JXTA, In support of the Migration of Users Across Smart Spaces.

Martin Commins
Department of Computer Science,
Trinity College Dublin.

A dissertation submitted to the University of Dublin,
in partial fulfilment of the requirements for the degree of
Master of Science in Computer Science.

September 2002

Declaration

I declare that the work described in this dissertation is, except otherwise where stated, entirely my own and has not been submitted as an exercise for a degree at this or any other university.

Signed: _____

Martin Commins

16th September 2002.

Permission to lend/and or copy

I agree that Trinity College Library may lend or copy this dissertation upon request.

Signed: _____

Martin Commins

16th September 2002.

Acknowledgements

I would like to thank my supervisor, Mr. Declan O'Sullivan, for his guidance and help throughout the year. His wealth of knowledge and ideas were of enormous contribution to the completion of this project.

Thanks to the MSc NDS class of 2002.

Thanks to Karolina for her motivation and support throughout the last year.

Last but certainly not least, thanks to my parents, Denis and Julie, for their support and encouragement throughout all of my college years.

Contents

List of Figures

Abstract

"Smart Spaces" are environments with traditional computing hardware as well as embedded computers, information appliances, and multi-modal sensors. The goal of a smart space is to sense, recognise and understand the tasks that are being undertaken by users of the space, and respond in a way that brings tangible benefits to the users in the execution of those tasks. In the area of Smart Space research, information about the user is generally encapsulated in what is called "context information". However, recent research into Smart Spaces has primarily focussed on isolated environments where user tasks are limited to within a single Smart Space. Little research has been undertaken into what support is needed for users that migrate between Smart Space environments.

This project describes the development of *MIGRA*, a system that allows for the dynamic interconnection and efficient exchange of user context information between Smart Space environments. A key challenge for *MIGRA* is the management of context information to ensure it reflects the inherent diversity of Smart Space users.
A core aspect to the implementation of *MIGRA* described in this writing is the investigation into the applicability of using the peer-to-peer JXTA software infrastructure to support user migration.

Chapter 1

Introduction

1.1 Project Aims

As computing devices become the primary tool of communication at home and in the work place there is an inherent demand by users to avail of these devices ubiquitously in order to support their daily lives. This scenario personifies the concept known as "Nomadic Computing", where users have the ability to execute tasks anywhere at anytime [CDK01].

Rosenthal [RS00] defines a Smart Space as environments with traditional computing hardware as well as embedded computers, information appliances and multi-modal sensors allowing people to perform their tasks efficiently. As users move across environments they will be permitted access to a multitude of devices offering a range of services. A typical scenario is where a user who begins a web browsing session on the laptop in their office, continues browsing on their way home on the train via a Personal Digital Assistant (PDA) and then when entering the home environment the desktop computer continues the browsing session. These environments, the office, train and home, are known as Smart Spaces.

To achieve seamless migration of a user between Smart Space locations communication between the environments is fundamental. This task is further complicated by the intermittent, multitasking nature of Smart Space users who can have a multitude of tasks executing anywhere at anytime. As Smart Spaces will be interconnected, a major challenge is to ensure freedom of movement of participants into and out of the Smart Space regardless of the task(s) they are executing. This requires enabling the seamless transfer of their services and their devices between smart spaces. In order to enact a

handover of a Smart Space user session, information needs to be exchanged between the Smart Spaces about the services available in the intended destination smart space, and the visitor's usage of services within their current smart space. This information is needed so that the destination smart space can initiate or adapt services and resources within its space so that the user's tasks underway are seamlessly supported upon entry to the space.

A system that supports the migration of users between Smart Spaces raises significant issues. The purpose of this writing is to present an architecture, called *MIGRA*, to facilitate this migration of users and outline the inherent issues that arise from such an architecture, including how to eradicate the need for the user to reconfigure local devices every time they move location.

Implementation of *MIGRA* is based on Sun Microsystems [SUN] peer-to-peer networking technology called JXTA [JXTA]. A focus of this writing also investigates JXTAs applicability as an underlying infrastructure upon which Smart Space systems can be based.

1.2 Project Objectives

The principal objectives for the *MIGRA* system are outlined as follows:

- Allow the migration of a user from one Smart Space to another by simulating various Smart Space environments.
- Simulate the detection of users entering a Smart Space.
- Generate, store, process and migrate the users context.

By investigating related research projects in the area of user migration between Smart Spaces this writing proposes several requirements that such a system should support, in order to achieve the objectives stated.

1.3 Dissertation outline

Chapter 2, State of the Art, outlines related research projects in the area of user movement across Smart Spaces. This chapter also includes a description of distributed programming approaches that might support user migration, such as distributed shared memory and peer-to-peer.

Chapter 3 provides a detailed introduction to JXTA. This chapter presents the architecture of the technology, the main components and also the protocols that JXTA implements. Also described in this chapter is an example application, myJXTA, which is entirely based on JXTA.

Chapter 4 presents the requirements, architecture and design of the *MIGRA* system. The requirements are derived from the research in chapter 2. Chapter 4 also outlines the impact of the JXTA technology described in chapter 3 upon the design of *MIGRA*.

Chapter 5 examines the implementation of the *MIGRA* design presented in chapter 4. This includes a discussion on the decisions that were made and difficulties encountered during the implementation phase of the project.

Finally, chapter 6 summarises the conclusions of this work and suggests areas of future work.

Chapter 2

State of the Art

2.1 Introduction

Smart Space research has primarily focused on individual environments where services are limited to within a single boundary. The purpose of section 2.1 is to examine the state of the art of the few projects extant that deal with migration between smart space environments. The remaining sections of the chapter examine the state of the art of promising distributed programming approaches that were investigated that might support smart space user migration. Section 2.3 focuses on distributed shared memory approaches and section 2.4 on peer-to-peer networking.

2.2 Domain Case Studies

Little research has addressed ubiquitous users that move between Smart Space environments. To access services within a single domain users transparently interact with an ensemble of multi-modal sensors all managed by an underlying framework. These frameworks typically are isolated from each other thus hindering intermittent movement between these locations. The following section describes projects that address this issue of user migration across multiple Smart Spaces.

2.2.1 Cooltown

Cooltown [CoolT] is a Hewlett Packard (HP) [HP] open research project that aims to accommodate a complete smart space solution by providing all entities with a web presence, where entities are defined as people, places and objects. This means each entity has its own URL (Uniform Resource Locator). This project is born out of a vision [Bel]

of pervasive, ubiquitous, anywhere computing that will facilitate a wide variety of ubiquitous computing scenarios from emergency services to real-time navigation.

Cooltown is a web and content-based solution that leverages existing standard Internet technologies as its underlying framework such as HTTP, MIME and URL. Content-based infers that the users context is advertised in conjunction with the URL. This is achieved by means of cookies [COOK]. User context is described by three parameters: identity, location, and browsing capability. The adoption of a web-based approach addresses the inherent complications coupled with ubiquitous computing such as language, OS and hardware dependencies.

Cooltown consists of several hardware and software modules that implement an architecture that links web presences to people, places and objects. "*Beacons*" are inconspicuously placed hardware devices that transmit the URLs to handheld devices such as a mobile phone or PDA. The primary idea is that users can avail of the multitude of information relating to entities by simply sending a URL pointer to a beacon via an infrared connection.

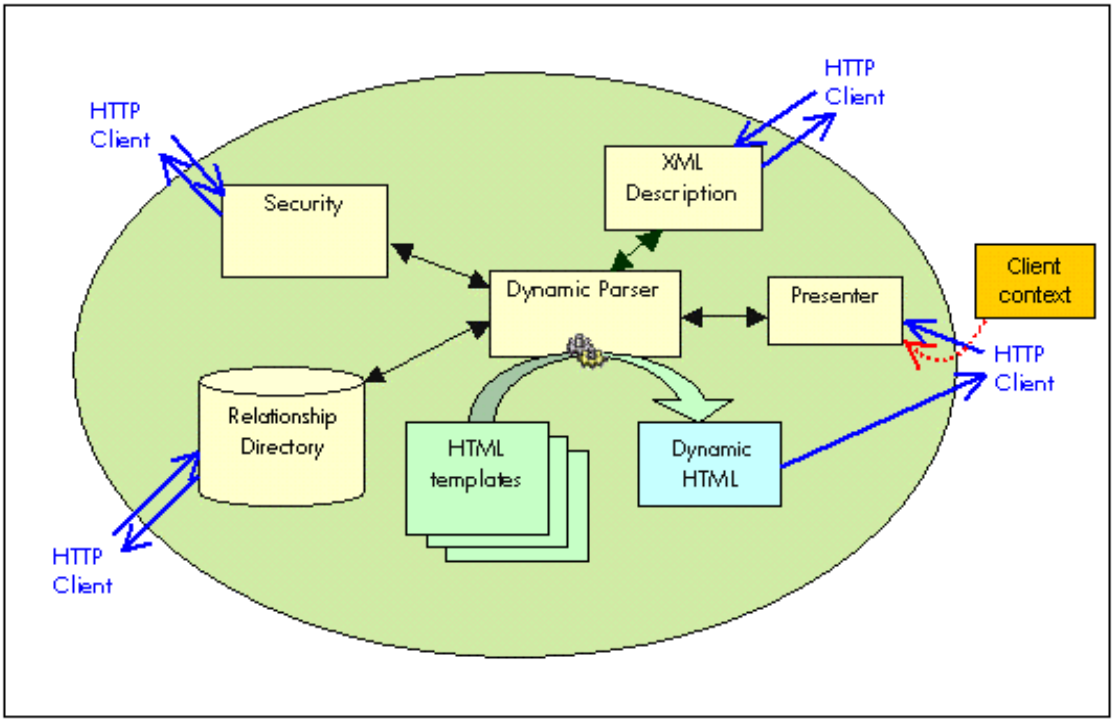


Figure 2.1: Cooltown Architecture.

Figure 2.1 depicts Cooltown's architecture [HPDC]. *XML Description* describes the raw information of entities that can be attained by other entities. The *Relationship Directory* is a repository that records the relationships between identities. An example of such a relation is a person that is accessing the services in a particular place. The person is said to be a '*resource'* of the place. Relationships need not always be formed between entities. For example, in the case where one entity just wants information relating to the other then all that needs to be accessed is the URL for that entity.

Users migrating from one location entity, i.e. smart space, to another can register as a resource with the new entity to avail of the services offered.
Debaty [HPDC] describes the process of migration between entities in the following way:
The person's Web presence needs to be registered in the Relationship Directory of the place. This is done using the URL for the register method of the place's (Smart Space) directory, which simply adds the person's Web presence in the place's directory.
The person might also want the room he is in to appear in his Web presence to let other people know where he is and how to contact him. Therefore he might want to register the room's Web presence in the Relationship Directory of his own Web presence.
The calling of the registration method can be either manual or automatic. Automatic execution personifies seamless migration while roaming between Cooltown Smart Spaces.

Coolbase is the open source framework offering from HP that was released under the General Public Licence [GPL] in an effort to broaden the community of Cooltown developers and ensure its interoperability.

2.2.2 MUSE

MUSE [MUSE] is a collaborative research project developed at UCLA designed to create middleware architecture for sensor smart spaces. MUSE is a realisation of the need for sensor smart spaces that exhibit a contextual understanding of the dynamic events

occurring in smart space environments and the ability to execute intelligent responses to such events.

The goal of MUSE is to provide an infrastructure that facilitates real-time capture, storage, processing display and analysis of the information generated. The MUSE infrastructure is written entirely in Java [JAVA].

Each smart space is populated with sensing devices and contextual data providers. These are referred to as *services*. A *community* is defined by the set of services it contains and its members are diverse as they can join/leave the community at any time. MUSE implements Jini [JINI] as its connectivity layer. Jini provides a means to track the diversity of the members in a particular community through its discovery and service interface mechanisms. JavaSpaces [JSPACE] technology, discussed in section 2.3, is also incorporated to provide a basis for shared communication pool for resources sharing [ACKMM00]. Smart Space environment history is maintained through a distributed XML (eXtended Markup Language) based memory element known as MIRA (Multimedia Internet Recorder and Archive) [MIRA]. MIRA is a subproject of MUSE being developed to add a memory component to sensor Smart Spaces.

Jini provides the underlying framework for each Smart Space, however, migration between such Jini federations has proved to be a stumbling block in the development of MUSE [GLJW00]. It has been addressed by the developers that there should be a means to track the use of services automatically as users move from one federation to another. One proposed solution is the incorporation of the distributed sensors and a shared memory environment at the base of the infrastructure. Lawton [GLJW00] suggests the sensors would be quantified in terms of energy cost and accuracy, and a Bayesian probabilistic model would be used to interpret the sensor results.

2.2.3 EasyLiving

EasyLiving [EASY] is a Microsoft research project aimed at the development of architectural infrastructure and technologies for intelligent environments. The purpose of the EasyLiving project is to develop an infrastructure capable of managing the multitude of heterogeneous applications in a smart space to result in a coherent user experience. The focus of this endeavour is moved from message passing paradigms to focus more on the shared representation of the intelligent environment. In his paper, Shafer [SMB00] describes the primary models under focus in EasyLiving as:

- The network information and current status of each connected device.
- The people known to be in the world and what is known about each person.
- The geometric relationships between people, places and things.

InConcert [InCon] is a Microsoft built middleware that is implemented instead of Java or CORBA [CORBA]. Brumitt describes the reasons for the implementation of InConcert instead of mainstream technologies in his paper [MKKS00]. The core of this discussion advocates that InConcert addresses the complications that are coupled with multi-threaded programming, InConcert is more efficient than pipelining message passing and machine addressing mechanisms provided by InConcert are independent of the machine. The conclusion is that, InConcert is a platform that allows asynchronous message passing, machine independent addressing and XML-based message protocols.

User movement is monitored through a myriad of sensors and cameras. Tracking of the user, in one smart space, is seamlessly executed by passing the task to the vision module in the perceived location that the user is entering. This is computed through complex computations based on current fields of view and heuristics of movement. All this information provides geometric knowledge, i.e. information about the physical relationships between the entities in the environment. This geometric representation is modelled on user interface devices. However, this movement is limited to only one geometric model. EasyLiving has, as of yet, found no means by which services can span

boundaries between spaces. Representation in one geometric model is deemed insufficient.

Migration of services, robust lookup services that support discovery and presentation of available resources in an understandable fashion to users are all areas of future work. Currently, EasyLiving is limited to a single room consisting of ten devices that can be configured dynamically. One to three users can avail of the EasyLiving space.

2.2.4 VIA

VIA (Verified Information Access) [DADS01] is a data centric application-level protocol analogous to common Internet peer-to-peer file sharing applications. VIA addresses user migration and data access between service discovery domains[1], that enforces cellular topology where data is inaccessible between cells. Through use of application-defined data schema, clusters organize themselves into a hierarchic structure that lends to efficient querying and leveraging of the resources placed on the network. XML is employed to describe the data that is produced by applications. This is called *metadata tags*.

Castro [DADS01] describes a scenario where a user roaming a campus taking photographs at will and storing them at various storage repositories. VIA would then process any queries relating to any of the photographs through filtering of the metadata tags.

VIA topology consists of gateways that act as mediators between ubiquitous devices in service discovery domains. This is illustrated in figure 2.2, taken from [DADS01]. There exists a main channel that broadcasts to all connecting gateways. All queries are directed through the main channel. VIA enables gateways to dynamically self-organize into

[1] Service Discovery Domain: A collection of services that are managed under the same umbrella by the same infrastructure. A typical example is a single coherent Smart Space environment that interacts with no other environments.

hierarchic structure based on the types of queries stemming from the main channel. This process reduces the number of irrelevant queries that could potentially swamp the network. Gateways then join clusters of other gateways that have similar data.

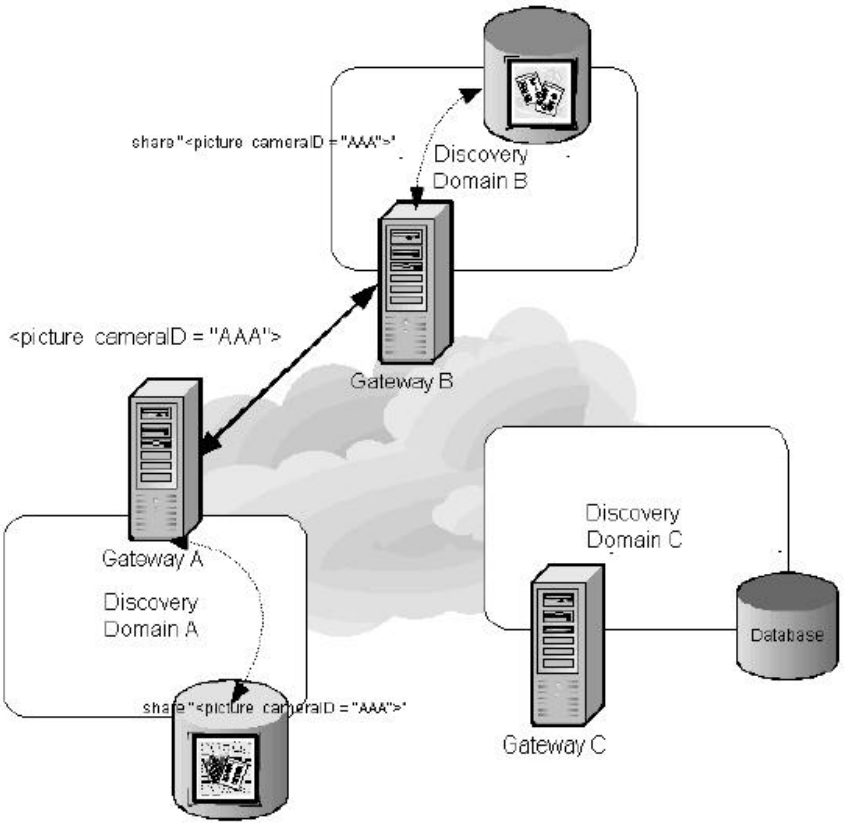


Figure 2.2: Discovery Domain Topology.

Smart Space migration is handled by VIA but there are indications it is not suited to real-time applications such as audio or video streaming.

2.2.5 The Aware Home

Context-aware application development is mainly restricted to controlled environments. One of the Aware Home [GATECH] goals is to create an infrastructure to aid in the rapid development of context-aware applications for un-controlled situations that occur in everyday living. The Aware Home is a living laboratory that focuses on ubiquitous computing for everyday activities and resides at Georgia Tech University. The home

consists of two independent living spaces and a shared basement where the control room for centralised computing services exists. The house is comprised of vision sensors that track the inhabitant's motions. Another approach to track user movement that is being investigated is the implementation of a Smart Floor [SMFL]. The smart floor interface is capable of identifying an occupant and tracking their movement. The identity would be computed in relation to the occupant's mass and the impact of their mass on the floor interface.

Ubiquitous sensing is not the only mode of interaction possible with the home. Rhodes [RMW99] argues that through wearable computers occupants can interact in a personalized manner. The main role such devices will play in the Aware Home is to draw on the house's data resources to cache important information. Research into movement in the home is still in its infancy.

2.2.6 Overview

The aforementioned case studies remain to the forefront of research as possible solutions to track occupant movement within the confines of Smart Space domains. Cooltown's web based approach is novel in the sense that it is the only project reviewed that is aiming to track users globally by utilising existing standard Internet technologies as its underlying framework. The other projects described have limited their scope to within the confines of building or, in the case of VIA, a university campus.

On the other hand, MUSE utilises existing distributed system technologies such as Jini and JavaSpaces to provide a global address space where user information is stored. However, this reliance on Jini and JavaSpaces would be undesirable to Smart Space developers aiming to provide the capability of user migration across domains, as communication via Jini and JavaSpaces outside the local network is difficult and inefficient. Perhaps the evolution of JXTA, described in chapter 3, will prove to be a viable alternative to MUSE, to achieve migration across domains. Another aspect of interest related to MUSE is the ability to store Smart Space history. This has the potential

to further enhance users experience within Smart Spaces by transparently setting the environments to suit user preferences.
VIA is the only technology that aims to employ a P2P topology. All of the other architectures incorporate the more traditional client/server approach. The results of this implementation are not yet defined. An interesting outcome of VIA's implementation could be the viability of P2P in user migration and if such an approach would be preferable to the client/server approach.

In summary, in all of the reviewed projects the fundamental goal is to provide an enhanced and coherent user experience.
The next section introduces some distributed programming technologies, some of which are implemented above, which could aid in the movement across Smart Spaces.

2.3 Distributed Shared Memory

A key challenge for pervasive computing environments is the management of user context in the face of constant change. Distributed shared memory is an abstraction that could be incorporated to meet the challenging needs of Smart Space environments to provide a unified virtual address space for storage of user context that would be accessible to all Smart Space processes on the network.

Distributed Shared Memory is an abstraction for sharing data between processes in computers that do not share physical memory [CDK01]. A layer of software simulates a logically shared memory space using a set of physically distributed local memory systems. Application processes operating at different sites have access to the same virtual address space and can cooperate by reading and writing values to the objects contained in the space, as illustrated in figure 2.3.

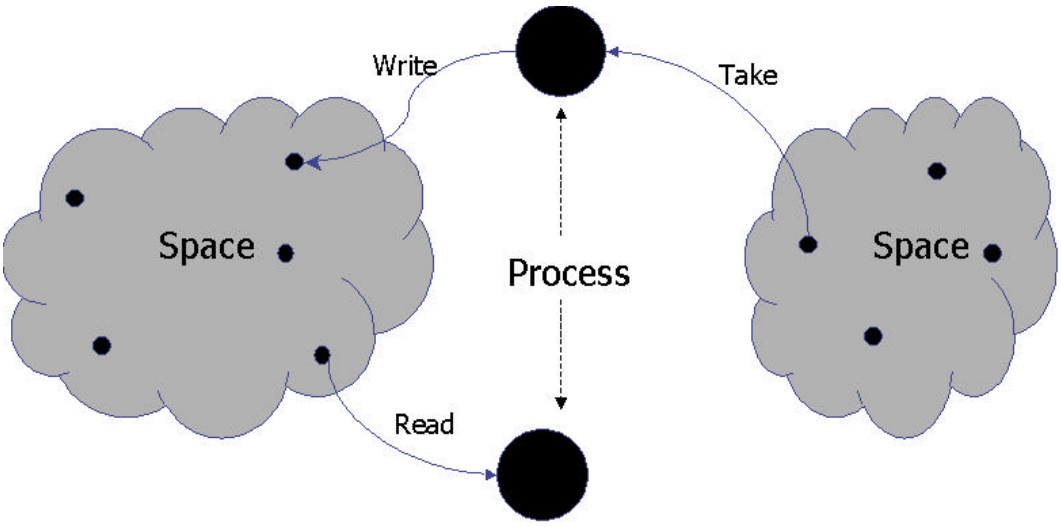


Figure 2.3: Distributed Share Spaces.

Both JavaSpaces [JSPACE] and TSpaces [TSPACE] incorporate the distributed shared memory model and are derivatives of the tuplespace system *Linda* [LIND]. Linda was originally developed to be a global communication buffer for parallel processing systems at Yale University. Tuple-based systems are based on the *"shared blackboard"* approach, where all clients can see tuples posted by others by issuing specific queries. Both JavaSpaces and IBM's T Spaces have inherited Linda's tuplespace model that facilitates a persistent communication buffer between servers across different platforms and are both essentially the same in functionality.

TSpaces and JavaSpaces are Java-based distributed object architectures and include development platforms, processing environments and mechanisms for addressing stored objects. Although Java-based, neither is compatible with the other. JavaSpaces and TSpaces provide a reliable repository source for individual Smart Spaces to store all contextual information relating to its entities. Examples of such entities could be user location and identity. The following sections offer a brief description of the principle functionalities involved in both JavaSpaces and TSpaces.

JavaSpaces

JavaSpaces address the need for distributed applications to store and share data and objects, through a means of a universal shared space. JavaSpaces are a service built atop of the Jini infrastructure and therefore require the implementation of Jini. This means the JavaSpace services can be located dynamically via Jini's lookup feature and can also be accessed by any entity in a Jini federation.
Edwards [CJED01] stipulates that the goal of JavaSpaces is to provide a centralised storage repository for Java objects. JavaSpaces provide a mechanism through which Jini services and clients can manipulate these stored Java objects.

JavaSpaces are both an API (Application Programming Interface) and a distributed programming model that allow a heterogeneous collection of computers to co-operate and offer space operations, distribution events, leases and transactions [HOMES].
Data is persistent, which means objects and data will survive even after process termination or until the lease on the data expires. Concurrent access to the shared data is built into the JavaSpace programming model. For object retrieval and manipulation, naming and location of the object is not essential. JavaSpaces are associative so objects can be located based on their content through a means of template matching techniques. The result of such content matching would return all relevant objects.

JavaSpaces provide a reliable repository source for individual Smart Spaces to store all contextual information relating to its entities. MUSE is one example of a Smart Space environment that employs the JavaSpaces programming model as a storage repository.

T Spaces

TSpaces is the middleware component that represents the core of the ubiquitous computing project at IBM [LMW99]. Similarly to JavaSpaces, IBM's TSpaces is a network middleware that provides a globally shared and associatively addressed memory

space capable of connecting heterogeneous systems into one common computing platform. That is, TSpaces allows devices to locate other devices and communicate, regardless of hardware, OS or location. Although Java-based, TSpaces is independent of the Jini infrastructure. TSpaces middleware caters for different application needs, depending on the client applications. [IBMUG] outlines the different facets of TSpaces technology as:

- A global persistent communication buffer.
- A lightweight database.
- A queue manager.
- A dynamic computation engine.

TSpaces are entirely written in Java and incorporate tuplespace terminology from project Linda. Tuplespaces are essentially an implementation of the Java vector class and contain name/value pairs. Methods of the tuplespace instantiations allow for manipulation on the individual tuples in the shared space. *Write, read, take* and *delete* are a few of the possible operations.

Each TSpace consists of a server, which can be run anywhere, and a client application that makes calls to the server. Clients read and write data to/from the server. Although TSpaces implement a lightweight database, there is no need for SQL statements. TSpaces provides its own simple query language.

At the time of writing, there existed no implementation of TSpaces in Smart Space environments. From the above reviews indications are that the distributed shared memory abstraction to distributed computing has a role to play in the migration of users across Smart Spaces. The implementation of distributed shared memory could provide a data storage repository as a backend resource where information relating to mobile users could be located from remote Smart Spaces and used accordingly. However, current implementations of this abstraction limit the reach to within the confines of a LAN. To date there exists no implementation that can span multiple networks.

Advantages:

The advantages of distributed shared memory include:

- **Simpler abstraction** – For the programmer it is conceptually easier to have shared access to all data as it eliminates the need for explicit communication.
- **Persistence of data** –As the shared address space continues to exist after processes terminate, any data written to it will persist beyond the lifetime of the process.
- **Better performance** – Achievable in situations where a process exhibits good locality of reference.

Disadvantages:

The disadvantages of distributed shared memory include:

- **Thrashing** – thrashing can occur when several processes compete for the same data item, or falsely shared data item.
- **Object Tracking** – mechanisms to track the location of the remote data must be in place.
- **Minimisation of Communication** – communication overhead should be minimised when accessing remote objects.
- **Concurrency** – concurrency control must be in place to prevent "dirty reads" of shared data.

Although this approach was not pursued in the project, this review of distributed shared memory indicates that such a construct could have a major role to play in the migration of users' context between Smart Spaces. Implementation of this model would in fact eradicate the need to move the users context around the network. Each process would simply access the context in its place, update it and place it back in the unified virtual address space. However, the inability to span the address space across different network

subnets coupled with the disadvantages mentioned prompted pursuit of other approaches. The next section describes the P2P networking paradigm, which was the approach pursued in the project.

2.4 Peer-to-Peer (P2P)

The purpose of this section is to introduce the P2P networking paradigm that is essentially based on a decentralised architecture. P2P has the potential to effectively aid the distribution of users context in a non-deterministic ad hoc manner. Nodes on a P2P network can each offer a dynamic information repository whereby users context can be stored and addressed reflective of its content. This provides a means for other remote peers to perform context-based retrieval on the context information. This discussion begins in the next section with a comparison of P2P to the popular client/server model. Section 2.4.2 proposes three candidate architectures that are based on this model. The final section then highlights the general advantages and disadvantages of the P2P computing paradigm.

2.4.1 Centralised versus Decentralised Network Applications

The client/server approach [CDK01] to distributed computing is based on a centralised architecture, illustrated in figure 2.4, and implemented in many of today's network applications. This model is based on the segregation of computers by function. It generally takes the form of a request message from a client that is sent to a server requesting a particular resource. The server then executes the request and returns a response to the client. This client/server architectural model provides no definite model for scalability, primarily due to its dependency on a central server. As the number of clients increases, the load and bandwidth on the central server also increases. This has the potential to result in bottleneck at the server, meaning no more requests can be resolved. However, centralized client/server architectures also exhibit many advantages, such as

simplifying management tasks, providing easy access to all content and control of user access.

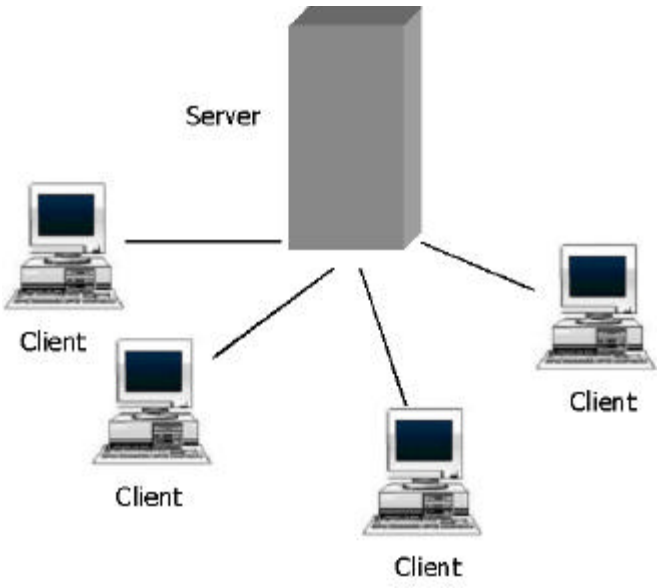


Figure 2.4: Centralised Client/Server Model.

By contrast, P2P applications are based on a decentralised architecture that offers a direct alternative to the traditional client/server model [PSW01]. Each peer in a P2P network is both a client and a server. P2P applications are analogous to telephone systems where a user can both speak (send information) and listen (receive information) [Harold00]. The P2P architecture leverages resources at the edge of the network such as bandwidth, computational power and storage by employing a topology where all peers communicate symmetrically and have equal roles. Services disseminate across the network and are prevalent on all or almost all participating peers, thus accentuating service coverage and access. Such dispersion eliminates dependency on a central server and eradicates the prospect of single point of failure.

The following section describes three different architectural approaches to P2P applications. The first architecture is a pure P2P design that is completely decentralised. The remainder of the section then discusses hybrid P2P architectures that employ a combination of centralised and decentralised topologies.

2.4.2 P2P Architectural Designs

Pure P2P

High availability of services that are disseminated across a P2P network coupled with the lack of dependency on a central server are the main advantages of a pure P2P application. This means P2P systems can continue to operate in the presence of failure. This architecture is depicted in figure 2.5.

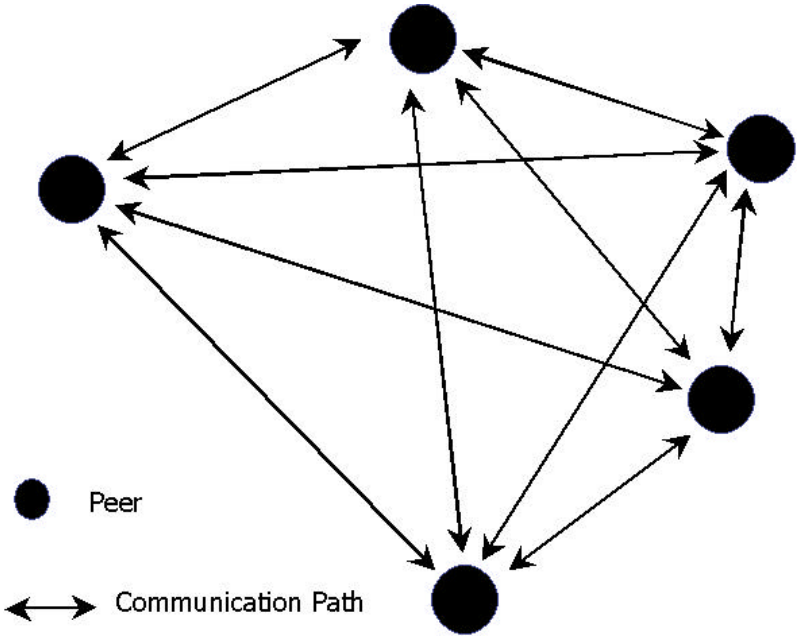


Figure 2.5: Pure P2P Architecture.

Peer discovery is dynamically executed through means of network broadcasting such as IP multicasting or from local configuration schemes that can prompt the peer with whom to communicate. The advantages and disadvantages of implementing this design are discussed in section 2.4.3.

P2P with Simple Discovery

Today, very few systems employ the pure decentralised approach. Most systems consist of a topology that incorporates a hybrid of centralised and decentralised architectures where a central server is used to authenticate users. This hybrid architecture is implemented mainly due to the complications that exist in the creation of P2P infrastructures. P2P with simple discovery, depicted in figure 2.6, is an example of such a hybrid topology. Discovery of peers that employ such a design work in a similar manner to the pure P2P architecture except that it relies on a central server for the discovery of other peers. Each peer informs the central server of its existence. Other peer applications then use the central server to download a list of peers present on the network. To retrieve specific content from peers, the application would exhaustively contact each peer on the list. Peers with the content would then respond by passing back the requested information. Similarly to the aforementioned client/server model the major downfall of this design is that it hinges on the availability of the central server. Another issue is the expense, from a network resource perspective, incurred through exhaustive communication between peers. This can lead to a drain on network resources such as bandwidth.

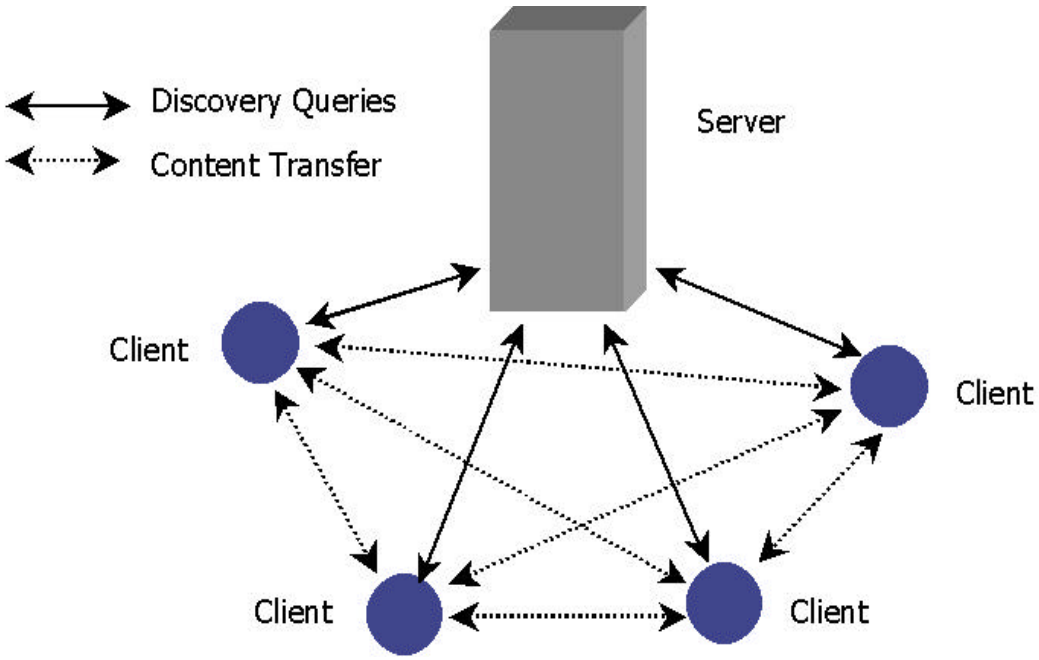


Figure 2.6: P2P with Simple Discovery

P2P with Discovery and Lookup Server

This model extends the previous discovery server so that it includes content lookup functionality. In this case, the peer application not only registers with a discovery service, but it also registers information relating to the content it possesses that is available for sharing. Peer applications requesting content must then query the central server. The server will in turn respond with a list of all peers that possess the associated content. The peer application can then contact those peers directly. The advantage of this system is that it means an application doesn't have to query all peers on the network. Only one request to the central server is necessary to identify the location of the desired content. Again, the major drawback is the dependency of the central server that can result in a single point of failure, particularly in this case as the server is a pertinent part of content sharing among peers.

2.4.3 Advantages and Disadvantages relating to P2P Architectures

As Minar [MDST01] pointed out P2P systems are by no means the silver bullet of distributed computing. This section aims to outline some of the general advantages and disadvantages inherent when deploying P2P systems. It is important to note that this section is generic in its nature and that the points discussed may vary depending on the P2P architecture implemented and the deployment environment of the system.

Advantages

- Fault Tolerance – Lack of dependency on a central server means that services can be dispersed at multiple peers on the network. The more peers ubiquitously present on the network that advocate the service, then the lower the probability of queries going unanswered.

- No Single Point of Failure – For reasons just stated, failure of a peer would not have catastrophic effect on the entire network, as responsibility of providing services resides with other peers.
- Bandwidth Consumption – P2P networks can implement a variety of communication channels and traversal paths that can reduce the load on network bandwidth. This is in contrast to the client/server model where there exists a specific route to servers.

Disadvantages

- Bandwidth Consumption – Contrary to reasons stated in the advantages of P2P systems, the incorporation of IP multicasting to aid in the discovery of peers could increase network traffic dramatically in situations where the communication is intended for one peer only but is received by all peers on the network.
- Availability – Peers that share content or services that are unique to the network may starve other peers of such resources should the peer become isolated or terminate.
- Security – The far-flung nature of P2P networks means systems tend to be difficult to manage and that data in the system is never fully authorative.

2.5 Summary

This chapter introduced some interesting projects that are ongoing in the area of user migration within Smart Space domains. Although none of the aforementioned projects deal exclusively with the movement of users across Smart Spaces, some fundamental aspects relating to Smart Space computing were brought to light, many of which provide the core requirements of this project. Also discussed was the Distributed Shared Memory abstraction and two implementations of this concept, JavaSpaces and TSpaces. This approach provides a potential alternative to the peer-to-peer approach adopted in the project. This work was undertaken to provide a contrast. The chapter then concludes with

an introduction to peer-to-peer programming, the approach that was chosen for investigation. The P2P implementation that was chosen as the basis of the project is Sun Microsystem's JXTA platform, which is described in the next chapter.

Chapter 3

JXTA

3.1 Introduction

This chapter offers a detailed description of project JXTA [JXTA], Sun Microsystems, Inc., network programming and computing platform. The purpose of this discussion is to introduce the reader to JXTA, which at the time of writing is a relatively new technology.

3.2 JXTA Architecture

JXTA was released as an open source project in April 2001. JXTA is an abbreviation for "Juxtapose" which means putting things side-by-side. JXTA at its core provides developers with a framework to create interoperable P2P applications and services. JXTA is designed for ad hoc, pervasive, and multi-hop P2P network computing [JXSPEC] and allows any connected heterogeneous device on the network – from a mobile phone to PDA, from PC to server – to communicate and collaborate with one another.
Sun's claim is that JXTA is designed to solve the many deficiencies inherent in distributed computing, some of which are:

- Peer Discovery – JXTA is not dependent on a central server where peer information can be replicated and retrieved. This means peers must be able to discover other peers on the network. JXTA also provides a solution for the discovery of peers that reside outside of the LAN setting.
- Interoperability – Most legacy P2P applications, such as USENET [USE], depend on vendor specific protocols that prevent compatibility with other P2P

applications. JXTA aims to provide a common interoperable set of protocols that are compatible with other P2P systems. This means interoperability amongst a range of devices from mobile phones to desktop computers.

- Platform Independence – JXTA endeavours to eradicate the need to re-write low-level core protocols for each platform upon which applications are deployed. This means abstracting the core protocols to ensure interoperability across a wide spectrum of operating platforms.
- Ubiquity - JXTAs prerogative is to also allow any device with a digital heartbeat (e.g. PDA, laptop, desktop etc.) to be part of the JXTA network.

The JXTA framework consists of three layers – core layer, service layer and the application layer. These are described as follows:

- **Core Layer**

  The core layer encapsulates the essential functionalities that are common to P2P networking. The elements of the core layer are:

  - **Peers** – Peers are nodes on a P2P network. In terms of JXTA a peer is any networked device that implements one or more of the JXTA protocols. Peers can range from mobile phones to desktop computers.

  - **Peer Groups** – A peer group is a group of peers that share a common set of services whereby the members of the group can cooperate and collaborate. A more detailed explanation of the peer group concept is described in section 3.6.3.

  - **Network Transport** – Network transport is a layer that provides a mechanism to handle the transmission of data over the network. Wilson [Wil02] breaks the network transport concept into three constituent parts:

    - **Endpoints** – the initial source or final destination of any piece of data being transmitted over the network.

- **Pipes** – uni-directional, asynchronous, virtual communications channels connecting two or more endpoints.
- **Messages** – containers for data being transmitted over a pipe from one endpoint to another.

- **Advertisements** – are structured XML documents that describe all JXTA resource such as peers, peer groups and services. A detailed explanation of JXTA advertisements is provided in section 3.3.

- **Protocols** – JXTA protocols deal explicitly with a fundamental behaviour of a P2P system. There are six protocols provided by JXTA. These are the *Peer Discovery Protocol (PDP), Peer Resolver Protocol (PRP), Peer Information Protocol (PIP), Pipe Binding Protocol (PBP), Rendezvous Protocol (RVP)* and the *Endpoint Routing Protocol (ERP)*. These are described in detail in section 3.4.

- **Service Layer**

  The service layer contains all the services that may exist in a JXTA network. This layer is optional as operation of a P2P system is possible without service layer implementation. CMS (Content Management System) [CMS] is an example of a service that can be implemented to manage content within a JXTA network.

- **Application Layer**

  The top layer is the application layer, which utilizes both the service layer and the core layer. A JXTA application will typically be simultaneously using some aspects of conventional client-server networking juxtaposed with JXTA-based P2P functionality [Li02]. The boundary between the application layer and the service layer is often not clearly defined as both layers can rely on each other thus making it difficult to define if a network module is a service or an application, e.g. JXTA-shell [JShell] . Often the presence of a GUI indicates the module is an application.

The following diagram depicts the JXTA software architecture described above:

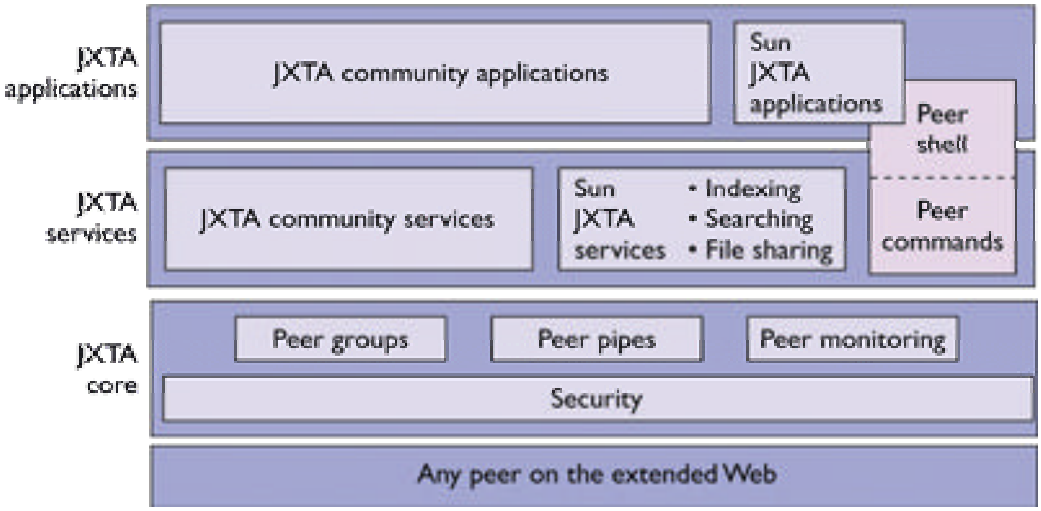


Figure 3.1: JXTA three-layer architecture.

JXTA was designed in a modular fashion. This allows developers to implement substrates of the platform in order to adapt their applications. Each protocol present at the core in JXTA deals exclusively with an essential concept that permeates most P2P systems. These protocols are examined in section 3.4.

The following section describes JXTA advertisements, which are fundamental to JXTA as they provide the backbone to all communication within a JXTA network.

3.3 JXTA Advertisements

An *advertisement* is a structured XML document used to describe all JXTA network resources – peers, peer groups, pipes and services. Each entity in JXTA notifies other peers of its existence on the network by publishing[2] XML documents that describes the resource. Each advertisement has an expiration time associated with it. This ensures that

[2] Publish is a term used to describe how information is placed on to the network. In this context the information communicated to other peers via a form of network multicasting.

stale advertisements that describe non-existent resources will time out. Expiration properties alleviate the need for a centralised control repository that ensures resources are active. A peer that locates a stale advertisement will be required to re-discover the advertisement, if possible. The following is an example advertisement, *PeerAdvertisement* that describes a JXTA peer resource:

```
<?xml version="1.0"?>

<!DOCTYPE jxta:PA>

<jxta:PA xmlns:jxta="http://jxta.org">
      <PID>
            urn:jxta:uuid-
59616261646162614A78746150325033BE6154A9F71A...
      </PID>
      <GID>
            urn:jxta:uuid-68B8A7A691684F9C9E05971D66D78ED602
      </GID>
      <Name>
            node4
      </Name>
      <Svc>
            <MCID>
                  urn:jxta:uuid-DEADBEEFDEAFBABAFEEDBAB...
            </MCID>
            <Parm>
                  <Addr>
                        tcp://localhost:9759/
                  </Addr>
                  <Addr>
                        jxtatls://uuid- ....   /TlsTransport/jxta-
WorldGroup
                  </Addr>
                  <Addr>
                        jxta://uuid- .... /
                  </Addr>
                  <Addr>
                        http://JxtaHttpClientuuid-5961626901603.../
                  </Addr>
            </Parm>
      </Svc>
</jxta:PA>
```

- *<Name>*- describes the name of the peer. In the above example it is node4.
- *Description <Desc>* - provides a description of the peer.
- *Peer ID <PID>* - uniquely identifies the peer on the JXTA network.
- *PeerGroup ID <GID>* - details the group identification number.

XML advertisements describe all facets of JXTA resources and are replicated throughout each and every network. XML is self-describing, extensible and simple. For these reasons and the fact that XML is language neutral and can be processed by any programming language capable of parsing textual strings, it is employed by JXTA. XML is not only confined to describing advertisements but is also the language of choice to describe the messages of the various messaging protocols implemented by JXTA, such as Peer Discovery Protocol and Peer Resolver Protocol.

The intent of the JXTA protocols is to be as pervasive as possible, and easy to implement on any transport. Few assumptions are made relating to the underlying physical transport network and the deployment environment of peers. The following sections examine JXTAs protocols.

3.4 JXTA Protocols

The JXTA specification [JXSPEC] consists of six independent protocols, which are illustrated in figure 3.2. Each protocol deals explicitly with a fundamental behaviour that permeates most P2P systems. Independence from the underlying physical transport protocols and programming languages are a major feature of JXTA. The current implementation[3] can theoretically be executed on top of TCP/IP, HTTP and Bluetooth transport protocols. The current reference implementation is in the Java programming language but work is under-way for C, C++ and Perl implementations.

[3] At the time of writing the current implementation is 65E.

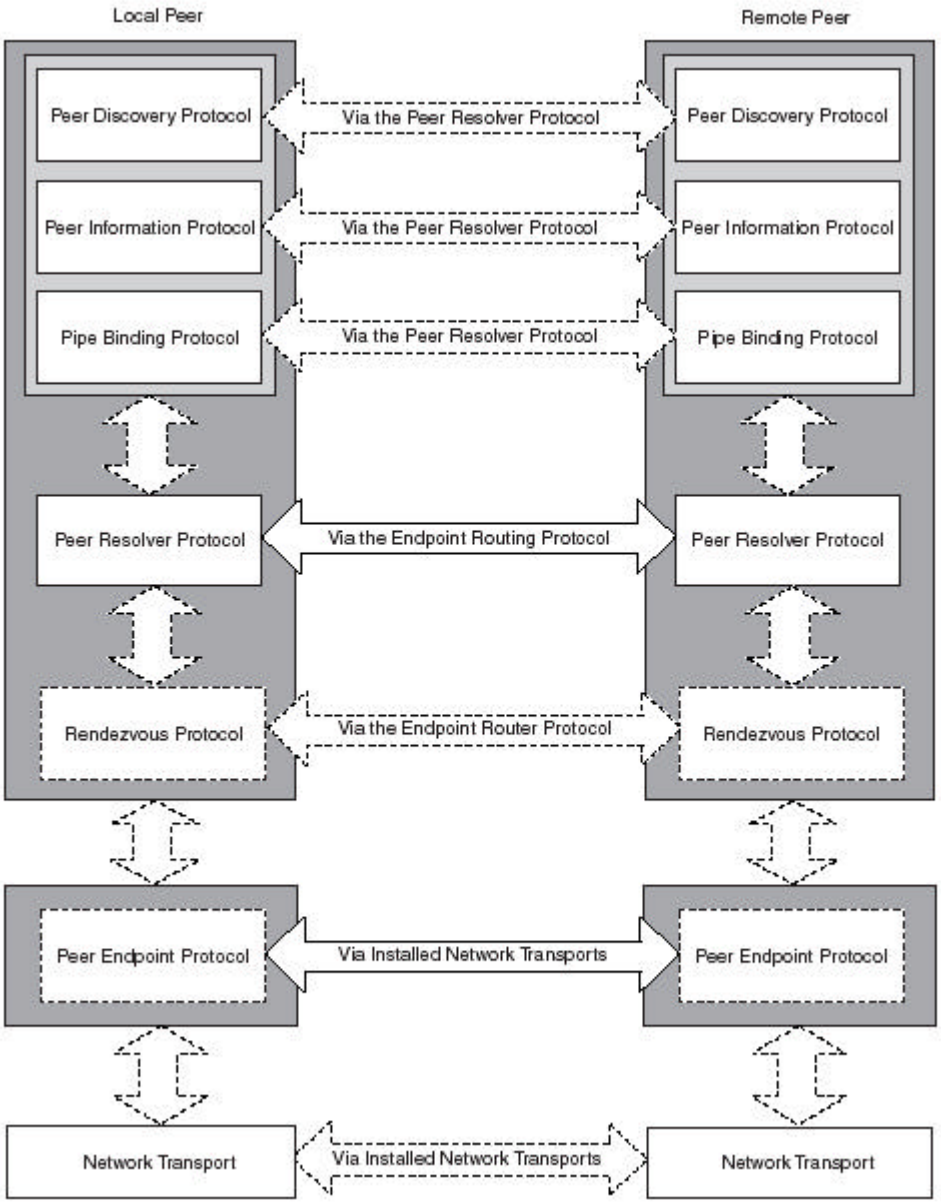


Figure 3.2: JXTA Protocol Stack.

The above diagram depicts the JXTA protocol stack. Each protocol is independent of the others. This means that not all protocols need be employed by a P2P application, only those necessary. Collaboration of the protocols is intended to provide dynamic transparent discovery, organization and communication amongst peers. Each protocol conversation is divided into two peers, a local and a remote peer. The remote peer is responsible for incoming messages and the processing of these messages.

3.4.1 Peer Discovery Protocol (PDP)

The Peer Discovery Protocol (PDP) is used to discover any published JXTA resource advertisements. Such JXTA resource advertisements can describe peers, peer groups, virtual channels of communication (pipes), peer services and peer group services. Discovery is group specific, which means that any request for advertisements will not be propagated to any peers that are not members of the requesting peers group. Discovery is two-fold as both local and remote discovery operations are possible. Local discovery searches the local cache for advertisements and remote discovery uses the Resolver protocol (PRP), which is described in the next section, to find advertisements. In the Java API, advertisements that are discovered are stored locally at the requesting peer. This means commonly requested advertisements would be replicated more frequently around the JXTA network. At its core the J2SE (Java 2 Standard Edition) [JAVA] binding of this protocol uses a combination of IP multicast to the local subnet and the use of rendezvous peers. Rendezvous peers incorporate a method of sending requests from one known peer to the next to dynamically discover information. Peer discovery is non-deterministic. A peer may receive zero, one or more responses to a discovery request.

3.4.2 Peer Resolver Protocol (PRP)

The JXTA Peer Resolver Protocol (PRP) is based on the request/response model of message passing. It is used to send a query to another peer or group of peers and receive a response. The PRP is the underlying protocol for supporting generic requests and is incorporated by PDP for the discovery of the various network resources. However, PRP is unreliable so there exists no guarantee of the safe delivery of query messages and response messages. JXTA provides no methodology to handle this scenario so guaranteed message delivery is up to the application developer. Both the query and response messages are XML based around a payload of information defined by the JXTA specification. An example scenario that could implement this protocol could be the case

where user messages require an acknowledgement of receipt. The response to the request could simply contain confirmation that the requesting message was received.

3.4.3 Peer Information Protocol (PIP)

Like PDP, the Peer Information Protocol (PIP) implements the PRP. Once a peer has been located, via its *PeerAdvertisement*, the status information relating to that peer can be retrieved. Using the endpoint information contained in the advertisement the peer can be contacted directly. Information such as peer uptime, traffic and service usage can be obtained via PIP.

3.4.4 Endpoint Routing Protocol (ERP)

An endpoint is an interface to a network transport that allows data to be sent across the network. The Endpoint Routing Protocol (ERP) provides the capability to allow applications to control and examine the topology of the network via various request/query messages defined in the protocol specification. Often routing information is necessary for sending messages between peers. When peers are requested to send a message to a particular endpoint address they check their local cache first for a route to the address. If no route exists they request a route from all of their known relay peers. Peers with the appropriate information will return the route information as an enumeration of hops [JXSPEC].

3.4.5 Rendezvous Protocol (RVP)

As previously stated, message passing is confined to the context of the group from which it originates. The Rendezvous Protocol (RVP) propagates messages within the specified group on behalf of peers from within the same group. The primary purpose of RVP is to

provide a service whereby peers can connect to a rendezvous peer to avail of a service that will propagate messages on their behalf. This is particularly useful in the scenario where members of a peer group exist outside a private network. The rendezvous peer is then used to propagate messages from the peers inside the private network to the existing peers outside.

To obtain the rendezvous service a peer must obtain a lease from the rendezvous peer. This lease specifies the amount of time that the peer can avail of the rendezvous service before it must renew its lease. To ensure that peers will receive propagated messages before the lease times out the Java reference API provides a class that periodically runs a thread. This thread uses the discovery service to find rendezvous peers and renew the lease.

3.4.6 Pipe Binding Protocol (PBP)

Pipes are virtual channels of communication in a JXTA network that are used to transmit data. [BGK02] describes pipes as a mechanism through which peers can interconnect directly, even through firewalls. Pipes are essentially a layer on top of multiple communication protocols to support relayed communications via gateway peers. Pipes provide an abstraction to hide the fact that there may exist relay peers in between the source and destination peers. Pipes, like many other resources, in a JXTA network are described by an XML advertisement called a *PipeAdvertisement*. In order to transfer data a direct connection is not mandatory. A pipe can consist of one Output Pipe endpoint (source) and one or more Input Pipe endpoints (target). To achieve communication, a pipe must be bound to a peer endpoint. The Pipe Binding Protocol (PBP) specifies the process of endpoint binding. The PBP is layered on top of the Endpoint protocol so it has the capability to use multiple transport protocols such as HTTP and TCP/IP. The ability to use the HTTP transport protocol provides a means to penetrate firewalls so data can be transmitted seamlessly.

There are three types of pipes:

- *JxtaUnicast* Unicast, unsecure and unreliable. This type of pipe is used for one-way communication.
- *JxtaUnicastSecure* Unicast, secure (using TLS). This pipe is similar to the aforementioned except the data transferred is protected using a virtual TLS connection between endpoints.
- *JxtaPropagate* Diffusion pipes. This type of pipe is used to send one-to-many messages.

3.5 Services

In order to understand the purpose behind *peer services* and in particular *peer group services* it is appropriate to first explain the concepts that these services rely upon.

3.5.1 Module Advertisements

JXTA implements a module framework abstraction that defines services and applications on a peer or peer group. In other words, modules define the code to be executed in order to run the service or application. Modules are defined in three separate advertisements – *Module Class, Module Specification* and *Module Implementation Advertisement*. The primary function of each, as outlined by Brookshier [BGK02], is as follows:

- Module Class – Defines the specific behaviour and expected API for each JXTA binding. The principal function of this advertisement is to describe what the module is intended from a high-level human readable perspective. The advertisement has the following parameters:
    - *Module Class ID* – Unique identifier used to reference the module class.
    - *Name* – Name of the module. This is used for searching and identification.
    - *Description* – Used to describe the modules intent. This is also used for searching.

- Module Specification – This advertisement is the specification of a module. Module specification advertisements are used to describe how a module is invoked and how it may be used. The following are a subset of the elements that make up the module specification advertisement:
  - *Module Spec ID* – Unique identifier that specifically defines the module.
  - *Name* – Name of the specification.
  - *Creator* – Name of the creator of the specification.
  - *Parameters* – A list of parameters used by the implementation.

- Module Implementation – A module implementation advertisement describes one of the implementations of a module specification. Essentially, this advertisement is used to launch the module code. This advertisement contains the following elements among others:
  - *ModuleSpecID* – ID that uniquely identifies the specification being implemented.
  - *Compatibility* – Describes the execution environment.
  - *Code* – Contains a reference used to load and execute the code of this implementation. For Java, this is a fully qualified class name.

The essential idea behind the separation into three separate modules is three-fold [Wil02].

- The separation of concerns in the module specification must ensure that JXTA is language/platform independent. Therefore, distinguishing between C++ and Java implementations is essential to P2P systems and is realised by the module abstraction.
- Extensibility is a major issue when specifying module advertisements that are used to implement JXTA resources. The fundamental reason behind this is to allow developers to change the capabilities of the modules dynamically to reflect the changes in the actual implementation.
- The third reason is to relate the meta-data describing a module specification and the meta-data describing the module's class.

Both service and application modules can be initialised started and stopped via the appropriate methods contained in the implementation code. The underlying concepts relating to service modules are similar to those of application modules.

JXTA services in their simplest form are hosted by a single peer and essentially perform a task on behalf of a remote peer. A peer service is accessible only on the peer that published the service. A Peer Group Service is a single service with multiple instances redundantly installed and available at multiple locations on the network. JXTA provides a core set of services that can be employed. These are discussed in the following section.

3.5.2 Core Services

Although not mandatory, the core services provide a building block upon which basic communication with other peers is possible and also allows developers to specify new purpose-built services. Such Services are often customized versions of the core JXTA services. The core services are *Discovery Service*, *Membership Service*, *Access Service*, *Pipe Service, Resolver Service* and the *Monitoring Service*.

- Discovery Service – Provides access to the Peer Discovery Protocol (PDP). Access to this service is limited to the context of the group to which the peer is a member. The discovery service searches for JXTA resources. JXTA resources are defined as peers, peer groups, pipes and services.

- Membership Service – The membership provides access to the Peer membership protocol. It provides a means to filter users joining the group through a process that requires prospective peers to meet certain requirements. Approved peers become members and are passed a credential that proves their membership.

- Access Service – The service works in conjunction with the previous service. Its function is to ensure peers are authentic members of the group.

- Pipe Service – This service manages and creates virtual channels of communication between peers in a peer group. This service is also specific to the group context.

- Resolver Service – This service implements the resolver service to distribute queries among peers. This is based on the request/response model.

- Monitoring Service – This service provides a means to monitor other peers in the peer group. The reasoning behind such a service is application dependent. For example an application may want to limit the usage on certain peers. The monitoring service would provide the capability to observe traffic from such peers.

Incorporation of these services into a P2P system is optional, but often recommended. For example, an important point to note is that services are discovered via the core Peer Discovery Protocol (PDP) so omission of this service will prevent the discovery of other services.

3.5.3 Peer Groups

Peer groups are a central concept upon which JXTA is based. Peer groups provide a meaningful partition for peers to cooperate and collaborate. The main characteristics of a peer group are the following:

- Peers self-organize into groups.
- A unique identification number uniquely identifies each group.
- Peer groups provide a security domain within which peers can be authenticated and monitored.
- Only members of the group have access to the group services.
- A peer group provides a set of services called *Peer Group Services.*

3.5.4 Peer Group Services

Peer groups provide a set of services that are associated with that particular group. JXTA provides core services that may be implemented or built upon to develop user-defined peer group services. Instances of the peer group services are replicated on multiple peers across the group to provide an indomitable service. This ensures high availability of the service to members of the group, even in the presence of failure. Groups can comprise any number of services and services are not unique to groups. Services that a group offers are specified in the *PeerGroupAdvertisement*.

3.6 myJXTA

To reinforce what exactly JXTA is about and to highlight the potential of the technology this section will present a brief introduction to "myJXTA"[MYJX]. The myJXTA application is one of the first and fully featured applications developed. This application is designed to run on desktops and handheld devices that support J2ME (Java 2 Micro Edition) [J2ME] platforms.

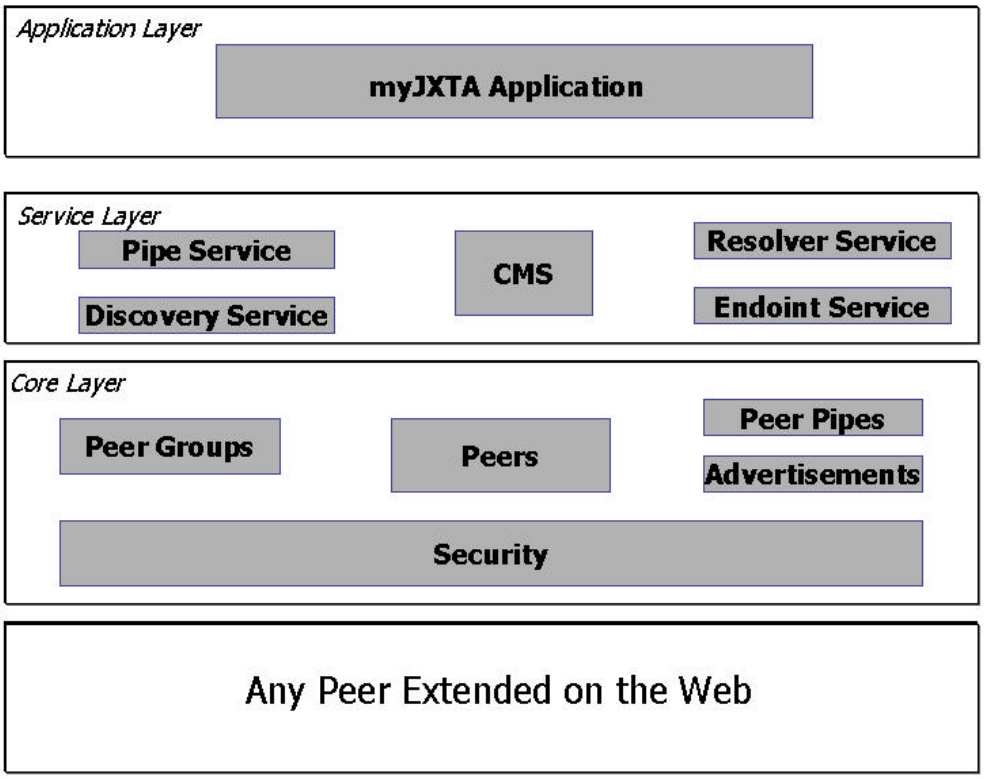


Figure 3.3: myJXTA Architecture.

The myJXTA application is essentially a composition of 2 main applications, a chat and a file sharing application. The chat application allows users to engage in one-to-one chats and group chats. Group chat is synonymous with the popular "chat room" concept. The second application incorporates the Content Management System (CMS) service to allow users to share files. Also incorporated into myJXTA are the core services. Figure 3.3 depicts an architectural layout of myJXTA and some of the components involved.

In myJXTA the concept of a peer essentially represents a user. This representation is due to the nature of the application as it provides a means for users to interact with each other. Each peer in myJXTA is a member of a universal peer group known as the NetPeerGroup. The NetPeerGroup is the root of all peer groups and is accessible to all myJXTA peers. This hierarchy provides a means to search for other peers and peer groups on a global scale. All peer groups created by the myJXTA application are children of the NetPeerGroup. Peer groups in this case provide a meeting place where other peer (users) can communicate and share files.

Users of myJXTA can join and leave peer groups that are discovered. These groups are discovered at start-up, as the myJXTA peer is initially a member of the universal NetPeerGroup. All peers in JXTA are members of the NetPeerGroup. However not all peers may be discovered. Some may reside on isolated networks preventing them from being discovered outside of the LAN. In myJXTA users also have the capability to create and destroy their own peer groups. More information relating to the myJXTA project is described in [MYJX].

3.7 Summary

This chapter offered a detailed description of a subset of Sun Microsystems's P2P networking and computing platform, known as JXTA. The fundamental reason behind examining JXTA was to lend the reader a thorough understanding of the technology and its concepts. In the following chapter it is described how such a technology was incorporated in the development of a platform to simulate the migration of a user from one network location to another.

Chapter 4

Design

4.1 Introduction

It is the aim of this chapter to present the requirements, architecture and design for the *MIGRA* system that supports the migration of a user's context information across smart spaces. Section 4.2 presents the requirements that are derived from the research in chapter 2. Also chapter 2 outlines a number of P2P application architectures that are possible. Section 4.3 considers the two hybrid architectures and outlines how our system might have been implemented using these. Section 4.4 then overviews the *MIGRA* architecture that is based on the pure P2P architecture approach. Finally, section 4.5 details the design of *MIGRA*.

4.2 Requirements

This section lists and describes some of the requirements upon a smart space user migration support system derived from the state of the art survey in chapter 2.

- **Empowerment of Spaces –** O' Sullivan [OSW02] has indicated that very little consensus as of yet has emerged regarding the architectural approach most suited for smart space environment's computing infrastructure. However, O' Sullivan [OSW02] goes on to further point out that there is consensus on the need for the environment to gather information about user's activity within the smart space. This implies that user information should be available to other smart space environments. This further ensures users context would effectively reflect the diverse real-time actions performed by the user.

- **Awareness of user context** – Dey, in his paper [DAS99], stipulates that in order for a smart environment to provide services to its occupants, it must be able to detect its current state or context and determine what actions to take based on the context. For this reason there must be global consensus as to what the users context is. In other words, users context information must directly reflect the users location, activities and be readily available for any other smart space locations on the network to process. There should be no conflict between instances of this context information.

- **Multi-user** – Smart space environments are not limited solely to a single user. Take for example a number of offices (i.e. smart spaces) on a single floor in a building, all of which are interconnected. Each office would have a primary user and perhaps associated users that may frequently avail of the services present. Each user would not be confined to one office. In some cases there could be a multitude of users all cooperating and collaborating in a single smart space. To cater for such a scenario the infrastructure must provide users with the ability to migrate between locations and avail of the services in each environment in a seamless manner. This implies each smart space environment must be capable of managing more than one user.

- **Empowerment of Participants –** Smart Spaces need to provide intuitive user-friendly interfaces at a level that will benefit even users that are not technically adept. The challenge is to integrate user interfaces so that they become an inherent part of the tools that the user would naturally avail of from within a task environment [OSW02].

- **Web Presence –** In [KBMB] significant progress in empowering users is achieved through a concept known as *"Web Presence"*. The information contained in a users web presence describes that user and presents details on how collaboration and communication with the user can be achieved. The key concept here is that the information would be globally available to each smart space infrastructure.

- **User detection** – Traditionally the presence of a person has been identified through the use of multi-modal sensor technology and smart cards. Such techniques are necessary in order to gather information about users activities whilst within the smart space. User detection is also a process whereby the infrastructure can differentiate between its occupants.

- **Ubiquitous Services** – In order to provide seamless and transparent interaction to the user there exists a need for services to be available at any point on the network. Smart Space systems must provide users the capability of suspending their computing tasks in one environment and resume them in another. Concerning this, there are two main solutions. The first is a scenario where a service is uniquely located at particular location(s) on the network. Users would be able to access the service from a remote location transparently. The second approach would be to replicate the services at every location. In this case, the user accesses the closest service instance. In both cases responsibility to adapt the service to reflect the users context lies with underlying infrastructure.

Wang [WG00] proposes an infrastructure that manages many low-level activities so that users can interact with a computing system directly in terms of high-level tasks that they wish to accomplish. From Wang's paper [WG00] and the requirements described above the key requirements on the underlying infrastructure are derived. These requirements are described as the following:

- **Global Access to User Context** – Providing a means for the smart space environment infrastructure to access users context will allow users to continue their tasks in remote locations seamlessly.

- **Information Persistence** – A key challenge for pervasive computing environments is the management of user context in the face of constant change. Therefore, the underlying system should provide a means to update user context to reflect its diversification.

- **Proactive Guidance** – ‘O Sullivan [OSW02] and Dey [DAS99] both underline that smart spaces should offer proactive guidance to assist users in completion of their tasks in an alternative manner.

With these requirements in mind the remainder of this chapter outlines the design options that were considered for actual implementation. A description of the actual design implementation is then provided.

4.3 Design Choices – Mapping to JXTA

As described in chapter two, many of today’s P2P applications employ a hybrid topology combining centralised and decentralised architectures. This coupling provides the core of the first two design architecture options that were considered.

4.3.1 P2P Architecture with Simple Discovery

The first architectural option employs the simple service discovery technique discussed in chapter two. This architecture consists of a central server that is interconnected with a number of peer applications. A peer application represents a user. The users unique identity would be directly associated with the peer application instance. This mapping would occur during the detection of the user entering the Smart Space. A Smart Space would comprise of any number of peer applications in the same physical location, each of which would reside on separate devices. In this model a peer group would represent a Smart Space domain, where a Smart Space domain is a number of physically disseminated Smart Spaces. Each peer would transparently announce its existence, location and its proprietor, i.e. user, to the central server. The server would store this mapping of the user to the application and the applications location in a storage

repository. This information would then be available to other peer applications upon request.

When a user is detected after migrating from one location to another the infrastructure at the new Smart Space location would send a query to the central server requesting a list of all peers active. The server would then return the list, step 1 in figure 4.1. As with the architecture described in section 2.4.2 the peer would exhaustively contact each peer on the list requesting if it has context information associated with the user detected. The peer with the relevant context would then respond by returning the context information. In the figure below, Peer 4 has the relevant context information. This information would then be processed at peer 1 and the context information would be updated to reflect the users movement. The advantages and disadvantages of this architecture are discussed in the following.

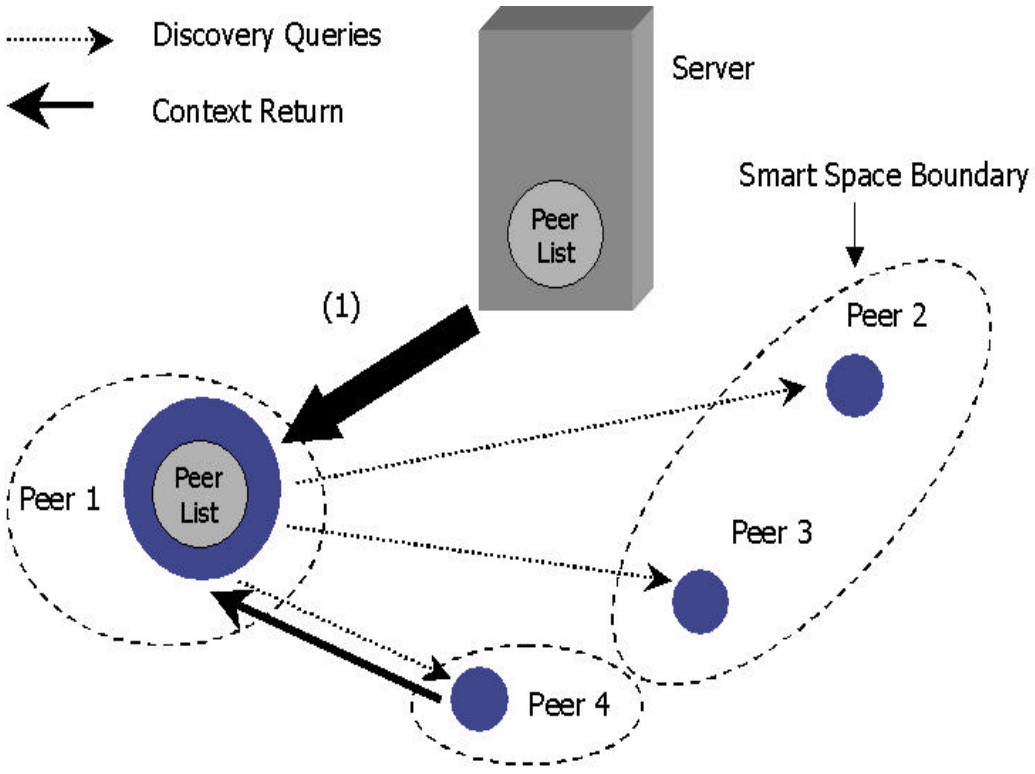


Figure 4.1: Simple Discovery.

Advantages:

- The central server would contain an active list of all users on the network. This means that peers only have to contact the central server to determine the location of the other users on the system that are active.
- Peer applications could be pre-configured with the address of the server to avoid the need to implement discovery of the server.

Disadvantages:

- The system would require a polling mechanism where peers would periodically contact the server to notify the peer's existence on the network.
- The context information would not be stored at the central server. This implies that in order to retrieve a users context each peer would have to be contacted and queried.
- There would exist a potential for conflicting user context unless a mechanism to prevent this was employed. Such a mechanism may involve more messages passing to ensure synchronization of users context.
- Retrieval of users context on a highly populated system would require a significant amount of message passing. This ratio would increase exponentially with additional peers making additional requests. Such high bandwidth consumption could drain other network resources.
- Reliance on the central server means there is a single point of failure for the system.

For the reasons stated this architecture was not sufficient to effectively simulate a smart space environment. The next section discusses a design that is based on the hybrid architecture discussed in section 2.4.2.

4.3.2 P2P Architecture with Discovery and Lookup

This topology consists of a P2P network based around JXTA that is complemented by a central server, as illustrated in figure 4.2. The representation of users and Smart Spaces remains the same as the previous design. This means a peer application represents a user. A Smart Space would consist of one or more peer applications and a peer group would map to a Smart Space domain.

The primary difference between this architecture and the aforementioned is the function of the central server. The server in this case stores a list of the peers that are active on the network and the users identity that is associated with that peer. This information would be passed to the central server during user detection. Once fully formed, users context would also be stored at the central server.

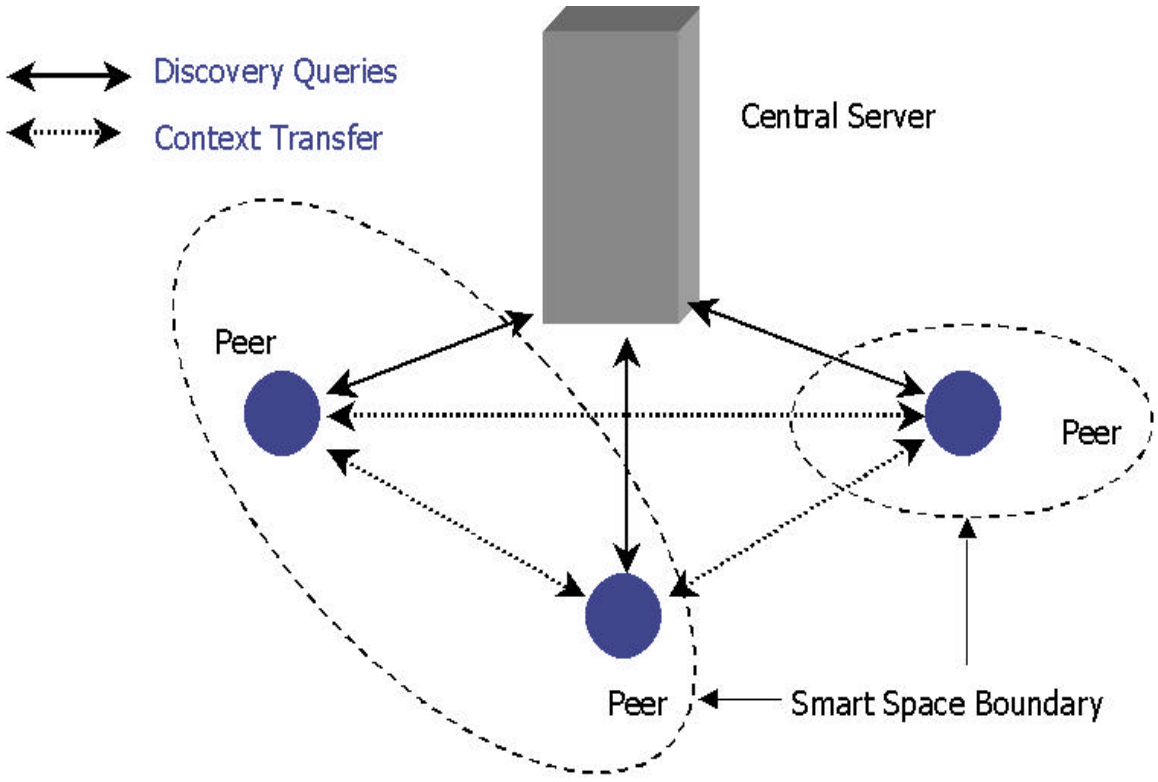


Figure 4.2: Hybrid Architecture.

The logic behind the incorporation of a central server is that in the event of network failure/partition the server shall possess a relatively up-to-date version of the users context. Having user context stored in such a centralised repository would allow the system to operate regardless of failure at a particular peer. However, there exists a possibility that the context stored at the server is out-of-date. Another drawback to implementing this architecture is the ever-present dependency on the central server. This implementation also hinders the main features of JXTA such as dynamic peer discovery and caching of information at each individual peer. The advantages and disadvantages are summarised below.

Advantages:

- All users context is easily retrievable at a centralised location. This implies there would be no need of dynamic discovery of remote peers on the network.
- The persistence of context information would mean the system would be resilient, to a degree, of network failure/partition.
- Location of context information could be located via template matching techniques at the server. The context that best matches the request and which is most up-to-date would be returned. This could avoid retrieval of stale context information.
- Bandwidth consumption would be minimized from the previous design, as the location of the context information would be retrieved through one request to the server.

Disadvantages:

- Failure of the central server would be catastrophic to the system. No requests could be resolved in such an event rendering the system useless.
- There would need to be a mechanism to prevent out-of-date or conflicting context information.

- Similarly to the previous design there would need to be a means for peers to periodically poll the server to ensure their existence on the network.

These are the principal reasons that this architecture was finally disregarded in preference for that which is discussed in the next section.

On the other hand, one fundamental aspect of this and the previous architecture is carried into the chosen architectural option. That is, the mappings between JXTAs core concepts and those inherent in Smart Spaces:

- A peer application represents a user;
- A Smart Space consisting of a number of peer applications;
- A peer group represents a Smart Space domain.

The architectural option chosen for MIGRA is the Pure P2P option and the MIGRA architecture is outlined in the following section.

4.4 The *MIGRA* Architecture

The design of *MIGRA* complemented the JXTA software architecture by building upon the core services provided and introducing two new services that reside in the service layer. These services are in turn utilised by the *MIGRA* application that resides in the application layer. Figure 4.3 illustrates the *MIGRA* software architecture.

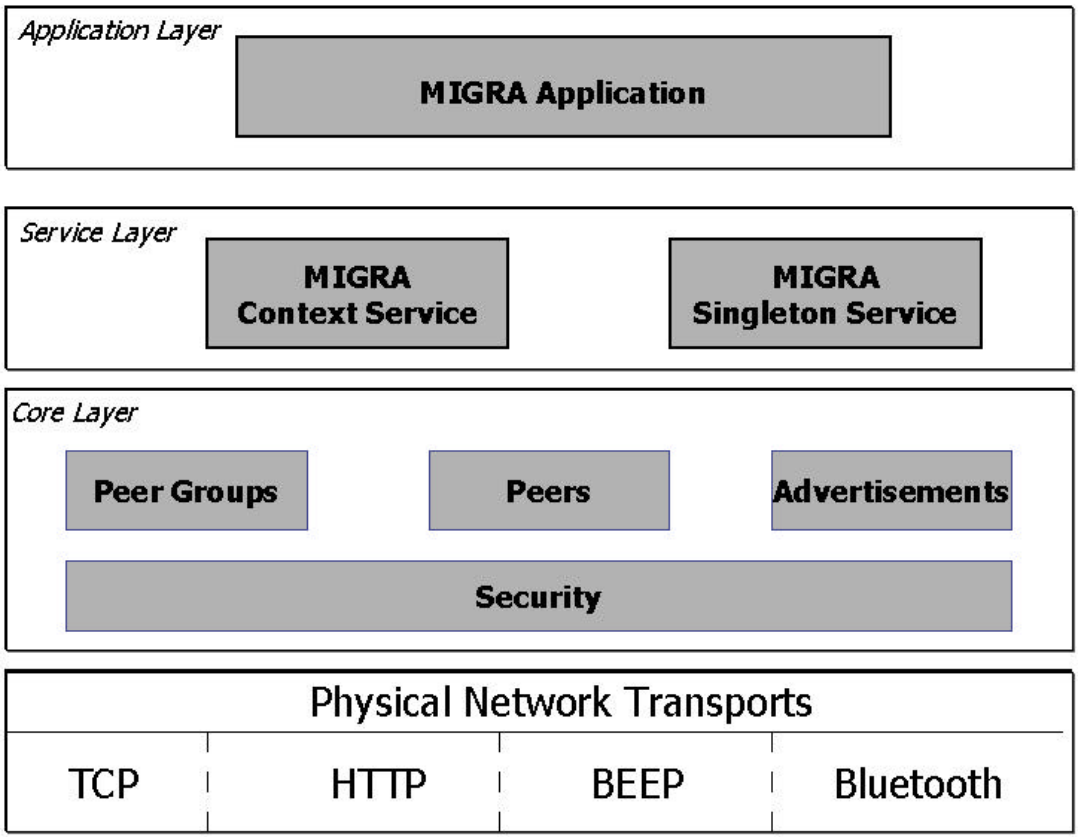


Figure 4.3: Software Architecture of *MIGRA*.

Core Layer

The core layer of *MIGRA* is derived directly from that of JXTA. The core layer provides the elements that are essential to P2P networking. A full description of this layer is offered in chapter 3. The following describes aspects of the JXTA core layer that is implemented by *MIGRA.*

- **Peers –** peers in *MIGRA* essentially represent the user that avails of the application running on the peer. The mapping between the user and the peer application occurs during the detection process, which simulates a user entering the Smart Space.

- **Peer Groups** – *MIGRA* peer groups represent a number of Smart Spaces, i.e. a Smart Space domain. Each Smart Space is a physical location containing one or more users.

- **Network Transport** – XML messages are used as a transport mechanism to send textual data across the network via various message-passing protocols such as the Peer Resolver Protocol. *MIGRA* implements this concept as a means to communicate between remote Smart Space infrastructures.

- Pipes are another concept that fits into this category. Pipes are described in section 3.4.6. In essence, pipes provide a means to create a virtual channel of communication between one or more peers that may not be capable of connecting directly. The primary use of pipes is to provide a mechanism to transport data between peers that may reside on different networks. In the case of *MIGRA* the local network was the working environment and no heavyweight data is transmitted, only text messages. For this reason pipes were not used in the *MIGRA* design.

- **Protocols** – section 3.4 provides a detailed discussion of the JXTA protocols. Of the six core protocols four were implemented into *MIGRA*. Implementation of the protocols is achieved via the core services discussed in section 3.6.2.

    - **Peer Discovery Protocol (PDP) –** the PDP provides the basis for discovery of JXTA resources. In *MIGRA* this protocol provides the crux of the context service for discovering peers and context advertisements. This is discussed in more detail in section 4.5.2.

    - **Peer Resolver Protocol (PRP)** – similarly to the context service the singleton service implements one of the JXTA protocols, the Peer Resolver Protocol. The role of the PRP in the singleton service is discussed in section 4.5.1.

- **Rendezvous Protocol (RVP)** – the rendezvous protocol is responsible for propagating messages within a peer group. Implementation of this protocol into *MIGRA* was transparent during development as it is used by the Peer Resolver Protocol to propagate messages.

- **Endpoint Router Protocol** – the ERP provides the basis for communication by enabling pipes or messaging. JXTA provides two methods to pass information between peers, the resolver service and the pipe service. These services are wrappers for sending and receiving messages using a peer's local endpoints. The ERP is responsible for transmitting the data over the actual underlying network by providing peers with a route to the destination endpoint.

The following protocols were not implemented into the *MIGRA* architecture. The reasons are as follows:

- **Peer Information Protocol (PIP)** – the current implementation of the PIP is limited to only discovering status such as uptime and traffic of remote peers. The requirements of *MIGRA* did not constitute implementation of this protocol.

- **Pipe Binding Protocol (PBP) –** implementation of the PBP is only necessary in systems that employ pipes. As for reasons previously mentioned *MIGRA* did not require the incorporation of pipes.

Service Layer

On the *MIGRA* service layer, data exchange formats are defined for both query and contextual data. Both services defined are based on the current reference implementation of JXTAs peer discovery protocol (PDP) and peer resolver protocol (PRP). Reasons for

the implementation of the PDP and PRP into the *MIGRA* services are discussed in sections 4.5.1 and 4.5.2.

Application Layer

Implementation of this architecture implies *MIGRA* peers live on the application layer, utilising the functionality provided by the *MIGRA* services in conjunction with the JXTA services. Scalability in terms of additional applications or added functionality to those extant is a feature that is inherent in JXTA applications located in the application layer. This attribute is prominent feature inherited by *MIGRA*.

All *MIGRA* applications are completely separate and dispersed across the network on remote devices in an ad hoc manner. User interaction at each peer application is independent of all others. The next section describes a typical *MIGRA* network where each peer depicted would represent a separate instance of a *MIGRA* application that in turn represents a user.

Network Topology

No peer ever knows the entire network orientation. Discovery of each user is performed in an ad hoc and pervasive manner. The following diagram is a typical scenario where there may exist five users active on the network who reside in four separate smart spaces. Each user, through means of the *MIGRA* application, will have the ability to discover every other user. The communication paths of the discovery are also indicated. This topology is synonymous to that of pure peer-to-peer.

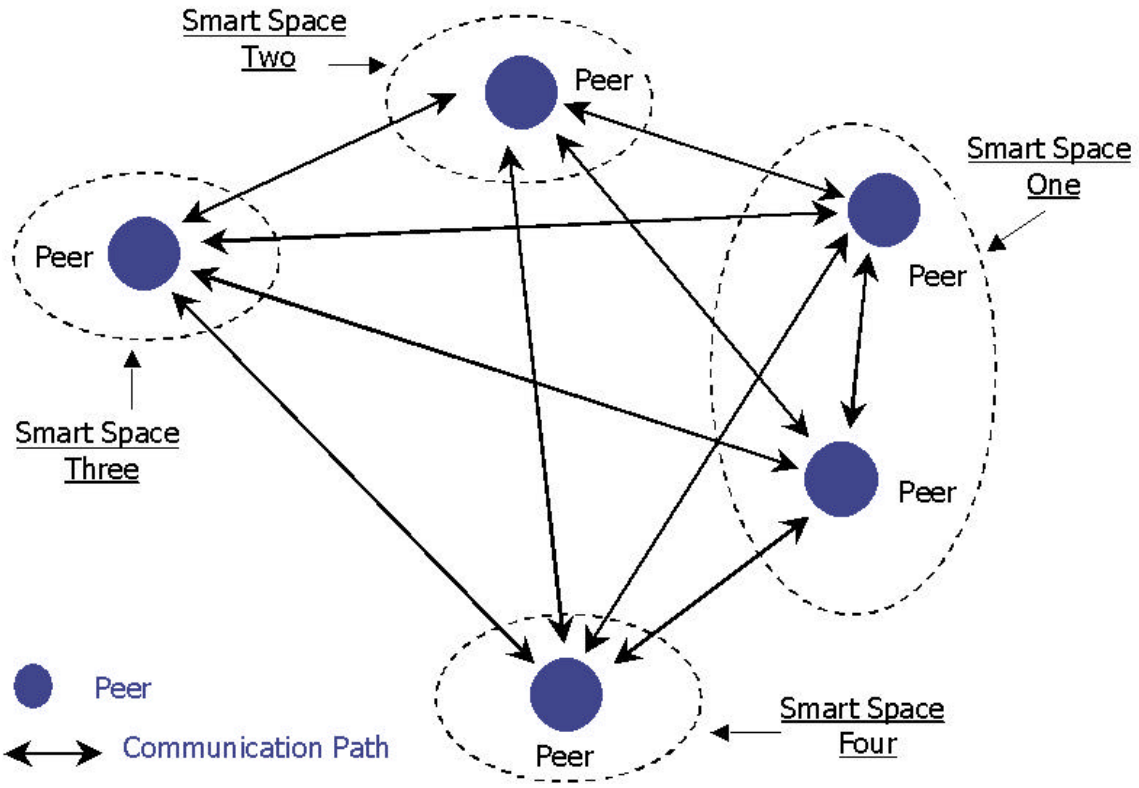


Figure 4.4: Topology of a typical *MIGRA* network.

This network virtualisation is independent of the underlying heterogeneous physical transports that may exist. An explanation describing this is beyond the scope of this writing but is outlined in [JXSPEC]. The following section describes the design of *MIGRA*.

4.5 *MIGRA* Design

This section attempts to describe how the modules of the architecture interact and the purpose behind their construction. *MIGRA* is based on the pure P2P architecture described in section 2.3.2. *MIGRA* utilises much of JXTAs core functionalities that are described in chapter 3. This section will deal explicitly with the components that were built to work in conjunction with JXTA in order to provide an infrastructure to migrate users from one smart space to another.

4.5.1 Singleton Service

The primary purpose of the singleton service is to ensure that there is only one application active per user on the *MIGRA* network. The Singleton Service must also support a means to provide users with the capability to gain domain awareness, i.e. notification of the users in their smart space domain when requested. This means the singleton service has to be designed generically so that different forms of messages can be processed. Rather than to add a completely new message passing protocol to handle each separate function all information should be embedded into each singleton request and response message. The Singleton Request Message must consist of two fundamental elements.

- Terminate – This element type will inform *MIGRA* applications to terminate (figure 4.5, step 14). If the value of this element is null then the request will be ignored by the remote application.

- Information – This element type will request applications to return information containing the current proprietor and the exact location of the application. Again, the null value of this element means the request will be ignored by the processing application.

For each of these message requests there will be an appropriate response. The following elements make up the Singleton Response Message:

- Terminated – is the response to the *Terminate* request message. This will be only returned by the one application that is requested to terminate (figure 4.5, step 15).
- Status Information – is the response to the *Information* request. This response message will contain two attributes, the identity of the user associated with the particular responding application and the location of that user.

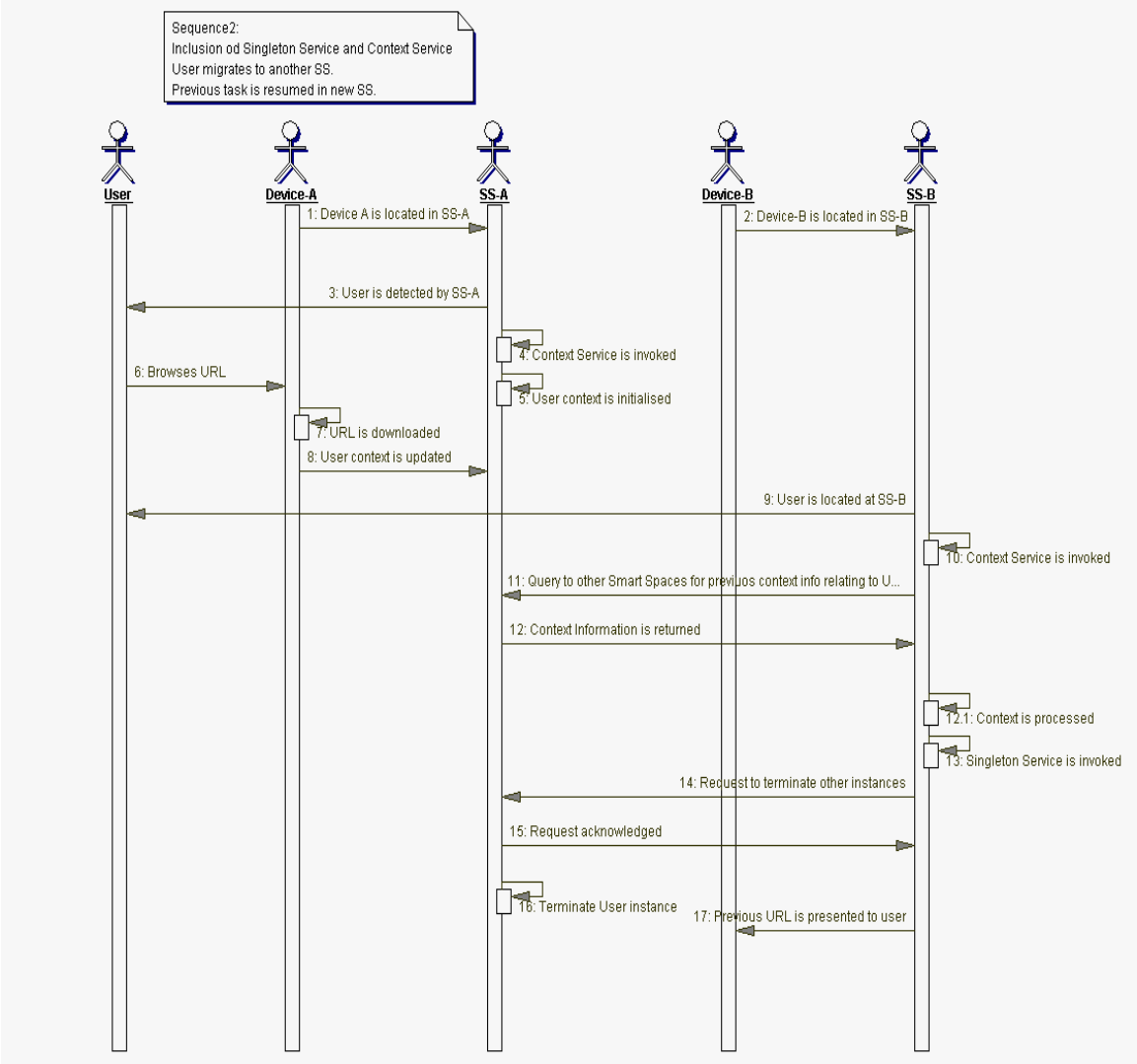


Figure 4.5: The incorporation of Singleton and Context Services.

In order to process the singleton messages there must exist a means for the application to be notified when such messages are received. The singleton service employs listener objects as a way to receive notification of arriving messages. Each message arrival will signal a singleton event. The following diagram represents this design.

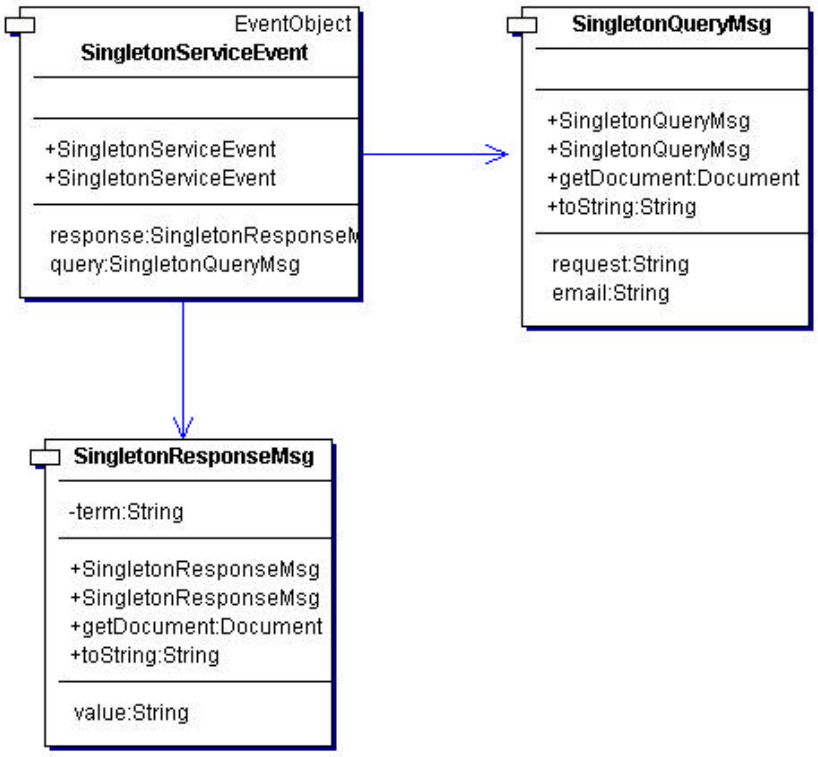


Figure 4.6: Singleton Service.

This design implies that there will also be only one copy of users context information present on the network at one time. The Context Service is designed with this in mind.

4.5.2 Context Service

A key challenge in the design of the context service is the management of users context in the face of constant change. A single web browsing session can include a significant number of URLs for any one user. This means a users context will need to reflect such actions. The Context service is designed with this diversity in mind and is illustrated in figure 4.7.

The operation of the context service will be managed by five interconnected operations. The following explains these operations in more detail.

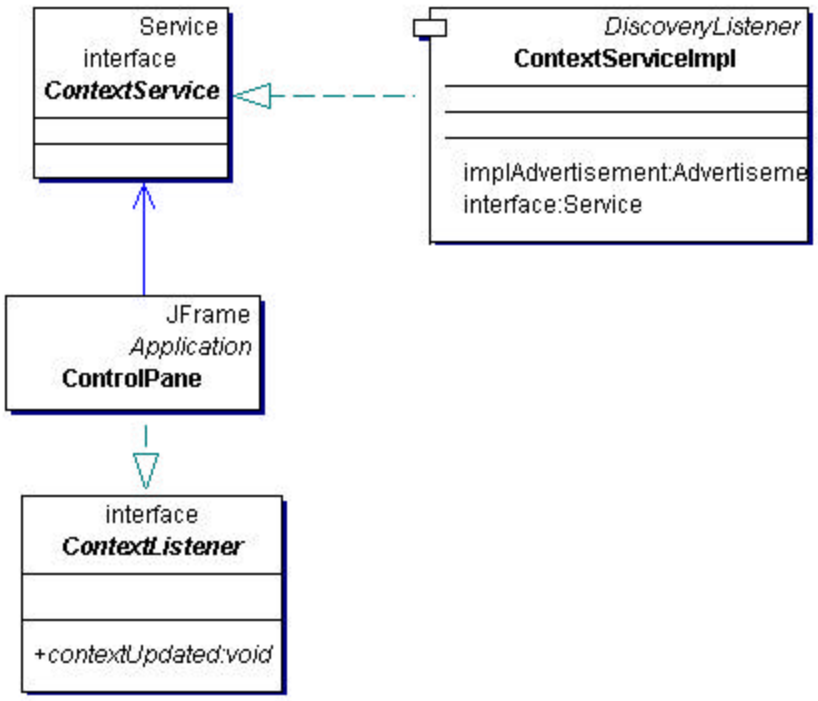


Figure 4.7: Context Service.

Context Generation

When a user enters a room the detection simulation receives the users unique identifier. The next step is to determine if the user has been active at another location or is entirely new to the *MIGRA* network. If the user is active then the previous context is retrieved via the discovery service provided by the context service. If the user is new, then the system must accumulate the context information over time as the user interacts with the system.

Context Persistence

The role of this function is to store the users context information locally for retrieval by other applications. This caching is achieved via JXTA.

Update Context Information

The context service must be capable of echoing the persistent changes made by users and ensuring that this information is then cached and up-to-date when requested.

Context Discovery

If users were active on the network then the context system will provide a method by which to retrieve their most recent context information. This is achieved via the discovery service provided by JXTA.

Process Context Information

In all operations there must be a means to process the context information in order prompt the system to act appropriately. This method will consist of parsing the context messages and retrieving the desired information.

A fundamental aspect to the context service is the format of the context itself. This next section outlines the context format that complements the proposed architecture.

4.5.3 Context Information

Context embodies all information pertinent to a user. Dey [DAS99] defines context to be any information that can be used to characterise the situation of an entity, where an entity can be a person, place or physical or computational object. Cooltown [CoolT] on the other hand defines context information into three properties, *identity*, *location*, *and browsing capabilities*. *MIGRA* should encapsulate both definitions and provide a basis through which the context is extensible. *MIGRA* context information should be capable of containing information related to users most recent actions such as previously accessed URLs, exact timings in an MP3 song, and the users preferences. Each *MIGRA* context object advocates the following attributes:

- Email Address – uniquely identifies and distinguishes users in the smart space domain.
- Preference – is an attribute that is used to contain the users URL. This provides a means to associating a web presence with users of the system.
- Location – this attribute will be hard-coded into the system and will specify the location that the application resides. Location will be used to ascertain a users last known location on the network.

- Activity – the purpose of the activity attribute will be to contain information as to what activity the user last executed. This should range from web browsing to audio streaming. However, for this prototype the user will be confined to browsing.
- URL – will be a child attribute of activity. URL specifies the last known URL that a user accesses. The purpose of the URL attribute is to identify a resource in such a way as to enable the browser to locate that resource.
- Time – the primary function of this attribute is to allow easy extension of the application for such activities as audio streaming and/or also to determine the time an activity was executed. This attribute could be used to determine the exact location in a song that a user last heard. This would then be used as the starting point of the song when the user enters the new domain.

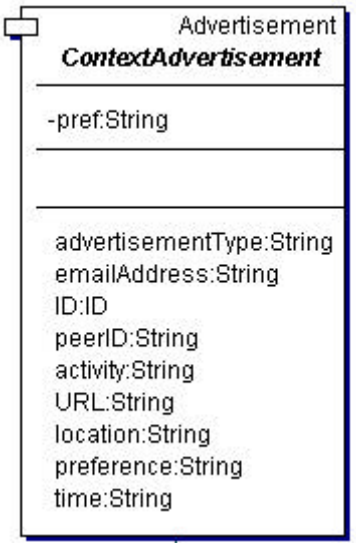


Figure 4.8: Context Advertisement.

As the user progressively interacts with the system the context object should populate. The final attribute should be retrieved when the user interacts with the web browser.

4.5.4 Application Interface

To ensure efficient utilisation of these services *MIGRA* needs to provide a means to track users actions whilst in a smart space. Similarly to that proposed by O' Sullivan [OSW02] and underlined in the requirements section *MIGRA* should provide a graphical user interface to allow users to execute the core functionalities of everyday computing such as web browsing and audio listening. Another requirement of such an interface for this system is to ensure users have the capability to view other users in their domain and their whereabouts. Such an application will provide a central point of control where the scope of the user input can be monitored. The following class diagram proposes these functionalities and also incorporates the application interface as a listener for events occurring in the singleton and context services. This provides a way for applications to receive notification of arriving messages.

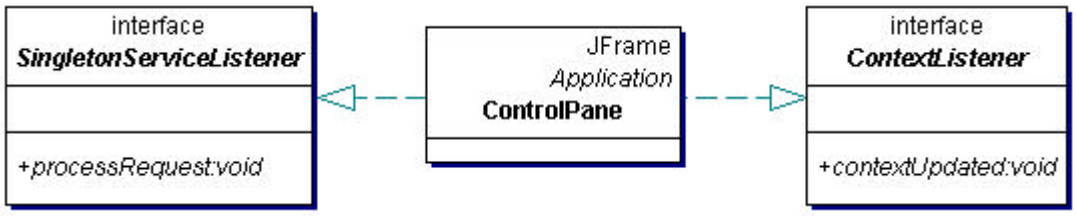


Figure 4.9: Class Diagram of Application Interface.

4.6 Summary

In this chapter the requirements, architecture and design of prototype *MIGRA* was outlined. Also presented were two hybrid architectures that outlined how the system might have been implemented. The following table is illustrates how the requirements outlined impacted on the various modules of the architecture.

Requirement	Module	Description
Empowerment of Spaces:	Context Service	
Multi-user:	All modules	Through replicating the peer on various devices throughout the network the system becomes multi-user, where users are independent of each other.
Awareness of User Context:	Singleton Service, Context Service	
User detection:	Singleton Service, Context Service	
Ubiquitous Services:	All modules	Through replicating the peer on various devices throughout the network the services would be readily available.
Empowerment of Users:	ControlPane GUI	
Web Presence:	ControlPane GUI	This requirement is not fully realised for reasons described in section 6.1

The following table details how the requirements of the infrastructure impacted the design of the system.

Requirement	Module	Description
Global Consensus of Context:	Singleton Service, Context Service	A combination of both services contributes to ensuring that user context is unique and valid.
Information Persistence:	JXTA	JXTA provided a Cache Manager that is transparent to the developer. This allowed the persistence of the user context.
Proactive Guidance:		This requirement is an objective of future work.

At various stages during these assessments we have highlighted *MIGRA*s dependency on the JXTA platform and the fundamental role it plays. The next chapter will describe the implementation of the system and the reasoning behind some of the implementation decisions.

Chapter 5

Implementation

5.1 Introduction

This chapter outlines the implementation of *MIGRA* based on the design presented in chapter 4. Section 5.2 describes the platform used to deploy the system on. Sections 5.2 to 5.8 present each module of the system shown in Figure 5.1. For each module a description of implementation is included and issues that were encountered during implementation are discussed.

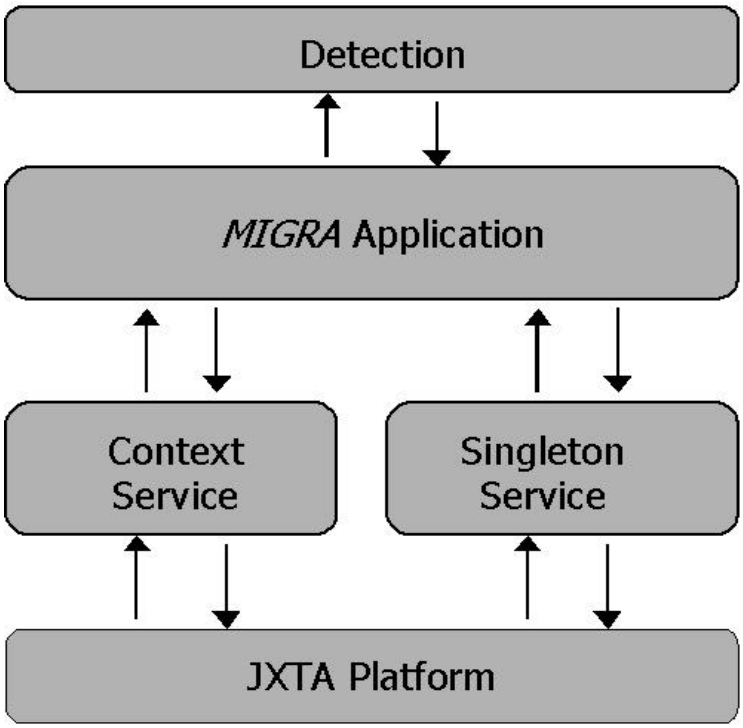


Figure 5.1: Basic components and interaction between components.

5.2 JXTA Platform Module

JXTA is an active project with stable releases of the platform available every few months. Implementation of *MIGRA* used the latest stable release, 65e, dated July 2002. The JXTA platform required the Java 2 platform standard edition SDK, at least version 1.3.1 is necessary.

5.3 User Detection Module

The main functionality of user detection is to inform the Smart Space that a user is present and to derive the users identity. Implementation of user detection had to be simulated due to the fact there was no access to off-the-shelf sensing equipment in the project. This simulation was achieved through presenting the user with a “detection” graphical user interface (GUI) that was tied into the *MIGRA* application. The purpose of the detection GUI was to ensure that the user entered their unique identifier, i.e. email address, before allowing them access to the main application. Users also have to indicate if they are new to the smart space or are migrating from another smart space. New users will be prompted to indicate their web presence before being allowed access to the *MIGRA* application. The following flow chart represents this procedure.

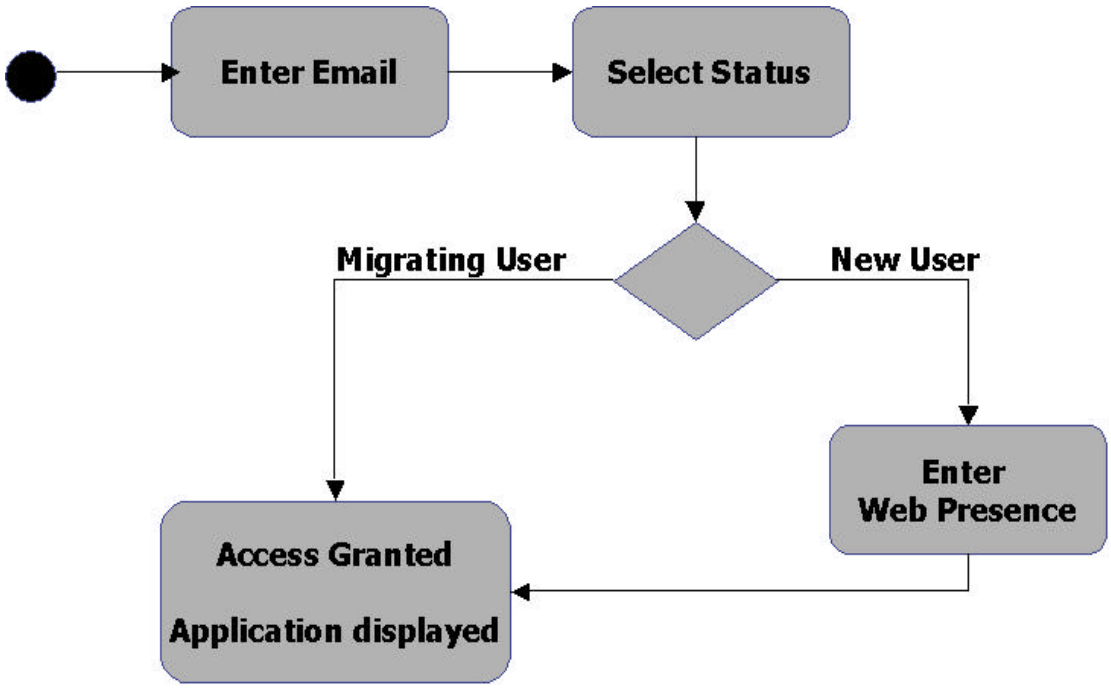


Figure 5.2: Implementation of User Detection.

The following diagram is a class diagram representation of the user detection implementation. From the relationship between the classes, it is clear that the simulation was tied into the *MIGRA* application.

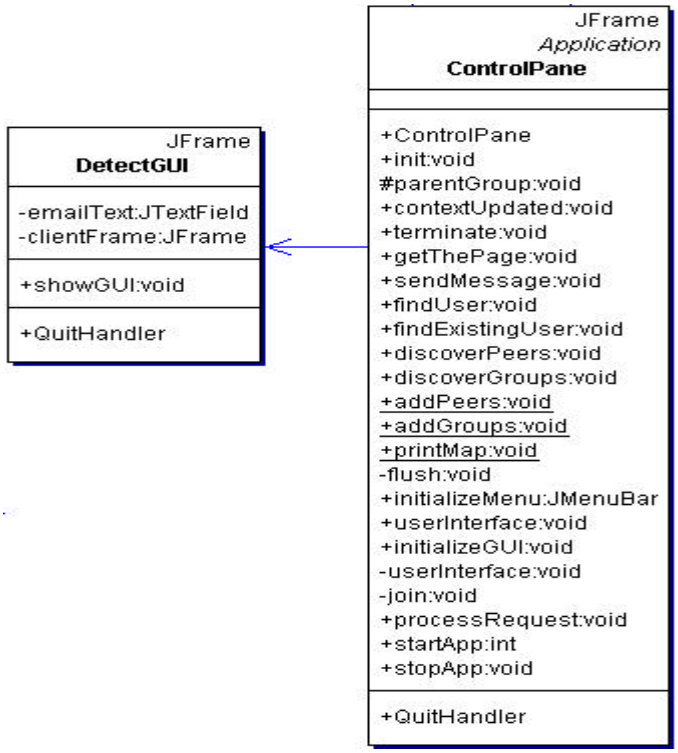


Figure 5.3: Class diagram representing User Detection.

The decision to tie user detection into the main application was to ensure the process in some way modelled real life detection, i.e. sensing of user and retrieval of the user identity. The user detection was designed to serve this particular prototypes purpose and it is recognized that further modelling would be necessary to implement real life sensing.

5.4 Context Information Representation

Users context information should encapsulate all facets that relate to users latest activities in Smart Space environments. A break down of the elements of a users context advertisement is described in section 4.5.3.

```
<?xml version="1.0"?>
<!DOCTYPE jxta:ContextAdvertisement>
<jxta:ContextAdvertisement xmlns:jxta="http://jxta.org">
        <Type>
                SmartSpaces
```

```
        </Type>
        <PeerID>
                urn:jxta:uuid-
                59616261646162614A787461503BB9A0F7801DBE71FAED03
        </PeerID>
        <EmailAddress>
                martin@cs.tcd.ie
        </EmailAddress>
        <Activity>
                browsing
        </Activity>
        <Url>
                http://localhost
        </Url>
        <Location>
                SSA
        </Location>
        <Presence>
                http://www.cs.tcd.ie/Martin.Commins
        </Presence>
        <Time>
        </Time>
</jxta:ContextAdvertisement>
```

The context advertisement employs a format that allows for the efficient exchange of the context information with other peer applications.
This exchange is achieved via the Context Service, the implementation of which is described in section 5.5. Implementation of this advertisement is loosely based on the general advertisement implementation approach that is described in Wilson in [Wil02].

Context advertisements are created from the abstract class **ContextAdvertisement** that is derived from JXTAs **net.jxta.document.Advertisement** class. Contained in the **ContextAdvertisement** class are the basic accessor methods that are used to retrieve the advertisements parameters, such email address, location and activity etc. Also included in this class is a method that returns the numeric ID that is used to uniquely identify the content in the cache.

A subclass is derived from the **ContextAdvertisement** class that contains all of the parsing functionality of the advertisement. JXTAs **net.jxta.document** class provides this parsing and formatting functionality of the XML advertisements. The following class diagram represents the relationship between these two advertisement classes.

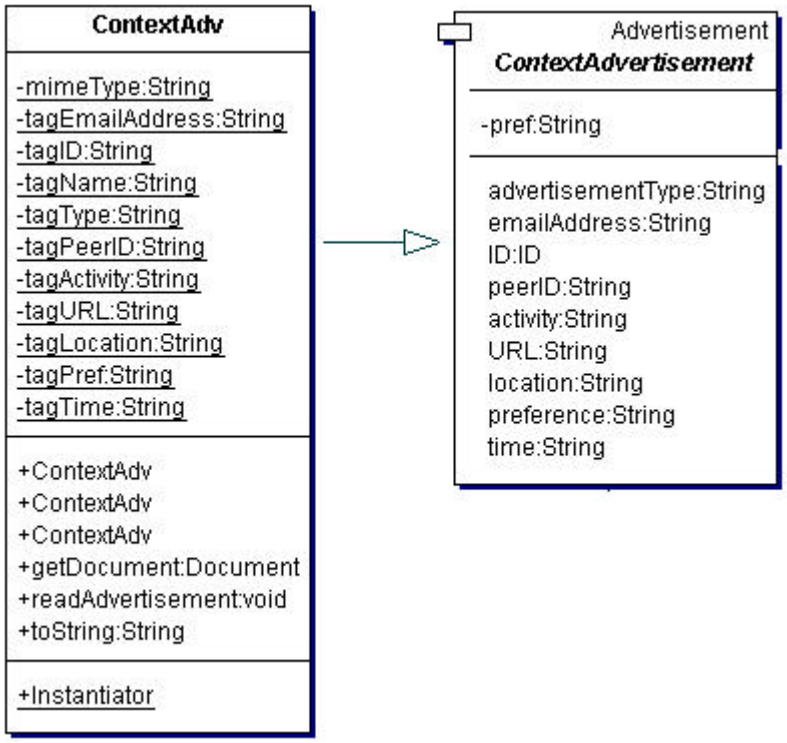


Figure 5.4: Context Information.

The major difficulty in designing the context advertisement was the decision of what information to include in each context advertisement. The contents of each advertisement make the fundamental connection between user and their activities in the Smart Space. This context advertisement was designed to serve this prototyping purpose only and it is recognized that further modelling is required and is a research issue in its own. The core information held is the name of the user, the location of the user and the activity of the user. All this and more information is encapsulated in an XML document. An example listing is described above.

5.5 Context Service

As described earlier in section 4.5.3 the main functionality of the context service is to ensure the effective exchange of context information between Smart Space systems. This service provides a fundamental link between the underlying platform and the user interface, as illustrated in figure 5.1.
The **init()** method of the context service provides it with a peer group that it uses to obtain the core discovery service. Rather than implement its own discovery techniques for the location of specific context advertisements the context service uses JXTAs discovery service.

The discovery service is bound to the peer group instance. The scope of this service and the context service resides within the group to which it is associated. This means that only peers that are active members of the group will receive any request made via the context service.

The fact that such services reside within the group context means that peers migrating from other Smart Space domains could not be catered for in the sense that their context information could not be retrieved. One solution to this was to create the context service so that it would span other Smart Space domains. Such a design would require a priori knowledge of the other domains upon which it would span. Because of the non-deterministic nature of users and their movements, to anticipate the domains the context service should span would be a difficult task. As stipulated by Brookshier in [BGK02], it is more effective to create an application that joins multiple domains. This was the approach implemented.

The **ContextService** interface provides the two methods that provide its basic functionality, **persistContext()** and **findContext().** The first method uses the discovery service to cache the context information locally.

```
discovery.publish(contextInfo, DiscoveryService.ADV,
                                                default_lifetime);
```

The parameters of this method call are the actual context information, the classification of the advertisement type and the time limit before the advertisement becomes stale and is removed from the cache.

The second method is a utility method provided to limit the scope of queries for context information. A query for a particular context advertisement uses the **getRemoteAdvertisements()** method that queries all remote members within the group.

discovery.getRemoteAdvertisements(null, DiscoveryService.ADV,
tagEmailAddress, emailAddress, 0);

Such a request returns context advertisements that contain the tag **tagEmailAddress** that contains the attribute **emailAddress**. If no such context exists then no advertisement would be returned.

Notification of discovered advertisements invokes the **discoveryEvent()** method. This method is located in the **ContextServiceImpl**. This class contains the implementation methods that are defined in the service interface **ContextService**.

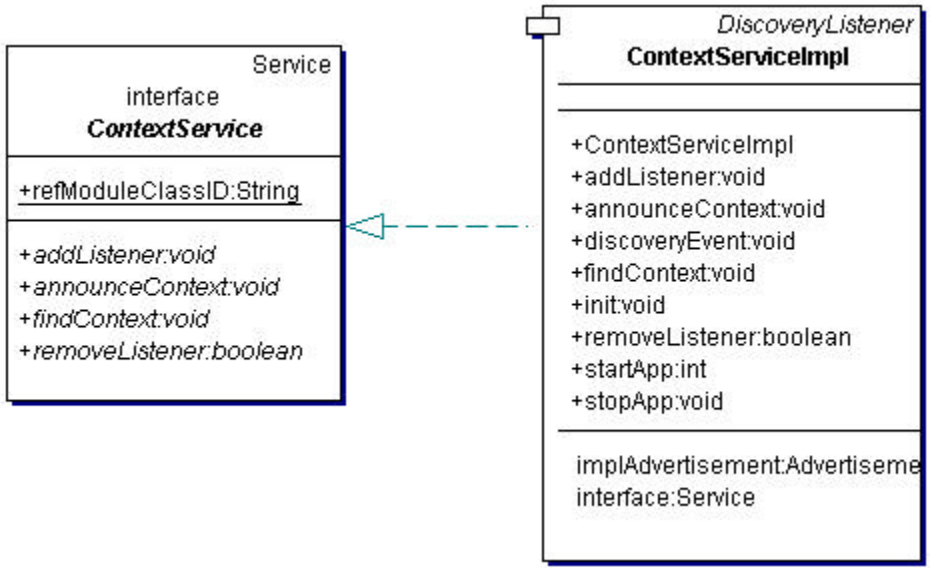


Figure 5.5: Context Service

The **discoveryEvent()** method contains references to the parsing logic described in the **ContextAdvertisement** class. In *MIGRA* this information is dispatched to the listener objects of the context service. An example of such a listener object is the detection application. This application could receive notification of the context information of a user that has just been detected as migrating from one Smart Space to another. This information would then be used to allow the users to continue their most recent activity in their new Smart Space location.

5.6 Singleton Service

As section 4.5.1 indicated the purpose of the singleton service is two-fold:

- To ensure there is only one application active per user on the network.
- Provide a dynamic list of users active on the network.

The purpose behind the first requirement is to ensure that no two separate peer applications contain user context relating to the same user. Initially it was hoped that user context relating to all users would be cached at various locations around the network. However, this solution was not implemented for two reasons. The first reason was that it was not possible for the *MIGRA* application to locally discover context advertisements that were cached from a remote peer application. The reason behind this is thought to be a fault with the registration method of advertisements with JXTAs **AdvertisementFactory**. After numerous requests to the JXTA mailing lists no solution was established. The second reason was bandwidth consumption. It was thought that on a highly populated network such traffic resulting from many users moving intermittently would be undesirable particularly as many users would only visit a subset of locations on a network. However, because of the initial reason this theory was never investigated.

The second purpose of the singleton service is to provide a means to query members on the network of their existence. Active peer applications representing users would reply to this request by returning their name and location. This, in effect, provides the user with

awareness of their Smart Space domain by providing them information on the other users that populate the domain. The following describes the implementation of the singleton service.

The interface class **SingletonService** defines the methods that must be implemented in the **SingletonServiceImpl** class. This relationship is depicted in figure 5.6.

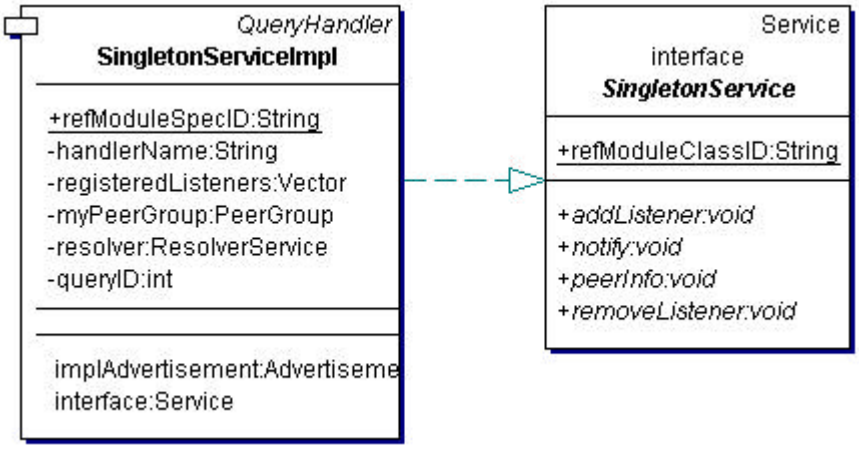


Figure 5.6: Singleton Service.

Shown below is the code for the interface for the singleton service.

```
import net.jxta.service.Service;
import com.newriders.jxta.chapter11.context.Context;


public interface SingletonService extends Service
{

   public static final String refModuleClassID =
      "urn:jxta:uuid-128E938121DD4957B74B90EE27FDC61F05";

   public void addListener(SingletonServiceListener listener);

   public void notify(String request, Context context);

   public void peerInfo(String request, Context con);
```

```
    public void removeListener(SingletonServiceListener listener);
}
```

The variable **refModuleClassID** is a unique identifier that is only associated with the singleton service module and is used to find the service on a peer group. This service lookup method is described later. The remainder of the code snippet underlines the methods that must be implemented by the singleton service implementation. Each of these methods is now described.

The method **addListener()** simply adds objects as listeners for events occurring in the singleton service. Mehod **removeListener()** simply removes the object as a singleton service listener.

The remaining two methods contain the core functionality of the singleton service. The first method, **notify()**, is called when a user has been detected in a Smart Space and has just migrated from another location. The parameters of this method include the request and email address of the detected user. The request element contains a string requesting a remote peer application that is associated to the email address to terminate.

```
SingletonQueryMsg equery = new
SingletonQueryMsg(request,con.getEmailAddress());

// Wrap the query in a resolver query message.
ResolverQuery query = new ResolverQuery(handlerName,
        "JXTACRED", localPeerId, equery.toString(), queryID++);
```

The above code snippet achieves this. The message is wrapped in a Singleton Query Message and then in turn wrapped in a resolver message. As previously mentioned the singleton service uses the Peer Resolver service provided by JXTA for request/response type messaging. The resolver service then publishes the message to the network. Any peer application that is associated to the email address contained in the request will terminate. But before terminating the process will return a response to indicate that it is about to terminate. This code comprises of:

```
String answer = "terminated";
er = new SingletonResponseMsg(answer, mailAddress,
                                                    smartSpace, URL);
 // Wrap the response message in a resolver response message.
 response = new ResolverResponse(handlerName, "JXTACRED",
                                                  query.getQueryId(), er.toString());
```

Method **peerInfo()** works in exactly the same manner as that described above. The difference is in the request parameter. This method is used to query all peer applications on the network of the user that is associated with the application and the location of the peer application. When a peer application receives the singleton query message it parses the message. When the element **<Request>**is parsed the process retrieves the value of this element. If the value is **terminate** then the action described for the **notify()** method is taken. If the value is **peerInformation** then the process creates a singleton response message and embeds the name of the user associate with the process and the location of the process. In a manner similar to that already described the response message is wrapped in a resolver message and returned to the requesting peer application.

Having described how both *MIGRA* services were implemented the next step is describe how these services are started as peer group services and the creation of the actual peer group.

5.7 Smart Space Domain Initialisation

Before deploying the context and singleton service as group services it is first necessary to create the group to which they reside and then add the services to the group's parameters. The following steps describe this process. The key concept here is that any reference to a peer group actually represents a Smart Space domain in the *MIGRA* system.

1. **Obtain the discovery service of the parent group:**

In the case of *MIGRA* the parent group is the NetPeerGroup. The parent's discovery service is eventually used to publish the peer group to the network.

```
DiscoveryService discovery = netPeerGroup.getDiscoveryService();
```

2. **Obtain a preformed Module Implementation Advertisement:**

   In *MIGRA* the NetPeerGroups module implementation advertisement is used. For this process the parent group can be any group. The reason the NetPeerGroup was chosen is so the system can avail of the core services, such as the discovery service and resolver service, which are provided by the NetPeerGroup group.

```
ModuleImplAdvertisement implAdv =
        netPeerGroup.getAllPurposePeerGroupImplAdvertisement();
```

3. **Create the Module Advertisements:**

   This step involves the creation of the Modules classes described in section 3.6.1. Each service requires it own unique *Module Class, Module Specification* and *Module Implementation Advertisements.* The following code snippet involves the creation of the Module Implementation Advertisement describing the context service.

```
ModuleImplAdvertisement contextImplAdv = createModuleImplAdv(
        implAdv, contextSpecAdv,
        "The reference Context service implementation",
        "ie.tcd.jxta.migra.impl.context.ContextServiceImpl");
```

4. **Add the context and singleton services:**

   Using the NetPeerGroups module implementation advertisement obtained in step 2, the context service and the singleton service are then added.

```
services.put(contextClassAdv.getModuleClassID(), contextImplAdv);
services.put(singletonClassAdv.getModuleClassID(),
                                        singletonImplAdv);
```

These parameters are then set on the implementation advertisement of the new *MIGRA* peer group.

```
implAdv.setParam((StructuredDocument) params.getDocument(
    new MimeMediaType("text", "xml")));
```

5. **Publish the advertisements:**

   All module advertisements are then published to the network. The code below describes how the module class of the context service is published both locally and remotely using the discovery service obtained in part 1.

```
discovery.publish(contextClassAdv, DiscoveryService.ADV);
discovery.remotePublish(contextClassAdv, DiscoveryService.ADV);
```

   The parameters contain the module class and the classification of advertisement that the module is. In this case the advertisement is the third classification, which describes every resource such as services and context information, except peers and groups.

6. **Create a group advertisement and publish it:**

   The next step is then to create a new group advertisement and publish the group advertisement to the network.

```
PeerGroupAdvertisement groupAdv =
        newGroup.getPeerGroupAdvertisement();
discovery.remotePublish(groupAdv, DiscoveryService.GROUP);
```

   The parameters of the **remotePublish()** method are the group advertisement and the type of advertisement, in this case it's a group type.

7. **Start the services:**

The final step is then to start all of the services, as described by the following code.

```
init(newGroup, groupID, implAdv);
parentGroup(netPeerGroup);
startApp(null);
```

This code could have been replaced by the following snippet that would initialise and start the services of the group in an encapsulated fashion. However, the *MIGRA* system requires a reference to the parent peer group to provide users with the capability to monitor all of the groups active on the network.

```
newGroup.startApp(null);
```

As described in section 3.6.1 modules also describe applications. The following section details the *MIGRA* application implementation.

5.8 The *MIGRA* Application

As outlined in the design in chapter 4, the purpose of the *MIGRA* application is to provide users with an application that allows users to intuitively execute the core functionalities of everyday computing such as web browsing and audio listening. However, the actual application services offered to the user were of secondary concern and for the purposes of this project the only service implemented by the application is web browsing. This application service was considered to have the kind of characteristic that would test our design, namely the frequent updating of context information as the user moved between URLs.

In conjunction with this browsing service the application also provides the user with the ability to view other members in their smart space domain and their associated web presence. Implementation of this was achieved via the singleton service, which is

outlined in section 5.6. The information that is returned from the singleton service contained the other users email address, location and URL representing their web presence. This information was then displayed in the application GUI. Also included into the application is the ability of the users to view other smart space groups that are active on the local network. This is now described.

Step 7 of the previous section highlighted that the *MIGRA* application required a reference to the parent peer group. This reference allows discovery of groups that also have the NetPeerGroup as the parent group. Obtaining this reference posed some difficulty during implementation. Initially it was thought that the **startApp()** method was the only means by which services of peer groups could be initialised and started. However, it was eventually realised that direct invocation of the initialisation and start methods was possible. Moreover, this meant a new method could be implemented that could receive a reference to the parent group as a parameter after initialisation and before starting the service.

Once the user chooses the option of retrieving information on other Smart Space domains, a thread is created that will discover all the peer groups, i.e. Smart Space domains, which are active on the network. This thread stays active waiting for other domains to be discovered until the peer application is terminated. The following code represents a remote request for group advertisements to be returned.

```
disco = parentGroup.getDiscoveryService();
disco.getRemoteAdvertisements(null, DiscoveryService.GROUP, null,null, 0);
```

Also discovered could be the children of these groups and so on. What the NetPeerGroup provides is essentially a view of all groups below it. This functionality is beneficial as group discovery is limited to the actual group and the children of that group. As the NetPeerGroup is the root of all peer groups this provided a global view of all peer groups below it, which in the case of *MIGRA* provided a global view of all the smart space domains present.

Another aspect of this is that *MIGRA* provides the functionality of users to actually join the groups discovered. For implementation purposes all groups that were present on the local network contained the same services as the group described in the previous section. The ability to join other domains contained no security measurements. JXTAs *MembershipService* could have been employed whereby a peer applies for membership to the group. Membership to the group could be limited to specified users. These users would be passed credentials upon which the group membership service could validate every time a user attempts to access a group service. However, the current implementation of JXTA has not fully implemented this concept and membership to a group can be achieved by simply instantiating the peer group and skipping the membership procedure. This remains an objective of the future work for both JXTA and *MIGRA*. The following describes *MIGRAs* process of allowing users to join different domains.

Smart Space Domain advertisements that are discovered by the **PeerGroupDiscoveryThread** are stored in a mapping of the group name to the group advertisement. This information is stored in Java's utility class **HashMap**. This structure provides a one-one mapping of Smart Space domain names to Smart Space advertisements. By scrolling onto the name of the group from the GUI the peer application and the user, then becomes a member of that domain. This functionality is contained in the following snippet.

```
groupChooser.addListSelectionListener(
      new ListSelectionListener() {
         public void valueChanged( ListSelectionEvent e)
         {
           String group_name = groupChooser.getSelectedValue().toString();
           PeerGroupAdvertisement pg_adv =
                 (PeerGroupAdvertisement) groups.get(group_name);
           join(pg_adv);
         }
       }
    );
```

The join method referenced in the above code automatically instantiates the domain advertisement, **pg_adv**, to allow the user to join the group. This method is described below.

```
private void join(PeerGroupAdvertisement pgAdv)
{
    try
    {
      currentGroup = parent.newGroup(pg_adv);
    }
    catch (PeerGroupException ex)
    {
      System.out.println("could not instantiate peerGroup");
    }
}
```

To obtain the core discovery service of the new group, all that is executed is following:

```
currentGroup.getDiscoveryService();
```

To avail of the context service of this group the following code is executed:

```
ModuleClassID class_ID = (ModuleClassID) IDFactory.fromURL(new
                              URL((ContextService.refModuleClassID)));
ContextService cSrvc =
                (ContextService) newGroup.lookupService(class_ID);
```

5.9 Conclusion

This chapter has described the implementation of the *MIGRA* system introduced in chapter 4. The next chapter outlines the conclusions reached, work accomplished and goals achieved during the course of this project. Also proposed in the next chapter are possible areas of future work.

Chapter 6

Conclusions

6.1 Introduction

The main goal of this dissertation was to design and implement a system that supports the migration of user's context information across smart spaces. In chapter one the core objectives of this dissertation were described as:

- Allow the migration of a user from one Smart Space to another by simulating various Smart Space environments.
- Simulate the detection of users entering a Smart Space.
- Generate, store, process and migrate the users context.

By investigating related research projects in the area of user migration between Smart Spaces this writing proposed several requirements that such a system should support, in order to achieve the objectives stated. The following section evaluates the resultant implementation against these requirements.

6.2 Evaluation of Implementation against requirements

Empowerment of Spaces, *Awareness of user context* and the allowance for *multiple users* were the primary requirements upon which implementation focussed. This section provides an evaluation on how these requirements were implemented in *MIGRA*.

- **Empowerment of Spaces:**

  Retrieval of the users context information is abstracted through the user detection simulation process and user interaction with the main application. The information retrieved from the user is processed by the context service, which ensures an accurate encapsulation of the context information to reflect the users actions within the Smart Space. Each Smart Space comprises of multiple users interacting with multiple applications. Each application contains the context information that best describes the activity of the user and their action whilst in the Smart Space.

- **Awareness of User Context:**

  Global consensus of the user context in *MIGRA* is achieved through ensuring that there is only one instance of users context on the network. The impact of this design prompted the implementation of the singleton service. This approach was adopted so that there could be only one peer application that would be actively responsible at any one time for the context information for a user and that peer would be located in the Smart Space where the user is currently active. However, as discussed in section 6.2, a drawback of this design is that a user can only have one application active in the Smart Space to which they reside.

- **Multi-user:**

  Each device in the system contained a peer application. Each peer application represented a single separate user. These constraints meant that every active peer application on the network signifies the existence of a user. Therefore, *MIGRA* is a system that supports multiple users migrating within a Smart Space operator domain. Future development of *MIGRA* could investigate associating multiple users with single devices, as in the case where users would share computer terminals.

Although the three requirements described above where focal to the design of *MIGRA* they were not the only requirements to be considered and implemented. The following presents and evaluates the remaining requirements that permeated the design of *MIGRA*.

- **Empowerment of Users:**

  MIGRA incorporates an intuitive user interface, *ControlPane*, coherent to that of everyday web browsing applications. Also included are two basic sidebars that allow the users to view other users in their smart space domain and other Smart Space domains active on the network. By simply scrolling on the name(s) of the other users in the Smart Space domain the user will be presented with the associated web presence of the member. This presence will be displayed in the web browser section of the application. By selecting a Smart Space domain from the sidebar the user can become a member of the selected domain and avail of the services offered. The current implementation of *MIGRA* consists of different groups that all comprise of the same *MIGRA* services and application interfaces. Future evolvement of the *MIGRA* architecture should address this to allow for users to dynamically join groups and avail of their unique services.

- **Web Presence:**

  Web pages are associated to users but they are not dynamically created. The pages are static and should exist before start-up. Viewing of users web presence is described in the previous section. The Cooltown project [HPDC] is the origin of the web presence concept and describes full implementation of this concept into the Cooltown Smart Space system. This approach could be considered for future implementations of *MIGRA*.

- **User Detection:**

  Simulation of the detection of users entering a smart space was achieved via a series of graphical user interfaces. Users migrating had simply to enter in their email address. This prompted the context service that would retrieve the users context from the previous remote location users had interacted with. New users

are prompted with a secondary step, the indication of their web presence. One area of consideration could be the interaction of *MIGRA* with real-life multi-modal sensors.

- **Ubiquitous Services:**

  The current implementation only provides a web browsing service to the user. Even this service is limited in its capabilities as it is based around Java's JEditorPane that is used to display URLs. This approach is constrained in the sense that it cannot render URLs that incorporate cascading style sheets (CSS) [CSS], graphics and certain images.

6.3 Evaluation of *MIGRA*

Experience from the prototyping work indicates the following:

- The P2P approach holds promise for building a Smart Space environment that is characterized by the need to dynamically sense, recognize, understand devices/services and reason about their functionality at runtime. However, P2P is not the panacea of Smart Space computing. P2P's flexibility can be its own downfall. Difficulty in the control of admittance of users to the network can result in exponential growth that could drain network resources. On the other hand, dynamic information repositories, redundancy, fault tolerance and content reflective addressing are potential benefits of P2P. These attributes coupled with the possible independence from the underlying physical network transport imply P2P as a suitable candidate upon which to base Smart Space environments.

- Sun Microsystem's JXTA platform, although still immature, provides good support for development of P2P networked applications through its abstraction of the core protocols that represent the essentials to P2P computing. The employment of XML provides a well-understood and supported format for data

exchange. On the other hand, as JXTA is a relatively new technology some protocols are as of yet not fully implemented. For example, aspects such as trust and security are not fully implemented by JXTA. Implementation of the membership service can be overridden through simple instantiation of group advertisements. This means that becoming a member of a group is a simple task. This has the potential for unwarranted membership of unauthorised users to groups. Another area of some debate within the JXTA development community is JXTAs abstraction from the underlying physical transport. Some argue that JXTA should sacrifice this abstraction for performance by focussing on applications that rely on TCP networks.

However, in defence of JXTA, once the learning and understanding involved in a new technology was completed, the entire coding of the *MIGRA* system was accomplished in a period of eight weeks. Therefore, *MIGRA* is a positive recognition of the capabilities of JXTA as a peer-to-peer development platform.

- *MIGRA*s design needs to evolve, such that a user's context is not tied to the peer in the Smart Space in which the user is currently active. By distinguishing between a user context peer and a user application peer, we will be able to allow a user to have applications active in several smart spaces, but still have only one context peer per user responsible for stewarding the information about that user. Such a design is illustrated below.

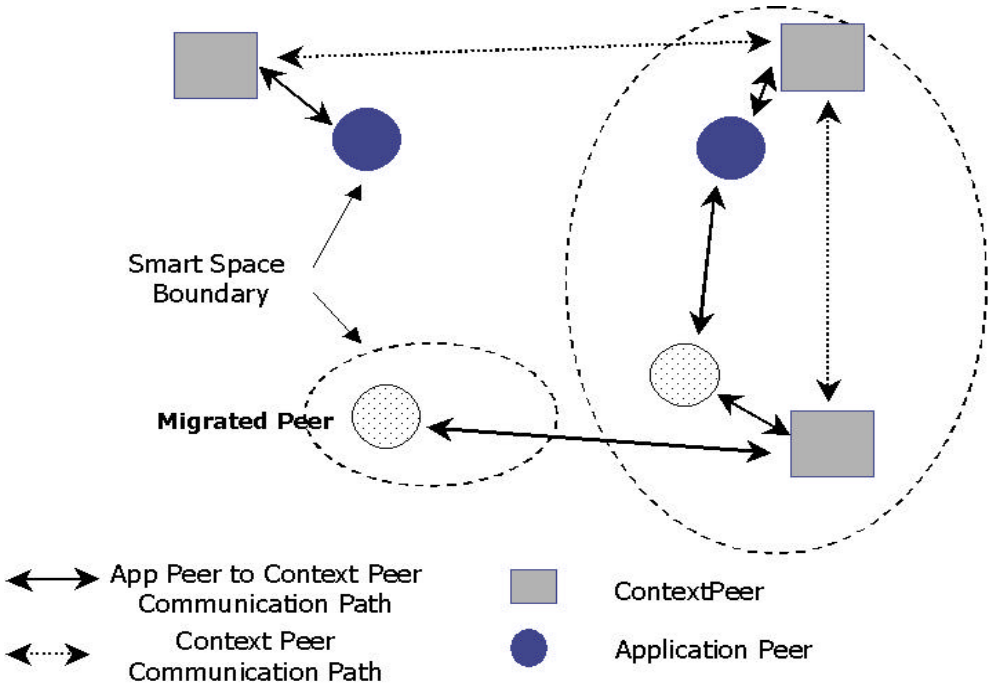


Figure 6.1: Implementation of Context Peers.

6.4 Future Work

This writing has offered a specific approach to the issue of migrating users context between Smart Space environments. Future work based on this approach could include:

- The evolvement of the *MIGRA* architecture to consider the intermittent, multitasking nature of Smart Space users today where users could leave multiple independent applications executing in various domains. A proposed architecture for this approach is presented in figure 6.1.

- The project did not aim to populate the application with multiple user services. Although, the current implementation of *MIGRA* has been designed in a generic way to allow the addition of new services such as MP3 players and video streaming applications. Also, adapting the application to work with a fully

featured Java web browser could extend the capabilities of the current web browsing application.

- Security is an issue that is of major concern when developing P2P applications. This issue was not considered during this writing. One future solution could be the evolvement of JXTAs membership service.

Bibliography

[ACKMM00] Dinesh Ajmera, Paul Castro, Ted Kremenek, Murali Mani, Richard Muntz, UCLA, "My Building Knows Where I am! Using Jini Technology as a Framework for Supporting Smart Spaces", In *JavaOne 2000* conference, http://mmsl.cs.ucla.edu/muse/slides/TS-1452.pdf.

[Bel] Hewlett Packard, "Cooltown Beliefs", http://www.cooltown.hp.com/dev/beliefs.asp.

[BGK02] Daniel Brookshier, Darren Govoni, Navaneeth Krishman, "JXTA: Java P2P Programming", Sams, 2002.

[CDK01] George Coulouris, Jean Dollimore, Tim Kindberg, "Distributed Systems: Concepts and Design", Addison-Wesley, third edition, 2001.

[CJED01] Keith W. Edwards, "Core Jini", second edition, Prentice Hall, 2001.

[CMS] Content Management System, http://cms.jxta.org.

[COOK] HTTP Cookies, http://wp.netscape.com/newsref/std/cookie_spec.html.

[CoolT] Hewlett Packard. Cooltown developer network. http://www.cooltown.com.

[CORBA] CORBA, http://www.corba.org.

[CSS] Cascading Style Sheets, http://www.w3.org/Style/CSS/.

[DADS01] Paul Castro, Benjamin Greenstein, Richard Muntz, Chastschik Bisdikian, Parviz Kermani, Maria Papadopouli, "Locating Application Data Across Service Discovery Domains", http://citeseer.nj.nec.com/462861.html.

[DAS99] Anind K. Dey, Gregory D. Abowd and Daniel Salber. "A Context-Based Infrstructure for Smart Environments". 1999, Georgia Institute of Technology.

[EASY] EasyLiving, http://www.research.microsoft.com.

[GATECH] Cory D.Kidd, Robert Orr, Gregory D. Abowd, Christopher G. Atkeson, Irfan A. Essa, Blair MacIntyre, Elizabeth Mynatt, Thad E. Starner, Wendy

Newstetter, “The Aware Home: A Living Laboratory For Ubiquitous Computing Research”, 1999, Georgia Institute of Technology, http://www.cc.gatech.edu/fce/ahri/.

[GLJW00] George Lawton, “Building the Infrastructure for Ubiquitous Computing, JavaWorld Article, http://www.javaworld.com/javaworld/javaone00/j1-00-smartspaces.html.

[GPL] General Public Licence, http://www.fsf.org/copyleft/gpl.html.

[Harold00] Elliotte Rusty Harold, “Java Network Programming”, O’Reilly, second-edition, 2000.

[HOMES] JavaSpaces, http://www.javaspaces.homestead.com/.

[HP] Hewlett Packard, http://www.hp.com.

[HPDC] Hewlett Packard, Philippe Debaty, Deborah Caswell, “Uniform Web Presence Architecture for People, Places, and Things”, White Paper, http://cooltown.com/dev/wpapers/uniform-webpresence/uniform-webpresence.asp.

[InCon] InConcert, http://research.microsoft.com/easyliving/.

[IBMUG] TSpaces user guide, http://www.almaden.ibm.com/cs/TSPaces/html/UserGuide.html

[JAVA] Java 2 Standard Edition SDK, http://java.sun.com.

[JINI] http://www.jini.org.

[JShell] JXTA Shell, http://shell.jxta.org.

[JSPACE] JavaSpaces, http://java.sun.com/products/javaspaces/

[TSPACE] TSpaces, http://www.almaden.ibm.com/cs/TSpaces/

[JXSPEC] JXTA Specification, http://spec.jxta.org/v1.0/docbook/JXTAProtocols.html

[JXTA] JXTA, http://www.jxta.org.

[J2ME] Java 2, Platform Micro Edition, http://java.sun.com/j2me/.

[KBMB] Hewlett Packard, Tim Kindberg, John Barton, Jeff Morgan, Gene Becker, "People, places, things: web presence for the real world", White Paper, Cooltown developer network.

[Li02] Sing Li, "JXTA peer-to-peer programming with Java, Early adopter", Wrox Press, 2001.

[LIND] Project Linda, http://www.cs.yale.edu/Linda/linda.html

[LMW99] Tobin J. Lehman, Stephen W. McLaughey, Peter Wyckoff, "T Spaces: The next Wave", Proceedings of the 32nd Hawaii International Conference on System Sciences, 1999, http://citeseer.nj.nec.com/lehman99spaces.html

[MDST01] Nelson Minar, *Distributed Systems Topologies*, O'Reilly Network, http://www.oreillynet.com/pub/a/p2p/2001/12/14/topologies_one.html, 2001.

[MIRA] MIRA, Multimedia Internet Recorder and Archive, http://mmsl.cs.ucla.edu/muse/services.shtml#mira

[MKKS00] Barry Brumitt, Brian Meyers, John Kruman, Amanda Kern, Steven Shafer, "EasyLiving: Technologies for Intelligent Environments", Microsoft Research, September 2000, http://research.microsoft.com/users/barry/research/huc2k-final.pdf.

[MUSE] http://mmsl.cs.ucla.edu/muse/.

[MYJX] myJXTA, http://instantp2p.jxta.org.

[OSW02] Declan o'Sullivan, Vincent Wade, "A Smart Space Management Framework", Knowledge an Data Engineering Group, Trinity College Dublin, 2002.

[PSW01] Manoj Paramewswaran, Anjana Susarla, Andrew B. Whinston, "P2P Networking: An Information-Sharing Alternative", IEEE, July 2001.

[RMW99] Bradely J. Rhodes, Nelson Minar, Josh Weaver, "Wearable Computing Meets Ubiquitous Computing: Reaping the best of both worlds", MIT Media Lab, 1999, http://xenia.media.mit.edu/~rhodes/Papers/wearhive.html.

[RS00] L.Rosenthal, V. Stanford, *NIST Information Technology Laboratory Pervasive Computing Initiative,* IEEE Ninth International Workshops on

Enabling Technologies: Infrastructure for Collaborative Enterprises, June 14-16 200, NIST USA.

[SMB00] Steve Shafer, Brian Meyers, Barry Brumitt, “The EasyLiving Viewpoint on Ubiquity”, Microsoft Research, Submitted to Workshop on Infrastructure for Smart Devices, http://www.inf.ethz.ch/vs/events/HUK2kW/Shafer.html.

[SMFL] Robert J. Orr and Gregory D. Abowd, “The Smart Floor: A Mechanism for Natural User Identification and Tracking”, 2000, Georgia Institute of Technology http://www.cc.gatech.edu/fce/smartfloor/index.html.

[SUN] Sun Microsystems, http://www.sun.com.

[USE] Usenet, http://www.usenet.org.

[WG00] Zhenyu Wang and David Garlan, “Task Driven Computing”, Technical Report from CMU, USA, CMU-CS-00-154, May 2000.

[Wil02] Brendon J. Wilson, “JXTA”, New Riders, 2002, http://www.brendonwilson.com/projects/jxta.

[XML] extensible Markup Language, http://www.w3.org/XML.

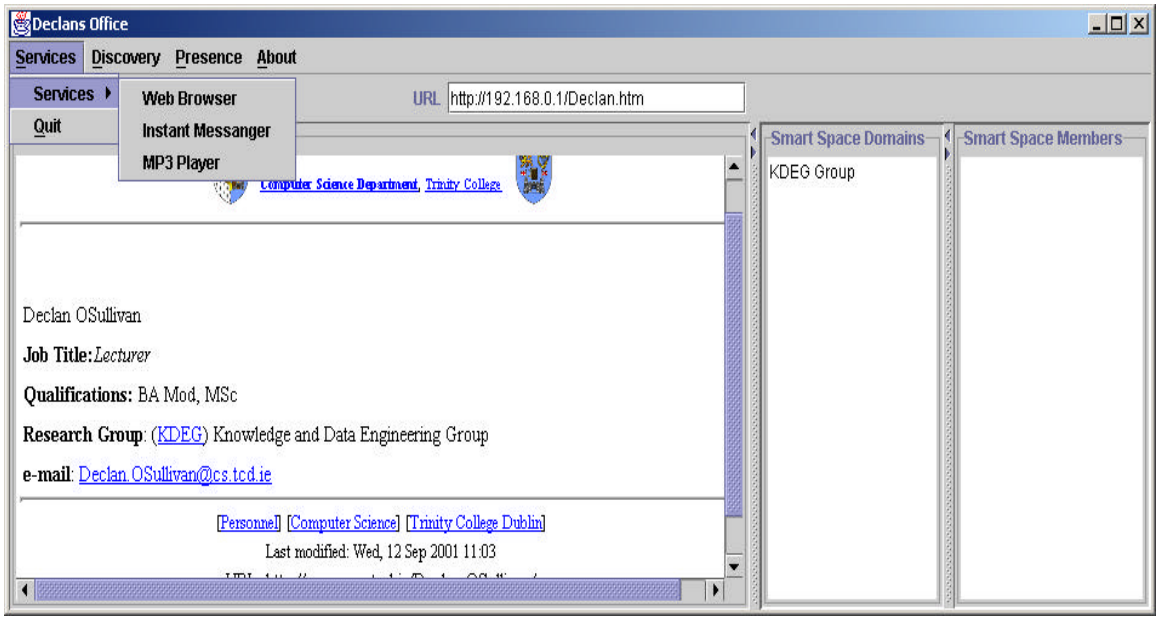
Appendix A

The following presents screen shots of *MIGRA* a various stages of user interaction.

This GUI is implemented by *MIGRA* to simulate user detection upon entry to a Smart Space.

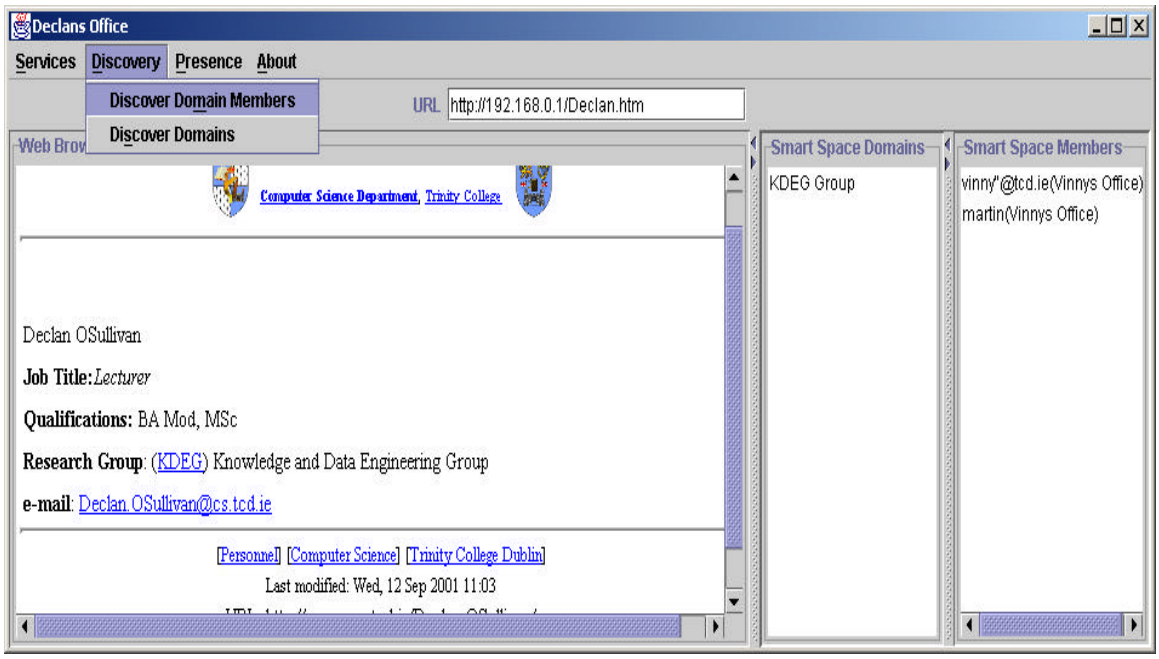
User detection GUI.

The following screen shot represents the application in a stage where the user has performed a search on other Smart Space domains that are active on the network. The result is "KDEG Group".



***MIGRA* application.**

Figure 8.3 represents the application after the user has searched for active Smart Space domains and a search on active users in their local domain. The result of the users active is depicted in the far right sidebar of the application.



Discovery of users in the local Smart Space Domain and other Smart Space Domains.