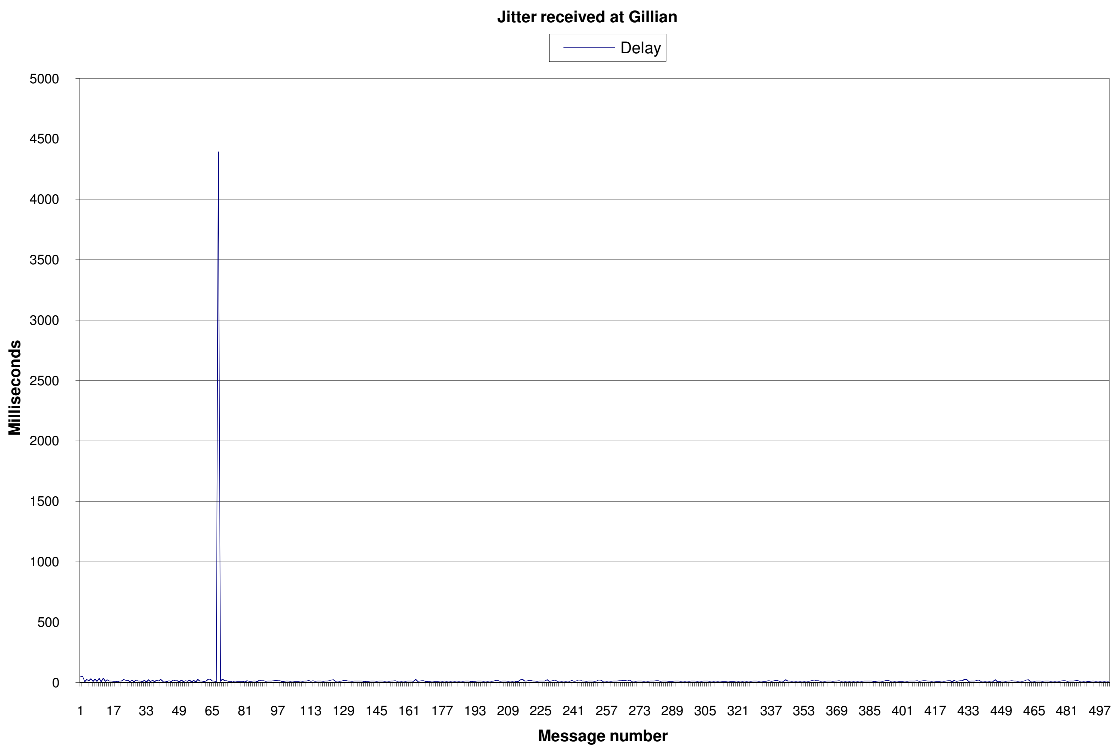


Figure 29 - Jitter received from the distributed server Julian on the distributed server Gillian

From these figures we can see that the delay from distributed server Gillian to distributed server Julian is always between 7 and 62 milliseconds, in fact most below 30 milliseconds, with the exception of some isolated values in the order of 4-5 seconds which are obviously due to network loss or to processor overload or some other external factor. Jitter is normally between 0 and 50 (normally below 30), with the same isolated values.

Values for delay and jitter for the case where the transmission was from Julian to Gillian are similar.

4.6. EVALUATING COMMON APIS

We have also tested the persistence sub-package on its own in order to estimate network bandwidth usage. For this, we designed a program to compare the savings made in relation to java Serialization and Externalization mechanisms and to allow us to calculate the size of a message for a common case. This common case was chosen to be the transmission of a position update (position sensor, for example). Then we ran a program to evaluate the size of the message that is produced applying each of the three methods, and printed the results. The obtained results were 92 bytes for Serialization, 83 bytes for externalization and only 22 bytes for SENSACT persistence.

4.7. EVALUATING THE MANAGEMENT ARCHITECTURE

For the testing phase of the management architecture, we configured the system with one UE (emulated with a program made over the middleware STFPAN on Java Wireless Toolkit 2.5), one central server (running on another computer), and one distributed server (running in the same computer as the central server). The HSS was emulated by one program of ours, that makes part of our middleware and that emulates essential function of the HSS that our application servers need. This is the minimum configuration.

First we started the central server, then the distributed server, then the UE. The UE connected to the central server by SIP and by SSL over TCP to the distributed server, after it had negotiated the session and the respective QoS properties.

We then, after some time, started the management application on the same computer where the servers were running.

First we connected to the central server. Then we connected to the distributed server (the only one). Then we tested every menu option. The results were acceptable for the scenario.

4.7.1. SCALABILITY ANALYSIS

As for scalability, we can analyze it in the following way. When the statistics are disabled, there are no messages going around in the system. Scalability is not a problem. Authorization of users only involves the central server and management application. Revoking of authorization only involves the central server and management application. Scalability is not a problem.

When generating statistics, the various UE (can be many of them per distributed server) periodically send their reports to the distributed server, which aggregates it into one only report and sends it periodically to the central server. Therefore, the central server only receives each aggregated UE report from the distributed server and each distributed server report. The number of reports received by the central server depends on the number of distributed servers, not on the number of UEs (i.e. the number of users on the system). Generated reports from the central server are generated and kept on the central server. The management application connects to the central server and possibly to a distributed server. The system is therefore scalable.

As for resetting statistics, only one message is diffused only one time for the entire system. As for checking if the user is running, only the central server and management application are involved. As for enabling statistics and disabling statistics, only one message is diffused by the entire system one only time. So the system is kept scalable.

Of course there is a performance penalty in enabling statistics generating and collecting over not having the system generating statistics generation and collecting, but this is unavoidable, and tells the manager that he should use this capacity sparingly, mostly to test the system while still in alpha or beta testing and occasionally, to track some bugs that may exist.

4.8. BUILDING A CONCEPT APPLICATION

Having built the Status Transmission Framework 2.0 Network Middleware for Large-Scale Mobile and Pervasive Augmented Reality Games, and tested its functions in isolated fashion, it is now time to demonstrate it working with a concept application.

4.8.1. THE OBJECTIVE

The objective of programming and executing this concept application, which is made of various modules, is to demonstrate that it is really possible, given the right hardware, to use the Status Transmission Framework Software to build the kind of mobile augmented reality applications it was conceived for.

4.8.2. THE CONCEPT

The concept of the game is assumedly simple: you enter a world (real/virtual) where in the middle of the real world you will find unreal structures, enemies, possibly firings, and other real players. Real players are your friends. The objective is to kill enemies and do not get killed by them. If you do get killed by the enemies, you get out of the game (you get disconnected after GAME OVER).

The apparatus of each player is as follows: Each player carries with him a central gaming device (possibly a mobile phone, but the requirement is that it must support 3GPP and MIDP 2.0 and CLDC 1.1). This device has installed STF PAN with the Concept Application Main Game Device Application, which have to be activated by the user when it wants to play.

The user also carries with him a pointing device called a "VirtualLightRay" device, a Bluetooth sensor with STF SENSACT installed, supporting orientation sensors and Bluetooth technology, besides CLDC 1.1.

Finally, the last concept device is the Glass/Camera apparatus, which is really two devices on one: A camera, which is a imaging sensor (CLDC1.1, Bluetooth, SENSACT), which sends images to the central device, and the Glasses, a output device, which display the real and imaginary data to the user's eyes, which are an actuator (CLDC1.1,Bluetooth,M3D,MMAPI). In the example in hand, all the devices are emulated using Java Wireless Toolkit 2.5 from Sun Microsystems, so they are also using MIDP 2.0, even if, in some situations, it was not needed. But this is the only emulator at hand.

4.8.3. THE ARCHITECTURE

The distributed game database contains all the data for the imaginary part (the part not on the real world) of the scenery, including positions, vertices, faces, appearances (such as colours, materials, etc.), and is located at each of the distributed servers.

The distributed game database filler is a program we have made to fill the database for testing purposes with structures, lights, enemies, etc..., and with the correct appearances for the initial start of a demonstration game.

The central game application is responsible for session initiation and termination and basically extends the basic central server that the STFServer middleware provides.

The distributed application is responsible for the application main logic at the distributed level and does so by extending functions provided by some classes of the base STFServer middleware.

The main game device application is responsible for the application logic at the terminal level including late join, initialization and communication and final termination, and does so by extending functions provided by some classes of the base STFPAN middleware.

There are two sensor applications in the concept application: The imaging camera on the glasses and the "VirtualLightRay".

Those imaging camera simply uses MMAPI to take pictures of the real world and transmit them through Bluetooth through the central device using SENSACT.

The VirtualLighRay simply gets the current orientation (where we are pointing at) and if we press the button, sends the indication via SENSACT to the central game device.

The actuator application is the glasses, which display the mix of real and virtual world based on the image they receive by SENSACT from the central game device and the virtual world info they receive also by SENSACT also from the central game device.

Note that the central game device knows the location of the user and transmits this location to the glasses.

4.8.10. RUNNING THE CONCEPT APPLICATION

To run the concept application, first you need to run the HSS (Home Subscriber Server) simulator included in the package of the STFServer API. Then you run on another or the same computer the STFConceptCentralServer. Then you run the STFConceptDistributedServer. You may run more distributed servers. Then you run the clients. In any order, you run the MIDP clients: ConceptLookingGlasses, ConceptLookingCamera, ConceptVirtualLightRay, and ConceptCentralMidlet.

You may do this for each user you want to connect to each distributed server.

When testing a user, activate (Press the button to activate) the sensors or actuators before you start the session setup process in the ConceptCentralMidlet MIDP Application.

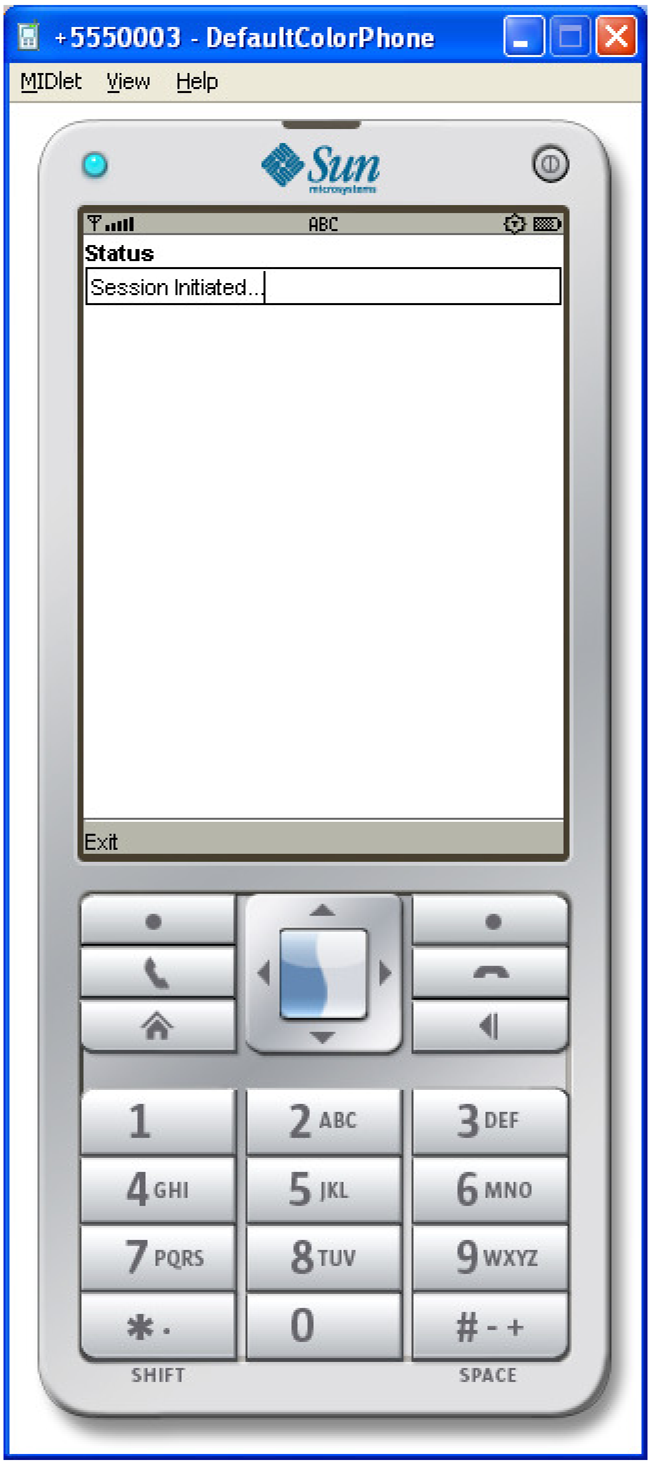
Figure 30 is a screenshot of the ConceptCentralMidlet MIDP application running in a mobile phone emultator (Java Wireless Toolkit from Sun Microsystems) whit the session already initiated. The function of the ConceptCentralMidlet is to run on the central device of the personal area network and manage the session,report status to the user, and let the user control when and how to enter and leave the game.

Figure 31 is a screenshot of a camera sensor (ConceptLookingCamera driver application) with a MIDP display. Normally we would join this with the glasses in Figure 32 and will have only the glasses display, but for simulation purposes we have chosen to separate the camera from the glasses. The function of the sensor is to provide images of the real world to the glasses. In a real driver we would not use MIDP.

Figure 32 is a screenshot of the ConceptLookingGlasses driver application which implements a simulated (obiously on a Java Wireless Toolkit from Sun Microsystems) 3D augmented reality glasses display using as camera input the images from the camera in Figure 31.

Figure 33 represents the ConceptVirtualLightRay sensor which represents a virtual pointing device, which uses 3D orientation sensors to point to objects in the augmented reality world and so permit interaction.

The reason because even the sensors and actuators are implemented as MIDP applications is that no emulation platform exists, at least to our knowledge, that allows us to test in a sandbox consisting only of CLDC 1.1 and Bluetooth APIs, and the Mobile Media and Mobile 3D APIs. These are only available in conjunction with emulators for mobile phones (which include MIDP 2.0), such as the one used, the Java Wireless Toolkit, from Sun Microsystems. However, ConceptLookingCamera, ConceptVirtualLightRay (the sensors) and ConceptLookingGlasses(the actuator) do not need MIDP 2.0, only CLDC 1.1. The only application which needs MIDP 2.0 is ConceptCentralMidlet, which runs on the central game device.



Status

Session Initiated...

Figure 30 - ConceptCentralMidlet working with a session active

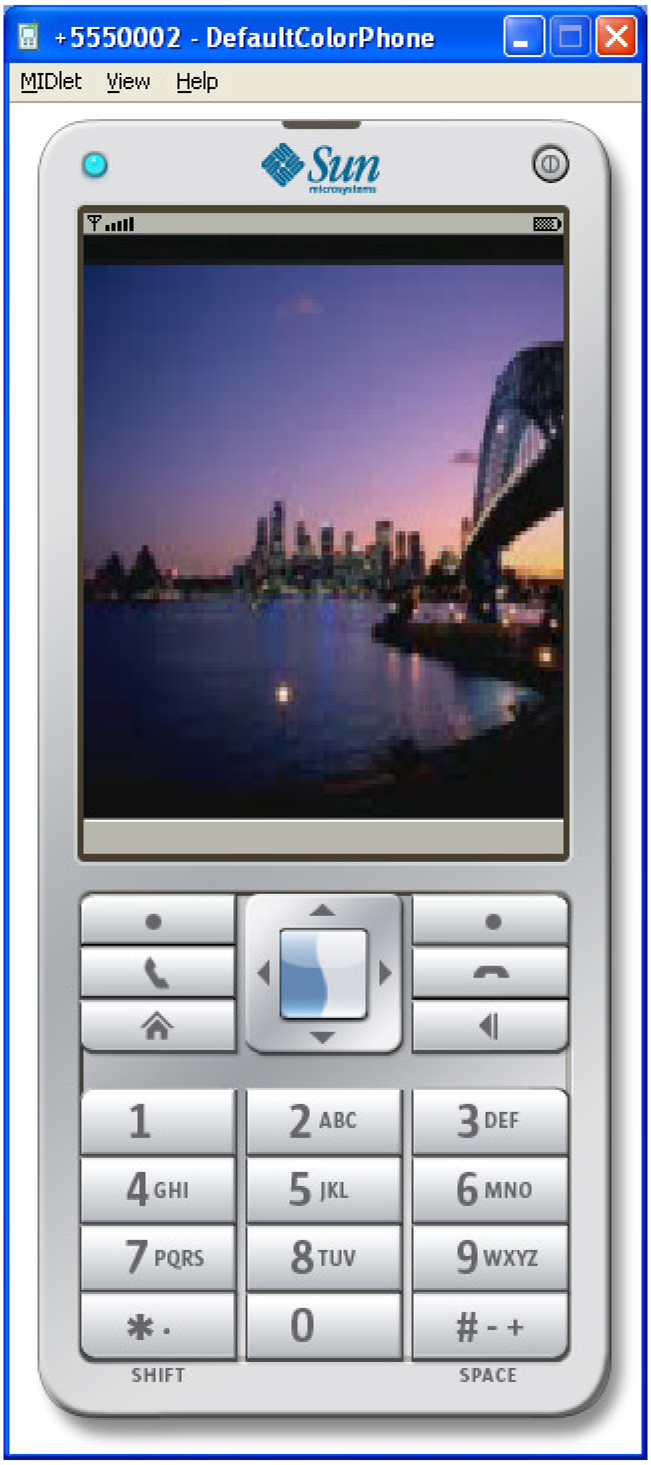


Figure 31 - ConceptLookingCamera working with a session active

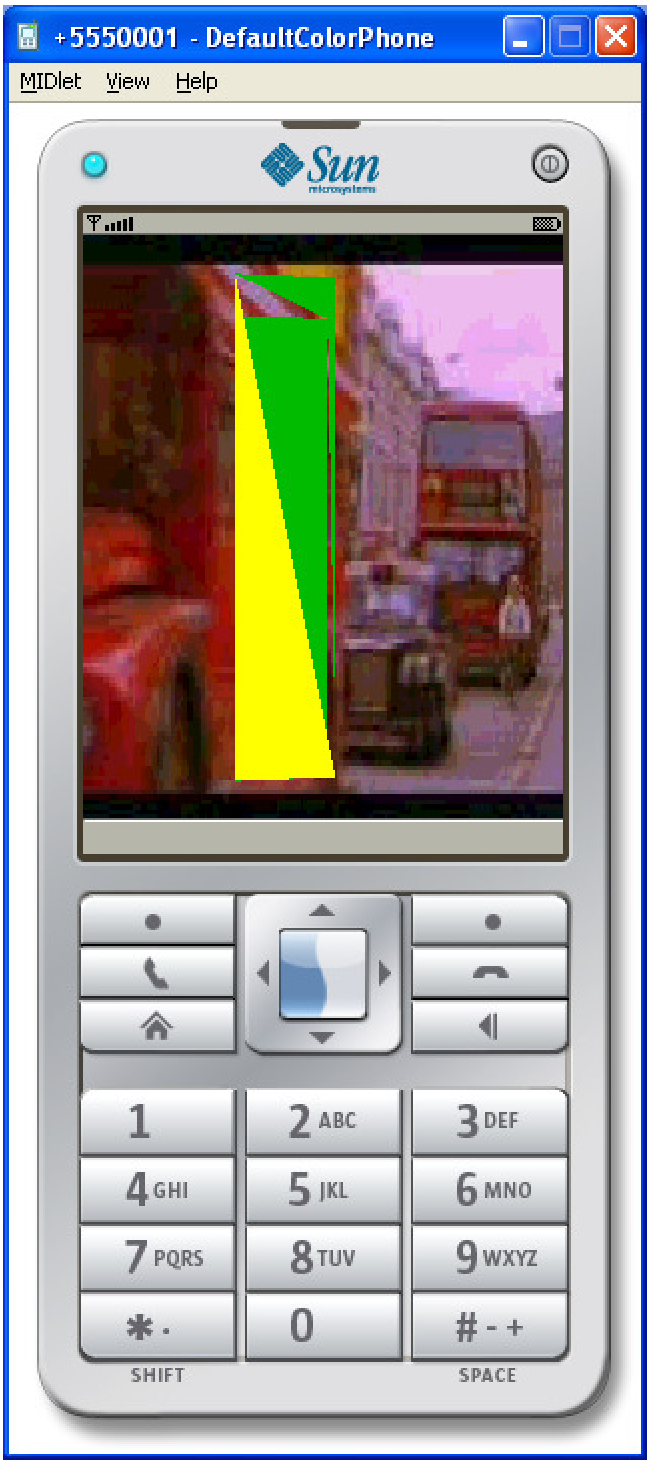
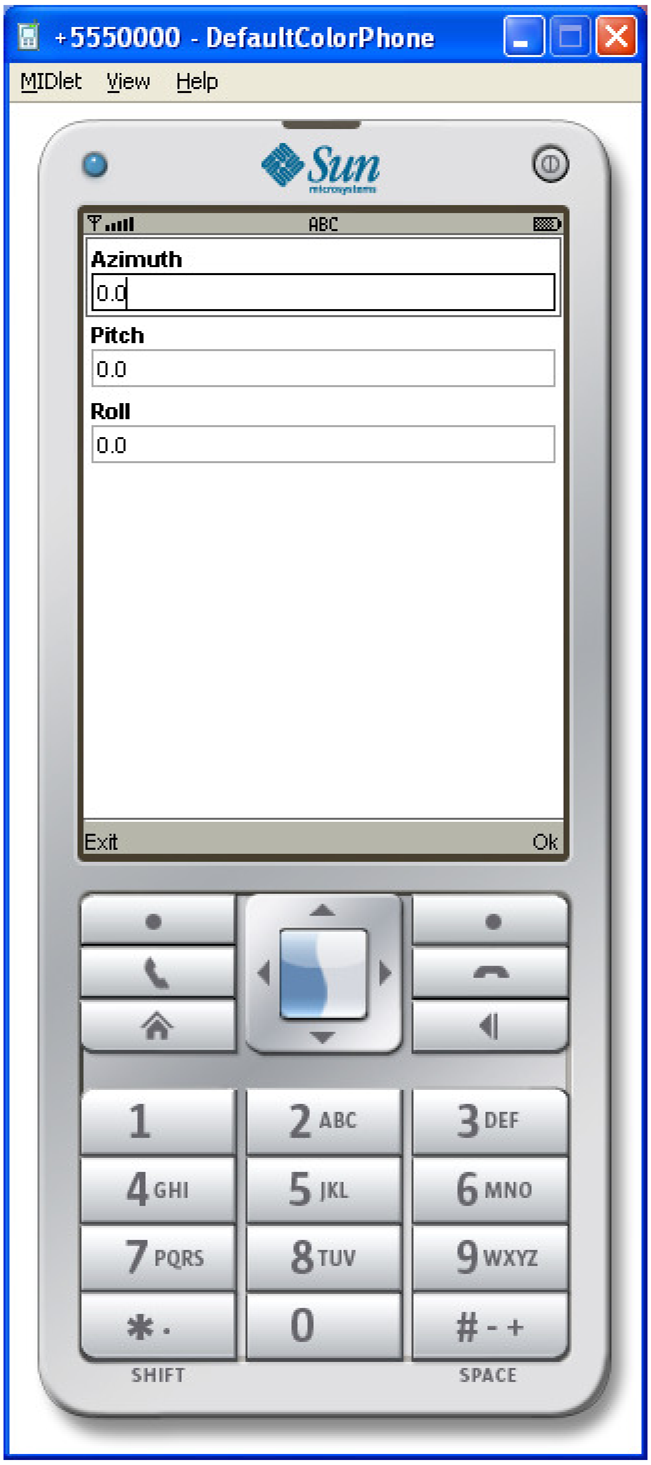


Figure 32 - ConceptlookingGlasses working with a session active



Azimuth
0.0
Pitch
0.0
Roll
0.0

Figure 33 - ConceptVirtualLightRay working with a session active

5. CONCLUSIONS AND FUTURE WORK

In this thesis we have proposed, built and evaluated a network middleware for large-scale mobile and pervasive augmented reality games, which works over a 3GPP network. As such, we have fulfilled our primary objective, stated in the introduction chapter.

This middleware has evolved from previous work in the area of interactive distributed multimedia, more specifically in state transmission for a collaborative virtual environment middleware platform, the Status Transmission Framework (STF)[7][8]. This platform extended ARMS - Augmented Reliable corba Multicast System [9] -with capabilities for the handling of state transmission for distributed collaborative virtual environments.

We also achieved all the other we have proposed ourselves in the introduction chapter. Mechanisms were proposed, built and evaluated that dealt with network issues of:

- Mobility: Host mobility is handled by the system not only by mechanisms specific of 3GPP but also by the distributed servers. The main game device moves from server to server depending on its location.
- Quality of Service (QoS): Architecture for QoS in Java for 3GPP networks was developed as part of our network middleware for mobile and pervasive large scale augmented reality games.
- Security: A whole architecture for security was built as part of our middleware for mobile and pervasive large scale augmented reality games, including authentication and authorization and encryption, and in doing so, increasing the difficulty of cheating and improving the privacy of user data.
- Management of Networks and Services: A management architecture specific to our middleware for mobile and pervasive large scale augmented reality games was built to enable management of the behaviour specific to the middleware and the analysis of execution statistics. The management of the 3GPP (UMTS) network may be done through the tools available to the network operator.
- Discovery: The sensors and actuators are dynamically discovered by the main game device using a messaging architecture that extends the Bluetooth SDP protocol, through the use of the SENSACT and STF PAN APIs.
- Ad-hoc networking and dynamic configuration: An ad-hoc network constituted of sensors, actuators and a main game device is formed and auto configured by the SENSACT and STF PAN APIs, according to orders from the application, using service discovery, and then configured with the main game device emitting values for the actuators and receiving values from the sensors.
- geospatial location and orientation: The middleware enables the application to use location and orientation in any way it sees fit and supports location and orientation in base messages and protocols. For example, partitioning is based on location.

- Also mechanisms were also proposed, built and evaluated that deal with architectural issues of:

- scalability: All the architecture (the three levels, and the distribution of the load and data) was though taking scalability as target. We have proved the architecture is the most scalable one from the possible alternatives in the evaluation chapter;
- consistency: Consistency algorithms were used and supported to guarantee consistency.
- multimedia data heterogeneity: The system supports 3D data and state, which may be mixed with audio and video by the applications and with real data from the real world. As such, we have real multimedia data heterogeneity.
- data distribution and replication: The system distributes data trough the central server, distributed server, databases at the central and distributed server sites which may have replication, main game device and sensors and actuators.

As such, we can conclude that all the objectives of this work were fulfilled.

As for papers published and the contributions they have made, we have already stressed in the introduction that the main contribution of this PhD was the building, testing and evaluation of a network middleware for mobile and pervasive large scale augmented reality games. To our knowledge, there no other similar middleware in existence. However, as sub contributions, several papers were writed and published about aspects of the middleware architecture, protocols and components. These were:

- The paper published on CoNext 2005 [10], which contributed with a general view about what we were planning to do with this PhD and consequently, with this middleware (basically, a position paper).

- The paper published on EuroCon 2005 [11], which contributed with a general architecture for the system we were planning to build for the middleware.

- The paper published on InfoCom 2006 [13], which contributed with an API (the SENSACT API), which allows sensors and actuators to speak to each other through a Bluetooth ad-hoc network which is then organized so that the STFPAN API running on the central device can control all sensors and actuators running on the personal area network of the user.

- The paper published on SOFTCOM 2006 [14], which contributed with a new reliable multicast protocol that works over IPv4 or IPv6, nak based, source ordered and many-to-many communications based and that avoided duplicates, which avoided the nak implosion problem and reported errors to the application when these are unavoidable.

- The paper published in CSNR 2007 [12], which contributed with the full architecture for the basic system of the middleware (including SENSACT, STFPAN and STFSERVER), with results from testing.

- The paper published in WORLDCOMP 2007 [17] in the conference IWCN 2007, that contributed with an analysis of the scalability of the system and that proves that our system architecture is the most scalable from the many alternate architectures we may have choosed.
- The paper published in NGMAST 2007[15] that contributes with QoS APIs for the system (both for the UE and for the distributed servers), based on PDP Context activation and RSVP protocol, to allow the system to be correctly used with the IP Multimedia Subsystem.

- The paper published in SOFTCOM 2007 [16], that contributes with a security architecture that complements the security already provided by 3GPP(UMTS) and allows our system to guarantee privacy, authentication and authorization.

As with any work, we can never say the work is complete. It is always possible to make improvements. It is the case with this work also.

5.1. FUTURE WORK

Future work in the Status Transmission Framework possibly will include new kinds of sensors and actuators included by default on the SENSACT API and the STF PAN equivalent API, and further testing of this API on real devices (not simulated ones) if that becomes possible.

Also in the SENSACT API and correspondent communication layer on the STFPAN API, it would also be interesting to implement support for other personal area network communication technologies like ZIGBEE, for example, which is a direct contender of Bluetooth, but not so much implemented on the mobile world.

We also should, of course optimize and extend the API as required as future research is made.

A bottleneck which in the future, if possible, we should investigate ways to remove is the fact that we use TCP to make the communications between the UE (STFPAN API) and the current allocated distributed server (STFSERVER API), though through a message oriented protocol we created especially designed for augmented reality and that utilizes available bandwidth adaptively.

The fact that we use TCP and not any form of multicast protocol on these connections is a fact that hinders performance of the whole system, but that is justified that is impossible, due to current J2ME specifications, which do not give us the possibility to use multicast for mobile network connections.

As for mobility support, the system supports mobility through the moving of the user, besides at the level of the UMTS network, between the various cells using the UMTS network technology and related methods, at the distributed game application server level through switching the gaming server that is currently communicating with the user. Each gaming server is responsible for a different gaming area.

Currently, there is no direct connection between the two levels. It would probably be beneficial to the system if we find a way in the future to integrate these two levels.

For example, instead of managing a rectangular area, the application server could manage a group of cell coverage areas. Communication between the 3GPP(UMTS) layer and the middleware layer should make that operation simpler.

As for system security, we could research ways to further improve the security architecture of the middleware. If we, as already mentioned, implemented other communication technologies into SENSACT and STFPAN we would need to integrate security also with these technologies. We could also introduce new encryption algorithms into STFPAN and STFSERVER, but, because of maintaining high software compatibility, we should maintain J2SE compatibility, so, it is better to wait for J2SE support for these algorithms before actually supporting them.

For system manageability, in the future we could research ways in which to integrate closer the management architecture of our middleware for mobile and pervasive large scale augmented reality with the management architecture of UMTS.

Future work on the QoS APIs will include implementing, testing and evaluating those APIs on real devices and not on emulated or simulated ones.

Finally, it should be interesting to see a real-life proof-of-concept application, with real augmented reality devices, speciality built, instead of a simulated one. That would require the collaboration of a hardware builder.

REFERENCES

[1] M. Weiser, “The Computer for the Twenty - First Century”, Scientific American, pages 94–104, Sept. 1991.

[2] 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

[3] 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

[4] 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

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

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

[7] João Orvalho, Pedro Ferreira and Fernando Boavida, “State Transmission Mechanisms for a Collaborative Virtual Environment Middleware Platform”, Springer-Verlag, Berlin Heidelberg New York, 2001, pp. 138-153, ISBN 3-540- 42530-6 (Proceedings of the 8th International Workshop on Interactive Distributed Multimedia Systems – IDMS 2001,Lancaster, UK, September 2001)

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

[9] 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).

[10] 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

[11] 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

[12] Pedro Ferreira, João Orvalho and Fernando Boavida, “A middleware architecture for Mobile and Pervasive Large Scale Augmented Reality Games”, to be published in Proceedings of the Conference on Networks and Services Research (CSNR) 2007, Fredericton, New Brunswick, Canada, May 2007

[13] Pedro Ferreira, João Orvalho and Fernando Boavida, “Middleware for embedded sensors and actuators in mobile pervasive augmented reality”, in Proc. of the INFOCOM 2006 (IEEE XPLORE), INFOCOM 2006 Student Workshop, Barcelona, April 2006

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

[15] Ferreira, P. and Orvalho, J. , "Quality of Service APIs for a 3GPP Mobile and Pervasive Large Scale Augmented Reality Gaming Middleware", in Proc. of the 2007 Internation Conference on Next Generation Mobile Applications, Services and Technologies (NGMAST 2007), pp. 179-184, Next Generation Mobile Applications, Services and Technologies 2007 (NGMAST 2007), Cardiff, Wales, UK, September 2007

[16] Ferreira, P. and Orvalho, J. and Boavida, F. , "Security and Privacy in a Middleware for Large Scale Mobile and Pervasive Augmented Reality", in Proc. of the 2007 International Conference on Software , Telecommunications and Computer Networks (SOFTCOM 2007), The 2007 International Conference on Software , Telecommunications and Computer Networks (SOFTCOM 2007), Split-Dubrovnik, Croatia, September 2007

[17] Ferreira, P. and Orvalho, J. and Boavida, F. , "Scalability and Mobility in a Network Middleware for Large Scale Mobile and Pervasive Augmented Reality Games", in Proc. of the 2007 International Conference on Wireless Networks (ICWN2007), Part of WORLDCOMP 2007, pp. 304-310, WORLDCOMP 2007, The 2007 International Conference on Wireless Networks (ICWN2007), Part of WORLDCOMP 2007, Las Vegas, Nevada, USA, June 2007

[18] Satyanarayanan M., "Pervasive computing: Vision and Challenges", IEEE Personal Communications, Vol. 8, No. 4, pp. 10-10, August 2001

[19] Deborah Estrin, David Culler, Kris Pister, Gaurav Sukhatme, "Connecting the Physical World with Pervasive Networks", IEEE Pervasive Computing, January-March 2002, pp. 59-69

[20] Vxworks, http://www.windriver.com/

[21] Geoworks, http://www.geoworks.com/

[22] Chorus, http://www.sun.com/chorusos/

[23] Windows Mobile, http://www.microsoft.com/windowsmobile/software/default.mspx

[24] PalmOS, https://pdn.palm.com/regac/pdn/index.jsp

[25] TinyOS, http://sourceforge.net/projects/tinyos/

[26] Tim Kindberg, Armando Fox, "System Software for Ubiquitous Computing", IEEE Pervasive Computing, January-March 2002, pp. 70-81

[27] Cynthia A. Patterson, Richard R. Muntz, Cherri M. Pancake, "Challenges in Location-Aware Computing", IEEE Pervasive Computing, April-June 2003, pp. 80-89

[28] Woodrow Barfield, Thomas Caudell, "Basic Concepts in Wearable Computers and Augmented Reality", in "Fundamentals of Werable Computers and Augmented Reality", edited by Woodrow Barfield and Thomas Caudell, ISBN 0-8058-2901-6 (cloth), ISBN 0-8058-2902-4 (pbk), Lawrence Elrbaum Associates, 2001

[29] Rick LaRowe and Chip Elliot, “Computer Networks for Wearable Computing”, in “Fundamentals of Werable Computers and Augmented Reality”, edited by Woodrow Barfield and Thomas Caudell, ISBN 0-8058-2901-6 (cloth), ISBN 0-8058-2902-4 (pbk), Lawrence Elrbaum Associates, 2001

[30] Cartsten Magerkurth, Adrian David Cheok, Regan L. Mandrik, Trond Nilsen, “Pervasive Games: Bringing Computer Entertainment Back to the Real World”, ACM Computers in Entertainment, Vol. 3, No. 3, July 2005, Article 4A.

[31] Steve Benford, Rob Anastasi, Martin Flintham, Adam Drozd, Andy CrabTree, Chris Greenhalgh, Nick Tandavanitj, Matt Adams, Ju Row-Farr, “Coping with Uncertainty in a Location-Based Game”, IEEE Pervasive Computing, July-September 2003, pp.34-41

[32] Kumar Ranganathan, “Trustworthy Pervasive Computing: The hard security problems”, Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops (PERCOMW’04), IEEE Computer Society, 2004

[33] Roshan K. Thomas, Ravi Sandhu, “Models, Protocols and Architectures for Secure Pervasive Computing: Challenges and Research Directions”, Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops (PERCOMW’04), IEEE Computer Society, 2004

[34] Uwe Hansmann, Lothar Merk, Martin S. Nicklous, Thomas Stober, “Pervasive Computing”, Second Edition, Springer, ISBN 3-540-00218-9, 2003

[35] Elliot D. Kaplan, “Understanding GPS Principles and applications”, Artech House Publishers, Boston, London, 1996

[36] Axel Küpper, “Location-Based Services – Fundamentals and Operation”, John Wiley & Sons Ltd, ISBN 978-0-470-09231-6, 2005

[37] Java Community Process, “Location API for J2ME”, http://www.jcp.org/en/jsr/detail?id=179

[38] Sumi Helai, “Standards for Service Discovery and Delivery”, IEEE Pervasive Computing, July-September 2002, pp. 95-100

[39] The uPnP Forum , http://www.upnp.org/

[40] The Salutation Consortium, http://www.salutation.org/

[41] E. Guttman, C. Perkins, J. Veizades, M. Day, “Service Location Protocol, version 2”, RFC 2608, Network Working Group, IETF, June 1999

[42] The 3rd Generation Partnership Project (3GPP), http://www.3gpp.org/

[43] Sumit Kasera, Nishit Narang, “3G Mobile Networks – Architecture, Protocols and Procedures”, McGraw Hill, ISBN 0-07-145101-3, 2004

[44] R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. Swarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, V. Paxson, “Stream Control Transmission Protocol”, RFC2960, Network Working Group, IETF, October 2000

[45] Miikka Poikselkä, Georg Mayer, Hisham Khartabil, Aki Niemi, “The IMS – IP Multimedia Concepts and Services in the Mobile Domain”, John Wiley & Sons Ltd, ISBN 978-0-470-87113-3, 2005

[46] Gonzalo Camarillo, Miguel A. Garcia-Martin, “The 3G IP Multimedia Subsystem – Merging the Internet and the Cellular Worlds”, John Wiley & Sons Ltd, ISBN 978-0-470-87156-0, 2005

[47] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, E. Shooler, "SIP: Session Initiation Protocol", RFC3261, Network Working Group, IETF, June 2002

[48] P. Calhoun, J. Loughney, E. Guttman, G. Zorn, J. Arkko, "Diameter Base Protocol", RFC 3588, Network Working Group, IETF, September 2003

[49] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", RFC 1889, Network Working Group. IETF, January 1996

[50] 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

[51] 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

[52] Debashis Saha, Amitava Mukherjee, Somprakash Bandyopadhyay, "Networking Infrastructure for Pervasive Computing – Enabling technologies and Systems", Kluwer Academic Publishers, ISBN 1-4020-7249-X, 2003

[53] "Bluetooth Specification Version 2.1 + EDR", available online at http://www.bluetooth.com/

[54] http://www.bluetooth.com/

[55] Sudhir Dixit, Ramjee Prasad(Editors), "Wireless IP and Building the Mobile Internet", Universal Personal Communications, 2001

[56] M. Pullen, M. Myjack, C. Bouwens, "Limitations of Internet Protocol Suite for Distributed Simulation in the Large Multicast Environment", RFC 2502, IETF, February 1999

[57] Reliable Multicast Transport (IETF Working group), http://www.ietf.org/html.charters/rmt-charter.html, Acessed April 2006

[58] T. Speakman, J. Crowcroft, J. Gemmell, D. Farinacci, S. Lin, D. Leshchiner, M. Luby, T. Montgomery, L. Rizzo, A. Tweedly, N. Bhaskar, R. Edmonstone, R. Sumanasekera, L. Vicisano, "PGM Reliable Transport Protocol Specification", RFC 3208, IETF, December 2001

[59] Pedro Ferreira, "Transmissão de estados em ambientes de realidade virtual distribuídos e colaborativos", M.Sc. thesis, Universidade de Coimbra - FCTUC - Departamento de Engenharia Informática, October-2002

[60] MPEG, http://www.chiariglione.org/mpeg/

[61] OMG, http://www.omg.org/

[62] CORBA, http://www.corba.org/

[63] R. Braden (Ed.), L.Zhang, S. Berson, S. Herzog, S. Jamin, "Resource Reservation Protocol – version 1 Fuctional Specification", RFC 2205, IETF Network Working Group, September 1997

[64] J. Wroclawski, "The Use of RSVP with IETF Integrated Services", RFC2210, IETF Network Working Group, September 1997

[65] S.Shenker, J. Wroclawski, “General Characterization Parameters for Integrated Service Network Elelements”, RFC 2215, IETF Network Working Group, September 1997

[66] J.Wroclawski, “Specification of the Controlled-Load Network Element Service”, RFC 2211, IETF Network Working Group, September 1997

[67] S.Shenker, C.Partridge, R. Guerin, “Specification of Guaranteed Quality of Service”, RFC2212, IETF Network Working Group, September 1997

[68] Robert Lloyd-Evans, “QoS in integrated 3G networks”, Artech House Publishers, ISBN 1-58053-351-5, 2002

[69] J. Case, M. Fedor, M. Schoffstall, J. Davin, “A Simple Network Management Protocol (SNMP)”, RFC1157, Network Working Group, IETF, May 1990

[70] “Digital cellular telecommunications system (Phase 2); Network management (NM); Part 1: Objectives and structure of Network Management (GSM 12.00)”, ETSI, December 1997

[71] Shinji Uchiyama, Kazuki Takemoto, Kiyohide Satoh, Hiroyuki Yamamoto, and Hideyuki Tamura, “MR Platform: A Basic Body on Which Mixed Reality Applications Are Built”, Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR’02), IEEE, 2002

[72] Wen tongCAI, Percival XAVIER, Stephen J. TURNER, Bu-Sung LEE, “A Scalable Architecture for Supporting Interactive Games on the Internet”, Proceedings of the 16th Workshop on Parallel and Distributed Simulation (PADS.02), IEEE, 2002

[73] Stefan Fiedler, Michael Wallner, Michael Weber ,“A Communication Architecture for Massive Multiplayer Games”, Proceedings of NetGames2002, April 16-17, 2002, Braunschweig, Germany.

[74] Jalal Al-Muhtadi, Shiva Chetan, Anand Ranganathan and Roy Campbell, ”Super Spaces: A Middleware for Large-Scale Pervasive Computing Environments”, Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops (PERCOMW’04), IEEE Computer Society, 2004

[75] Bruce Thomas, Ben Close, John Donoghue, John Squires, Phillip De Bondi and Wayne Piekarski ,“First Person Indoor/Outdoor Augmented Reality Application: ARQuake”, Personal and Ubiquitous Computing (2002), No. 6, pp. 75–86, Springer-Verlag London Ltd, 2002

[76] P. Beskow, P. Halvorsen, and C. Griwodz. Latency Reduction in Massively Multi-player Online Games by Partial Migration of Game State, In: Second International Conference on Internet Technologies and Applications, ed. by Vic Grout, Denise Oram and Rich Picking, pp. 153--163, University of Wales, NEWI, Wrexham, UK., Centre for Applied Internet Research (CAIR) (ISBN: 978-0-946881-54-3), 2007.

[77] Tino Pissysalo, Petri Pulli, “A Picocell-Based Architecture for a Real-Time Mobile Virtual Reality”, Proceedings of the 9th Euromicro Workshop on Real Time Systems (EUROMICRO-RTS '97), pp. 144-151, 1997, IEEE

[78] Taesoon Park, Namyoon Woo, and Heon Y. Yeom, ”An Efficient Optimistic Message Logging Scheme for Recoverable Mobile Computing Systems”, IEEE TRANSACTIONS ON MOBILE COMPUTING, VOL. 1,NO. 4, OCTOBER-DECEMBER 2002, pp. 265 – 277, IEEE

[79] Sumi Helal, “Pervasive Java”, IEEE Pervasive Computing, January-March 2002, pp. 82-85

[80] 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), IEEE, 2004

[81] M. Handley, V. Jacobson, C. Perkins, "SDP: Session Description Protocol", RFC 4566, Network Working Group, IETF, July 2006

[82] CLDC – Common Limited Device Configuration 1.1, http://jcp.org/en/jsr/detail?id=139

[83] MIDP – Mobile Information Device Profile 2.0, http://jcp.org/en/jsr/detail?id=118

[84] Java APIs for Bluetooth (JSR-82), http://jcp.org/en/jsr/detail?id=82

[85] SIP API for J2ME, http://jcp.org/en/jsr/detail?id=180

[86] Mobile 3D Graphics API for J2ME, http://jcp.org/en/jsr/detail?id=184

[87] Mobile Media API, http://jcp.org/en/jsr/detail?id=135

[88] 3GPP TS23.207 V6.0.0, "3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; End to End Quality of Service (QoS) concept and architecture (Release 6)", September 2005

ANNEXES

ANNEX I – LIST OF PUBLISHED PAPERS

1. 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
2. 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
3. Pedro Ferreira, João Orvalho and Fernando Boavida, "A middleware architecture for Mobile and Pervasive Large Scale Augmented Reality Games", to be published in Proceedings of the Conference on Networks and Services Research (CSNR) 2007, Fredericton, New Brunswick, Canada, May 2007
4. Pedro Ferreira, João Orvalho and Fernando Boavida, "Middleware for embedded sensors and actuators in mobile pervasive augmented reality", in Proc. of the INFOCOM 2006 (IEEE XPLORE), INFOCOM 2006 Student Workshop, Barcelona, April 2006
5. Pedro Ferreira, João Orvalho and Fernando Boavida , " Sixrm: Full Mesh Reliable Source Ordered Multicast" , in Proc. of the SoftCom2006 - 14th International Conference on Software, Tellecommunications & Computer Networks, SoftCom2006 - 14th International Conference on Software, Tellecommunications & Computer Networks, Split, Croatia, September 2006
6. Ferreira, P. and Orvalho, J. , "Quality of Service APIs for a 3GPP Mobile and Pervasive Large Scale Augmented Reality Gaming Middleware", in Proc. of the 2007 Internation Conference on Next Generation Mobile Applications, Services and Technologies (NGMAST 2007), pp. 179-184, Next Generation Mobile Applications, Services and Tecnhologies 2007 (NGMAST 2007), Cardiff, Wales, UK, September 2007
7. Ferreira, P. and Orvalho, J. and Boavida, F. , "Security and Privacy in a Middleware for Large Scale Mobile and Pervasive Augmented Reality", in Proc. of the 2007 International Conference on Software , Telecommuncations and Computer Networks (SOFTCOM 2007), The 2007 International Conference on Software , Telecommuncations and Computer Networks (SOFTCOM 2007), Split-Dubrovnik, Croatia, September 2007
8. Ferreira, P. and Orvalho, J. and Boavida, F. , "Scalability and Mobility in a Network Middleware for Large Scale Mobile and Pervasive Augmented Reality Games", in Proc. of the 2007 International Conference on Wireless Networks (ICWN2007), Part of WORLDCOMP 2007, pp. 304-310, WORLDCOMP 2007, The 2007 International Conference on Wireless Networks (ICWN2007), Part of WORLDCOMP 2007, Las Vegas, Nevada, USA, June 2007

ANNEX II – SENSACT API INTERFACES AND CLASSES

The Table 26, Table 27, Table 28, Table 29 and Table 30 all show the classes and respective functions of the several packages of the SENSACT API.

Class or interface	Function
BluetoothChannel	A Bluetooth communication channel implemented using L2CAP communications.

Table 26 - SENSACT package pt.uc.dei.lcst.stf.sensact.link classes and functions

Class or interface	Function
Actuator	An interface that defines a new actuator
ActuatorValue	This interface defines an actuator value to be acted upon
SensactManager	This interface defines a manager for a sensor or actuator
Sensor	This interface defines a new sensor
SensorValue	Defines a sensor value for the sensor to transmit values
ActivateSensorMessage	A message that activates the sensor
ActuatorDiscoveryMessage	An actuator discovery message
ActuatorStartMessage	A message that starts the actuator
ActuatorStopMessage	A message that stops the actuator
BTSensactManager	Main sensor or actuator manager class for Bluetooth technology.
SensorDiscoveryMessage	A sensor discovery message
SensorStopMessage	A message that stops the sensor
StatisticsLog	A statistics logger.

Table 27 - SENSACT package pt.uc.dei.lcst.stf.sensact.main classes and functions

Class, interface or exception	Function
GenericChannel	A generic communications channel interface
GenericChannelListener	A generic communications channel listener interface
GenericChannelData	A generic channel data message
ChannelException	Base exception for all exceptions related to generic channels of communication.

ChannelReadException	Exception reading a generic channel
ChannelWriteException	Exception writing to a generic channel

Table 28 - SENSACT package pt.uc.dei.lsct.stf.sensact.net classes and functions

Classes and interfaces	Function
PersistentObject	Interface for objects that want to be persistent on STF and CLDC 1.1
PersistentNameArchive	Contains an archive of all ids and names of classes persisted at the moment by this persistence framework
PersistentObjectInputStream	A persistent object input stream for reading persistent objects
PersistentObjectOutputStream	A persistent object output stream for writing persistent objects.

Table 29 - SENSACT package pt.uc.dei.lcst.stf.sensact.persist classes and functions

Class	Function
AccousticFrequencySensorDiscoveryMessage	An acoustic sensor (Frequency) discovery message
AccousticFrequencySensorValue	An acoustic sensor (Frequency) value
AccousticPhaseSensorDiscoveryMessage	An acoustic sensor (Phase) discovery message
AccousticPhaseSensorValue	An acoustic sensor (Phase) sensor value
AccousticPitchSensorDiscoveryMessage	An acoustic sensor (Pitch) sensor discovery message
AccousticPitchSensorValue	An acoustic sensor (Pitch) sensor value
AccousticVolumeSensorDiscoveryMessage	An acoustic sensor (Volume) sensor discovery message
AccousticVolumeSensorValue	An acoustic sensor (Volume) sensor value
BiologicalBodyTemperatureSensorDiscoveryMessage	A biological sensor (Body temperature) discovery message
BiologicalBodyTemperatureSensorValue	A biological sensor (Body temperature) value
BilogicalHeartRateSensorDiscoveryMessage	A biological sensor (Heart rate) discovery message
BilogicalHeartRateSensorValue	A biological sensor (Heart rate) sensor value
BiologicalNeuralActivitySensorDiscoveryMessage	A biological sensor (Neural activity) discovery

	message
BiologicalNeuralActivitySensorValue	A biological sensor (Neural activity) value.
BiologicalRespirationRateSensorDiscoveryMessage	A biological sensor (Respiration rate) discovery message.
BiologicalRespirationRateSensorValue	A biological sensor (Respiration rate) value.
EnvironmentalHumiditySensorDiscoveryMessage	A environmental sensor (Humidity) discovery message
EnvironmentalHumiditySensorValue	A environmental sensor (Humidity) value
EnvironmentalTemperatureSensorDiscoveryMessage	A environmental sensor (Temperature) discovery message
EnvironmentalTemperatureSensorValue	A environmental sensor (Temperature) value
MechanicalAccelerationSensorDiscoveryMessage	A mechanical sensor (Acceleration) discovery message
MechanicalAccelerationSensorValue	A mechanical sensor (Acceleration) sensor value
MechanicalDisplacementSensorDiscoveryMessage	A mechanical sensor (Diplacemement) discovery message.
MechanicalDisplacementSensorValue	A mechanical sensor (Displacement) value
MechanicalForceActuatorDiscoveryMessage	A mechanical actuator (Force) discovery message
MechanicalForceActuatorValue	A mechanical actuator (Force) value
MechanicalForceSensorDiscoveryMessage	A mechanical sensor (Force) discovery message
MechanicalForceSensorValue	A mechanical sensor (Force) value
MechanicalMassSensorDiscoveryMessage	A mechanical sensor (Mass) discovery message
MechanicalMassSensorValue	A mechanical sensor (Mass) value
MechanicalPositionSensorDiscoveryMessage	A mechanical sensor (Position) discovery message
MechanicalPositionSensorValue	A mechanical sensor (Position) value
MechanicalShapeSensorDiscoveryMessage	A mechanical sensor (Shape) discovery message
MechanicalShapeSensorValue	A mechanical sensor (Shape) value
OpticalBrightnessSensorDiscoveryMessage	A optical sensor (Brightness) discovery message
OpticalBrightnessSensorValue	A optical sensor (Brightness) value
OpticalEmissivitySensorDiscoveryMessage	A optical sensor (Emissivity) discovery message
OpticalEmissivitySensorValue	A optical sensor (Emissivity) value

OpticalLightWaveFrequencySensorDiscoveryMessage	A optical sensor (Light wave frequency) discovery message
OpticalLightWaveFrequencySensorValue	A optical sensor (Light wave frequency) value
OpticalLuminanceSensorDiscoveryMessage	A optical sensor (Luminance) discovery message
OpticalLuminanceSensorValue	A optical sensor (Luminance) value
OpticalRefractionSensorDiscoveryMessage	A optical sensor (Refraction) discovery message
OpticalRefractionSensorValue	A optical sensor (Refraction) value

Table 30 - SENSACT package pt.uc.dei.lcst.stf.sensact.pack classes and functions

ANNEX III – STFPAN API INTERFACES AND CLASSES

The Table 31, Table 32, Table 33, Table 34, Table 35 and Table 36 show the classes and corresponding functions for each of the packages of the STF PAN API.

Class, interface or exception	Function
STFApplication	Interface UE main game device applications must implement
STFKey	Interface for message keys
STFLatejoin	Interface for latejoining methods
STFMessager	Interface for messaging methods
STFPartitioner	Interface for partitioning methods
STFSessionListener	Interface for UE main game device applications that manage sessions must implement
STFState	Interface for message state including positioning information
STFStatistics	Statistics collecting interface
DisableStatsUE	Message that disable statistics collecting in an UE
EnableStatsUE	Message that enables statistics collecting in an UE
ResetStatsUE	Message that resets statistics collecting in an UE
StartUEGameActivity	Starts Game activity in an UE
STFChannelManager	Media communications channel manager. Implements messaging, latejoining, partitioning and statistics.
STFLateJoinBeginMessage	Message that starts late joining process.
STFLateJoinCheckpointMessage	Message that starts a checkpoint process.
STFLateJoinEndMessage	Message that ends a latejoin process.
STFLateJoinMessage	Base message of all latejoin messages
STFLateJoinPartMessage	A latejoin message (a retransmitted state message)
STFLateJoinRequestMessage	A latejoin request message.
STFLateJoinStreamMessage	A message indicating the latejoin streaming process is about to begin
STFLocalCache	A class that implements the local cache of messages
STFPartitionMessage	A Message indicating we are going to start partitioning

STFProperties	The configuration properties of STF.
STFSessionManager	The STF session manager. Manages the session through SIP.
STFStateMessage	A STF state message. Includes key and state.
STFStatisticsData	Statistics data.
STFStatisticsReportedMessage	A message that includes statistics data
STFStatisticsReportMessage	A message requesting statistics data
STFTime	A class implementing time sync.
STFTimeHeartBeatMessage	A time sync heart beat message
STFTimeMessage	The base message for all time sync messages.
STFTimePingMessage	A time net sync ping message.
STFTimePongMessage	A time net sync pong message
StopUEGameActivity	Stops all game activity.
SwitchServersMessage	Message to switch game servers.

Table 31 - Classes of STFPAN package pt.uc.dei.lcst.stf.pan

Class, interface or exception	Function
Actuator	This class is to be implemented by all actuators
ActuatorValue	This interface is meant to be implemented by classes that which to represent values to be actuated upon
Sensor	An interface that every class that communicates with a sensor must implement.
SensorValue	Interface that all sensor values for specific sensors must implement.
AccousticFrequencySensor	A class to communicate with an acoustic sensor (Frequency).
AccousticFrequencySensorDiscoveryMessage	An acoustic sensor (Frequency) discovery message
AccousticFrequencySensorValue	An acoustic sensor (Frequency) value
AccousticPhaseSensor	A class to communicate with an acoustic sensor (Phase)
AccousticPhaseSensorDiscoveryMessage	An acoustic sensor (Phase) discovery message
AccousticPhaseSensorValue	An acoustic sensor (Phase) sensor value
AccousticPitchSensor	A class to communicate with an acoustic sensor (Pitch)

AccousticPitchSensorDiscoveryMessage	An acoustic sensor (Pitch) sensor discovery message
AccousticPitchSensorValue	An acoustic sensor (Pitch) sensor value
AccousticVolumeSensor	A class to communicate with an acoustic sensor (Volume)
AccousticVolumeSensorDiscoveryMessage	An acoustic sensor (Volume) sensor discovery message
AccousticVolumeSensorValue	An acoustic sensor (Volume) sensor value
ActivateSensorMessage	This message activates a sensor
ActuatorDiscoveryMessage	This is the base class from which all actuator discovery messages must derive
ActuatorStartMessage	This message starts an actuator
ActuatorStopMessage	This message stops an actuator
BiologicalBodyTemperatureSensor	A class to communicate with a biological sensor (Body temperature)
BiologicalBodyTemperatureSensorDiscoveryMessage	A biological sensor (Body temperature) discovery message
BiologicalBodyTemperatureSensorValue	A biological sensor (Body temperature) value
BilogicalHeartRateSensor	A class to communicate with a biological sensor (Heart rate)
BilogicalHeartRateSensorDiscoveryMessage	A biological sensor (Heart rate) discovery message
BilogicalHeartRateSensorValue	A biological sensor (Heart rate) sensor value
BiologicalNeuralActivitySensor	A class to communicate with a biological sensor (Neural activity)
BiologicalNeuralActivitySensorDiscoveryMessage	A biological sensor (Neural activity) discovery message
BiologicalNeuralActivitySensorValue	A biological sensor (Neural activity) value.
BiologicalRespirationRateSensor	A class to communicate with a biological sensor (Respiration rate)
BiologicalRespirationRateSensorDiscoveryMessage	A biological sensor (Respiration rate) discovery message.
BiologicalRespirationRateSensorValue	A biological sensor (Respiration rate) value.
BTPANManager	This class manages a Bluetooth Personal Area Network of sensors and actuators (SENSACT based)
EnvironmentalHumiditySensor	A class to communicate with an environmental sensor (Humidity)

EnvironmentalHumiditySensorDiscoveryMessage	A environmental sensor (Humidity) discovery message
EnvironmentalHumiditySensorValue	A environmental sensor (Humidity) value
EnvironmentalTemperatureSensor	A class to communicate with an environmental sensor (Temperature)
EnvironmentalTemperatureSensorDiscoveryMessage	A environmental sensor (Temperature) discovery message
EnvironmentalTemperatureSensorValue	A environmental sensor (Temperature) value
MechanicalAccelerationSensor	A class to communicate with a mechanical sensor (Acceleration)
MechanicalAccelerationSensorDiscoveryMessage	A mechanical sensor (Acceleration) discovery message
MechanicalAccelerationSensorValue	A mechanical sensor (Acceleration) sensor value
MechanicalDisplacementSensor	A class to communicate with a mechanical sensor (Diplacemement)
MechanicalDisplacementSensorDiscoveryMessage	A mechanical sensor (Diplacemement) discovery message.
MechanicalDisplacementSensorValue	A mechanical sensor (Displacement) value
MechanicalForceActuator	A class to communicate with a mechanical actuator (Force)
MechanicalForceActuatorDiscoveryMessage	A mechanical actuator (Force) discovery message
MechanicalForceActuatorValue	A mechanical actuator (Force) value
MechanicalForceSensor	A class to communicate with a mechanical sensor (Force)
MechanicalForceSensorDiscoveryMessage	A mechanical sensor (Force) discovery message
MechanicalForceSensorValue	A mechanical sensor (Force) value
MechanicalMassSensor	A class to communicate with a mechanical sensor (Mass)
MechanicalMassSensorDiscoveryMessage	A mechanical sensor (Mass) discovery message
MechanicalMassSensorValue	A mechanical sensor (Mass) value
MechanicalPositionSensor	A class to communicate with a mechanical sensor (Position)
MechanicalPositionSensorDiscoveryMessage	A mechanical sensor (Position) discovery message
MechanicalPositionSensorValue	A mechanical sensor (Position) value
MechanicalShapeSensor	A class to communicate with a mechanical sensor (Shape)
MechanicalShapeSensorDiscoveryMessage	A mechanical sensor (Shape) discovery message

MechanicalShapeSensorValue	A mechanical sensor (Shape) value
OpticalBrightnessSensor	A class to communicate with a optical sensor (Brightness)
OpticalBrightnessSensorDiscoveryMessage	A optical sensor (Brightness) discovery message
OpticalBrightnessSensorValue	A optical sensor (Brightness) value
OpticalEmissivitySensor	A class to communicate with a optical sensor (Emissivity)
OpticalEmissivitySensorDiscoveryMessage	A optical sensor (Emissivity) discovery message
OpticalEmissivitySensorValue	A optical sensor (Emissivity) value
OpticalLightWaveFrequencySensor	A class to communicate with a optical sensor (Light wave frequency)
OpticalLightWaveFrequencySensorDiscoveryMessage	A optical sensor (Light wave frequency) discovery message
OpticalLightWaveFrequencySensorValue	A optical sensor (Light wave frequency) value
OpticalLuminanceSensor	A class to communicate with a optical sensor (Luminance)
OpticalLuminanceSensorDiscoveryMessage	A optical sensor (Luminance) discovery message
OpticalLuminanceSensorValue	A optical sensor (Luminance) value
OpticalRefractionSensor	A optical sensor (Refraction) discovery message
OpticalRefractionSensorDiscoveryMessage	A class to communicate with a optical sensor (Refraction) discovery message
OpticalRefractionSensorValue	A optical sensor (Refraction) value
SensorDiscoveryMessage	Base class for all sensor discovery messages
SensorStopMessage	This message stops a sensor
StatisticsLog	Statistics logger for testing and debugging

Table 32 - Classes of STFPAN package pt.uc.dei.lcst.stf.pan.comms.area

Class, interface or exception	Function
BluetoothChannel	A Bluetooth communication channel implemented using L2CAP connection
STCPChannel	A Secure TCP (SSL) Communications channel

Table 33 - Classes of STFPAN package pt.uc.dei.lcst.stf.pan.comms.link

Class, interface or exception	Function
SDP	Represents a full protocol string SDP, with possibly multiple session descriptions

SDPBandwidthInformation	Bandwidth information in SDP
SDPConnectionInformation	A SDP Connection Information Element
SDPMediaDescription	A SDP Media Description
SDPSessionDescription	Describes a session in SDP
SDPSessionRepeatTimes	Describes session repeat times
SDPSessionTime	Represents a SDP session time
SDPTimeDescription	A SDP Time Description
SDPException	An exception that is thrown when there is an error in interpreting SDP protocol strings

Table 34 - Classes of STFPAN package pt.uc.dei.lcst.stf.pan.comms.sdp

Class, interface or exception	Function
GenericChannel	A generic channel interface
GenericChannelListener	Generic channel listener interface
GenericChannelData	A generic channel data message
ChannelException	Base exception for all channel operations related STF channels
ChannelReadException	The channel could not be read
ChannelWriteException	The channel could not be written to

Table 35 - Classes of STFPAN package pt.uc.dei.lcst.pan.net

Class, interface or exception	Function
PersistentObject	A persistent object
PersistentNameArchive	Archive of names and correspondent ids of persistent object classes
PersistentObjectInputStream	A persistent object input stream
PersistentObjectOutputStream	A persistent object output stream

Table 36 - Classes of STFPAN package pt.uc.dei.lcst.stf.pan.persist

ANNEX IV – STFSERVER API INTERFACES AND CLASSES

The Table 37, Table 38, Table 39, Table 40, Table 41, Table 42, Table 43, Table 44, Table 45, Table 46, Table 47 and Table 48 show the classes and corresponding functions for each of the packages of the STF Server API.

Class, interface or exception	Function
STFApplication	Interface for STF supported applications
STFKey	Interface for the transmitted state key classes
STFLateJoin	This is the interface used by applications that want to late join the channel, that is, receive old recorded state transmission messages to rebuild the current state of the distributed application
STFMessager	Interface for the classes that can transmit STF messages
STFPartitioner	Interface for partitioning message handlers
STFState	Interface for the transmitted STF State classes
STFStatistics	Statistics collecting interface
DisableStatsUE	Message that disable statistics collecting in an UE
EnableStatsUE	Message that enables statistics collecting in an UE
ResetStatsUE	Message that resets statistics collecting in an UE
StartUEGameActivity	Start all game activity in a UE.
STFLateJoinBeginMessage	Message that starts late joining process.
STFLateJoinCheckpointMessage	Message that starts a checkpoint process.
STFLateJoinEndMessage	Message that ends a latejoin process.
STFLateJoinMessage	Base message of all latejoin messages
STFLateJoinPartMessage	A latejoin message (a retransmitted state message)
STFLateJoinRequestMessage	A latejoin request message.
STFLateJoinStreamMessage	A message indicating the latejoin streaming process is about to begin
STFLocalCache	A class that implements the local cache of messages
STFPartitionMessage	A Message indicating that a UE is going to start partitioning
STFProperties	The configuration properties of STF.
STFServerChannelManager	The STFServer media manager. Manages the media channel.

STFStateMessage	A STF state message. Includes key and state.
STFStatisticsData	Statistics data.
STFTime	A class implementing time sync.
STFTimeHeartBeatMessage	A time sync heart beat message
STFTimeMessage	The base message for all time sync messages.
STFTimePingMessage	A time net sync ping message.
STFTimePongMessage	A time net sync pong message
STFTimeVoteDecidedMessage	A time sync distributed vote decided message.
STFTimeVoteMessage	A time sync distributed vote message
STFTimeVoteRequestMessage	A time sync distributed vote request message
StopUEGameActivity	Stops all game activity in a UE.
UEConnectionPool	A UE connection pool.

Table 37 - STFSERVER classes of package pt.uc.dei.lcst.stf

Class, interface or exception	Function
HSS	A partially emulated Home Subscriber Server (HSS) for testing purposes.
RoutingTableEntry	A routing table entry for the HSS Diameter server.
SLF	A partially emulated SLF (Subscription Locator Function) for testing.
STFCentralGameAppServer	A base central game application server.
STFGameAppServer	A base distributed game application server

Table 38 - STFSERVER classes of package pt.uc.dei.lcst.stf.applications

Class, interface or exception	Function
STCPChannel	A SSL/TLS (over TCP) channel of communications
STFManagerApplication	A STF manager application
STFManagerMainForm	The STF manager application’s main form.

Table 39 - STFSERVER classes of package pt.uc.dei.lcst.stf.applications.manager

Class, interface or exception	Function
AggregateDSTStatsMessage	Aggregate distributed server statistics message.
AggregateDSTUEStatsMessage	Aggregate distributed statistics of UEs message

AggregateGlobalStatsMessage	Aggregate global statistics message
AllowARMessage	Allow AR message
AllServerNames	All server names message
AskForAggregateDistServerStatsMessage	Ask for aggregate distributed server statistics message
AskForAggregateDistServerUEStatsMessage	Ask for aggregate distributed statistics of UEs message
AskForAggregateGlobalStatsMessage	Ask for Aggregate global statistics message
AskForAllServerNames	Ask for all server names message
AskForCentralServerStats	Ask for central server statistics message
AskForServerAddressMessage	Ask for server address message
CentralServerStats	Central server statistics message
CheckAuthorized	Check if authorized message
CheckAuthorizedResult	Check if authorized result message
CheckRunning	Check if running on distributed server
CheckRunningResult	Check if running result
DisableStatistics	Disable statistics message
EnableStatistics	Enable statistics message
LocatedServer	Located server message
LocateServer	Locate server message
RemoveUserRunning	Remove user running on distributed server
ReservationCompletedMessage	Reservation completed message
ResetStatistics	Reset statistics message
ServerAddressMessage	Server address message
ServerNotifyMessage	Server notify message
StartGameActivity	Start all game activity
StartReservationMessage	Start the reservation message
STFStatisticsReportedMessage	STF Statistics reported message
STFStatisticsReportedUEMessage	STF statistics reported for a UE message
STFStatisticsReportMessage	STF statistics report message
STFUEStatisticsReportMessage	STF UE statistics report message
StopGameActivity	Stop all game activity

TerminateManagerCommunications	Terminate manager communications
UserAuth	User authorization message
UserAuthError	User authorization error message
UserRevokeAuth	User revoke authorization message
UserRevokeAuthError	User revoke authorization error message

Table 40 - STFSERVER classes of package pt.uc.dei.lcst.stf.applications.messages

Class, interface or exception	Function
COSEVChannel	A ARMS normal Corba Event Service Channel of communications
COSEVChannelData	Represents generic channel data representation for transmission in ARMS interface related channels of communications.
MulticastCOSEVChannel	An ARMS Sixrm based multicast corba event service channel of communications.
MulticastToStandardCOSEVChannel	An ARMS multicast to standard corba event service channel of communications.
SMulticastCOSEVChannel	A secure multicast ARMS Sixrm based event channel
StandardToMulticastCOSEVChannel	An ARMS standard to multicast corba event service channel of communications.

Table 41 - STFSERVER classes of package pt.uc.dei.lcst.stf.arms

Class, interface or exception	Function
DiameterApplication	The interface that applications that want to receive Diameter messages have to implement.
AccountingRealtimeRequiredAVP	Accounting real time required Attribute Value Pair
AccoutingRecordNumberAVP	Accounting record number Attribute Value Pair
AccountingRecordTypeAVP	Accounting record type Attribute Value Pair
AccountingSubSessionIDAVP	Accounting sub session ID Attribute Value Pair
AcctApplicationIDAVP	Accounting application ID Attribute Value Pair
AcctInterimIntervalAVP	Accounting interim interval Attribute Value Pair
AcctMultiSessionIDAVP	Accounting multi session ID Attribute Value Pair
AcctSessionIDAVP	Accounting session ID Attribute Value Pair
AuthApplicationIDAVP	Authorization application ID Attribute Value Pair
AuthGracePeriodAVP	Authorization grace period Attribute Value Pair

AuthorizationLifetimeAVP	Authorization lifetime Attribute Value Pair
AuthRequestTypeAVP	Authorization request type Attribute Value Pair
AuthSessionStateAVP	Authorization session state Attribute Value Pair
AVP	Represents a base attribute value pair (AVP).
AVPHeader	A Attribute Value Pair header
AVPResolver	Resolves AVPs for the diameter base protocol
AVPSequence	Represents a sequence of Attribute Value Pairs
ClassAVP	Class Attribute Value Pair
DestinationHostAVP	Destination host Attribute Value Pair
DestinationRealmAVP	Destination realm Attribute Value Pair
DiameterACA	Accounting answer
DiameterACR	Accounting request
DiameterASA	Abort session answer
DiameterASR	Abort session Request
DiameterBaseResolver	Resolves Diameter Messages for the diameter base protocol.
DiameterCEA	Capabilities Exchange Answer (CEA)
DiameterCER	Capabilities Exchange Request (CER)
DiameterCommand	Represents a Diameter Command
DiameterDPA	Disconnect Peer Answer
DiameterDPR	Disconnect Peer Request
DiameterDWA	Device Watchdog Answer
DiameterDWR	Device Watchdog Request
DiameterERROR	Diameter Error
DiameterHeader	A diameter header
DiameterManager	The Diameter manager. Manages the diameter protocol for a diameter application.
DiameterPeer	Represents a diameter peer.
DiameterRAA	Re-authorization answer
DiameterRAR	Re-authorization Request
DiameterSTA	Session Termination answer
DiameterSTR	Session Termination Request

DisconnectCauseAVP	Disconnect cause Attribute Value Pair
E2ESequenceAVP	End to End sequence Attribute Value Pair
ErrorMessageAVP	Error message Attribute Value Pair
ErrorReportingHostAVP	Error reporting host Attribute Value Pair
EventTimeStampAVP	Event timestamp Attribute Value Pair
ExperimentalResultAVP	Experimental result Attribute Value Pair
ExperimentalResultCodeAVP	Experimental result code Attribute Value Pair
FailedAVP	Failed Attribute Value Pair
FirmwareRevisionAVP	Firmware revision Attribute Value Pair
HostIpAddressAVP	Host IP Address Attribute Value Pair
InBandSecurityIDAVP	In-band security ID Attribute Value Pair
MultiRoundTimeOutAVP	Multi-round timeout Attribute Value Pair
OriginHostAVP	Origin Host Attribute Value Pair
OriginRealmAVP	Origin Realm Attribute Value Pair
OriginStateIDAVP	Origin state ID Attribute Value Pair
ProductNameAVP	Product Name Attribute Value Pair
ProxyHostAVP	Proxy host Attribute Value Pair
ProxyInfoAVP	Proxy information Attribute Value Pair
ProxyStateAVP	Proxy state Attribute Value Pair
RealmRoutingTableEntry	Realm routing table entry
ReAuthRequestTypeAVP	Re-authorization request type Attribute Value Pair

RedirectHostAVP	Redirect host Attribute Value Pair
RedirectHostUsageAVP	Redirect host usage Attribute Value Pair
RedirectMaxCacheTimeAVP	Redirect maximum cache time Attribute Value Pair
ResultCodeAVP	Result code Attribute Value Pair
RouteRecordAVP	Route record Attribute Value Pair
SessionBindingAVP	Session binding Attribute Value Pair
SessionIDAVP	Session id Attribute Value Pair
SessionServerFailoverAVP	Session server failover Attribute Value Pair
SessionTimeoutAVP	Session timeout Attribute Value Pair
SupportedVendorIDAVP	Supported Vendor ID Attribute Value Pair
TerminationCauseAVP	Termination cause Attribute Value Pair
UserNameAVP	User Name Attribute Value Pair
VendorIDAVP	Vendor ID Attribute Value Pair
VendorSpecificApplicationIDAVP	Vendor Specific Application ID Attribute Value Pair
DiameterException	A Diameter Exception

Table 42 - STFSERVER classes of package pt.uc.dei.lcst.stf.diameter

Class, interface or exception	Function
AsociatedIdentitiesAVP	Associated identities Attribute Value Pair
ChargingInformationAVP	Charging information Attribute Value Pair
ConfidentialityKeyAVP	Confidentiality key Attribute Value Pair
CXDXAVPResolver	A AVP resolver for the CXDX diameter application
CXDXAVPSequence	A CDDX diameter application AVP sequence
CXDXExperimentalResultAVP	A CXDX Experimental result Attribute Value Pair

DeregistrationReasonAVP	De-registration reason Attribute Value Pair
DiameterCXDXResolver	A CXDX diameter application command message resolver
FeatureListAVP	Feature list Attribute Value Pair
FeatureListIDAVP	Feature list id Attribute Value Pair
IntegrityKeyAVP	Integrity key Attribute Value Pair
LocationInfoAnswerCommand	Location Information Answer (LIA) command
LocationInfoRequestCommand	Location Information Request (LIR) command
MandatoryCapabilityAVP	Mandatory capability Attribute Value Pair
MultimediaAuthAnswerCommand	Multimedia Authorization Answer (MAA) command
MultimediaAuthRequestCommand	Multimedia Authorization Request (MAR) command
OptionalCapabilityAVP	Optional capability Attribute Value Pair
PrimaryChargingCollectionFunctionNameAVP	Primary charging collection function name Attribute Value Pair
PrimaryEventChargingFunctionNameAVP	Primary event charging function name Attribute Value Pair
PublicIdentityAVP	Public identity Attribute Value Pair
PushProfileAnswerCommand	Push Profile Answer (PPA) command
PushProfileRequestCommand	Push Profile Request(PPR) command
ReasonCodeAVP	Reason code Attribute Value Pair
ReasonInfoAVP	Reason information Attribute Value Pair
RegistrationTerminationAnswerCommand	Registration Termination Answer (RTA) command
RegistrationTerminationRequestCommand	Registration Termination Request (RTR) command
SecondaryChargingCollectionFunctionNameAVP	Secondary charging collection function name Attribute Value Pair
SecondaryEventChargingFunctionNameAVP	Secondary event charging function name Attribute Value Pair
ServerAssignmentAnswerCommand	Server Assignment Answer (SAA) command
ServerAssignmentRequestCommand	Server Assignment Request (SAR) command
ServerAssignmentTypeAVP	Server assignment type Attribute Value Pair
ServerCapabilitiesAVP	Server capabilities Attribute Value Pair
ServerNameAVP	Server name Attribute Value Pair

SIPAuthDataItemAVP	SIP authorization data item Attribute Value Pair
SIPAuthenthicateAVP	SIP authenticate Attribute Value Pair
SIPAuthenthicationContextAVP	SIP authentication context Attribute Value Pair
SIPAuthenthicationSchemeAVP	SIP authentication scheme Attribute Value Pair
SIPAuthorizationAVP	SIP authorization Attribute Value Pair
SIPItemNumberAVP	SIP item number Attribute Value Pair
SIPNumberAuthItemsAVP	SIP Number of Authorization Items Attribute Value Pair
SupportedApplicationsAVP	Supported applications Attribute Value Pair
SupportedFeaturesAVP	Supported features Attribute Value Pair
UserAuthorizationAnswerCommand	User authorization answer (UAA) command
UserAuthorizationRequestCommand	User authorization request (UAR) command
UserAuthorizationTypeAVP	User authorization type Attribute Value Pair
UserDataAlreadyAvailableAVP	User data already available Attribute Value Pair
UserDataAVP	User data Attribute Value Pair
VisitedNetworkIdentifierAVP	Visited network identifier Attribute Value Pair

Table 43 - STFSERVER classes of package pt.uc.dei.lcst.stf.diameter.cxdx

Class, interface or exception	Function
CurrentLocationAVP	Current location Attribute Value Pair
DataReferenceAVP	Data reference Attribute Value Pair
DiameterSHResolver	A SH Diameter application command message resolver
ExpiryTimeAVP	Expiry time Attribute Value Pair
IdentitySetAVP	Identity set Attribute Value Pair
MSISDNAVP	MSISDN Attribute Value Pair
ProfileUpdateAnswerCommand	Profile update answer (PUA) command
ProfileUpdateRequestCommand	Profile update request (PUR) command
PushNotificationRequestCommand	Push notifications request (PNR) command
PushNotificationsAnswerCommand	Push notifications answer (PAN) command
RequestedDomainAVP	Request domain Attribute Value Pair
ServiceIndicationAVP	Service indication Attribute Value Pair
SHAVPResolver	A SH diameter application AVP resolver

SHAVPSequence	A SH diameter application AVP sequence
SHExperimentalResultAVP	SH Experimental result Attribute Value Pair
SubscribeNotificationsAnswerCommand	Subscribe notifications answer (SNA) command
SubscribeNotificationsRequestCommand	Subscribe notifications request (SNR) command
SubsReqTypeAVP	Subscribe notifications request type Attribute Value Pair
UserDataAnswerCommand	User data answer (UDA) command
UserDataAVP	User data Attribute Value Pair
UserDataRequestCommand	User data request (UDR) command
UserIdentityAVP	User identity Attribute Value Pair

Table 44 - STFSERVER classes of package pt.uc.dei.lcst.diameter.sh

Class, interface or exception	Function
GenericChannel	A generic channel interface
GenericChannelListener	Generic channel listener interface
GenericChannelData	A generic channel data message
ChannelException	Base exception for all channel operations related STF channels
ChannelReadException	The channel could not be read
ChannelWriteException	The channel could not be writed to

Table 45 – STFSERVER classes of package pt.uc.dei.lcst.stf.net

Class, interface or exception	Function
PersistentObject	A persistent object
PersistentNameArchive	Archive of names and correspondent ids of persistent object classes
PersistentObjectInputStream	A persistent object input stream
PersistentObjectOutputStream	A persistent object output stream

Table 46 - STFSERVER classes of package pt.uc.dei.lcst.stf.persist

Class, interface or exception	Function
SCTPSocketOptions	Represents SCTP protocol socket options
SCTPInputStream	A sctp input stream

SCTPOutputStream	A sctp output stream
SCTPServerSocket	A sctp server socket
SCTPSocket	A sctp socket
SCTPSocketImpl	A sctp socket implementation
SCTPSocketImplFactory	A sctp socket implementation factory

Table 47 - STFSERVER classes of package pt.uc.dei.lcst.stf.sctp

Class, interface or exception	Function
ConnectionVerifiyer	A connection verifier (verifies if authorized).
STCPChannel	A SSL/TLS (over TCP) channel

Table 48 - STFSERVER classes of package pt.uc.dei.lcst.stf.uelink