

State of the Art: Middleware in Smart Space Management

Sinead Cummins: Cork Institute of Technology
Alan Davy, Jason Finnegan, Ray Carroll: Waterford Institute of Technology

Abstract

Ubiquitous computing, and specifically Smart Spaces, is an emerging paradigm for interactions between people and computers. Its aim is to break away from desktop computing to provide computational services to a user when and where required. Large numbers of heterogeneous computing devices provide new functionality, enhance user productivity, and ease everyday tasks. In home, office, and public spaces, ubiquitous computing will unobtrusively augment work or recreational activities with information technology that optimizes the environment for people's needs (Manuel Román et al, 2002). The aim of the M-Zones program is to do novel research into the management concerns connected to such spaces.

The rationale behind this paper is to review the existence (or lack) of a suitable middleware infrastructure for the development and management of applications and services in ubiquitous computing environments. This paper will present current practice in middleware infrastructure, analyse the requirements of a ubiquitous computing environment, and draw conclusions as to its relevance to the M-Zones project and future research required.

This paper will begin by giving an overview of current middleware practices, analysing the requirements of a ubiquitous computing environment, and drawing conclusions as to its relevance to M-Zones. It will then proceed to look at specific aspects of middleware such as service creation (OSA/Parlay), networking architectures (Jini) and management using agent-based technology.

Keywords: Smart Spaces, Middleware, Jini, Agents, Parlay, Service Creation Environments, Application Servers

1 Introduction

This paper provides a state of the art overview and a discussion of important approaches in the field of middleware. Special focus lies on the point of view of Smart Spaces and Managed-Zones (M-zones) with their specific requirements. A Smart Space is a physical space rich on devices and services that are enabled to act autonomously on behalf of an individual user. An M-Zone is an administrative domain that deals with all issues on managing one or more Smart Spaces in order to run services and devices in an efficient way.

A major topic for Smart Spaces is the provision of a middleware. Such a middleware should hide specific (and mostly proprietary) aspects of the infrastructure (devices) to ease the development and deployment of components for Smart Spaces. This is an important basis for intra- and inter-domain service offerings as well as their management. Furthermore, this provides users the opportunity to roam between Smart Spaces or even M-Zones without losing their well-known service environment.

This paper investigates available and mature middleware technology from three different viewpoints. Section 2 starts with a general introduction into middleware and a categorization of middleware

technology. This general overview should enable the reader to understand the different approaches of middleware technology deployed today. Furthermore, the section 2 introduces three technologies in more detail, whereas all of these three technologies have been identified as potential candidates for a Smart Space middleware by the M-Zone program. These technologies are Jini (for device access), mobile and intelligent agents (as a technique to generate a flexible service environment, including decision making components and mobility components) and finally OSA/Parlay as the momentarily only solution to access a telecommunication network as a service provider.

Section three defines the requirements of Smart Spaces with regard to middleware technology. These requirements are kept generic (e.g. scalability, re-configuration, quality of service) in order to allow the identification of criteria for the evaluation of middleware technology. The requirement part of section three is followed by detailed discussions of Jini, Agents and OSA/Parlay.

Section four describes the future directions of research work within the M-Zones program. The authors of this paper intend to give guidelines and recommendations for the application of middleware technology to Smart Spaces. Furthermore, the applicability of the three candidate technologies will be investigated. This process leads to a number of important questions like:

- Is the integration of Smart Spaces with telecommunication networks feasible?
- Is management by delegation a suitable paradigm for Smart Space management?
- What approaches can be taken to integrate middleware and management, e.g. to build a decentralized management system that incorporates SNMP and (mobile) agents?

This paper can not give answers to those questions. This paper is intended to provide the necessary background information that allow researchers of the M-Zones program to define, specify and later also to deploy a distributed management platform for Smart Spaces that is build upon state of the art middleware technology.

2 Overview

The following section starts with a general introduction into middleware and a categorization of middleware technology. This general overview should enable the reader to understand the different approaches of best practice middleware technology deployed today. Furthermore, this section introduces three technologies in more detail, whereas all of these three technologies have been identified as potential candidates for a Smart Space middleware by the M-Zone program. These technologies are Jini (for device access), mobile and intelligent agents (as a technique to generate a flexible service environment, including decision making components and mobility components) and finally OSA/Parlay as the momentarily only solution to access a telecommunication network as a service provider.

2.1 Middleware

The term middleware is used today to describe any technology that provides an abstraction from the heterogeneous components of computer environments, like hardware platforms, networks, protocols, operating systems, applications, groupware, and databases. Middleware ties components of a distributed application together and solves the problem of scalability (Bakken, 2002). It supports application deployment using well defined, probably standardized, Application Programming

Interfaces (API). Those APIs and related interfaces decouple the applications from technologies as shown on Figure 1- Middleware – Logical View of Middleware

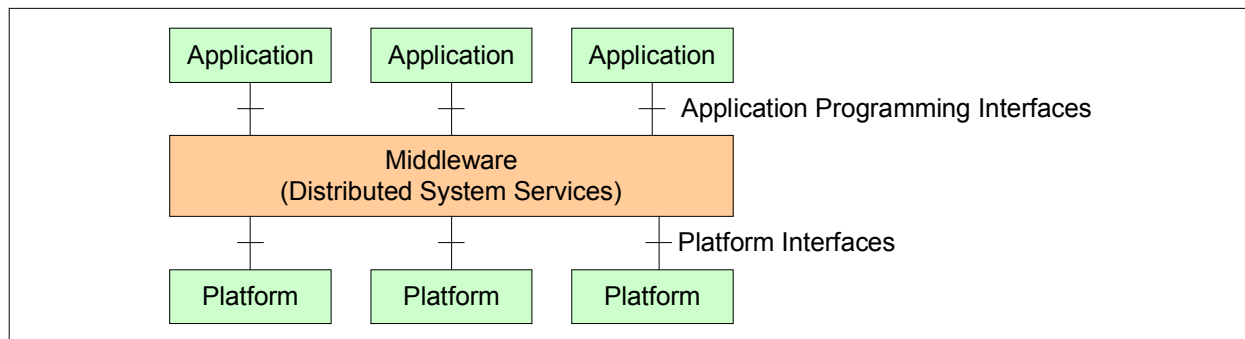


Figure 1- Middleware – Logical View of Middleware

Two different types of middleware are present: Inter Process Communication (IPC) and Distributed Transaction Processing (DTP). IPC based middleware systems operate with a direct communication among distributed components. IPC based middleware is made up of Remote Procedure Call (RPC) middleware, Message Oriented Middleware (MOM), and Object Request Broker (ORB).

DTP based middleware realizes the link between a client and any kind of database. Those middleware systems define abstract interfaces to databases for handling transactions and concurrency. DTP based middleware includes portable Transaction Processing (TP) Monitors and interconnected Database Management Servers (DBMS).

2.1.1 Paradigm of Client – Server

Distributed objects communicate in a client/server relationship. The client calls operations on a server and receives return values. The server offers those operations on its interface(s), runs the business logic once an operation is called, and generates the return value. The technical realization of interfaces on the server to be used by the client and the communication between client and server is the task of the middleware.

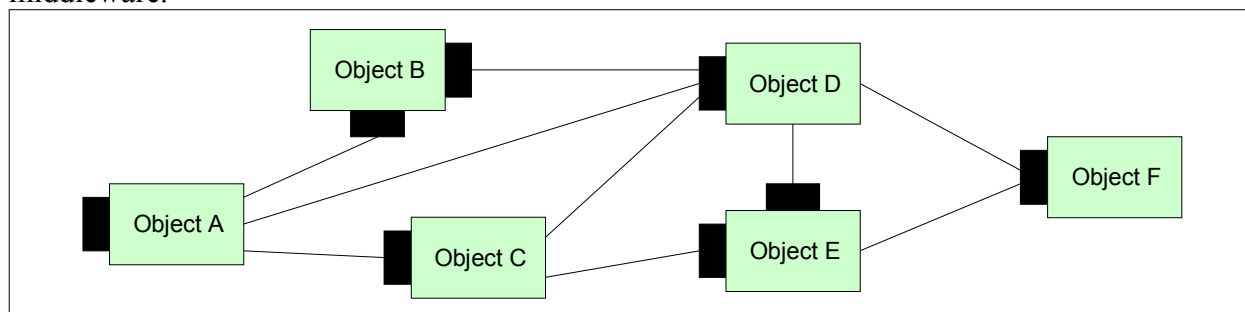


Figure 2 - Middleware – Distributed Objects

The actual relationships between distributed objects are not defined by the paradigm. Each object can access every other object. The definition of the relationships needs to be done in the development process of the distributed application. Hierarchical structures are not supported. They need to be realized individually.

2.1.2 Message Oriented Middleware

Message Oriented Middleware accomplishes the communication of application components through the exchange of records called messages (A. Campbell et al, 1999). Messages are strings of bytes that have meaning to the applications that exchange them. MOM is event-driven. It can support asynchronous as well as synchronous communication. However, there is no general standard or common set of specifications. At least three different techniques for the exchange of messages are available.

Message passing applies a direct communication model. A message is sent directly from one application to another. The model is connection-oriented. A logical connection between applications needs to be established. The exchange of messages can be either asynchronous (via a polling model or by callback routines) or synchronous (sender blocks until a message is returned).

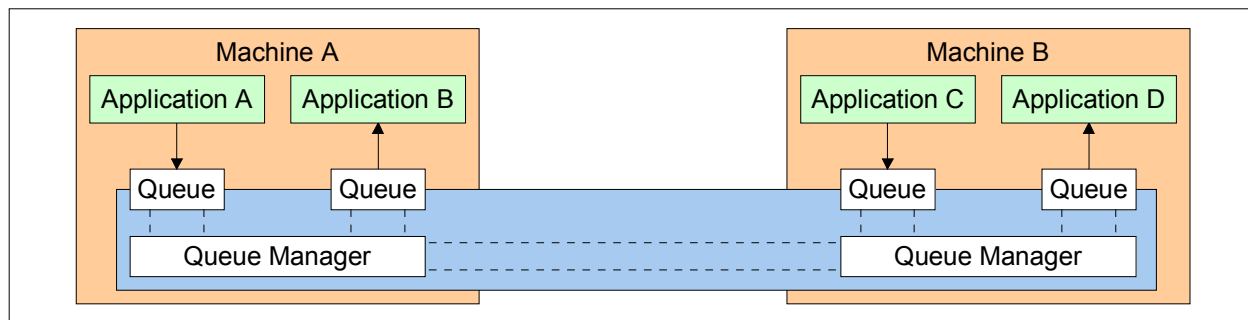


Figure 3 - Middleware – Message Queuing

Message Queuing applies an indirect communication model. Applications communicate via a message queue. This model is connectionless. Messages are put in queues for immediate or subsequent delivery. A Queue Manager is responsible for message handling and delivery. Message queuing implies the support for different types of Quality of Service (QoS), such as reliable message delivery (no packet loss on the network), guaranteed message delivery (messages are delivered immediately or eventually, at least after a specified period of time), and assured non-duplicate message delivery (message are delivered only once).

Publish and Subscribe is based on trading. Publishers produce information and offer them. Subscribers sign in to certain topics and receive messages. Publishers and Subscribers usually do not know the others existence. Applications can be loosely coupled increasing the flexibility of the design and operation of a system. Those systems can be reconfigured dynamically without interrupting operations. New applications can be added without disturbing existing applications. For more information refer to (M Erzberger, M Altherr, 1999).

2.1.3 Remote Procedure Call

The Remote Procedure Call is a common accepted and widely used technique for the communication between applications in distributed environments. All major operating systems support client/server communication via logical RPC interfaces. That is, clients and servers have local, logical interfaces that

are called stubs. Those stubs realize proxy objects for remote functions. To the client, the stub looks like the remote procedures thus providing location transparency (hiding the actual location of the server).

The logical function calls on the RPC interface are mapped to physical invocations on the stub. The latter one has the knowledge on how to access the network's transport layer to send the request to the server and to receive the reply for the client. The client itself usually blocks until the server replies to the remote call. On server side, the stub calls the requested function (procedure), as the client would have done if called locally. The programmer of the server has to provide some functionality to support a remote call, like directory, security, etc.

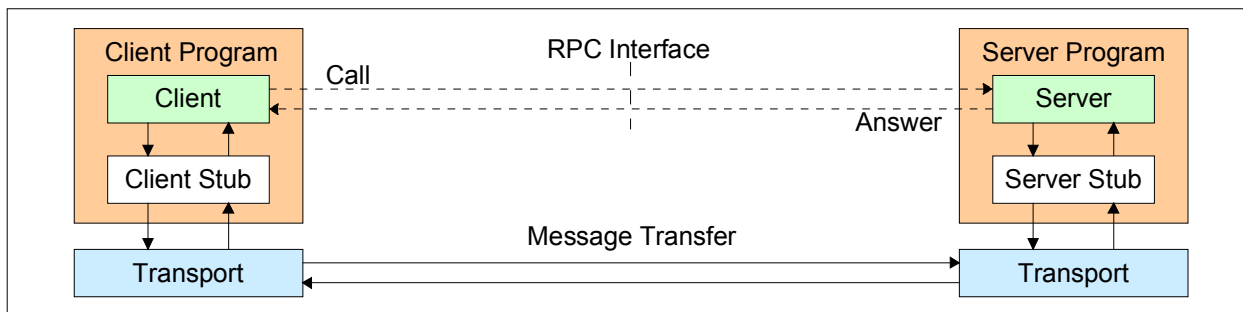


Figure 4 - Middleware – Remote Procedure Call

To use the network's transport layer for data exchange between client and server, the data has to be streamed through a communication channel. The serialization of the data for the transport inside of the network is called marshalling. The serialization requires that all data to be transported be pre-described, including type, format, and length. Such a description is to be provided by the programmer of the server in form of an interface definition using an Interface Definition Language (IDL). An IDL is a high level, universal notation that is capable to describe interfaces independent of the actual programming language (Bakken, 2002).

2.1.4 Object Request Broker

Object Request Brokers can be classified into a group compliant to the Object Management Group's CORBA and one compliant to Microsoft DCOM. In the following paragraphs, a look at a generic ORB architecture is explained, which is the basis of both CORBA and DCOM (J. Bacon et al, 2000).

The ORB is a software component that enables objects to initiate requests and to receive responses. All inter-object communication happens via the ORB, independent of the actual location of the objects (local or remote). The ORB enables objects to communicate over networks with different communication protocols and to reside on different hardware platforms. It provides the mechanisms by which objects make requests and receive responses. Furthermore, it is responsible for managing and locating the objects, supporting both inter-object communication and communication between objects and external services.

Most ORB products provide a set of components as specified by the Object Management Group (OMG). At first, an IDL is used as notational language to describe interfaces of objects. It is

implementation-neutral and needs specifications for the mapping to implementation languages. An Interface Repository contains all IDL definitions for attributes, operations, user-defined types, and exceptions. The Basic Object Adapter (BOA) represents the interface between the ORB and server applications. It dispatches objects that the server application maintains, and exchanges messages with the server objects. The Static Invocation Interface (SII) is a stub-based interface used by client programs in order to invoke services on application objects. Finally, the Dynamic Invocation Interface (DII) provides a generic interface that does not require stubs, but supports dynamic construction of object invocations by the client program at runtime.

The ORB establishes a client/server relationship among objects. A client can transparently invoke any method on any server object. The ORB intercepts the call, looks for an object that has implemented the call, passes the parameters to that object, invokes the method, and returns the result. Clients can invoke either 'one way requests' when no result is expected or synchronous requests.

The ORB itself is responsible for locating the server object by use of the implementation repository, to exchange data between client and server (including marshalling), to invoke the server (dynamic server invocation), and to recover from failures of the server object. The ORB provides objects with services like naming, lifecycle, property, relationship, query, and licensing.

2.1.5 Distributed Transaction Processing

Transaction Processing (TP) has been used for mainframe-based applications where TP Monitors have managed the limited number of connections to databases. They have acted as connection concentrators, reducing overhead and increasing the performance of database access. Today, TP Monitors are used as a solution for transactional integrity in database environments (P.A. Bernstein, 1990). They support the so-called ACID properties that describe the quality of transactions:

- **Atomicity:** All operations that an application performs, which involve updates to any kind of resource, are grouped into a "unit of work." This unit is referred to as atomic, meaning it is indivisible. Any partial completion (due to system failures) will be rolled back.
- **Consistency:** At the end of a transaction, all resources that have participated will be in a consistent state.
- **Isolation:** Concurrent access to shared resources by different units of work (performed by different applications) is coordinated so that they do not affect each other. Transactions that compete for resources are isolated from each other.
- **Durability:** All updates to resources that have been performed within the scope of a transaction will be persistent or durable.

The standard for DTP represents a basis for TP Monitor products. It allows programmers to share resources keeping the overall system in a consistent state. Databases are connected to the TP Monitor via a standardized interface called XA.

The XA interface is the bi-directional interface between a transaction manager and a resource manager. The XA interface is not an ordinary Application Programming Interface (API). It is a system-level interface between DTP software components (Technical Standard a, 1992).

Applications use the TX interface to communicate with the TP Monitor. The TX (Transaction Demarcation) is the application-programming interface by which the application program calls the Transaction Monitor to demarcate global transactions and direct their completion (Technical Standard b, 1995). The Standard Transaction Definition Language (STDL) provides a TP Monitor independent API in the form of a higher layer, procedural language for the description of transactions.

2.2 Jini

2.2.1 Middleware and Jini

In the following review, Jini will be examined under headings, which describe the key requirements of any network Middleware technology. These broadly are designed to provide access to services on the network, in an abstract and flexible manner, in order to satisfy requesting users. Users typically can be software client processes or possibly specific applications run by human users. The middleware thus stands in the middle between the servers and the clients facilitating the communication. For networking Middleware this general aspiration can be broken into requirements in the following areas.

- Networking
- Service Description
- Service Connection
- Connection Management
- Reliability
- Scalability
- Security

Jini has specific capabilities in each of these areas, which we examine separately. In addition, we will examine the main competing technologies to Jini, before reaching a conclusion as to its applicability to the M-Zone research.

2.2.2 Introduction to Jini

Jini is a network infrastructure that runs on top of Java, which allows services to dynamically join and exit without impact on the network or the network users. As a service based architecture, Jini provides a solution for the evolving ubiquitous computing requirements. Kumaran [2002,Preface] explains that “It abstracts both the devices and software under a service notion and supports dynamic community formation and dissolution.” This community, or federation is described by Mahmoud [2000,pg 244] as “a set of services that can work together in a single distributed computing space, to perform a task” Jini is a technology that will allow developers and manufactures create a range of computerised devices that can instantly connect both wired and wireless into a network to share services regardless of the underlying operating system or hardware. “Jini software gives network devices self-configuration and self-management capabilities; it lets devices communicate immediately on a network without human intervention.” Stang & Whinston [2001,pg33]. The networks are described as “self healing” Stang &

Whinston [2001,pg33], in that devices can leave a network without affecting the operation of the remaining devices, this eliminates network failure in the case of machine crashes or power surges. The means by which devices can achieve this is by leasing- a fundamental concept in Jini.

A Jini federation is made of look-up servers clients and services. A client finds a service by specifying a Java object type, and interacts with that service through a java object of that specific type. The java object can be downloaded from the network without the client having any knowledge of the service beforehand. We will now look at each of these fundamental concepts and how they are applicable to middleware software.

2.3 Agents

In section 2.1 a taxonomy of today's common middleware approaches was detailed. However agents are a new and evolving form of middleware technology and so are not often considered in such classifications. Agents differ from other middleware technologies in that they cannot be addressed under just one of the listed middleware approaches. Instead agents can incorporate a number of middleware technologies and concepts such as message passing, RPC, distributed object technology, etc. The volume of research carried out in this field indicates that agents need to be considered in this taxonomy as an emerging middleware approach.

The term "agent", in the computing context, has exploded in popularity in the last number of years. A simple count of the number of references to agents in the first 3 issues of IEEE Pervasive Computing could be taken as an indication of the current research popularity of agents. However 'Agent' can be a very ambiguous word and is often used to describe software elements that would not be classified as agents in the context of this paper. The purpose of this overview is to clarify that context as well as introduce agents to the novice reader. This will be done by examining where agents have evolved from, what exactly an agent is and the types of agents that exist.

2.3.1 History of Software Agents

Software agents have their roots in Artificial Intelligence (AI). Some of the earliest works on AI were the neural network (McCulloch & Pitts, 1943), Vannevar Bush's (1945) depiction of a computer assisted future in his article "As we may think" and the Turing Test (Turing, 1950). The idea of an agent originated with John McCarthy in the mid-1950's, and the term was coined by Oliver G. Selfridge a few years later. "They had in view a system that, when given a goal, could carry out the details of the appropriate computer operations and could ask for and receive advice, offered in human terms, when it was stuck" (kay, 1984).

Software agents really began to emerge from work that involved Distributed Artificial Intelligence (DAI). One of the approaches developed under this title was to use multiple, distributed intelligent agents or Multi-Agent Systems (MAS). According to Nwana (1996) this research first started in 1977 and is the first of two main strands of research into agent technologies. This strand concentrated on issues such as "...the interaction and communication between agents, the decomposition and distribution of tasks, coordination and cooperation, conflict resolution via negotiation, etc." This strand was much more focused on pure AI and many of the agent mechanisms were inherited from AI technology.

The second strand started in the early 90s, and here agents moved from their pure AI roots into a much wider variety of applications (e.g. Personal Agents, Information Agents, Negotiation Agents etc). Pattie Maes, now one of the world's leading voices on agents, lead the way with many well-received papers including "Agents that Reduce Work and Information Overload" (Maes, 1994). Agent research continued through the 90s and into the new millennium with much attention moving to the latest agent paradigm of mobile agents.

2.3.2 What are Agents?

An agent is a piece of software. However, the problem is that numerous definitions of how agents differ from conventional software exist (Franklin & Graesser, 1996) and reaching a consensus on a single definition is virtually impossible. The Oxford English dictionary provides the following definition of the word Agent:

"...a person or thing that exerts power..." and "...who acts or has the power to act"

When talking about software agents most definitions attempt to identify a set of properties they believe an agent should command. Some of these properties are (Nwana 1996), (Kampis 1998):

- Autonomous: act on its own without being controlled by others or by outside forces.
- Cooperative (Communicative): interact with other agents to fulfil some goal.
- Intelligent (learn, reason, adapt): acquire knowledge, process this knowledge in order to arrive at some conclusion and adjust based on this conclusion.
- Reactive: act as a response to some stimuli, event or condition.
- Proactive: act in advance to deal with an expected difficulty (anticipatory).
- Mobile: move from one location to another in order to fulfil a goal.

An agent does not necessarily possess all of these attributes and often there is a trade off between them. For instance, a key attribute in a mobile agent will obviously be mobility. This does not mean that it is not intelligent or cooperative. It still may have these properties, if required, but perhaps to a lesser extent than mobility. However, there is one property that distinguishes an agent from a common piece of software: autonomy. This power of self-government is the basic attribute required by all agents. This leads to the following definition of an agent: "A software component that can autonomously fulfil some goal or set of goals". The context of this discussion focuses on agents that are autonomous. Definitions from other areas, e.g. management agents or search agents, are not within the scope of this discussion.

One issue that has caused confusion in defining agents is the overuse (and misuse) of the term agent itself. In recent times more and more technologies appear to use the word agent to describe software components. Looking at the properties above many of these so-called agents could not justify the use of the term.

2.3.3 Relevance to M-Zones

M-Zones' focus is on the management of mobile and dynamic environments. Agent technology provides intelligent, flexible and mobile software that can facilitate a variety of management functions in such environments. Through the incorporation of the characteristics listed in section 2.3.2 (such as mobility, intelligence, cooperation etc) agents can be a very powerful technology, especially in

dynamic environments such as smart spaces, where intelligence and mobility can facilitate smarter and more efficient management.

2.4 Parlay/OSA

The purpose of the Parlay/OSA API's is to allow third party applications access to features of the telecommunication networks. Parlay provides a standard way for applications to access information such as user location or account balance from the telecoms secure servers. The Parlay APIs allow access to the telecoms network through a Parlay gateway. This gateway is inside of the telecom network and is customised for that particular network. Applications that want to make use of the parlay gateway are run on application servers. These application servers can be on the same intranet as the Gateway or can access it over the Internet.

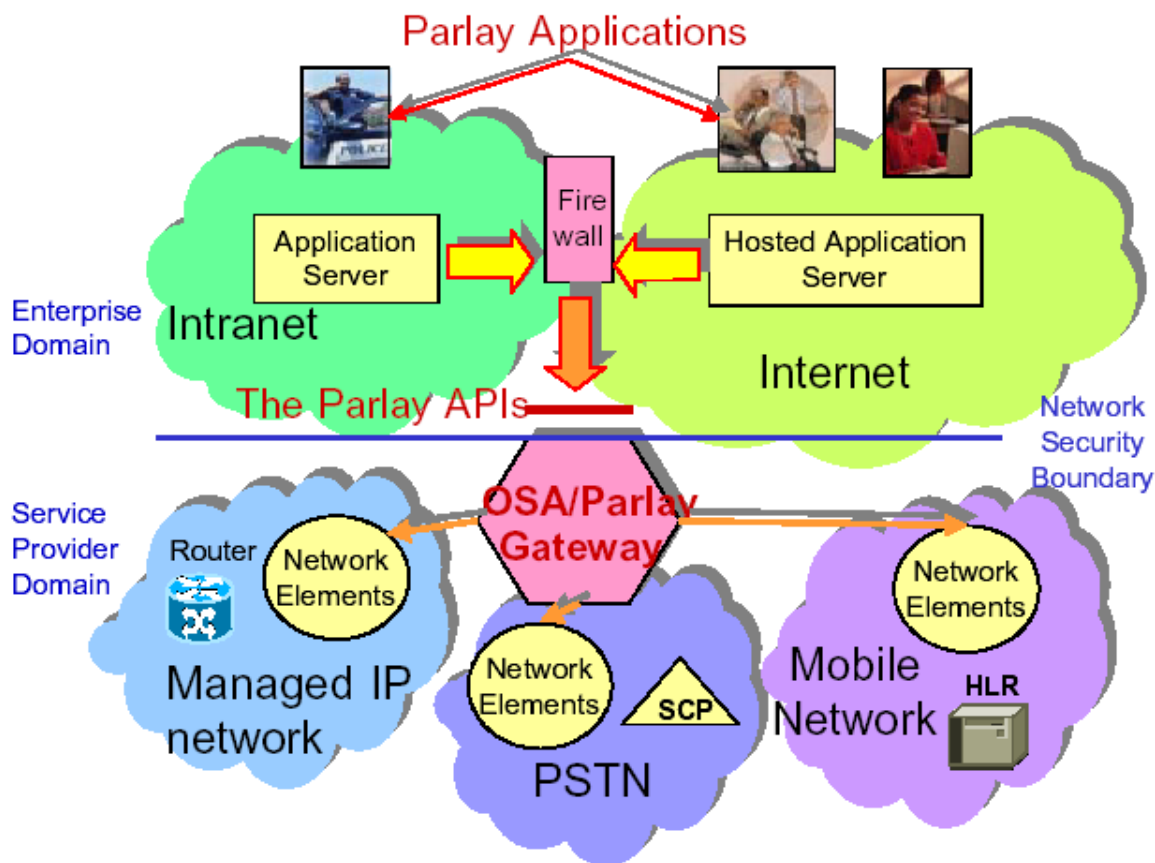


Figure 1 - Parlay gateway between networks

Relevance to M-Zones

The M-zones programme is aiming to “ Develop novel information and communications management technology to support dynamic, integrated management of participants, information appliances, and smart space infrastructure.”

The Parlay/OSA API's allow would allows us to integrate local smart space management with existing GSM , 3G and PSTN(Traditional Telephone) Phone Networks. This would enable M-zones management to switch sessions from Wide area Networks to Local Area Networks seamlessly and to locate Users at all times when outside of Smart Spaces.

3 Analysis

The following section defines the requirements of Smart Spaces with regard to middleware technology. These requirements are kept generic (e.g. scalability, re-configuration, quality of service) in order to allow the identification of criteria for the evaluation of middleware technology. With these requirements in mind this section is followed by detailed discussions of Jini, Agents and OSA/Parlay. Each of these three emerging middleware technologies deals with different aspects of the Smart Space. Jini is a middleware technology developed for smart devices access. Mobile and intelligent agents may be considered as a technique to generate a flexible service environment, including decision-making components and mobility components. OSA/Parlay momentarily is considered to be the only solution to access a telecommunication network as a service provider. The purpose of reviewing these middleware technologies is to highlight the fact that current state of the art middleware technologies will have to change the way they operate if they want to survive in the emerging smartspace environment (Kurt Geihs, 2001).

3.1 *Middleware*

The few ubiquitous computing environments that exist today tend to be highly specialised and based on application-specific software. Applications developed for interactive environments should be able to interconnect and manage large numbers of disparate hardware and software components. They should operate in real-time; dynamically add and remove components to a running system without interrupting its operation; control allocation of resources; and provide a means to capture persistent state information. Frequently these components are not designed to cooperate, so not only do they have to be connected, but also there is a need to express the "logic" of this interconnection. In other words, inter-component connections are not merely protocols, but also contain the explicit knowledge of how to use these protocols. Thus, viewing the connections simply as an application-programming interface is not enough. Cooperation among different applications is also difficult to achieve without a common platform. In order to model applications in this domain the need to define a common design methodology based on new paradigms independent from the technology is important. A model to abstract the main components of a ubiquitous computing environment is needed in order to formalize the development of interactive environment applications. These components may be classified into three abstraction layers:

- Physical deals with technological constrains.
- Middleware defines structure and the cooperation of abstract services.
- Application concerns the user interfaces. (Maffioletti, 2001)

Thanks to these abstractions a middleware will present a uniform access abstraction for different ubiquitous devices, allowing them to interact and cooperate. This will allow the writing of applications scaling both on services offered, and on devices composing the system. The model is intended to provide a standardized view of basic interactive environment functionality.

3.1.1 Breakdown of requirements

The main requirements that must be provided by a middleware infrastructure for services in ubiquitous computing would fall under the following headings.

3.1.2 Mobility

New wireless communication technologies provide connectivity for laptop computers and personal digital assistants, phone organizers offer the processing power, and application-level protocols such as the wireless application protocol—a tiny first step—allow convenient applications to run on these devices. Undoubtedly, these developments point toward the widely expressed goal of accessing and processing information almost “anywhere and any time.” Independent of a person’s current location, one can already use voice communication via mobile phones almost anywhere, and have worldwide access to personal e-mail, bank accounts, and home-country news over the Web.

Mobility introduces a key technical challenge because the available resources vary widely and unpredictably. Communication bandwidth and error rates change dynamically in wireless communication networks, a mobile system’s battery power decreases, portable devices can be temporarily switched off or unreachable because of network partitions, and the monetary cost of communication can vary significantly. Envisaging a middleware system that makes these dynamics transparent is difficult; so middleware must support the applications to explicitly accommodate these changes. Another mobile-computing issue, location awareness, demands that mobile-computer applications know their operating environment for context-dependent activities, such as giving directions or employing more or less stringent security mechanisms.

The characteristics of Message Oriented Middleware would be very suitable in the ubiquitous environment. MOM does not require constant connections to be held between communicating devices, as it can pass messages asynchronously, accounting for ad hoc connections in the ubiquitous environment.

3.1.3 Heterogeneity

The ubiquitous computing vision assumes that future computing will comprise diverse computing devices ranging from large computers to microscopic, invisible processing units contained in objects that are used in daily life, (e.g. light switched, mugs, etc). These computing devices may all need to communicate over different network mediums and protocols, (e.g. 802.11b, Bluetooth, IrDA, etc.).

Present middleware technologies such as CORBA can be implemented in different languages, letting heterogeneous ORBs talk to each other. Other OOM technologies such as SOAP communicate in a structured format (XML) over common network protocols (HTTP). Message oriented middleware can also provide data communication over diverse networks, such is the case for Java Messaging Service where as long as the applications communicating are running in a Java Virtual Machine (JVM) corresponding to their platform, communication is not a Problem.

Middleware for a ubiquitous environment must follow the characteristics of present middleware technologies in regards to the way they can be implemented in heterogeneous environments, and communicate over heterogeneous networks with dissimilar protocols.

3.1.4 Scalability

The middleware has to scale well for both a large number of cooperating services, which realize the application in each application context, and a large number of devices involved each time the application are used. Services represent the logical dimension of the application, while devices represent its physical dimension.

Different middleware technologies can handle being scaled in different ways. OOM such as CORBA would be considered a connection oriented middleware; this is to say if two or more orbs are to communicate, there must be an open connection between them. Middleware must provide applications with consistent functionality the more a system scales. In a ubiquitous environment the amount of connections between devices at any time could be immense. The management of these connections could require more resources than available, which is not desirable in such a resource-restricted environment.

Message oriented middleware mostly works on a connectionless bases, where the sender and receiver may be disconnected when communication is initiated. When a system, using MOM, is scaled to an environment with possibly hundreds of devices communicating with each other, such as a ubiquitous environment, messages being passed may build up because of devices being disconnected for unpredictable amounts of time. This may compromise the performance of critical systems running in a ubiquitous environment.

A possible solution to the scalability issue for ubiquitous environments is to group small areas together, such as a room. In this room, because the devices in the room may be made up of static and dynamically connected devices, it would be safe to assume a limited number of devices may be in operation at any one time. Using these assumptions, different middleware solutions may be used in combination with each other in order to control the scale of the environment. For example in a small-scale environment, an OOM may be used to control connections between devices, such as the .NET framework, which is based on XML Web services communicating through basic Web protocols. A collection of these environments could be considered another group communicating at a higher level, connecting these environments together. Using a method like this would break down the environment, into controllable scaling environment.

3.1.5 Configurable / Re-configurable

It is becoming increasingly apparent that most existing middleware technologies cannot accommodate the great diversity of application demands in areas like mobile and ubiquitous computing. The main reason for this is the black-box philosophy adopted by existing middleware platforms. In particular, existing platforms offer a fixed service to their users and it is not possible to view or alter the implementation of this service—that is, they are closed systems. This basically means that all network and connection information is hidden from the application layer, thus not allowing applications see changes in network topologies. Applications should be able to be configured to any network topology (802.11x, GPRS, Bluetooth, etc), thus the underlying middleware should not be considered as a black

box paradigm but more of a book that can show certain required information about the underlying network topology to the application layer.

Devices in ubiquitous environments will automatically detect other devices; forming ad hoc networks spontaneously such as Bluetooth technology. A middleware infrastructure for a ubiquitous environment will have to re-configure itself to changes in the network, and also be able to reflect this change at the application layer. The Java-based Jini system appears to anticipate these requirements. Jini defines a middleware infrastructure for spontaneous networking in which Java objects can discover, join, and interact with communities of devices. Whether the Jini approach will scale to the requirements of ubiquitous computing, however, remains unclear.

3.1.6 Quality of Service

Increasing concerns about service quality have led to several proposals that advocate integrating QoS management into networking and distribution infrastructures. QoS management at the middleware and application levels aims to control attributes such as response time, availability, data accuracy, consistency, and security level. From the Internet perspective, QoS concerns seem to arise automatically when commercial applications meet a best-effort communication environment. When clients must pay for a service, they are certainly concerned about QoS, and they will expect to pay less for lower quality. From a ubiquitous computing perspective, the guarantee of QoS is an even greater issue. For example if a heart monitor was part of a ubiquitous environment, then the guarantee of the quality and delivery of this important data could mean life or death. Also for public acceptance of a ubiquitous environment, people must be able to trust the security of personal data that can under go potential total monitoring (Kurt Geihs, 2000).

At present current products such as IntServ and DiffServ are being tested to guarantee QoS over IP. The new Internet protocol Ipv6, may also provide in itself QoS, but these are yet to be developed to a level of open use and acceptance. For CORBA middleware, the Object Management Group recently proposed new messaging extensions that support QoS guarantees such as message delivery and error handling. Research projects have produced specific Corba-based platforms for handling individual QoS categories such as real time (D.C. Schmidt, 1998) or replication and fault tolerance (S. Maffeis, 1996). Others have addressed generic frameworks for generating and operating customized systems for various QoS categories (C. Becker et al, 2000). Although interesting QoS management into middleware architectures is essential, a procedure for doing so has yet to be agreed upon.

The ubiquitous computing environment introduces extra issues regarding QoS that present QoS architecture for the Internet or wired networks do not cover. This new environment introduces problems such as wireless channel fading and mobility (M. Naghshineh 1997). These are two very important elements that have to be addressed if QoS architecture is to be successfully mapped to a ubiquitous computing environment. As these problems can affect the performance of a network and application, it is often proposed that QoS be handled at the middleware layer (K. Nahrstedt, 2001).

(Klara Nahrstedt, 2001) suggested that QoS-aware middleware architecture for ubiquitous computing environments has to address four main aspects of QoS. First, QoS specification to allow description of application behaviour and QoS parameters; Second, QoS translation and compilation to translate specified application behaviour into candidate application configurations for different resource conditions; Third, QoS setup to appropriately select and instantiate a particular configuration; Finally,

QoS adaptation to adapt to runtime resource fluctuations. Under these four aspects the problems that the ubiquitous environment introduces to QoS may be tackled.

An example of a problem introduced by ubiquitous environments would be the transmitting and receiving of multimedia data to a wireless media player application. Applications such as these can run using different configurations, i.e. streaming videos using ISO's MPEG layer 2 (ISO/IEC, 1994) encoding versus ITU's H.261 (ITU-T, 1993) encoding. The former method is for high quality video storage and transmission, which would require a large amount of network resources to be available. The latter being used for low bit-rate videoconferencing over narrowband integrated services digital network lines (ISDN) and telephone lines, respectively. In a ubiquitous computing environment due to fading signals, network bandwidth may drop considerably. In this event a QoS aware middleware infrastructure should be able to recognize that a change in network performance has occurred, notice that the application can handle different configurations, and adjust the video encoding method to suit.

In this example it is clear that the application layer needs to know some knowledge about the network layer and visa versa. QoS aware middleware should provide an abstract link between the two layers, controlling the application when network connectivity fluctuates. Another issue that should be considered is handing off between two networks. A hand off is where a mobile device moves from one network to another, vertical hand off being when a device moved from one network topology to the next (Wi-Fi to GPRS), and horizontal hand off being roaming between similar network topologies (Wi-Fi to Wi-Fi). QoS aware middleware should be able to adjust applications to react to any changes in network resources.

In ubiquitous computing environments, the question of guaranteed QoS is open. Will applications require a single Service Level Agreement (SLA) for it to function within acceptable parameters, or just based on a best effort approach? One possibility is that for each SLA, there is a subset of agreements that follow a set of further SLA's for different situations. This approach takes into account the unavoidable characteristics of the ubiquitous computing environment, such as network fading and network hand off.

Present technologies that provide some form of Quality of Service would include 2KQ (Klara Nahrstedt, 2000b), Agilos (B. Li et al, 1999), TAO (D. Schmidt et al, 1999), and QuO (J. Zinky et al, 1997). These middleware infrastructures provide a method of application configuration and adaptation to environment changes through QoS programming environments. In a ubiquitous computing environment, containing a wide variety of devices, network topologies and applications, it would be necessary to have custom designed QoS resolutions to suite different configuration changes applications may need to undergo in the event of fading or mobility (Klara Nahrstedt, 2001). This implies that the most suitable application service paradigm for ubiquitous computing is adaptive in nature; hence the provision of this adaptive quality may be provided at the middleware layer (Markus Lauff, 2000).

3.2 Jini

3.2.1 Networking – Federations

Jini relies on a network for its existence. A Jini network consists of a network of many services. Applications are created by combining these services in groupings called federations. Li [2000,pg12] explains that a Jini client can use the services of a federation by first joining that federation. This same client can then at a later stage, leave this federation and join another federation in order to access its services. Any service within a Jini federation may make use of other services in order to perform its own task. This means that a Jini service can also act as a client of another Jini service. The lookup service then controls actions within the federation.” The lookup service decouples the client-to-server relationship, enabling distributed dynamic reconfiguration.” [Li, 2000,pg14]

A Jini federation is made up of three vital components: servers, clients and lookup services. Every Jini network must have one or more lookup servers. The lookup server is where all the client and services information is exchanged. Here “All the services announce their availability and unavailability, so that clients can notice the existence of them” [Rong 2001, pg 148].

3.2.2 Service Description – Attributes

Edwards [2001,pg 246] explains, “Attributes are Java objects that are attached to service proxies” Services attach these attributes when they publish their proxies, and clients can search for proxies by looking for certain attribute patterns. They can then download the attributes of a service to examine them more closely. Attributes are used to add extra descriptive information to a service. For example a printer may have a location attribute to specify where the printer is, or a status attribute to determine if there is paper in the tray or not. These attributes are Java objects that are attached to the service proxy.

3.2.3 Service Connection – Lookup

A Jini service finds the lookup service in which it must describe itself through a multicast protocol called discovery, it then registers in lookup services with its proxy along with some attributes. The proxy is a java object, which is capable of carrying out the services duties and the attributes simply describe the information of the service. The lookup service maintains a map between each Jini service and its attributes. “When a new Jini-enabled device advertises its service, the lookup server adds that information to the map.” [Stang & Whinston 2001, pg35]. A Jini Client can then request from the lookup server a list of Jini servers that match the requested attributes. The client can then select the desired server from the returned list. The client downloads the proxy of the wanted service and uses this proxy to communicate with the service itself and the lookup service no longer participates in the rest of the communication process between the client and the service.

A unique feature of Jini technology is its lack of necessity to transfer significant executable code between Jini devices. “Jini software sends only the code for the interface that the client uses to communicate with the server; the rest of the program remains on the server and actually executes there” [Stang & Whinston 2001,pg 36] This strategy overcomes the problem of slow speeds faced by

comparable strategies such as applets. Jini systems download less “because a Jini network usually only transfers the results of whatever code the server executes” [Stang & Whinston 2001, pg36]. When a Jini client receives the lookup service reply that responds to its request for a service, it initiates a connection with the server. The Jini specification does not stipulate how an application implements a service. The only requirement is that a Jini service must implement a specific interface; and it is this interface that the server gives to the client. The client has no prior knowledge of any server and only knows how to communicate with the server through the given interface.

3.2.4 Connection Management – Leasing

Jini’s approach to self-healing is through a technique called leasing. Jini requires that resource holders continually renew their leases on resources. Edwards[2001,pg80] considers the scenario of a digital camera which has joined a federation. The camera announces the fact that it is available for use. However consider the scenario where the user indiscriminately dislodges the camera from its cradle without turning it off. To the other members of the federation, this may look like a partial failure situation, they may not be able to determine if the remote host to which the camera is connected has gone down, if it’s simply slow to answer, if it’s not answering network traffic because a change in its configuration, if the camera’s software has crashed or even if the camera has been smashed with a hammer. Regardless of how it was disconnected, it has not had a chance to unregister itself from the middleware system before disconnecting. Services that wish to use the camera will see it registered but will not be able to use it. From the point of view of the lookup service for middleware, if service registrations are never properly cleaned up they will accumulate and slow down operation. The system is not self-healing, partial failures aren’t recognised and cleaned up, and services that hold resources on behalf of others may grow without bound. This flawed system will also eventually require explicit human intervention to administer the system.

Jini uses leasing as a means to overcome these problems. Rather than granting access to resources for an unlimited amount of time, the resource is “leased” to some consumer for a fixed period of time. In order to maintain a resource, a resource consumer must show continued interest in the resource. A Jini lease may be denied by the grantor of the lease. They can be renewed by the holder. Leases will expire at a predetermined date unless they are renewed. This provides a lightweight and distributed management of resource allocation. In the case of the dislodged camera, since the it will not grant new lease or re-grant any existing leases, no new users will be allowed to use it and existing users will unregister themselves automatically. Leasing ensures that the management of persistent storage and services used by the members of a Jini community is distributed and virtually maintenance free.

3.2.5 Reliability

Stang and Whinston [2001,pg33] define reliability as a measure of “how well a device or network performs in the presence of disturbances”. However processes often crash, or someone will shut them down. Or the machine running the process will crash or stop. Jini is capable of managing these changes due to the fact that it expects devices to randomly move in and out of the network. In Jini when a server becomes unavailable, the client automatically goes looking for an alternate server. “Once it locates another server process (or the failed process comes back up), the Jini client can reconnect. If no server

is available, the client waits or informs the user” [Stang & Whinston, 2001,pg34] This functionality is transparent to the user.

Edwards[2001,pg62] explains that “Jini supports serendipitous interactions among services and users of those services”. Services can enter and exit a federation in a very “lightweight” manner. Interested parties can be automatically notified when the set of available services changes. Furthermore, every device or service that connects to a Jini community carries with it all the code necessary for it to be used by any other participant in the community.

These features combined make Jini virtually unique among commercial-grade distributed systems infrastructures. They make a Jini community virtually administration-free. Edwards [2001,pg63] explains “A cooperating group of Jini services will be resilient to changes in network topology, service loss, and network partitions in a clean way.”

3.2.6 Scalability

A system is defined as being scalable by Stang and Whinston [2001,pg34] if “the overhead required to add more functionality is less than the benefit that functionality provides.” Thus if adding a server to the network makes it more difficult to accomplish a task, the system is not scalable. “Adding Jini services to a system gives clients more choices in the devices they can communicate with.” [Stang & Whinston, 2001,pg 34]. Thus more devices providing the same service increase a Jini system’s reliability.

“Jini addresses scalability through federation.” [Edwards 2001,pg 64] Federation is the ability for Jini communities to be linked together or federated into larger groups. The ideal size for a single Jini community is a workgroup, consisting of a number of printers, scanners, PDA’s and network devices required by a group of 10 to 100 people. Jini groups can then be brought together in a community in order to make their resources sharable. Access to services within other workgroups is possible through federation. “Specifically, the Jini lookup service-the entity responsible for keeping track of all the services in the community- is itself a service” [Edwards 2001,pg 64]. A lookup service of one community can register itself in other communities offering itself as a resource for the community.

A Jini federation is self-managed; a device inserts itself into a network. Jini technology “dynamically discovers the services it needs for processing client requests” (Stang &Whinston 2001,pg 34). Jini federations are designed to exist in a flexible environment thus making them more scalable.

3.2.7 Security

Within Jini, the main security issues are to prevent unauthorised or unknown clients from accessing protected services. Jini tackles this by ensuring that remote clients can be prevented from even seeing what services are offered by the network. Only when they become part of the federation or network are they allowed see its services. This provides a first line of defence.

The architecture of Jini also fundamentally protects against Viral attack. Jini servers provide a client the interface with which it may access the services of code residing on a server. The services are accessed through the network and the code itself is typically not transferred. Viruses on the other hand for infection, typically require code mobility and use this mobility to transfer executables and infect

other computers. Stang & Whinston (2002,pg34) explain that “In a Jini environment, before code can move onto another machine, it must satisfy the client’s security policy.” The system administrator can ensure that only trusted machines are permitted to download code.

3.2.8 Competing Technologies

“For Jini to accomplish network nirvana, it must become widely adopted” [Clark 1999,pg 18] Standing in its way of achieving this are a few strongly backed competitors.

Universal Plug and Play (UPnP)

Microsoft’s Universal Plug and Play (UPnP) is open distributed networking architecture for pervasive, peer-to-peer connectivity of intelligent devices. UPnP supports pervasive, invisible networking, zero configuration, and dynamic formation of device communities. UPnP is built using established open Internet standards such as IP, UDP, TCP, HTTP, XML, and SOAP, for various devices to discover and join a network dynamically. No configurations, no device drivers. UPnP internetworking is based on IP standards. Thus every device joining the UPnP community acquires a network address of its own. UPnP is platform and language independent and is geared toward dynamic ad hoc networking of devices, however it does not handle services provided by software implementation or any other entity. UPnP is dependent on SOAP as a protocol for RMI. It does not discuss security aspects within the network. “Currently, it depends on the presence of other security components, such as Firewall, authentication, and authorization servers, to provide security functions.”[Kumaran 2002,pg287] UPnP registers the various devices with DNA. It does not have a separate lookup service as in Jini. Microsoft’s partners in this initiative include Intel and Hewlett-Packard. As such it is a system for ad-hoc networking but does not address the requirements of large scale service deployment. Jini works both at the ad-hoc and enterprise level.

HP JetSend

JetSend, from Hewlett-Packard, is a device-to device communication technology that allows devices to negotiate information exchange intelligently; JetSend enables devices to transfer data without needing to know anything about the other device. JetSend is both complementary and competitive to Jini. JetSend can work with both wired and wireless protocols. “In the corporate network, JetSend technology can become an alternative to fax by allowing you to send documents from one location to another by printing directly onto a high-quality printer.”(Kumaran 2002,pg288) This however assumes the existence of a JetSend-enabled device at both ends of the network. The communication model does not require driver installation or configuration, which makes it easy to plug and play with different devices, JetSend does not depend on a transport protocol; hence it can work with any bi-directional transport protocol, including TCP/IP, IR, RF, Bluetooth, IEEE1394, and others. JetSend can work with various networking technologies and can coexist with such technologies as UPnP, and Chai ApnP. Since Jini is protocol independent, it can embrace JetSend as a communication protocol for communication between devices.

JetSend, is targeting a specific requirement, that of remote printing, using JetSend as the communication technology. While this is an initial starting point, it is limiting for other service/communication requirements.

HP Chai

Hewlett-Packard's Chai is a full-featured Java based appliance platform for developing and running applications on embedded devices. The ChaiAppliance communications is based on Web standards such as HTTP and XML, rather than on Java. However, ChaiAppliance was developed in the Chai Virtual Machine, which is HP's clean-room developed version of Java. This could possibly make ChaiAppliance Plug and Play useful as a bridge between Jini and UPP.

3.2.9 Why Not Jini

“Because Jini was explicitly designed to cover such a wide range of possible deployment scenarios-all the way from servers down to light switches- there are a few hardware situations where Jini is not appropriate”(Edwards 2001,pg 103) If a device is truly isolated or is so fundamentally uninteresting that it would not be used over a network, then Jini is probably not appropriate. However, fewer and fewer devices are now truly isolated. If a device is unlikely to be in the vicinity of a device with a JVM, and needs to be so cheap that it cannot have a JVM embedded on it, then Jini is not a viable solution.” Java has to exist somewhere in the loop for devices to participate in Jini”(Edwards 2001,pg 103). This is the primary limitation that of minimum hardware/performance capabilities.

3.2.10 Conclusion

Jini is a possible network middleware for the new age of ubiquitous computing. It enables communication between applications and devices in a heterogeneous network environment. “Jini portends the movement of computing from general-purpose machines to specialised processors.” (Stang & Whinston 2001,pg38) However Jini technology is still in its infancy. “Because Jini technology represents a fundamental shift in application development, adoption has been slow, but enthusiastic”(Stang & Whinston 2001,pg38) The problem of noise filtering in portable devices has not yet been solved. Also Jini is still viewed by many as primarily for embedded systems. Although the responsibility of controlling an enterprise network is less than an attractive prospect to many human network administrators, it is unnerving for many to let a mere “piece of software” do the work, while is a possibility of Jini.

3.3 Agents

Currently middleware and management systems are separate entities, however, the integration of these two has become the focus of much research. This section will address agents for management while keeping the general middleware requirements identified (section 3.1.1) in mind. Agent technology implicitly addresses many of these middleware characteristics (e.g. mobility, scalability, heterogeneity etc), hence the following analysis of agents will inherently address many of these.

3.3.1 Types of Agents

The definition of an agent is more ambiguous than would be preferred. The lack of a concise agent definition in turn causes a lack of clarity when discussing the different categories of agents. A simple web search using the term agent will return a large amount of references, many using different adjectives to describe the agent type. For example words such as Intelligent, Information, Mobile,

Trading, Personal, Search, Negotiation, knowbot, softbot, etc may all be used to describe the agent type. This is, too say the least, a problem when trying to classify agents into types. Outlining an agent typology can be as contentious an issue as identifying an agent definition and varying opinions are common. Nwana (1996) lists seven types of agent:

- Collaborative
- Mobile
- Information / Internet
- Interface
- Smart
- Reactive
- Hybrid

The paper goes on to provide a critique of its classification detailing the opinions of two independent reviewers. Both suggested that agents defined by what they do (e.g. interface, collaborative), had been confused with agents defined by the technology that enables them (e.g. mobile, reactive).

Alternatively, Magedanz, Rothermel and Krause (1996) classify agents as either single-agent or multi-agent systems. Single-agents are further sub-divided into either local or networked agents while multi-agents are sub-categorised into DAI-based agents and mobile agents. Local agents can access only local resources. Network agents are able to access both local and remote resources. DAI-based agents are focused on intelligent behaviour and the coordination of multiple agents. Mobile agents are obviously focused on the ability of the agent to move around a network.

These two classification models approach agents from a different perspective. The latter model takes an overall system classification approach, i.e. single or multi-agent, and further categorises based on the type of agent approach used to realise that system. The former of the two models describes agents at a more individual level without really taking into account the overall context. Here an agent is described more on what they are or what they do and many of these resemble the properties of agents discussed before.

In reality, due to the lack of a formal definition, most agents still tend to be classified in a haphazard manner. Often this is based on either the agent's purpose or some of its properties. This is similar to Nwana's classification but not restricted to the seven types listed. However, two of the most frequently discussed agent types are classified based on their properties. These are Intelligent and Mobile agents. In the context of this paper 'Intelligent' and 'Mobile' only suggests that these are the properties most focused on and not exclusive of other agent properties. This distinction is made as mobile agents, in some writings, fall under the banner of intelligent agent.

3.3.2 The Management Problem

The management scope of the system has been previously mentioned but an attempt has not yet been made to identify specific management tasks that need to be considered. Two of the most important areas of management that must be looked at are Services and Networks. Smart spaces will offer unlimited numbers of services to their users and as such these services, along with the networked

resources they use, will have to be managed. These two ideas are key to any smart space architecture, and therefore the management of these areas are also central to the M-Zones research.

The distinction has already been made between inter-zone management and intra-zone management. An important issue in this regard will be the understanding that both services and networks may be in separate domains that are under the control of separate operators, and possibly using different smart space architectures. Any smart space management architecture will have to take this into account by allowing management systems in different domains (and possibly of different types) to interoperate and share management information.

Service management must recognise the fact that services in emerging networks will be abundant, accessible from many locations by many concurrent users, created using heterogeneous technology and provided by operators in separate domains using differing architectures. Services may be required to use network resources or other services to fulfil their task. The services themselves will be much more complex incorporating various types of media and incorporating a variety of newly available functionality/technology (e.g. location information). Another consideration is that services are not necessarily static and may be able to migrate across domains on demand. Due to the high volume and mobility of end-users, service security will become a more complex area. These are just some of the service issues that will need to be addressed by a smart-space management architecture

The emergence of smart spaces and new service architectures in turn creates a need for new intelligent and adaptable networks and network management systems. One of the major issues that new management systems will have to face is a huge increase in the number of networked resources. Also due to the highly mobile characteristics of smart spaces, devices (users) will be able to attach and detach from the network in a highly dynamic way. It must also be possible to introduce and remove devices from the network with minimal effort and highly automated configuration. Furthermore these networks will have to be capable of multiple forms of communication (i.e. voice, data and video) which adds a further dimension to the management problem.

In effect, smart space environments call for new network and service management solutions, as current management approaches alone will not suffice. Today the most common type of management is done using Network Management Systems (NMS). These focus primarily on managing the network and do not facilitate the management of services (perhaps what is needed is a 'NSMS – Network and Service Management System'). Even in today's environment, these approaches are considered by many to be inadequate and so will clearly be unable to meet future network and service management demands. Cabri, Leonardi and Zambonelli (1999) identify a number of limitations of the traditional NMS approaches. These approaches are based on a centralised client-server model in which a central administrator client collects management information from other network components. A management protocol such as SNMP or CMIP is typically used to transport the management information. This approach is limited in terms of scalability and reliability and is best suited to small networks. Also these approaches are more suited to networks that deal with devices rather than with services and data provided by network nodes.

3.3.3 Agent Management vs. Agentless Management

Before progressing further it may be of value to look at some simple pros and cons of using agents in management by comparing agent management to agent-less management. Obviously the point of

incorporating agents into our management framework is to provide a solution that will improve the management capabilities of the system. This should provide more fault tolerant systems i.e. more automated, more intelligent fault correction;

Advantages of Agent-based Management

- **Reduced Network Traffic:** With agents, the management logic is delegated to remain locally on individual nodes, either temporarily (mobile agents) or permanently (intelligent agents). In this way the node is managed locally and therefore network traffic can be greatly reduced by potentially removing multiple messages (polling) across the network. Also, depending on the level of intelligence the agent has, it may be possible to filter unwanted information at a node before transmission, reducing network load when the agent returns its acquired data.
- **Robust:** Agent management solutions have the potential to be more robust as management is not solely dependent on a central point. So even if the network is down the agent may still be able to complete its task at its node.
- **Heterogeneous Execution:** Agents are typically written in a scripting language and therefore can run on a variety of heterogeneous systems.
- **Scalable Management:** The traditional centralised client/server approach suffers from scalability problems due to centralised nature of management. Agents lead to de-centralised management and hence more scalable management.

Advantages of Agentless Management:

- **Simpler deployment:** no need for agent or agent execution environment on target machine.
- **Widespread acceptance:** Existing management technology such as SNMP is widely accepted and this should not be underestimated.
- **Security:** With the proliferation of agents around a network there are some serious security concerns that need to be addressed in agents.

3.3.4 Mobile & Intelligent Agents

With regard to management the two most relevant agent types are Mobile and Intelligent agents. Many have been looking at the potential of these approaches in management for some time, with a large number concentrating on network management aspects (Gurer, Lakshninarayan & Sastry 1998) (Zapf, Herrmann & Geihs 1999). Aside from their central properties of mobility and intelligence, both these agent types often require cooperation with other agents. Consequently a method for allowing agents to communicate is essential to both. Many systems in the past have looked to KQML for agent communication. More recently another agent communication language has begun to emerge in the form of FIPA ACL (Agent Communication Language). The two are similar in many ways but KQML is more widely known, as FIPA ACL is only a new standard. Intelligent agents are the more established methodology but the mobile agent approach has built up a great deal of momentum in the last number

of years. In fact in recent years much of the research attention has switched from intelligent agents to mobile agents

Intelligent Agents

As stated before intelligence is based around an agent's ability to reason, learn and adapt to within its environment. Intelligence in these types of agents is usually derived from AI approaches such as Case Based Reasoning, Expert Systems, Neural Networks, etc (Gurer, Khan & Ogier).

When creating a knowledge-based, AI system (such as an expert system) a method of knowledge representation and reasoning has to be chosen. Knowledge representation is how the data or knowledge that is known to the system (and specific to the problem domain) is represented. This can be based on rules, frames, cases etc and is kept in a knowledge or case base. Knowledge reasoning is where the known information is used as a model for solving a new problem. Rule based systems consist of if-then (production) rules and facts. If some fact is matched successfully to the 'if' part of the rule, the 'then' part can subsequently be concluded. Starting with some initial fact, rules can be chained together to form more complex conclusions. With case-based previous experiences are stored in cases. The system then attempts to match the current problem to the similar experiences in these cases. If a similar experience is found the solution or diagnosis used in that case can be applied to the new problem and then this will be stored as a new case.

An expert system is a computer system that simulates the judgement and behaviour of a human (or organization) that has expert knowledge and experience in a particular field. Expert systems are knowledge based which means they already have specific knowledge about the application area and use this knowledge to solve problems. Expert system shells are expert system applications without the domain specific knowledge (rules) and can be purchased 'of the shelf' for quicker development.

Neural networks attempt to model the human brain and as such process information in a similar way. They consist of a large number of highly interconnected processing elements (neurons) working in parallel to solve a problem. Conventional computing uses known algorithms to solve problems (i.e. set of pre-programmed instructions) whereas neural networks learn by example. Neural networks do not know the solution, instead they learn by example and can adapt and modify their behaviour based on their environment (or more specifically their inputs).

Intelligent agents are often discussed as a potential technology for use in Network Management. This approach is based around the principle of deploying agents at nodes around the network to perform management tasks locally. Since they are intelligent as opposed to mobile (static agent) they remain permanently at their node. The focus here is on inserting intelligence into the node to allow it do a certain level of self-management. This delegation of management responsibility leads to a less centralised management system and along with reducing network traffic can increase automation and deliver a more robust and scalable system.

Intelligence, however, has many applications as it brings decision-making and adaptive capabilities to a system. This allows systems to be aware of changes in its environment and potentially re-configure/adapt themselves accordingly.

Mobile Agents

The concept behind mobile agents is that, as the name suggests, the agent is not bound to one location and can move to other network nodes. This can come under a number of banners including Remote Programming or Mobile Code and despite the fact that this has only received serious attention in recent times the concept behind it is rather old. As early as the 70s “remote (batch) job processing” was being done and in the 80s “function shipping” or “remote evaluation”.

When talking about agent mobility there are two main forms to discuss. These are Remote Execution and Migration. Remote execution is where an agent, including its code and data, is transmitted to a node/server where it is executed and its task ends here. With migration on the other hand, the agent’s execution state is also transmitted with the agent allowing it to suspend and resume its execution. Once suspended an agent can then be transmitted to another node and resume its execution there.

It is a generally accepted principle that mobile agents are developed in a machine independent language or interpreted language. The reason for this is that the execution environment an agent may be currently running on can vary greatly and developing the same agent for a number of different platforms is a huge overhead. Most of the mobile agent platforms available today, such as Grasshopper (www.grasshopper.de), are fully implemented in Java.

Mobile Agent Technology (MAT) was, in its early days, considered its own separate “religion” for distributed processing. “Remote programming is considered as an alternative to the traditional “Client/Server programming” based on the Remote Procedure Call (RPC) paradigm.” (Magndanz, Rothermel & Krause 1996). Even though both essentially had the same aim, MAT was believed by the majority to be mostly incompatible with Distributed Object Technology (DOT) such as the Object Management Groups (OMG) Common Object Request Broker Architecture (CORBA). CORBA, and DOT in general, have gained widespread acceptance within the telecommunications and networking environment due to its ability to support interoperability between objects in distributed heterogeneous domains. As CORBA allows location transparent access to objects, CORBA objects can be written in any programming language and running on any platform in a separate domain and can still interoperate. To facilitate this CORBA uses well defined interfaces.

However, the opinion expressed in the above quote has undergone a major shift in recent times. (Breugst, Hagen & Magendanz 1998) A common understanding has now been formed that MAT should enhance DOT rather than rival it. The reason for this is that both technologies offer different advantages in distributing software. CORBA agents are static and cannot move from the domain in which they were started. MAT however is designed to move so by combining both MAT and DOT the advantages of both can be extracted to realise a highly flexible distributed object environment. One major advantages of this is that (as was mentioned as a problem of new service management) service objects could now migrate across domains on demand or as required by the user. Another advantage is that non-agent (legacy) systems can be integrated into this environment. If the agent system uses CORBA then any legacy CORBA system or application can be incorporated into the new system.

3.3.5 Existing Management Solutions

It is a worthwhile exercise to look at some of the existing management solutions and architectures. Much work has been done in the field of telecommunications management, mostly through standards bodies such as OSI, ITU and TINA-C. One particular standard that is of interest is the Telecommunications Information Networking Architecture developed by the TINA-Consortium.

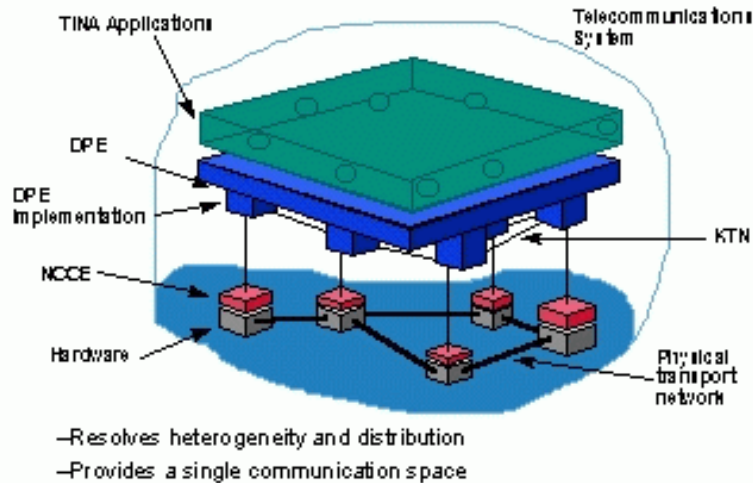


Figure 6 – TINA layered architecture

This architecture will not be discussed in much detail at this time but further investigation in this area is suggested. TINA contains four major architectures that define the concepts and principles for creating a TINA compliant system. These are the Computing, Service, Network and Management architectures and contain many concepts that may be of relevance to both agent-based management and to the M-Zones management objectives in general. The architecture can be split into different layers as can be seen in the diagram above. At the bottom layer is the physical hardware of the network. Upon this is the Native Computing and Communications Environment (NCCE) which is made up of heterogeneous Operating Systems, communications and support software. Above this is the Distributed Processing Environment (DPE) which is a distributed object environment very similar to systems such as CORBA. The top layer is the application layer and this is made up of application objects.

As can be seen from the discussions before regarding MAT and CORBA, any agent environment that combines the two has the potential to deliver great rewards. If the application /service objects in TINA's application layer are looked upon as Agents then many of TINA's recognised concepts can be applied (in particular from the Management architecture) to a Mobile Agent system.

Another management concept worth considering is the ever-present FCAPS (Fault, Configuration, Accounting, Performance, Security) acronym. These five management functions will be important when considering the management tasks involved in smart spaces.

3.3.6 Agent Standards

FIPA – Foundation for Intelligent Physical Agents

The FIPA organisation was started in 1996 in Switzerland and currently has almost 60 member organisations. The group aims to develop standards that promote the interoperation of heterogeneous interacting agents and agent-based systems. Its first standard, FIPA 97 was published in October 1997 and contains specifications for agent management, agent communication language and agent software integration. FIPA has so far primarily worried about high-level agent-to-agent communication and has not said much about mobility.

Since then, FIPA97 has been further extended with a FIPA 98 and FIPA 2000 release.

MASIF – Mobile Agent System Interoperability Facility

MASIF (<http://www.fokus.gmd.de/research/cc/ecco/masif>) is a joint submission of GMD FOKUS, International Business Machines Corporation, Crystaliz, General Magic, and the Open Group which was adopted by the OMG in 1998. MASIF, like FIPA, has its long-term focus on the interoperability of heterogeneous agent systems. The specification is a collection of definitions and interfaces that provides an interoperable interface for mobile agent systems. It contains two main interfaces: the MAFAgentSystem, for agent transfer and management, and the MAFFinder, for agent naming and locating (Milojicic et al 1998). MASIF has primarily considered mobility and, to a lesser extent, interoperability among heterogeneous agent systems. It does not say much about agent communication as mobile agents main consideration is mobility.

3.4 Parlay/OSA

In order to create, manage and deploy services for OSA/Parlay Networks, there are several issues which need to be addressed. The first issue I will explore is Application servers, as most services need a server to run upon (except for peer-to-peer services).

Service Platforms, Reusable components and Service Creation Environments are all methods being developed to simplify the creation and management of services. Service platforms extend upon reusable components and Service creation Environments extends upon Service platforms.

The OSA/Parlay interfaces are Middleware in the form described in paragraph 3.1.3, Heterogeneity, allowing various networks running assorted operating systems and communications software to communicate with one another.

Both the OSA and Parlay initiatives to define a common interface have both been industry led, with little academic involvement. There is currently no reference implementation or ideal model of an OS/Parlay gateway. The state of the Art is being led by companies releasing their own commercial implementations.

3.4.1 Application Servers

All Parlay applications have to be deployed on an application server of some sort to be able to access the Parlay gateway. This application server may also be inside the telecoms network, which would mean that applications could not be updated, maintained or monitored easily by the third party who created them.

Another option is to have trusted application servers under the control of the third party Service developer/provider, which would have access to the Parlay gateway. This would allow for ease of updating and maintenance of the applications but would require a lot of resources and is not feasible for smaller companies/individuals.

In order to keep maximum amount of control over applications that use its network some telecoms will be using Application Development Environments and Service Creation Environments (SCE) to create another layer of abstraction between the developer and the Parlay Gateway. By using these developer environments, network operators can limit the access of third party applications to network features as much as they want while also speeding up application development time.

The question of whether the telecom network operator or the developer should host the application server is mostly down to security as highlighted by Schneier (2001). How open does the telecom want their network to be? From past and current examples (i.e. Broadband) most telecoms keep a tight grip on their infrastructure and opening up their network to third parties is not something they like to do. From a commercial side the telecoms are relying on third party applications to increase their ARPU (Average Revenue Per User), so it is not clear which route the telecoms will take.

3.4.2 Reusable Components

The concept of building services for Parlay/OSA networks using reusable building blocks has been investigated by Prof Hanrahan and his Team at CETAS. In their paper, Nana(2002) they divide the code that makes up a service into service dependent and service independent. Their goal is to define generic reusable components that enable the easy creation of services for TINA, Parlay and other Next generation networks.

These reusable Components would include controlling sessions and streams. Session control is generic as most services require connecting to something and this connection has to be set-up, monitored and then destroyed. Streams are required for transferring data of any type over these sessions. The data being streamed could be anything, text emails, news stories, or video clips. Service management, such as billing mechanisms, fault management and load management are handled by Service Independent Components, but these features are dealt with more comprehensively in Service platforms

3.4.3 Service Platforms

An Alternative to using a number of generic components when creating a new service is to build the service on top of an existing platform that provides all commonly required features. This approach is explored in "A Service Platform for Internet-Telecom Services using SIP" - Bessler (200). They propose using thin clients in the form of HTTP and WAP at the user end, while the service provider would control the service platform that the clients connect to. The services and content for the platform would be created by third party content providers and by the service provider themselves.

The service platform itself handles subscriber management and is the sole point of authentication, Bessler (2000). This allows for a User to have only one contract and facilitates easier payment for

services. One problem with this approach is the requirement for a secure certification system or other authentication system.

The Lucent MiLife™ Solution

The MiLife™ Intelligent Services Gateway (ISG) Lucent (2002) is a Parlay Gateway, which also uses optional convenience classes unique to MiLife. To help developers create applications to use the MiLife ISG, Lucent have created an ISG Simulator as part of a Software Development Kit (SDK). The SDK allows the developer to create their application and test it on the simulator with script files to simulate users on the network. Sample programs and script files are included in the SDK and these show some of the features of the ISG in action. Applications can be written in any language that supports CORBA, but Java is recommended and used in all samples.

The ISG SDK is designed to help developers developing for the MiLife Parlay gateway, but due to the open nature of the Parlay/OSA any program developed for one gateway should work on them all. This is why it is important to avoid shortcuts such as the convenience classes in the ISG SDK that could tie a developer down to a particular implementation of a Parlay/OSA gateway. The MiLife SDK is available from Lucent now to registered developers, along with a list of compatible networks.

The NOMAD Project

The NOMAD project is a new project undertaken by WIT, Dundalk IT and Dun Laoghaire Institute of Art and Design. The aim of the project is to develop new framework models and systems to aid in the creation and deployment of services for 3G and wireless networks. The team working on the project are focusing on location-based services and are also investigating the Parlay/OSA interfaces. This project is in its early stages but will have a large overlap with the M-zones project in the area of Parlay/OSA or other 3G type services.

Nomad's objectives are to examine the state of the art in 3G, WiFi, SIP, and Parlay. They will then identify some service(s) that will utilise the technology across the various bearer networks, i.e. 3G, IP etc. They then propose that they will develop a service creation framework, identifying the building blocks required to deploy services, and specifying methodologies to do so.

Nomad will combine these technologies and resources to develop a set of services utilising 3G and WiFi technologies. It will probably utilise Parlay and SIP. It will endeavour to develop a framework for the creation of additional services, and also hopes to deploy a toolkit which will facilitate this.

3.4.4 Service Creation Environments

The concept of using a Service Creation Environment (SCE) to create services encompasses many of the advantages of a service platform. Using an SCE means building your service within rather than on top of an existing system. The SCE helps the service builder to take a specification of a service, to model it and then to implement it. In this paper Jagot (2002), the CETAS team proposes a model for a service creation environment based on a framework they have named SEMTA (Software Engineering based on Modelling for Telecommunications Architectures).

Hughes Software Systems SCE (Software Creation Environment)

Hughes Software Systems (HSS) is a leading member of the JAIN initiative that is trying to create standard interfaces for service creation and management. HSS is creating its SCE to comply with the Parlay/OSA standards so that applications created using it will be compatible with any telecommunication network that has a Parlay gateway. The SCE is designed to be useful for creating applications for different types of network including IP, PSTN and wireless networks.

The SCE uses a GUI to speed up the development of applications and services and is designed to allow reuse of service logic through the use of Service Independent Building-Blocks (SIB). A SIB is similar to a JavaBean in that it is a self-contained re-useable chunk of code. Examples of SIB's are 'Billing', 'Call Forwarding' etc

The advantages of using their service creation environment are given by HSS as "Rapid development and maintenance of revenue generating applications and services

Reduction in initial and operating costs for service creation and maintenance
 Faster time to market for delivery of new applications and services
 Quicker turnaround for customer specific application customization needs"
 HSS (2002)

The SCE Graphic Environment is a Java Applet running inside of a browser window which would allow for easy deployment of applications depending on the application server to be used.

The HSS SCE has an "Integrated testing environment " which would probably include a network simulator of some kind, allowing a developer to simulate customers using the application over the 3G/wireless/IP network.

TrueConverge Service Creation Environment

The TrueConverge Service Creation Environment (SCE) as described in TrueConverge (2002), is another example of a complete graphical user environment for designing creating, testing, packaging and deploying services. There is also a TrueConverge application server to run the completed applications on.

There is little to choose between the TrueConverge and the HSS SCE as neither company has yet released their products. A positive note for the future is that both companies are building the product around the JAIN specification, which should allow applications to be run on any JAIN Compatible network. "Write once, run anywhere." Sun Microsystems (2002)

Implementation Independent specification of Services

The purpose of the Parlay API's is to allow services access to network core features. However because of the different ways the networks are implemented it is necessary to model services in an implementation independent way, to allow their deployment across many different networks Sties (2002).

In an ideal world all services would be compatible with all Application servers and all Parlay/OSA Gateways, but in reality, services will have to run on various platforms. In order to minimize the work involved in creating the services for any network “A Generic and Implementation Independent Service Description Model” Sties(2001) has been proposed.

3.4.6 Service Deployment

Service deployment is, at its most basic, getting a working application from the developer’s computer to the application server where it will be run. This application server could be part of the Telecoms private network, or owned by the developer. Separate “hosting” companies may even own the application servers.

In the HSS Service Creation Environment Model, the SCE is responsible for passing on the finished application to the application server. This limits third party access to the servers and may be necessary if application server is inside the Telecoms internal network. This system would need lots of features to allow developers update applications already deployed, without having to start from the beginning. This system looks to be the most possible option for the security conscious network operators.

Using the MiLife SDK or simply writing an application from scratch means that a developer must then copy the application to the application server. Updating the application would mean removing the application and replacing it with an updated version.

The STARLITE Project

The STARLITE project is part of the European IST program and has been working for several years in the service creation and management area. At a recent conference Triglia (2002) summarized what they had achieved.

“Fast provisioning of new services to end-users has rapidly become the key issue in the modern telecommunications environment, often being the most important element determining the success of operators and manufacturers. The convergence of IT (Information Technology) and TLC (Telecommunications) determines a framework for new, potentially appealing services, like, for instance, Intelligent Network (IN) capabilities offered to Internet (IP) users.

The Project has implemented an integrated IT/TLC environment based on a distributed IN, exploiting Parlay API specifications, as well as using Distributed Object and Mobile Agent Technologies (DOT/MAT). Above that integrated

Environment, cross-network services have been designed, created and deployed. These services are derived from the integration between distinct network capabilities (PSTN and Internet) and are uniformly controlled from creation to provisioning. DOT and MAT are used as a unified framework addressing the service lifecycle (creation, deployment, provision and operation), allowing the design and implementation of compact and extendable service objects, used in order to program active network nodes according to time-varying needs. Parlay APIs are used in the Project, as intended by the Parlay Group, i.e. to allow service developers to create services in an open network-programming framework.”

4 Future Directions

4.1 *Middleware*

The aim of the M-Zones project is to develop a management system for multiple ubiquitous environments referred to as smart spaces. Present ubiquitous computing environments such as Active Campus (2002), Oxygen (MIT, 2003) and EasyLiving (Microsoft, 2003) have each been independently developed with no consideration of interoperability. These ubiquitous environments were not designed to be compatible with each other so a management system developed these would be specialized and not generic. It would be of great benefit if these ubiquitous environments were developed from a common middleware infrastructure leaving the heterogeneous nature of these ubiquitous environments transparent to the management system.

At present, research in the area of middleware infrastructure for services in ubiquitous computing environments is in its infancy. There are, however, some encouraging developments in emerging technologies such as Sun Microsystems Jini (Sun, 2003) platform which is a distributed computing framework that allows users to access any resources available on the network without the need of any configuration. Other following technologies have also been proposed for developing application to support context aware middleware: XML , SIP and SOAP . These technologies are being used in the development of Web Services, which is a way of describing and using services from anywhere in the internet, in a common way. Middleware providers such as CORBA and .NET are adding support for these technologies, which is pushing present day middleware closer and closer to complete interoperability and ubiquity. As part of the M-Zones project it would be of great benefit if the three smart spaces that are being developed at Trinity College Dublin, Cork Institute of Technology, and at Waterford Institute of Technology, were based on a common middleware infrastructure. This would make developing a management system for managing multiple smart spaces (i.e. M-Zones) a lot easier. Possible technologies that would be of benefit to M-Zones would include the use of Sun's Jini platform as a middleware for device communication within a smart space, and Web Services as a method of smart space to smart space communication.

4.2 *Parlay/OSA*

The issue of how to create, manage and deploy services for 3G or other wireless networks could become a problem once the network operators start to open their networks to third party services and applications. The central issue is security, and how much control the network operator will maintain over these services. It is still too early to tell, but it is likely that the network operators (O2, Vodafone etc) will incorporate complete SCE (Service Creation Environments) into their network. Developers will have to use this environment to develop applications for their network and will not have direct access to the Parlay interface.

The implication for the M-Zones project of exclusive integrated Service Creation Environments is that it is more difficult to exchange data with other telecom networks than it should be simply using the Parlay/OSA API's. The M-zones project should allow for a lot of movement in the Parlay/OSA API's if they are to be used, as the future of Parlay is by no means certain.

The Nomad Project, which has links to the M-zones programme, is developing a Service platform which plans to link to a Parlay/OSA Gateway. Any further research into developing services that will integrate with Telecommunications Networks, through a Parlay gateway or other means, will need to be done in partnership with the Network Operators.

4.3 Agents

The domain of applicability for agents is extensive. Much of the analysis of agents in this paper has leaned towards network management, largely due to the fact that NM is a long-established and widely addressed area. Within this area a number of relevant future directions exist. Using agents as an enabling technology for active networks is one such area. Active networks not only allow the network nodes to perform computations on the data but also allow their users to inject customised programs into the nodes of the network, that may modify, store or redirect the user data flowing through the network. Mobile and intelligent agents could easily assume the role of these active network programs (Karnouskos 2002), traversing the network and performing customised processing of data received at network nodes. Another, more definite, direction is incorporating agent properties such as mobility, intelligence, etc into existing management technologies like SNMP. This would provide decentralized SNMP management while maintaining the existing and widely used management protocol.

Moving away from Network Management, Service management presents, possibly, the most interesting area of application for agents in regard to M-Zones. Again the mobility and intelligent facets of agents can provide many benefits to smart spaces such as more adaptive and nomadic services. Services that can customize themselves based on a variety of variables, that can be intelligently composed from a number of other services and that might move/migrate on request (possibly with a user).

As has been stated the range of applications for agents in smart spaces is extensive. How agents will be used, if even at all, is very much an open question, but the fact that agents hold potential for smart spaces is undeniable.

5 Conclusion

This paper began by introducing the main concepts in best practice middleware technologies. Following this, a definition of requirements for Smart Spaces with regard to middleware technology were highlighted. These requirements were kept generic (e.g. scalability, re-configuration, quality of service) in order to allow the identification of criteria for the evaluation of middleware technology. A detailed discussion of three technologies namely OSA/Parley, Jini and Agents addressed how these could be used within smart spaces or managed zones of smart spaces (M-Zones). These three technologies address different aspects of smart spaces, satisfying different middleware requirements. What is evident is that no single technology reviewed completely satisfies the requirements outlined in the analysis section. This paper concludes that present best practice middleware will have to expand its horizons to cater for the new requirements outlined in the analysis section if they want to survive in the emerging smart space environments (Kurt Geihs, 2001).

In summary it is not clear at this point exactly what Smart Space middleware will consist of. This remains very much an open question. What is likely, however, is that various smart space architectures will emerge independently of each other which greatly increases the need for middleware to provide interoperability between heterogeneous systems.

6 References

Aberdeen Group 2002, Mobile Middleware Market to Triple by 2006, Retrieved November 27, 2002 from
<http://www.pdastreet.com/articles/2002/7/2002-7-30-Mobile-Middleware-Market.html>

Active Campus, Visited November 27, 2002 at
http://www.jacobsschool.ucsd.edu/newsletter/winter2002/active_campus.html

Bakken, D.E., "Middleware", Encyclopedia of Distributed Computing, Kluwer Academic Press, 2002.

Bacon, J. et al., "Generic Support for Distributed Applications", Computer, March 2000, pp. 68-76

Becker, C. & Geihs, K. "Generic QoS Support for Corba," Proc. Int'l Symp. Computers and Communication (ISCC 2000), IEEE CS Press, Los Alamitos, Calif., 2000, pp. 60-65.

Bernstein, P.A. "Transaction Processing Monitors", Communications of ACM, November 1990, 33(11): 75 - 86.

Bessler, A.V., 2000, A Service Platform for Internet-Telecom Services using SIPS.

Breugst, M., Hagen, L. & Magedanz, T., 1998, 'Impacts of Mobile Agent Technology on Mobile Communications System Evolution', IEEE Personal Communications, vol. 5, no. 4, pp. 56-69.

Bush, V. 1945, 'As We May Think', The Atlantic Monthly, July.

Cabri, G., Leonardi, L., Zambonelli, F. 1999, 'Network Management based on Mobile Agents using Programmable Tuple Spaces', Proceedings of the 4th International Conference and Exhibition on The Practical Application of Intelligent Agents and Multi-Agents, April.

Campbell, A., G. Coulson, and M. Kounavis. "Managing Complexity: Middleware Explained." IT Professional, IEEE Computer Society, 1:5, September/October 1999, 22-28.

Capra, L., W. Emmerich, and C. Mascolo: 2001, 'Middleware for Mobile Computing: Awareness vs. Transparency (position paper)'. In: Int. 8th Workshop on Hot Topics in Operating Systems.

Emmerich, W.: 2000, Engineering Distributed Objects. John Wiley & Sons.

Erzberger, M., Altherr, M., "Every DAD needs a MOM" Message-Oriented Middleware. September 1999,
<http://www.softwired-inc.com/pdf/technology/momdad-final.pdf>

Foundation for Intelligent Physical Agents, Available: www.fipa.org

Turing, A. M. 1950, 'Computing machines and intelligence', *Mind*.

Franklin, S. & Graesser, A. 1996, 'Is it an Agent, or just a Program: A Taxonomy for Autonomous Agents', *Proceedings of the Third International Workshop on Agent Theories, Architectures and Languages*, Springer-Verlag, 1996

Gelernter D., *Generative Communication in Linda*. *ACM Computing Surveys*, 7(1):80 – 112, Jan 1985

Grasshopper, Available: www.grasshopper.com

Gurer, D., Lakshminarayan, V., Sastry, A. 1998, 'An Intelligent Agent Based Architecture for the Management of Heterogeneous Networks', *Proc. of the 9th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*.

Gurer, DW, Khan I, et al., 'An Artificial Intelligence Approach to Network Fault Management.', http://www.sce.carleton.ca/netmanage/docs/An_AI_Approach.pdf

HSS, 2002, Hughes Software Systems, *Convergent network Solutions-Service Creation Environment* Available at http://www.hssworld.com/hss_mindsystem/archives/sce/docs/sce.pdf

ISO/IEC 13818-2 (MPEG-2 video), November 1994

ITU-T Rec. H.263, "Video Codec for Low Bit rate Communication". 1996.

Jagot, A.R., 2002, *A Meta- Service Creation Environment for the Next-Generation Network (NGN)* Centre for Telecommunications Access and Services, University of the Witwatersrand, Johannesburg

Kampis, G., 'The Natural History of Agents', <http://hps.elte.hu/~gk/Mirror/Agents.pdf>

Karnouskos, S. 2002, 'Realization of a secure active and programmable network infrastructure via mobile agent technology', *Computer Communications Journal*, Volume 25, Issue 16, pp. 1465-1476, October 2002

Kenneth Black et al, "Wireless Access and Terminal Mobility in CORBA", *OMG Meeting in Dublin, Ireland*, November 13-14, 2001

Kurt Geihs, "Middleware Challenges Ahead", *IEEE computer*, 2001

Lauff, M., Gellersen, H.W., "Adaptation in a Ubiquitous Computing Management Architecture". *Proceedings of the ACM Symposium on Applied Computing*. pp. 566--567, 2000

Li, B. et al, "A Control-based Middleware Framework for Quality of Service Adaptations", IEEE Journal of Selected Areas in Communications, Special Issue on Service Enabling Platforms, vol. 17 no. 9, pp. 16632-1650, Sept. 1999

Lucent, 2002, MiLife™ Intelligent Services Gateway - the operator need for service Mediation, Available at http://www.lucent.com/livlink/0900940380020c61_Brochure_datasheet.pdf

Maes, P. 1994 'Agents that Reduce Work and Information Overload', Communications of the ACM, vol. 37, no. 7 (July), pp. 31-40.

Maffeis S., "The Object Group Design Pattern," Proc. Conf. Object-Oriented Technologies and Systems (COOTS 96), Usenix, Berkeley, Calif., 1996, pp. 294-303.

Maffioletti S., "Requirements for an Ubiquitous Computing Infrastructure" <http://diuf.unifr.ch/~maffiole/Documents/Papers/ParadigmForUbiComp.pdf>, 2001

Magedanz, T., Rothermel, K. & Krause, S. 1996, 'Intelligent Agents: An Emerging Technology for Next Generation Telecommunications?', Proceedings of INFOCOM'96, San Francisco, USA, March 24-28.

Mahmoud Naghshineh, Marc Willebeek-LeMair, "End-to-End QoS Provisioning Multimedia Wireless/Mobile Networks Using an Adaptive Framework", IEEE Communications Magazine, no. 11, November 1997 pp. 72-81

Manuel Román et al, "Gaia: An OO Middleware Infrastructure for ubiquitous Computing Environments", 5th ECOOP Workshop on Object-Oriented and Operating Systems (ECOOP-OOSWS'2002), Malaga, Spain, June 11th, 2002

Mascolo, C., Capra, L., Zachariadis, S. and Emmerich, W., "XMIDDLE: A Data-Sharing Middleware for Mobile Computing". In Personal and Wireless Communications Journal, Kluwer. April 2002.

McCulloch, W. & Pitts, W. 1943, 'A Logical Calculus of the Ideas Immanent in Nervous Activity', Bulletin of Mathematical Biophysics, vol. 5, pp.115-133.

Microsoft's Easy Living project, Visited November 27, 2002 at <http://research.microsoft.com/easyliving/>

Milojicic, D. et al. 1998, 'MASIF---The OMG Mobile Agent System Interoperability Facility', Personal Technologies, vol. 2, pp117—129, September.

MIT Oxygen project, Visited November 27, 2002 at <http://oxygen.lcs.mit.edu/>

Mobile Agent System Interoperability Facility,
Available: <http://www.fokus.gmd.de/research/cc/ecco/masif/>

Murphy, A. et al, "LIME: A Middleware for Physical and Logical Mobility", The 21st International Conference on Distributed Computing Systems, April 16 - 19, 2001 Mesa, AZ

Nahrstedt, K., et al. "QoS-Aware Middleware for Ubiquitous and Heterogeneous Environments". IEEE Communications Magazine. 2001.

Nahrstedt, K., et al. "Distributed QoS Compilation and Runtime Instantiation" In Proceedings of the Eight IEEE/IFIP International Workshop on Quality of Service, pp. 198-207, June 2000.

Nana, P., 2002, Re-usable TINA service components incorporating the Parlay API for the Next Generation Network (NGN) Centre for Telecommunications Access and Services (CeTAS) University of the Witwatersrand, Johannesburg

Norman, D., The invisible Computer. The MIT Press, Cambridge, Massachusetts 0214, 1999

Nwana H. 1996, 'Software agents: An Overview', Knowledge and Engineering Review, vol.11, no. 3, pp. 205—244.

Petersen, K., M. J. Spreitzer, D. B. Terry, M. M. Theimer, and A. J. Demers: 1997, 'Flexible Update Propagation for Weakly Consistent Replication'. In: Proceedings of the 16th ACM Symposium on Operating Systems Principles (SOSP-16). pp. 288-301, ACM Press

Picco, G., Murphy, A. and Roman, G.-C.. LIME: Linda Meets Mobility. In D. Garlan, editor, Proc of the 21st Int. Conf. On Software Engineering, pages 368-377, May 1999

Raatikainen K. 2001. Functionality Needed in Middleware for Future Mobile Computing Platforms. Nov. Retrieved November 18, 2001, from <http://www.cs.arizona.edu/mmc/02Raatikainen.pdf>.

Roman,G-C. Christine Julien, "Using EgoSpaces for Scalable, Proactive Coordination in Ad Hoc Networks", Dept of Computer Science, Washington University in St. Louis, 2002

Roth, J., "A Communication Middleware for Mobile and Ad-hoc Scenarios", International Conference on Internet Computing (IC'02), 24.-27. June 2002, Las Vegas (USA), Vol. I, 77-84

Schmidt, D.C., Levine, D.L. and Mungee, S. "The Design of the TAO Real-Time Object Request Broker," Computer Comm. J., vol. 21, no. 4, 1998, pp. 294 324.

Schmidt, D., Levine D. and Cleeland, C. "Architectures and Patterns for High performance, Real-time CORBA Object Request Brokers," In Advances in Computers, Marvin Zelkowitz, Ed., Academic Press, 1999.

Schneier, B., 2001, Bruce Schneier Phone Hacking: The Next Generation, July 15, 2001 Available at <http://www.counterpane.com/crypto-gram-0107.html>

Sun Microsystems Jini project,

<http://www.sun.com/software/jini/overview/index.html> .
Visited March 5, 2003

Sties P., Kellerer, W., 2001, A Generic and Implementation Independent Service Description Model, Institute of Communication Networks Munich University of Technology (TUM)

Sties P., 2002, A service creation model for network spanning services
Peter Sties Institute of Communication Networks
Munich University of Technology (TUM)

Sun Microsystems, 2002, Sun Microsystems :The JAIN API's
Available at <http://java.sun.com/products/jain/overview.html>

Technical Standard a, "Distributed Transaction Processing: The XA Specification" C193 UK
ISBN 1-872630-24-3, February 1992

Technical Standard b, "Distributed Transaction Processing: The TX (Transaction
Demarcation) Specification", C504 UK ISBN 1-85912-094-6 April 1995

Triglia S., 2002, "STARLITE, a success story for Parlay", Parlay Meeting 9-11 July 2002,
Montreal Canada.

TrueConverge, 2002. TrueConverge Service Creation Environment
Available at
http://www.truetel.com/download/TrueConverge_Service_Creation_Environment.pdf

Vanegas R., et al., "QuO's Runtime Support for Quality of Service in Distributed
Objects," Proc. IFIP Int'l Conf. Distributed System Platforms and Open
Distributed Processing, Springer-Verlag, New York, 1998, pp. 207-223. IEEE
Personal Comm., vol. 4, no. 5, 1997, pp. 58-64.

Wolfgang Emmerich, "Software Engineering and Middleware: A Roadmap" In: A.
Finkelstein (ed): The Future of Software Engineering. pp. 117-129. ACM
Press. 2000.

Yau, S.S. and Karim F., "Reconfigurable Context-Sensitive Middleware for Pervasive
Computing", IEEE Pervasive computing, 2002

Zapf, M., Herrmann, K. & Geihs, K. 1999, 'Decentralised SNMP management with mobile
agents', sixth IFIP/IEEE International Symposium on Integrated Network Management.

Zinky, J. , Bakken, D. and Schantz, R., "Architecture Support for Quality of Service for
CORBA Objects," Theory and Practice of Object Systems, vol. 3, no. 1, Jan. 1997.