



Mobile Resource Awareness

Master of Science Thesis

Department of Teleinformatics, Royal Institute of Technology (KTH)

Erik Wiklander

e95_ewi@e.kth.se

Stockholm, February 21, 2001



Examiner:

Gerald Q. Maguire Jr.
KTH Teleinformatics
maguire@it.kth.se



Supervisor:

Theo Kanter
Ericsson Research
theo.kanter@era.ericsson.se

Abstract

New types of cellular networks are appearing on the market, such as GPRS and UMTS. In addition to this, an increasing number of places are being equipped with wireless LAN extensions to the Internet. Simultaneously, new mobile devices are becoming available that can perform complex computations based on events from various input devices and/or information sources. The combination of extended network connectivity and new mobile devices creates new opportunities for network computing.

This paper presents a scenario where network enabled applications can interact with mobile devices to create a *smart space*. The ambition is to create a system where mobile devices can be aware of what resources are present at a specific location. The prototype solution in this paper uses an XML-based protocol and a tuple space architecture as a message passing mechanism.

Table of Content

1	INTRODUCTION	1
1.1	TRENDS IN MOBILE COMMUNICATION	1
1.1.1	CONVERGENCE OF TELECOM AND DATACOM	1
1.2	FOURTH GENERATION WIRELESS (4G)	3
1.3	SMART MOBILE DEVICES	3
1.4	MOBILE INTERACTIVE SPACES	4
1.4.1	UBIQUITOUS COMPUTING	4
1.4.2	SMART SPACES	5
1.5	PREVIOUS EFFORTS ON SMART SPACES	5
1.6	PROJECT GOALS	6
1.6.1	REQUIREMENTS	6
1.6.2	IMPLEMENTATION	7
2	BACKGROUND	9
2.1	JINI	9
2.2	UNIVERSAL PLUG AND PLAY	11
2.3	SALUTATION	12
2.3.1	SYSTEM OVERVIEW	12
2.4	COMPARISON	13
3	TOOLS	15
3.1	XML	15
3.1.1	BACKGROUND AND HISTORY	15
3.1.2	DESCRIPTION	16
3.1.3	EXAMPLE	17
3.1.4	XML EXTENSIONS	17
3.1.5	COMMON USES	20
3.2	TSPACES	20
3.2.1	BACKGROUND – LINDA TUPLE SPACE	20
3.2.2	TSPACES MOTIVATION	21
3.2.3	PROPERTIES	22
3.3	SIP	24
3.3.1	FUNCTIONALITY	24
3.3.2	DESCRIPTION	25
4	PROPOSED SOLUTION	27
4.1	OVERVIEW	27
4.1.1	ASSUMPTIONS	27
4.1.2	FUNCTIONALITY	27
4.1.3	COMMUNICATION AGENTS	28
4.1.4	COURSE OF ACTION	28
4.1.5	USE OF XML	29
4.2	RETRIEVING TUPLE SPACE IP ADDRESS	30
4.2.1	WELL-KNOWN DOMAIN NAMES	30
4.2.2	ALTERNATE COMMUNICATION INTERFACE	30
4.2.3	MULTICAST	30
5	IMPLEMENTATION	33

5.1	SYSTEM COMPONENTS	33
5.2	AGENT DESCRIPTION.....	33
5.3	USE OF XML.....	33
5.3.1	ELEMENT CONTENT VS. ELEMENT ATTRIBUTE.....	33
5.3.2	USER AND APPLICATION PROFILES	34
5.3.3	EVENTS.....	35
5.4	DOWNLOADABLE CODE.....	35
5.4.1	JAR FILE CONTENT	35
5.4.2	ACCESSING THE CODE.....	35
5.5	AN EXAMPLE.....	35
5.5.1	EXAMPLE SCENARIOS.....	36
5.5.2	DESCRIPTION	36
6	CONCLUSIONS.....	39
6.1	TSPACES.....	39
6.2	XML	39
6.3	USEFUL APPLICATIONS.....	40
7	REFERENCES.....	41

1 Introduction

New types of cellular networks are appearing, or will appear on the market in the near future. These networks will allow end-to-end IP connectivity for mobile users. In this scenario, the telecom industry is likely to shift towards a more Internet centered network. At the same time, new mobile devices with increasing computing capabilities are becoming available that are so small that they can fit in your pocket.

This report proposes a solution for a system where mobile devices can access applications and resources that reside in a specific location. The system should work so that the client can be aware of services that appear in the network. This work is partially influenced by the paper “Event Driven, Personalizable, Mobile Interactive Spaces” [1], by Theo Kanter.

1.1 Trends in Mobile Communication

1.1.1 Convergence of Telecom and Datacom

Mobile telephony made it possible for people to talk while on the move. The Internet made it possible to turn raw data packages into useful services. Now these two technologies are converging, and this convergence is usually referred to as the Third Generation Wireless (3G).

WAP

The GSM cellular network has traditionally been a network for voice services only. But with the introduction of the Wireless Application Protocol (WAP) [2], a data service was added to the network. WAP provides the user with simple services such as electronic payment and informational services, but because of the limited bandwidth of GSM (only up to 9.6 Kbit/s) WAP is not well suited for delivery of services that requires more bandwidth (e.g. multimedia services). Yet another limitation of WAP is that the Internet can only be accessed if a WAP gateway is used. This puts further constraints on the end-to-end quality of service aspects and makes it impossible to make use of real-time Internet services using WAP. Even though WAP is a packet switched service, WAP is not considered to be part of a 3G network. (the current status of GSM is sometimes called 2.5G)

Third Generation Wireless

GPRS (General Packet Radio Services) [3] is a packet based wireless communication service that is based on GSM. GPRS promises a significant increase of data rates compared to GSM. GPRS will make it possible to directly access services that are located anywhere on the Internet. And because of the higher data rates, users will be able to take

part in video conferences and interact with multimedia Web sites using handheld computers.

The Enhanced Data Rate for GSM Evolution (EDGE) [3], which is the successor of GPRS, will increase bit-rate even further and thus boosting the opportunities for mobile applications. Both GPRS and EDGE are seen as evolutionary standards on the way to Universal Mobile Telecommunications Service (UMTS) [3]. UMTS is a standard that will make it possible for speeds up to 2 Mbps and offer packet based services to mobile computer or phone users no matter where they are located in the world.

Apart from the increased data rates in 3G networks the main advantage is that these networks are package-based and that they have support for IP. This implies that users can be constantly attached to the Internet as they travel. A virtual connection is always available to any other end point in the network. Another consequence of 3G is that it in theory should be cheaper than a circuit switched network since several users share the same capacity and only use the channel when packets are transmitted.

Wireless LAN

A wireless LAN (WLAN) is a technology, which lets a mobile user connect to a local area network through a wireless connection. The standard most commonly used for WLAN is called IEEE 802.11b [4]. Work on this standard began in 1990 and it was approved in 1997. The purpose was to create a global standard for radio equipment and networks, which use the unlicensed ISM (Industrial, Scientific and Medical) band, 2.4-2.5GHz, for computer speeds of 11Mbps. The standard includes specifications for both the physical layer and the MAC (Media Access Control) layer. Indoor range varies from 20 up to 120 meters and outdoors the range varies from 100 to 450 meters. Using directional antennas and if the line of sight is clear, much greater distances can be achieved outdoors.

Using multiple access points a wide area can be covered so that for instance an entire corporate building or a university campus can be covered. The advantages of using a WLAN instead of connecting to the network using e.g. Ethernet, is of course that no cables are needed. Students with laptops can then use the network no matter where on the campus they are.

Besides this, so-called hot spots are being equipped with wireless LAN extensions to the Internet. Examples of such hot spots are traffic junctions such as train stations and airports. Other possible hot spot locations are hotels. This allows travelers to get connected to the Internet at various times during a trip, so that they for example can check their e-mail etc. An example of such a solution is Telia's Homerun [5]. Several hotels and exhibition halls have signed up for this and the number of hot spots are increasing continuously.

1.2 Fourth Generation Wireless (4G)

Broadband Internet access is being provided in many urban areas, as for example housing co-operatives. It has been shown that it is possible to provide each apartment with 100Mbps at relatively low cost [6] using WLAN. When more and more buildings are being equipped with such WLAN extensions, relatively big area can be covered with WLAN in an urban area. One can see these WLAN extensions as islands of high bandwidth surrounded by an ocean of a 3G network. And if a standard were used for all of these islands it would be possible for mobile users to purchase abundance of bandwidth. In this scenario no prior subscription to Internet access with network operators is needed.

Thus, the properties of 4G are high bandwidth Internet access with end-to-end IP connectivity [6]. 4G will be a multitude of operator-less heterogeneous networks.

In traditional cellular networks it has been the network operator that provided for all the services, but with the new technologies appearing it is possible for third parties to deliver services directly to end users using IP. The combination of constant IP connectivity together with smart mobile devices makes it possible to create new services and applications. Applications can range from advanced services such as multimedia deliverance and online gaming to much simpler ones like a shared electronic blackboard.

1.3 Smart Mobile Devices

In the recent years the market has been flooded with handheld computers, or PDAs (personal digital assistant). These PDAs function as information storages and provide some computing capabilities. Some PDAs have a keyboard for receiving input, but most of them have an electronically sensitive touch screen on which handwriting can be received using a stylus pen.

Many of the PDAs that exist on the market today have some kind of wireless communication ability. An infrared interface is seen on most popular devices and many future PDAs are likely to be equipped with Bluetooth interfaces. Some PDAs have expansion packs to make it possible to connect a wireless LAN card. One example of a PDA with these capabilities is the Compaq iPAQ Pocket PC [7]. The iPAQ has a color display and a 200MHz StrongARM processor. The operating system used is Microsoft Windows for Pocket PC and this makes it easy to write your own applications for the PDA using Visual Basic or Visual C++. It is also possible to install a Java Virtual Machine. Windows Pocket PC is however memory consuming, and this combined with the color display puts the battery lifetime in the same range as with a regular laptop.

Another operating system for handheld computers is EPOC [8]. EPOC leaves a much smaller footprint than Windows and it is possible to build devices that consume less power and thus have longer battery lifetime. These features have made EPOC the OS of choice for mobile GSM phones with additional functionality of a PDA. A drawback of EPOC is that only the manufacturers may install programs, so it is not possible for third parties to construct programs. This may however be changed in the future.

Mobile phones and PDAs has been two separate markets that have evolved concurrently. But now we are experiencing a technical development that bears a resemblance to the convergence of telecom and datacom. Mobile phones and PDAs are being integrated into single device. In the near future we will probable see mobile phones with computing capabilities traditionally only found in PDAs. And these phones will of course have the ability to access 3G cellular networks.

1.4 Mobile Interactive Spaces

When network connectivity and computing capabilities increase for mobile devices it is possible to create interactive location based services. A solution for such an interactive space is proposed by Kanter in [1]. In this paper an architecture is presented where devices in the network can interact with nodes to create a resource aware application system. The application architecture should be able to work in an ad-hoc fashion, meaning that new devices and resources entering the network should become visible to users without any prior knowledge of that resource's existence. Furthermore, users should be able to invoke the different services and be able to interact with other users that are currently connected to that location. One can see this architecture as a form of mobile instant messaging service, such as ICQ [9] or AOL Instant Messenger [10]. The main differences from these are that the system is mobile and not only persons should be made visible but also services and devices. Examples of devices that mobile users would like to interact with could be a Web camera or a networked printer or some type of application server.

1.4.1 Ubiquitous Computing

The area regarding interactive spaces is closely related to that of ubiquitous computing. Ubiquitous computing is the method of enhancing computer use by making many computers available throughout the physical environment, but making them invisible to the user. The aim of this is to let portable personal devices interact seamlessly with the surrounding environment. Research of building smart homes is based on ubiquitous computing. The idea behind it is that there are so many devices with computing capabilities that are spread out, and they stand alone with no means of communication with each other. Therefore it would be desirable to connect all devices so that they could work as a single entity.

1.4.2 Smart Spaces

Another research area intimately connected to this is Smart Spaces. A smart space is a concept that embraces the notion of mixed-reality [11]. Mixed reality is the integration of the physical and virtual worlds as shown in Figure 1.1.

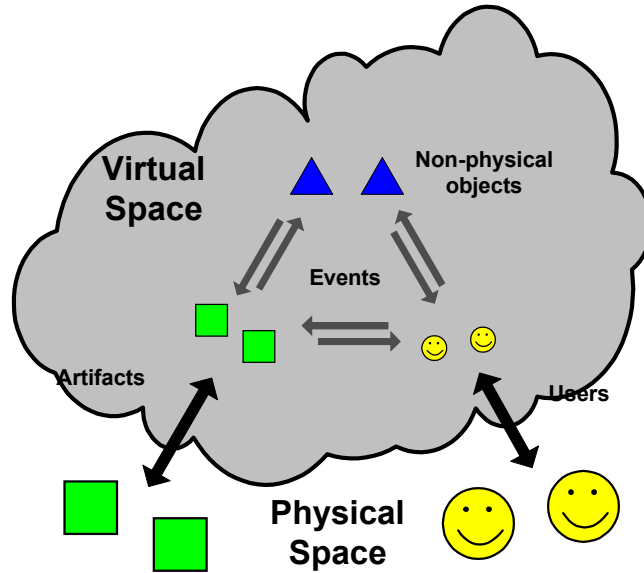


Figure 1.1: Mixed Worlds¹

In a smart space communication system, mobile artifacts, mobile users and resources have corresponding representational entities in the communication space. A smart space is basically no more than a physical space equipped with sensing capabilities (e.g. location, movement, visual, audio, etc.).

1.5 Previous Efforts on Smart Spaces

There are many ongoing research projects in the area of smart spaces. These projects build on previous work in ubiquitous computing and distributed system architectures. These have the aim to build multi-user and multi-device systems.

One project that bears a great deal of resemblance to what I am doing in this work is the Stanford Interactive Workspaces Project [12]. They are working with the technology called TSpaces combined with XML, but they have focused on a specific application area of use, namely a meeting room where users can share resources and information. The work at Stanford has put a lot of work on the graphical appearance of information.

Another projects worth mentioning is the Smart Space at the National Institute of Standards and Technology [13]. This project has another

¹ Figure 1.1 appears by courtesy of Theo Kanter

approach to the problem. They have realized a distributed network based on the Linux Operating System.

1.6 Project Goals

The goal of my work is to study the possibilities to create a resource aware service and to work out a solution for such a service and to build a simplified architecture, which has some of the characteristics of a smart space. In addition to the actual creating of the system, a study of existing systems has been performed.

1.6.1 Requirements

Before the actual implementation begins it is important to list the desirable characteristics of the system. The ambition is that the system has the following distinguishing features:

Small Footprint

The software should leave a footprint small enough so that it can fit into any device that should be able to connect to the system.

Flexibility

The architecture should be able to adjust dynamically to new datatypes. When a new message type is added into the system, this type of message should be accepted by all devices, or at least tolerated.

Ad-Hoc Situations

New devices should be able to enter or leave the system dynamically. It should work in an ad-hoc fashion on the application layer. New devices and applications should be able to connect without needing to do any additional configuration.

Event-Driven

The system should respond accordingly to events generated by any device, and the messaging should work in an asynchronous manner. Messages do not travel from point-to-point, instead messages are passed to the group as a whole and it is assumed that those who are interested listen to it.

Downloadable Code

As a consequence to the dynamic behavior of the system, clients need to have the ability to run downloaded code when they come across new types of applications.

Platform Independency

Since there are a multitude of different devices with various operating the system should be platform independent.

1.6.2 Implementation

The tools used when building this system are mainly XML and TSpaces. XML is used to create profiles for users as well as for applications and resources. XML is also the base for the protocol used for message passing. As will be explained in subsequent sections, XML is used because it has some features that are advantageous, including extensibility.

TSpaces is a tuple space framework that works as a message passing mechanism and a message repository. The reason I have chosen to work with TSpaces is that it among others includes support for XML and it makes use of asynchronous communication which makes it easy to write applications.

The programming language used is Java. The reason for this is that it is possible to run a Java Virtual Machine (JVM) on many different platforms, and thus attain some degree of platform independency. Another reason for using Java is that TSpaces is written using Java exclusively.

2 Background

In the recent years a number of technologies have appeared with the goal of connecting devices of various kinds to a network in a seamless fashion. These technologies are essentially frameworks that propose certain ways of device interaction with the ultimate goal of simple and scaleable device inter-operability. The way of which these technologies want to accomplish the goal varies and they have focused on different aspects of the complex of problems.

Possible devices that could be connected can be anything that occupies some kind of intelligence, e.g. intelligent appliances, wireless devices and PC's of all kinds. And with the emergence of many new smart devices, the issues regarding coordination between these devices has become a major research area.

Below I will give an explanation of three of the most well known frameworks related to this area; Universal Plug and Play, Jini and Salutation. Apart from the differences of these frameworks, below is a list of what devices should be able to do in each framework:

- *Ability to announce its presence to the network*
- *Automatic discovery of devices connected to the network*
- *Ability to describe its capabilities and understand the capabilities of other devices*
- *Self configuration without administrative intervention*
- *Seamless inter-operability with other devices*

2.1 Jini

Jini [14] is a coordination framework developed by Sun Microsystems. It has been specifically customized for the Java programming language. Inspiration for Jini was drawn from the original Linda coordination model using Tuple Spaces (a more detailed review of Linda will be given later). The original idea was called JavaSpaces, which later evolved into the framework that is called Jini today.

Before going into detail about how Jini works, it is important to know what kind of requirements that are made of a Jini system. A fundamental assumption is of course that there is some kind of network connecting the different devices. The network should have reasonable bandwidth and latency cannot be too severe.

Since Jini is Java centered, it is necessary for each device to have memory and some computing capabilities. This is needed because each device needs to run its own Java Virtual Machine (JVM). The reason why every device needs an own JVM is that downloadable code is an important

feature of Jini. All devices need to be able to download code and run it locally on its own JVM.

It should be mentioned that it is possible for device that is not Java-enabled connect to a Jini system. For this to work however, there has to be a device that works as a proxy or gateway. This device then has to perform protocol translations, but this is not a very attractive solution since it introduces a possible failure point in the system.

A group of devices that are connected in a Jini system is called a federation. Each member of such a federation is an autonomous device that can be aware of and cooperate with other devices. In Jini, a device is however not the key concept. Instead Jini uses the term service. A service could indeed be a device, but one can see a service as something far more general. Services can be seen as anything that can be used by a person, a program, or another service. Examples of services can be printing a document (using a printer device), or doing a mathematical computation (using a PC device).

The central point of any Jini system is the lookup service. This is the point to which other services register and where services look for other services. Before a device can join a federation it first has to find a lookup service. The first thing a device does after being plugged into the network is that it multicasts a request for any lookup service to identify themselves. The multicast is usually done over a LAN but it is also possible to do a multicast over the Internet. Another option is that the device knows about the lookup service beforehand and immediately connects to it without doing the multicast. When a lookup service receives a request message it immediately sends a response. This first message exchange is called *discovery*.

Once a device has discovered a lookup service, the device registers itself with the lookup service. The registration process is called *join* in Jini. A device registers itself with a set of properties. These properties are pairs of names and values. Other devices can then query this information to find services that are useful to this device. During the registration process it is also possible for services to upload some code to the lookup service.

The code that is uploaded to the lookup service basically works as a proxy that can be used to contact an interface on the device and invoke methods of that interface. This is accomplished using Java Remote Method Invocation (RMI). In short, RMI is a set of protocols that enable Java objects to communicate remotely with other Java objects. An important issue is that for this to work both devices should have Java embedded systems. It is however not mandatory for devices to use RMI. In the case when not using RMI, proprietary protocols are used and then the lookup service only works as a directory service.

When a service is accessed it is usually done so on a time restricted manner. The time of which access to the service is granted is called *lease*

time. Not all services are lease based, and those who are lease based can either be exclusive or non-exclusive. Exclusive leases assure that only one can use the service at a time.

It is possible to build an event passing system with Jini. Any object may subscribe to any other object for a notification when any kind of event occurs. An object is typically interested in some kind of state change in the other objects or it can also be an event that signals that a new service has appeared or an old one has disappeared. All devices do not support event subscription. In order to subscribe to an event notification, the device needs to have set up an event and notification interface.

2.2 Universal Plug and Play

As with the case of Jini, Universal Plug and Play (UPnP) [15] proposes a solution for peer-to-peer network connectivity of network devices. As will be shown, UPnP takes on a different strategy altogether. UPnP works at a much lower level than Jini.

The concept of Jini is to have an application level program that can run on any platform. In this case they have chosen Java. The approach of UPnP is instead to leveraging code that is implemented everywhere and the obvious choice is to make use of the TCP/IP protocol suite, and that is exactly what UPnP does.

The functionality of UPnP is divided into six separate steps. A short description of each step is given below.

The most basic step in UPnP is *addressing*. Each UPnP-enabled device must have a Dynamic Host Configuration Protocol (DHCP) client and search for a DHCP server in order to obtain an IP address. If there is no DHCP server available on the network, i.e. the network is unmanaged, then the device must use Auto IP to get an address. Auto IP is the procedure of automatically choosing an IP address in an ad-hoc IPv4 network.

After the fundamental step of addressing a device makes itself available to other devices by advertising its presence and what services it supplies. There has to at least one control point in the network that listens for advertisements and this control point keeps track of which devices that are currently connected to the network. As soon as a device has registered with a control point, the device can search for what devices or services are available. The exchange of messages is done by use of the UPnP discovery protocol. This protocol is based on the Simple Service Discovery Protocol (SSDP) [16]. Messages include information about type of service, identifier and a pointer to more detailed information. The discovery can be done either by a unicast or a multicast, but in common for both ways is that they use HTTP over UDP. When using HTTP in this way it is usually referred to as HTTPU (unicast UDP) or HTTPMU (multicast UDP). The messages are transported by HTTP but the semantic

is UPnP specific. Among the information in the discovery message is an URL to a detailed device description.

When the step of discovery is completed not much is known about the device. For more detailed information about each device some kind of description is needed. The URL provided with the discovery message contains a detailed description of each device. The UPnP description of the device is expressed in XML (XML is described in more detail in subsequent sections). The UPnP description contains information about vendor specific information such as model name and number. In addition it also includes a list of each commands the service responds to and the variables for each command. In the UPnP terminology commands are called actions.

When a control point has received a device description the control point can send control messages to the device in accordance with the description. Messages are passed using the Simple Object Access Protocol (SOAP) [17]. SOAP is an XML based protocol, which makes it possible for applications to communicate with each other independent of platform.

As stated above the description step of UPnP includes an XML representation of devices with their actions and variables. Eventing is the process of when a device informs the control point of its current status. A device can have one or more status variables and whenever these variables change the device informs the control point using General Event Notification Architecture (GENA) [18]. As with the case of SOAP, GENA is also an XML based protocol.

Presentation is an optional step in UPnP. If a device has a URL for presentation this can be accessed from a web browser and the current status of the device can be viewed. Some devices offer the possibility to control the device in addition to just monitoring the status.

2.3 Salutation

Salutation [19] is a framework with a somewhat different focus than both Jini and UPnP. It has drawn more from research on intelligent agents than the other two mentioned earlier. One notable feature of Salutation is that it tries to provide transport protocol independence as well as operating system independence.

2.3.1 System Overview

One key component in Salutation is an entity that is called a Salutation Manager (SLM). The SLM may be located locally or remotely. The complete system consists of an ensemble of SLMs that coordinate with each other. They act like agents that do everything on behalf of their clients. When a device is plugged into a network, it registers itself with a well-known SLM. All communication between devices is mediated

through the SLMs. The framework supplies a callback mechanism to notify devices of events in the system.

As stated earlier, Salutation wants to achieve transport independence. It is therefore possible for SLMs to communicate even if they are on different transport media. This is attained by using transport dependent modules that are called Transport Managers.

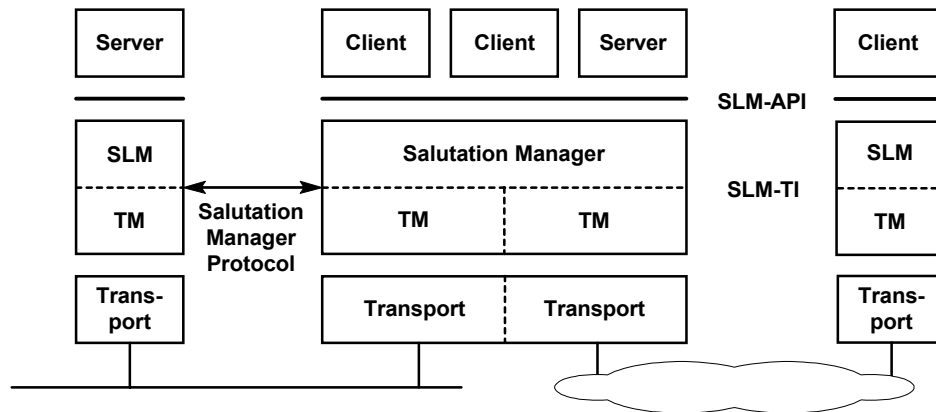


Figure 2.1: Salutation Overview

The Salutation Manager provides four basic tasks; service registry, service discovery, service availability and service session management.

The Salutation Manager can discover other remote Salutation Managers and determine the services registered there. The process of discovery is done so that the local SLM compares the service type with which the service registered itself. Communication between SMLs is done using Remote Procedure Calls (RPC).

Data description in Salutation uses a notation called Abstract Syntax Notation One (ASN.1) [20]. ASN.1 is an ISO/ITU standard based on the OSI model. It is a language that defines the way data is sent across dissimilar communication systems. ASN.1 ensures that the data received is the same as the data transmitted by providing a common syntax for specifying application level protocols.

Instead of lease management as used in Jini, SLMs can be asked periodically to check the availability of devices and report the status. This allows a service (device) to become aware when either have left the system.

2.4 Comparison

When working with frameworks like these, one should have in mind that it is needed to for the devices to have some standards for the operations. If no standards are introduced it is simply not possible for devices to coordinate. A problem that arises from this is maintaining a balance between standardization requirements and the use of proprietary protocols.

Jini is an example where developers have strained their efforts to make it as standardized as possible, as where on the other hand UPnP developers have come up with a lot of proprietary protocols. The advantage of the latter is that more device autonomy is attained.

It is of course difficult to decide which of these frameworks that is better, since it all depends on what you want to accomplish. But one drawback with Jini is of course that it requires a Java platform, and therefore it leaves a relatively large footprint on the device. But to the benefit of Jini is that it is fairly easy to build such a system.

One of the advantages of UPnP is that it is operating system independent. Even though this is the case, misgivings have been issued that Microsoft want to tie UPnP closely to the Windows OS. It is not easy to say whether this holds true or not, but since Intel has released a Linux API for UPnP [21] these misgivings will probably not hold true.

Perhaps the most viable one of these frameworks is Salutation. It has several pleasing features such as transport independency. But the price you have to pay for this is increased complexity and building a system with Salutation can be quite cumbersome.

But again, it must be stressed that it very much depends on what types of devices you want to connect. When dealing with only software applications Jini is a strong contender because of the fairly simple way of adding new services. But as I will show in the next sections I have not used any of these frameworks for constructing a smart space, I have built the foundation of a smart space using a set of different tools.

3 Tools

In order to completely understand the system that I have built, it is necessary to have a basic knowledge of the tools that are used. The two tools that I have primarily used are XML and TSpaces. XML is a markup language that can be used in many different contexts and TSpaces is a technology that has originated from the original Linda architecture for distributed computing.

3.1 XML

The Extensible Markup Language (XML) [22] is a document description language, which is used to describe the structure of documents. XML is a method of storing information in a simple, yet highly structured way. The language describes how tags are used to divide documents of structured data into different parts and how to identify these parts. Structured data can be any of a number of different things, e.g. spreadsheets, address books, financial transactions, etc. The rules and guidelines of how to use XML is an international recommendation for standardization by the World Wide Web Consortium (W3C).

3.1.1 Background and History

XML is a W3C standard since February 1998 but even though it has appeared fairly recently, the technology originates from an OSI standard from the 1980s called the Standard Generalized Markup Language (SGML). SGML does not specify any particular formatting, instead it specifies the rules for tagging elements. SGML is widely used for to manage large documents that are subject to frequent revisions and need to be printed in different formats.

Since SGML is a large and complex system it is not widely used on personal computers. But in the early 1990s, with the growth of Internet, SGML gained increased popularity, since the World Wide Web uses HTML, which is one of the ways of defining and interpreting tags according to SGML rules.

One can see XML as something that takes the best features of SGML and HTML and combines them into one. And one of the specific goals when XML was developed was that it should keep most of the descriptive power of SGML, but to remove as much of the complexity as possible.

When looking at an XML document it has many similarities with HTML, but whereas HTML is only used to convey layout information about a document, XML is used to represent structured data in a document.

3.1.2 Description

An XML document stores data in pure text format, but it is not intended for humans to read. The primary producer and consumer of XML data is a computer and not a person, but the text format provides the opportunity for programmers to easily debug and fix a broken XML file.

The most essential building block of XML is the tag. A tag is a word bracketed by the '<' and '>' characters. All tags XML tags must be closed. The text that is in between an opening tag and a closing tag is called *tag value*.

A set of opened and closed tags is usually referred to as an element. XML elements can be nested inside other elements, but only in a logical manner. The inner nested tags must always be closed before the outer nested tags are closed.

Tags can optionally contain attributes. Attributes are used to provide additional information associated with the tag. The attribute is a name/value pair separated by an equal sign. The value must be enclosed by a pair of single or double quotes.

An XML document that is consistent with the rules defined in the XML specification is termed well-formed. An XML document that is not well-formed can not be used at all. In order for an application that uses an XML document to know how data should be interpreted, an XML document can optionally contain a description of its grammar. The mechanism for describing the grammar of an XML document is known as Document Type Definition (DTD). The DTD describes what types of elements are allowed in a document and describes the structure of these elements. If the XML document contains a DTD and the grammar of the document conforms to that specified in the DTD, then the XML document is referred to as valid. The process of making sure that the document is consistent with its DTD is commonly termed validation.

As mentioned above there are many similarities between XML and HTML. But there are mainly three advantages that are notable with XML:

Extensibility

XML is a meta description format, i.e. by using XML you can create your own markup language with its own tags and attributes. But with HTML, you can only work with a set of predefined tags.

Structure

XML separates content and structure from presentation, it lets you structure information in a more logical and independent way. The interpretation of the data is completely up to the application that reads it.

Validity

An XML document can easily be tested for validity. It is important for an XML file to be consistent with the rules of XML. An XML file with a missing tag is not possible to use at all and the application that reads it must throw an exception of some kind.

3.1.3 Example

Below is an example of what an XML document might look like:

```
<?xml version="1.0"?>
<memo>
  <from class="person">Erik</from>
  <to class="group">friends</to>
  <subject>Greetings</subject>
  <message>
    Hello everyone!
  </message>
</memo>
```

The first line of every document always looks the same. It is there to prepare the application that handles the document that it is dealing with XML. Until now there is only one version available for XML, but the version attribute is there because of conceivable future upgrades.

As can be seen the `from` and `to` tags contain an attribute. This attribute is there to tell to what type of class the receiver and sender of the memo is. The attribute name is in this case `class` and the attribute values are `person` and `group` respectively.

3.1.4 XML Extensions

The XML 1.0 specification is what defines what tags and attributes are, but around XML, there is a growing set of optional modules that provide guidelines for specific tasks. In this section some modules that are of particular importance are listed.

XML Namespaces

Since authors of XML documents can make up their own tag names, some problems can arise. The same tag name can appear in two different documents in two completely different contexts. The potential problem of collision of tag names can be solved using XML Namespaces.

XML Namespaces is a specification that describes how you can associate a URL with every single tag and attribute in an XML document. Below is an example² of a namespace declaration, which associates the namespace prefix `edi` with the namespace name `http://ecommerce.org/schema`, and a qualified name serving as an element type:

² Example is taken from <http://www.w3.org/TR/REC-xml-names/>

```
<?xml version="1.0"?>
<x xmlns:edi='http://ecommerce.org/schema' >
  <edi:price units='Euro'>32.18</edi:price>
</x>
```

The prefix `xmlns` is used only for namespace bindings and is not itself bound to any namespace name.

Document Object Model

Document Object Model (DOM) [23] is a programming interface specification that is developed by W3C. DOM lets a programmer create and modify XML documents (and also HTML pages) as program objects. DOM objects can be manipulated using an object oriented programming style. The structure of nested XML documents can be represented as a DOM tree, which has the same structure as most file systems (e.g. AFS).

Resource Description Framework

The Resource Description Language (RDF) [24] uses XML Namespaces to link every piece of metadata to a file defining the type of that data. This mechanism lets authors invent new element names and publish those names so that a community can easily agree on standard terms for representing common data elements.

RDF provides interoperability between applications that exchange information on the Web. One main objective of RDF is to enable automated processing of Web resources. RDF is an initiative from W3C, building on earlier developments such as the Dublin Core [25] and the Platform for Internet Content Selectivity (PICS) [26].

RDF metadata can be used in a number of different ways including resource discovery to provide better search engine capabilities, cataloging for describing the content relationships available at a particular Web site and by intelligent software agents to facilitate knowledge sharing and exchange.

The syntax in which RDF is expressed is XML. The advantage of using XML as syntax is that XML provides for the explicit expression of semantics. In RDF, resources are defined as any objects that can be uniquely identifiable by a Uniform Resource Identifier (URI).

The following example is what an RDF document using XML might look like:

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf=http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:s="http://description.org/schema/">
  <rdf:Description about=http://www.example.com/john>
    <s:Creator>John</s:Creator>
  </Description>
</RDF>
```

To get an overview of an RDF document it can be advantageous to build a labeled directed graph. Figure X shows such a graph for this simple example:

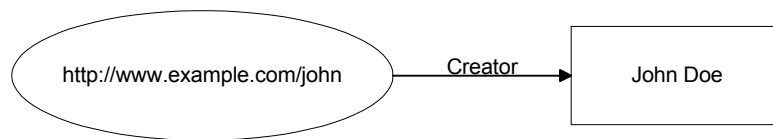


Figure 3.1: Labelled Directed Graph

RDF data consists of nodes and attached attribute/value pairs. Nodes can basically be any type of Web resource, i.e. anything that can be assigned a URI. Attributes are named properties of the nodes, and their values are either atomic (strings, number, etc) or other resources or metadata instances.

A more general form of the directed graph is shown in Figure X below:

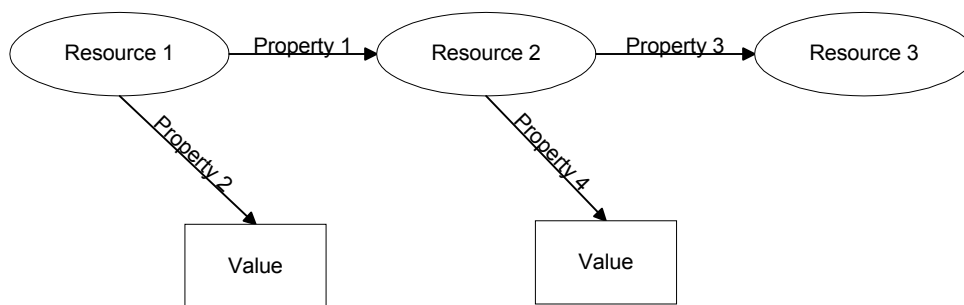


Figure 3.2: General Directed Graph

This model can often be very useful to get a quick overview of an RDF document. In these diagrams, the nodes are drawn as ovals and represent resources. Arcs represent named properties, and nodes that represent strings are drawn as rectangles.

3.1.5 Common Uses

As explained above XML can be used at many different occasions. One possible way to use XML is as a substitute for HTML, i.e. for online publishing on the Internet. Another way to use XML is for communication between networked applications and for information sharing in a consistent way. E-commerce organizations are to an increasing extent beginning to use XML in order to store information in a format that can be read by everyone.

In the recent years a number of different XML based protocols have appeared. Two of these are GENA and SOAP. These two protocols have the goal to create a common communication platform for computers of different operating systems. HTTP is the underlying protocol and since all major operating systems have this protocol installed XML and HTTP provide an at-hand solution to the problem of how applications running on different platforms can communicate with each other.

3.2 TSpaces

Another important tool in my solution is TSpaces [27]. TSpaces is an implementation of a tuple space architecture developed at the IBM Almaden Research Center.

This section contains information about what a tuple space is in general, and how it can be used. This section also includes information that is specific to the TSpaces implementation.

3.2.1 Background – Linda Tuple Space

The first efforts of making a tuple space were done in the early 1980s. It was David Gelernter of Yale University that first started to work on the Linda [28] programming language.

Linda has its origin from the parallel computing community. When distributed algorithms were first being developed, a need was seen to have some common memory for the different participating computers. And this is what a tuple space is, a globally shared, associatively addressed memory space organized as a bag of tuples. A tuple is simply a vector of typed values or fields.

The original tuple space architecture only had a few primitives. In order to place a tuple in the space a tuple is first created and then the `write` primitive is used. And in a similar manner, the `read` primitive is used to read a tuple from the space. When tuples are read, a query is placed with either an exact match of the tuple, but it is also possible to make use of a template to find matching tuples. A template is a tuple with some of its fields left blank. Instead the field only has a type definition. A field of a tuple and a template is said to match if it has the same type. A primitive that is similar to `read`, is `take`, which functions exactly the same but has

the consequence of deleting the tuple from the space. There are also blocking forms of read and take. These primitives wait for a matching tuple to be written to the space and then returns it. The non-blocking versions return a `tuple not found`, if no match is immediately available.

A tuple space provides a simple way for communication and synchronization. A process that wishes to share data simply generates a tuple and writes it to the space. This tuple can then easily be accessed by another process. This is not as efficient as a message-passing system, but it is easier to maintain because of these reasons:

Destination uncoupling

When sending a message in the system, it is not necessary for the sender to know the identity (or address) of the receiver.

Space uncoupling

Since tuple space uses associative addressing, tuple space is able to provide a globally shared memory regardless of machine or platform boundaries.

Time uncoupling

Tuples live independently of processes, and they can therefore outlive their creators, thus making it possible for time-disjoint processes to communicate seamlessly.

The combination of a standard sequential computation language and a small number of tuple space communication primitives produces a complete parallel programming language (e.g. C-Linda or FORTRAN-Linda). The work with Linda is still an ongoing work and since the earliest efforts it has been more and more refined. Many modified versions of Linda have also emerged, and TSpaces is one of these.

3.2.2 TSpaces Motivation

Although TSpaces is a derivation of Linda, it focuses on some other aspects. Not only is TSpaces used as a shared memory, but it also functions as a message passing system. One of the explicit goals of TSpaces is that it should function as a network middleware, i.e. a framework for connecting devices.

There are mainly two difficulties concerning a framework that attempts to connect devices, namely data representation and client communication.

The use of CORBA (Common Object Request Broker Architecture) [29] is one way to solve the problem of data representation. In CORBA it is possible to let objects of different languages and platforms to communicate by creating a standard data representation. Programs map

their data structure into a specific interface and to allow other programs to reference those data structures remotely using remote method calls.

Another approach is to make use of relational database systems, such as IBM's DB2, Oracle and Sybase. In this case data is stored in a database-specific format, dividing the program data into the primitive atomic types used by the database system. This way of doing it is however not well suited for most applications, since databases only have a restricted set of datatypes and do not have the capability to express complex relationships between entities.

TSpaces is a system that uses the tuple space model for interaction. Moreover, it extends the tuple space model with database features. TSpaces is implemented in the Java programming language and therefore inherits the ability to run on virtually any platform and to download new datatypes.

The TSpaces implementation of the tuple space architecture was developed with the goal to create a serviceable middleware package for developing highly effective distributed applications. It was specifically developed to support ubiquitous computing and it is implemented in the Java programming language and is thus to some extent platform independent. TSpaces provides developers with a distributed-object architecture including a development platform, processing environment, and addressing mechanism.

3.2.3 Properties

TSpaces implements the standard set of tuple space operators (such as `read` and `take`). In addition to these operators it includes set-oriented operators such as `scan` and `consuming scan`. These two are the set-equivalent versions of `read` and `take`. `Scan` returns every tuple that match the specified query. More advanced operators such as transactions are also included.

Another operation that is supported by TSpaces is *event notification*. The event notification feature is of particular importance since this lets objects subscribe to tuples that match any set template, i.e. objects do not have to search actively for a matching request. In the event notification mechanism we now have the possibility for an object to receive an event without having to have a pending query or needing to continuously query the tuple space. Thus reducing the necessary communication.

In the original tuple space architecture it is only possible to create tuples that contain the most common datatypes such as string integers and characters. But with TSpaces it also possible to create tuples that contain Java objects that are user defined. For this to work, the object has to implement the `Serializable` interface, and an equivalent class file must be available to both the clients and the TSpaces server.

As mentioned above, TSpaces has the functionality of a lightweight database. To support this there are different queries that can be performed on the space. These queries can be nested and become quite complex.

XML Support

Because of the extensible nature of XML, TSpaces developers decided to add support for XML since it is a flexible metadata language that can be used in a variety of contexts.

In order to make use of the XML support in TSpaces, an XML parser needs to be installed. The one used by TSpaces is called XML parser for Java (XML4J), and is based on the Apache Xerces parser.

Handling XML documents with TSpaces is very similar to handling ordinary tuples. When creating a tuple containing XML data, the whole XML string is passed to the constructor. When the XML tuple is passed to the TSpaces server the string is passed off and converted into a tuple tree. A tuple tree is basically just a tree of tuples, which has the exact same appearance as a DOM tree. Each tuple in the tree corresponds to a node in the XML DOM tree.

To extract XML tuples in a space, a specific query syntax is needed. The syntax supported by TSpaces is called XML Query Language (XQL) [30]. XQL is a notation for addressing and filtering the text of XML documents. The XQL notation is a proposed standard published by W3C.

XQL works as an easily understandable way to describe a class of nodes in a document. It is declarative rather than procedural, and to use it one simply describe the types of nodes to look for using a simple pattern modeled after directory notation.

Usage

For someone who is familiar with Java, it is fairly easy to get started with TSpaces. To make use of TSpaces you have to have a computer that runs the TSpaces server, and you also need a TSpaces client. A client is simply a program that makes calls to the server.

When starting the server it is possible to alter a number of different configurations. These settings are specified in a configuration file. One can specify which port number the server should listen to (the default value is 8200). And it is also possible to make use of the various security features available in TSpaces. Clients can be assigned different rights and they can have restrictions on how they can access the server.

The TSpaces distribution is also equipped with an HTTP server. This is useful for a number of reasons. First of all it possible to access a space with a web browser and to see what tuples are currently present in the space. And it also makes it possible to write applications that are HTTP

dependent. I will later show that my system depends on the HTTP server since this is where the downloadable code is situated.

Persistence

In TSpaces content in the space will be periodically checkpointed and all operations will be logged. Therefore it is possible to achieve persistence in the space. Inside TSpaces is IBM's own database system. There are two alternative tuple managers, one is based on IBM's main memory database technology and the other is based on DB2. Both managers provide persistence.

3.3 SIP

Session Initiation Protocol (SIP) [31] is an application layer control protocol. It is used for setting up sessions such as Internet multimedia conferences, Internet telephone calls, and multimedia distribution. SIP is a product of IETF and was designed with certain assumptions in mind. First it should be scalable since users can reside anywhere on the Internet and users can be invited to lots of sessions. Another assumption was that SIP should be built using already existing protocol tools. This means that people that are already familiar with e-mail and HTTP could easily understand the concepts of SIP. Even though SIP has some obvious strengths, it did not grow quickly until the interest in Internet telephony started to take off in 1996 and 1997. From then on SIP's use has grown exponentially.

SIP invitations are used to create a new session. These invitations contain information describing the type of session and lets participants agree on which media types can be used. Members in a session can communicate either via multicast or unicast. A noteworthy characteristic of this protocol is that it supports user mobility by proxying and redirecting requests to the user's current location. SIP handles not only invitations from persons but can also handle invitations for smart agents. For example, these agents could be the owner of media storage.

3.3.1 Functionality

When a SIP connection is set up between two or more session participants, there is no physical channel or network resource associated with that connection. The connection only exists as signaling state at the endpoints.

When setting up a new SIP session, a SIP URL is used to identify users. The SIP URL is similar to an e-mail address and has the form: `sip:user@host`. The user part is either a user name or a telephone number and the host part is either a domain name or a numeric network address. This SIP URL must then be translated into an IP address and port number. This translation is usually more than just a simple table lookup.

The translation can vary based on time of day or the status of the receiving party of the connection.

For non-local SIP resources requests are sent through a proxy server, which then forwards the request to the next hop server, which can be either the final user agent or another proxy server. A server can also be a redirect server, which informs the client of the address of the next hop server so that the client can contact it directly.

3.3.2 Description

SIP is a client-server protocol, and since a call participant may either generate or receive requests, SIP-enabled end systems must have both the functionality of clients and servers. This is usually called a user agent server. A caller that wishes to set up a connection sends an `INVITE` request to the other party. The message is not sent directly to the receiver, but rather it is sent to a proxy server that is responsible for getting the message to the right destination. When a user agent server receives an `INVITE` message it can respond to the sender in various ways, but some of the more common responses include acceptance, rejection, redirection, and busy. Another important request is the `REGISTER` request, which is used to inform a proxy of an address binding. This is particularly important where mobility support is concerned. To terminate an ongoing session the `BYE` message is used.

There are three main reasons why SIP is a protocol well suited for setting up sessions for transporting multimedia and they are:

Scalability:

To accomplish scalability SIP uses the same model as for the Internet in whole. The system is fast and simple in the core and smarter at the periphery.

Extensibility:

It is possible to extend SIP in many different ways. The reason for this is that Internet history has often shown that protocols are not always used exactly in the way they were intended.

Flexibility:

Even though SIP was developed mainly for Internet telephony-like services, SIP gives operators a lot of flexibility on regards of how to use the protocol.

In the system I have created SIP is not used at all. SIP is a part of this report since in future versions of my implementation it is conceivable to use SIP for message passing and other things.

4 Proposed Solution

This section and the next contain information of how the mobile resource awareness system works. In this section the system is explained on a higher level and how the system is experienced by a user.

4.1 Overview

4.1.1 Assumptions

This system is mainly supposed to work in wireless LAN environments. It is a possible application architecture in 4G enabled surroundings. It is however also conceivable to extend the capabilities of this architecture so that it functions in 3G networks, but the principal use is for a network with smaller coverage such as wireless LAN.

Before describing what the system should be able to do, it is important to know what the system *cannot* do. There is no support for IP address assignment. It is assumed that every device has some other means for obtaining an own IP address. To increase mobility support it is possible to use this system in correlation with Mobile IP [32]. In short, mobile IP makes it possible to make use of the same IP address when roaming different sub networks. This system does neither claim to solve problems regarding authentication, authorization and accounting (AAA). It is assumed that such solutions exist when entering a different network.

Another assumption is that all devices have the ability to run their own Java Virtual Machine. The reason for this is that all the communication is done through the TSpaces server and this is written in Java. If a device does not have the ability to run Java, it can still be made to function, but this demands a proxy that can run Java, which does all the communication on behalf of the device.

4.1.2 Functionality

The principal part of the system is the TSpaces server. It is through this all communication is done. The server has two different tasks. It works as a reflector of messages and as a message/data repository. Figure 4.1 shows how the TSpaces server works in theory.

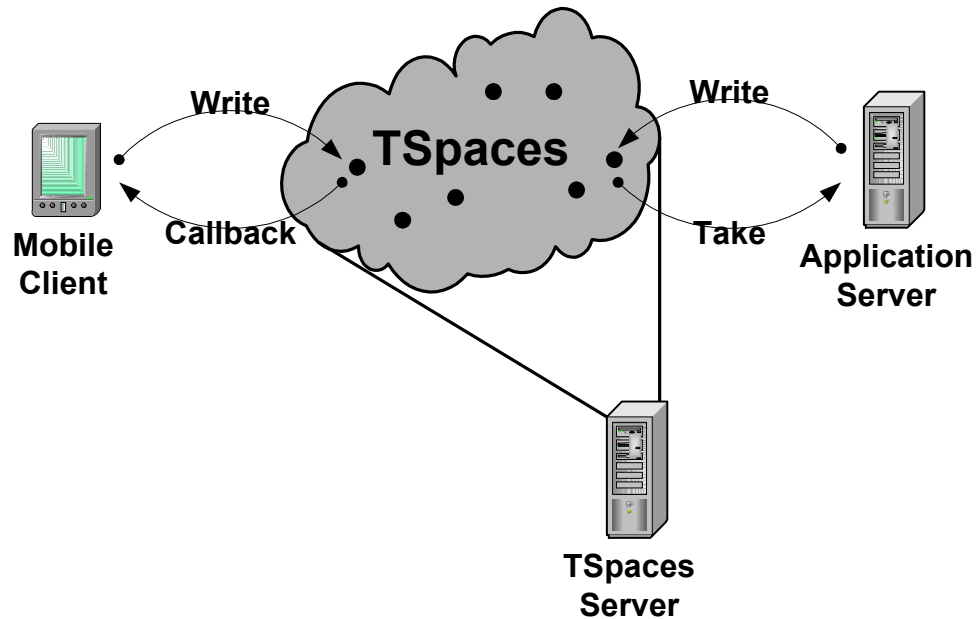


Figure 4.1: Tuple Space Functionality

In order to have a complete system, three logical entities need to be present. The most fundamental entity is the TSpaces server, the software for the server is available for free download and I have done no changes on this entity. By using the TSpaces primitives (i.e. `take`, `write`, etc.), full communication is achieved through this server.

Messages and information are written with help of tuples. These tuples are depicted as dots in the figure, and they have their own lifespan independent of the application that wrote it.

4.1.3 Communication Agents

Every entity in the system has an agent. All agents are connected to the space and can perform communication through the TSpaces server. They can scan the server for all types of messages that are of interest and they can subscribe to notifications when a specific type of message appears.

The goal when writing these agents has been to write them as general as possible to make it easy to extend the capabilities of them. All of these agents implement a basic interface, and every new application that should support this system needs to have an agent that implements the basic agent class.

4.1.4 Course of Action

When a client enters a communication space or when an application connects they have to follow a number of steps in order to make use of the functionality of the system. Conceptually, there is no difference in the way a client and an application connects to the space, everything is handled by

the communication agent that act the same way in both cases. These are the steps that has to be followed:

1. Retrieval of Tuple Space IP Address:

This first and must fundamental step can be reached in a number of different ways. One possible way is using multicast. A more thorough discussion of this step is given later.

2. Registration

Upon receiving the IP address of the server the communication agent is free to access the tuple space (unless the administrator has set up security restrictions). The agent registers to the server with a user profile that describes its own functionality and capabilities. This is done so that other agents can perform a search of the tuple space to find applications of interest.

3. Search

After the step of registration the agent scans the tuple space for applications. The agent can either search the space for all available applications or it can search for only those applications that are of interest at the moment.

4. Subscription

The subscription step is actually performed simultaneously as the search step. The agent reports to the tuple space that it is interested in changes or events in the space. Such changes can be when a new application is added or removed (or a new mobile client). Events can typically be application specific messages that are used for synchronization purposes.

5. Serve or Be Served

After completing the first four steps the application or user is ready for serious communication. Every application listens for requests and responds to them accordingly. In this step it is possible for a client to download necessary code to interact with the current application.

Steps 2 through 4 are basically all part of the registration process and these steps are all rather straightforward. Since this is a flexible system everything that happens in step 5 is very much application dependent. An implementation example is completely illustrated in the next chapter.

4.1.5 Use of XML

Every type of message and information is built using XML. XML is used because of its extensible nature, and applications should be able to

respond to new types of messages. The challenge here is to write the communication agents so they can handle all types of messages.

In my solution there are basically two different classes of XML messages namely XML profile messages and XML event messages.

4.2 Retrieving Tuple Space IP Address

An important step in the process of logging onto the system is step 1. There are several different imaginable ways of doing this. Even though this is an important issue, the way of retrieving the IP address is does not matter for my solution.

4.2.1 Well-Known Domain Names

The easiest way from a programmer's point of view is to not address this problem at all. By letting the user of the mobile client type in the domain name or the IP address of the communication space manually when entering a new location the problem is solved. The user could save different domain names and switch between them as he moves. When entering a new place there could for instance be signs on the walls that tell what the specific domain name of the particular location is.

Even though this solution is conceivable it is however not very practical for the user. And since one of the goals of the system is that it should work in a plug and play fashion this solution is no good.

4.2.2 Alternate Communication Interface

Yet another possible means for retrieving the server IP address is by using another communication interface. This could for example be Bluetooth or an infrared interface. Bluetooth has a smaller coverage than wireless LAN access points and therefore delimit different communication spaces with greater accuracy. With the help of a Bluetooth transceiver it would be possible to define a space within the range of the Bluetooth device.

To make an even finer distinction an infrared receiver and transceiver can be used. In this case one can think of a really limited space in respect of physical area or one can place several IR devices that sends out the same IP address. The problem with IR is that it needs line of sight in order to work. That means that a user cannot receive the signal if the device is in the pocket or in the briefcase. The user must consequently make an active decision and walk up to an infrared transceiver in order to log onto the communication space.

4.2.3 Multicast

Another possible way of solving this problem is to let the server multicast a message on a specific port at regular intervals to advertise its presence. The client agent always listens to this port and whenever it receives

information on this port it registers to the IP address of the server. This is a very straightforward solution but this also has some drawbacks depending on the application.

A network usually covers a rather big physical area, and thus a multicast on the network also covers the same physical area. The drawback of this is when you want the communication space to cover just a restricted area, as for instance a room or a part of the office. So the problem with multicast is that it covers a too big of area. This is of course dependent on what types of applications you wish to make use of. Sometimes it is perhaps wanted that the whole network should work together as a whole communication space and in this case the multicast alternative is preferable.

5 Implementation

This section gives a more detailed description of how the programming was actually done and explains how to add an application to the system. And to give a clearer understanding an example is described³.

5.1 System Components

The components needed to have a fully functional system are a TSpaces server, an HTTP server and an application server. These three entities can be located on the same physical computer since each one of them does not require a lot of resources. To make use of the system the user has to have a mobile client (the client does of course not necessarily need to be mobile).

5.2 Agent Description

Each entity (client or application) that wants to connect to the system needs to have a communication agent that communicates with other agents through the TSpaces server. The functionality of the agent can be different depending if it is a user or an application, but some features are common for all agents. Every agent implements an abstract class called `Agent`. This class contains the methods that are in common for all agents including an XML parser.

5.3 Use of XML

The use of XML plays an essential role in my implementation. XML is used both for user and application profiles as well for the eventing mechanism in the system.

XML supply the opportunity to format messages in a variety of ways. In order for the components in the system to be able to communicate at all there has to be at least some basic common syntax for the messages. The agents have to agree on some common tag names and they have to interpret them in the same way.

5.3.1 Element Content vs. Element Attribute

One of the first decisions to be made was to which extent I should make use of attributes in the different messages. The decision should be made in light of simplicity and readability. Below I show examples of two very simple XML documents that contain the exact same information but are presented in different ways.

³ All source code is available at http://www.e.kth.se/~e95_ewi/exjobb/code

```
<TITLE> Solitaire </TITLE> (1)
```

```
<TITLE value="Solitaire" /> (2)
```

The first row is an example of a tag containing the element data `Solitaire`, the second row is an example of an empty tag with an attribute, which has the value `Solitaire`. These two examples are conceptually the exact same thing, there is only a difference of semantics.

In my solution I made the decision to format all messages without attributes. This is mainly because it makes XML parsing much simpler when dealing with nested XML documents. The fact that I do not make use of attributes does not put any restraints on what can be supplied in the document. The disadvantage of not using attributes is that documents tend to get longer.

5.3.2 User and Application Profiles

In the registration process the agent writes the profile of the client or the application to the server in the form of an XML document. I have chosen to make the profiles as simple as possible at first. These profiles supply only a minimum of information about their capabilities. On the other hand it is easy to extend the profiles with new information since they are written in XML.

One should however not add new information in the form of tags and elements at random. It is important to know how the XML parser works. If the parser is assuming a particular format of the documents problems are undoubtedly going to arise. To write a general purpose XML parser in Java using one of many available tools⁴ is indeed a challenging task. If such a parser is available some type of intelligence still needs to be present in order to process this new type of information.

In order to minimize the time consuming work of writing such a general purpose XML parser I have limited the profiles to a predefined format. This might seem to be a great restriction in the system that I have built but this problem is instead solved using downloadable code. The format of the profiles is as follows:

```
<object>
  <type> [type | object] </type>
  <name> [name] </name>
  <jarurl> [URL] </jarurl>
</object>
```

The type can either be `user` or `device`. The name is a in the case of a user, a user name. If it is a device it is a name that describes what kind of device it is, e.g. a printer, a web camera etc. The jarurl contains a URL that points to where the downloadable code is situated. The code at this

⁴ Java API for XML Parsing or XML for Java

located is in the form of a Java Archive (JAR) and this can be invoked by a client. How this works is described in more detail below.

5.3.3 Events

There are basically two different kinds of events in the system. The first type of event is generated when a new entity is introduced or removed from the system. The other kind of event is created by the downloadable client and it is used to communicate with the application server.

These events are formatted as XML documents and the looks of them are application dependent. An example of how these events can be formatted is shown later in this section.

5.4 Downloadable Code

The main purpose for using code that can be downloaded is so that clients can interact with services without prior knowledge of this service. So the downloaded client contains some communication interface. The JAR file also typically includes a graphical user interface (GUI) so that the user can interact with the service.

5.4.1 JAR File Content

The content of the JAR file is very dependent of the specific application, but in common for all JAR files is that they contain the minimum amount of classes to support communication with the TSpaces server. The JAR file also has to contain a manifest file containing a Main-Class definition. This has to be present in order for the application that starts the downloaded code to know which class is used as entry point.

5.4.2 Accessing the Code

As described above, the user gets a reference to a JAR URL in the XML profile. This is situated at a HTTP server. This introduces a new logical entity in the system. But since the TSpaces server is equipped with a miniature HTTP server this is not a big problem.

To invoke a Java application class file that is packaged in a JAR file, I have used some of the classes that are present in the `java.net` package. This package includes classes that make it possible to start applications that are remotely located at an HTTP server.

5.5 An Example

Up till now I have only described how this system works and only briefly mentioned how this system can be used. A lot of the work during this project has been to develop a completely functional application that can be plugged into the system. The application I have developed is an electronic message board for mobile users. This message board is a screen that is

situated at a specific location where users can leave messages to each other.

5.5.1 Example Scenarios

One possible usage for such an application is to have an electronic message board placed at a location where people usually are waiting for each other, e.g. outside a restaurant or conference room. If for instance you are waiting for someone and this person is late you could post a message that tells this person that you have already gone in to eat. In this scenario the message board is used from peer-to-peer. These kinds of messages written to the board are examples of messages that are only relevant to a specific person at a specific time and place.

Another, perhaps more interesting, example is advertising directed at a specific group of people. In this scenario the message board is located at a public hot spot. Every user has its own profile specifying its interests and if there are many users logged on to the system with an interest in movies, the message board could play a trailer of the latest movie playing at the local cinema. This is an example of a one-to-many message.

5.5.2 Description

I have focused on the first of the scenarios mentioned above, where communication is on a peer-to-peer basis. For the application to work at all, the mobile user needs to have the communication agent installed and running on the mobile device. When the user enters a space in which a message board is present the message board displays a welcome message to the user. If the user has a message waiting the user is informed of this.

If the user wants to leave a message at the board the user can invoke the application and then code is downloaded and executed at the mobile device. The code that is downloaded contains a GUI and can set up a connection to the TSpaces server to communicate with the message board server.

Figure 5.1 shows how the downloaded GUI looks like. It contains fields for addressing, expiration and the message itself. In many cases the kind of messages that are used are only of importance for a limited amount of time. That is why the user has the ability to specify the period of time the message should be present at the server. After this time the message is automatically discarded by the server.

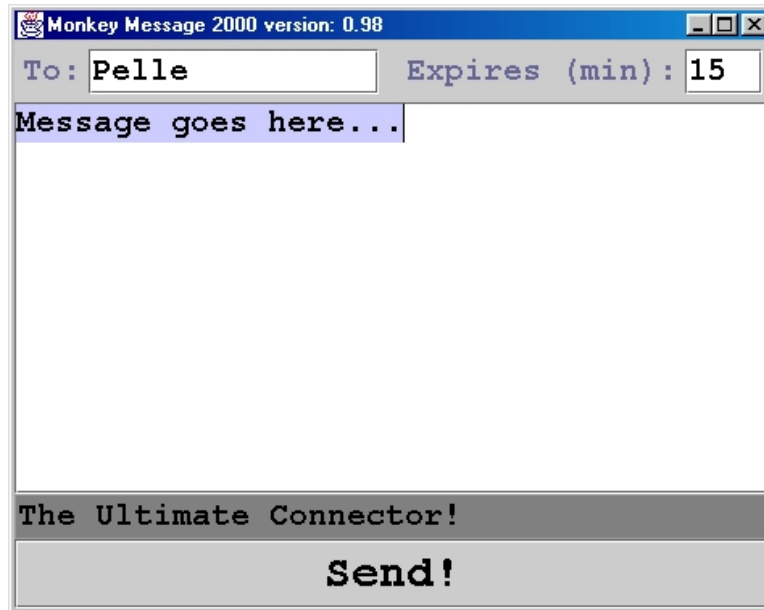


Figure 5.1: User GUI for Message Board

Upon clicking the send button an XML message is generated and written to the TSpaces server. The format of the message is as follows:

```
<action>
  <postmsg>
    <to> [username] </to>
    <from> [username] </from>
    <msg> [message body] </msg>
  </postmsg>
</action>
```

Whenever a message of this format is written to the server the recipient is informed about it through the event notification mechanism, or if the user is not currently logged on he is informed when he enters the space. The user can then decide whether to have the message displayed on the board or on the screen of the mobile device.

This example is simple but it shows how this system could be used. It could of course be discussed how useful this application would be in real life. If you want to inform someone where you have gone, it would probably be simpler just to give the other user a call or to send him an e-mail.

6 Conclusions

The work I have done in this thesis is related to many different areas. There has been no problem finding information about the various areas, the problem has rather been to limit the scope of the work and to find out what parts that are relevant. In this section I will try to summarize the opportunities and restrictions with the system that I have built and used.

6.1 TSpaces

TSpaces is a network middleware with some very nice features and the people at the Almaden research center are continuously making new improvements. The XML support of TSpaces is a trait that has many advantages but still, it is not optimal. For one thing every client needs to have an XML parser available to extract any information in an XML document at the TSpaces server. If the client is only interested in a small fraction of the XML document it would be much more pleasant if it would be possible to extract a tag element or attribute directly from the server. This is however one of the issues that will be confronted in future releases of TSpaces.

A very interesting article on the possible uses of TSpaces is presented in “A Universal Information Appliance” [33]. This article presents many similar ideas to what I have discussed in my thesis work. In short it talks about how a PDA can be used to interact with all sorts of devices using a TSpaces based system.

6.2 XML

I have on numerous occasions in this paper stated the many advantages of XML and it seems like every major company with any self-respect is looking for an XML based solution to every possible problem. There are mainly two questions that one should ask oneself before going headlong into XML when developing any kind of system. Why do I need XML? Should I believe the XML-hype?

The extensibility of XML is always considered to be one of the strongest feature of XML. This in combination with the many XML tools available is a strong motive for using XML in many situations. When creating web pages, or when building an e-commerce web page, or when structuring any kind of data XML can be an excellent choice of method. But the most winning reason for wanting to use XML is that it is entirely text based. This implies that every device can access the information regardless of platform and operating system.

In the work I have done in this thesis I have worked only with Java and TSpaces. If all devices in the system are Java-enabled the use of XML is somewhat superfluous. Using XML puts an unnecessary burden on each

device because of the required and problematical XML parsing. Furthermore, using XML messages for eventing doubtlessly adds additional overhead to the system.

Using XML to publish application and user profiles might seem to have a better employment than using XML for events. This is because of the fact that XML gives the profile writer the ability to come up with new ways of writing profiles. This advantage is however in many cases only imaginary since a new type of profile needs to be understood by every entity in the system in order for it to be effective. No matter by what means information is mediated in the system a predefined standard *must* be present.

My point is that if only Java-enabled devices are present there is not much motive for using XML. If I were to redo my work I would have used ordinary tuples for information passing rather than to use XML. In the system I have built the eventing and message passing could more easily have been accomplished if XML was *not* used.

6.3 Useful Applications

The starting point when developing a mobile resource awareness system should be how the system should be used, what applications and devices that should be made available in the system. When I started this work the usage of this system was very vague and in the beginning I should have focused more on defining how the system should work. Because of this uncertainty the result of my work is somewhat sprawled. The problem I was faced with was to make a general-purpose solution with no specific goal in mind. It would have been better to focus on a specific case and from there on draw some conclusions.

The most promising application is a form of mobile ICQ where a list of users (or *friends*) is shown in a window at the client. This gives the possibility to see which friends are present at the current location.

Service and resource discovery is one of the main application areas of use of the Bluetooth technology [34]. This area will undoubtedly obtain new insights when Bluetooth becomes more and more available. This type of service discovery has a somewhat different approach.

The way to design a resource awareness system varies quite a bit depending if the resources are hardware devices or software applications. The system that I have built is primarily built with respect of software applications although it is highly possible to connect a hardware device (although this requires some additional programming). If hardware devices were the main goal I would have designed it to be more similar to UPnP or Salutation.

7 References

- [1] T. Kanter, "Event-Driven, Personalizable, Mobile Interactive Spaces", Accepted for the second International Symposium on Handheld and Ubiquitous Computing (huc2k), 24-27 September 2000, Bristol UK.
- [2] WAP Forum (Wireless Application Protocol Forum Ltd.) Specifications, <http://www.wapforum.org/what/technical.htm>
- [3] 3GPP (Third Generation Partnership Project), 3G Specifications, http://www.3gpp.org/3G_Specs/3G_Specs.html
- [4] IEEE Wireless Standards Zone, <http://standards.ieee.org/wireless>
- [5] Telia Homerun: <http://www.homerun.telia.com/>
- [6] T. Kanter, "Do-It-Yourself-4G and Enabled Adaptive Mobile Multimedia Communication", accepted for the IEEE International Conference on Networking (ICN'01), July 10-13, 2001
- [7] Compaq iPAQ homepage, <http://www.compaq.com/products/handhelds/pocketpc/>
- [8] EPOC
- [9] ICQ (I Seek You), <http://www.icq.com>
- [10] AOL (America Online) Instant Messenger, <http://www.aol.com>
- [11] H. Tamura: "Mixed Reality: Merging Real and Virtual World," Journal of the Robotic Society of Japan, vol. 16 no. 6, pp 759-762, 1998.
- [12] Stanford Interactive Workspaces Project, <http://graphics.stanford.edu/projects/iwork/>
- [13] Smart Space Laboratory, National Institute of Standards and Technology, <http://www.nist.gov/smartspace/>
- [14] Jini Network Technology Specification, <http://www.sun.com/jini/specs/>
- [15] Universal Plug and Play (UPnP) Forum, <http://www.upnp.org>
- [16] Goland, Cai, Leach, Gu, "Simple Service Discovery Protocol," Internet Draft, October 1999
- [17] Simple Object Access Protocol (SOAP) Specifications, <http://msdn.microsoft.com/workshop/xml/general/soapspec.asp>
- [18] Cohen, Aggarwal, Goland, "General Event Notification Architecture," <http://www.upnp.org/draft-cohen-gena-client-01-txt>, September 2000
- [19] Salutation, <http://www.salutaion.org>
- [20] International Organisation for Standardization homepage, <http://www.iso.ch>
- [21] Intel UPnP Development Kit for Linux, <http://developer.intel.com/ial/upnp/>

- [22] Extensible Markup Language (XML) 1.0, W3C Recommendation, <http://www.w3.org/TR/2000/REC-xml-20001006>
- [23] Document Object Model (DOM), <http://www.w3.org/DOM>
- [24] Resource Description Framework (RDF) Model and Syntax Specifications, W3C Recommendation, <http://www.w3.org/TR/REC-rdf-syntax/>, February 1999
- [25] The Dublin Core Metadata Initiative, <http://purl.oclc.org/dc/>
- [26] Platform for Internet Content Selectivity, <http://www.w3.org/PICS/>
- [27] IBM TSpaces, <http://www.almaden.ibm.com/cs/TSpaces/index.html>
- [28] Carriero and Gelernter, "Linda in Context," Communications of the ACM 32, No. 4, 444-458, April 1989
- [29] CORBA Documentation, <http://www.omg.org/technology/documents/formal/index.htm>
- [30] XML Query Language (XQL) Proposal, <http://www.w3.org/TandS/QL/QL98/pp/xql.html>, September 1998
- [31] Handley, Schulzrinne, Schooler, Rosenberg, "SIP: Session Initiation Protocol," RFC 2543, <http://www.ietf.org/rfc/rfc2543.txt>, March 1999
- [32] Perkins "IP Mobility Support," RFC 2002, October 1996, <http://www.ietf.org/rfc/rfc2002.txt>
- [33] Eustice, Lehman, Morales, Muson, Edlund, Guillen, "A Universal Information Appliance," IBM Systems Journal, vol 38, November 1999.
- [34] Bluetooth Homepage, <http://www.bluetooth.com>