An Advanced Appliance Interaction Architecture

by

Darragh O' Sullivan, BSc.

Thesis

Presented to the

University of Dublin, Trinity College

in partial fulfillment

of the requirements

for the Degree of

Master of Science in Computer Science

University of Dublin, Trinity College

September 2005

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

Darragh O' Sullivan

September 12, 2005

Permission to Lend and/or Copy

Acknowledgments

A sincere thanks to my supervisor Alexis Donnelly for all his support and guidance throughout the dissertation. Also, I would like to thank my family for their support. Thanks to Amanda for her support throughout the year and her very helpful proof-reading towards the end of the dissertation. Finally, I would like to thank my classmates for their shared knowledge and making the year enjoyable.

Darragh O' Sullivan

University of Dublin, Trinity College
September 2005

An Advanced Appliance Interaction Architecture

Darragh O' Sullivan
University of Dublin, Trinity College, 2005

Supervisor: Alexis Donnelly

In the future, the ability to monitor and control home appliances over the Internet could become one of the conveniences that we wonder how we ever managed without. Also, the ability to interact with networked appliances within the home, using different user devices and modalities, could prove to be hugely beneficial to certain users, especially the disabled and elderly. Using a single device to discover and communicate multi-modally with many appliances based on different technologies could bring untold advantages to such people. There are a number of issues however, preventing these scenarios from being realised.

The dissertation firstly presents these challenges along with the requirements that need to be met. There are three main challenges that need to be addressed. The first challenge is communication in a heterogeneous environment where there are a variety of communication protocols being used by the networked appliances (Jini, HAVi, UPnP and many more). Secondly, it is also expected that there will be a broad range of user devices in operation in future home and vehicle networks. Users will come to expect visual, speech and possibly even gesture interaction all at once. Thus, multi-modal and multi-device interaction is required. This leads to research in generic user interface languages. Lastly, for the true value of networked appliances to be realised, wide-area access to appliances over the Internet should be provided and currently there is very poor support for this.

This dissertation reviews the state-of-the-art in multi-modal interaction, heterogeneous communication and wide-area access to networked appliances. Based on an analysis of the requirements and state-of-the-art, an appliance interaction architecture is proposed. The service-oriented OSGi (Open Services Gateway Initiative) framework is proposed as the basis of the architecture. OSGi is a standardised, protocol agnostic, lightweight services gateway. It has widespread industry support, and it provides a dynamic services platform allowing components such as appliance drivers and user interface transcoders to be deployed at run-time. As an evaluation of the feasibility of the proposed architecture, a proof-of-concept implementation is deployed and evaluated on the Oscar OSGi framework. It is established that an appliance interaction architecture implemented on the OSGi framework is a flexible environment for deploying services in a home or vehicle network.

Contents

List of Figures

Chapter 1

Introduction

1.1 Communicating with Networked Appliances

The advancement of technologies to enable communication with networked appliances is providing new opportunities, which enables the user to control and monitor networked appliances such as entertainment systems, air conditioners, lights etc., within the home or vehicle, or remotely over the Internet. Despite these opportunities, there are a number of major issues that are preventing the widespread deployment of networked appliances.

The problem with most current solutions is that they are restricted to a specific technology or service. Basic services controlling only a single appliance do not offer much value compared to traditional, non-networked appliances and may not justify the investment in the new infrastructure. For the true value of networked appliances to be realised, it is necessary to have a solution which allows communication with multiple appliances. This should be done regardless of their underlying discovery and communication technologies, be it UPnP (Universal Plug-and-Play), Jini, LonWorks, HAVi or any of the myriad of communication and discovery technologies in operation in the home network or vehicle network. In general, a practical home or vehicle network should consist of a gateway server seamlessly integrating communication with multiple appliances. Unless this integration is completely seamless to the user, then it is very unlikely that the use of networked appliances will ever take off.

Another issue that is preventing the widespread use of networked appliances is that, despite the wide range of solutions and standards that exist for interconnecting appliances inside the home, there is currently very little support for wide-area communication with networked appliances over the Internet. In order for wide-area protocols to communicate with networked appliances they need to meet the requirements defined for communicating with networked appliances. Solutions should provide support for invoking control actions, such as turning on the lights, querying appliances, and receiving notifications, for example, if the power usage increases above a certain threshold value [2].

Yet another issue is preventing widespread deployment of networked appliances. It is quite common for today's users to juggle between laptops, Palm Pilots and mobile phones. These user devices feature a broad range of user-interface technologies. For example, Java Swing might be used for a desktop PC, C for a Palm Pilot, WML (Wireless Mark-up Language) for a mobile phone, and VoiceXML for a voice application. This richness creates problems for user interface designers. It is necessary for user-interface designers to build and maintain numerous different source code bases. This requires effort and also means that the user interface designer needs to be proficient in all the different user-interface technologies. Obviously this is not possible, so, a flexible way to provide user-interfaces for user devices and their different modalities is needed to realise the full benefit of communication with networked appliances. This requirement leads to research in generic user-interface languages. It is also becoming increasingly more important to develop user-interfaces accessible and usable by a diverse user population with different requirements, skill levels, preferences and abilities. It should be possible to use these interfaces in a variety of contexts and through a variety of different user interface technologies. User-interface solutions should accommodate all potential users, particularly the disabled and elderly, and not just the technology savvy.

1.2 Why is this technology important?

Before going any further, it is necessary to discuss why this technology is important and who it may potentially benefit. After all, there is no real point in investing research effort into a technology that will not benefit society to some extent.

The purpose of any technology is to make people's lives easier. The technology being researched and developed in this dissertation has the potential to improve the quality of life for many different people with varying needs and abilities. The application space for such technology is huge and could potentially be applied to, but not limited to, home networks and automation, assistance for the disabled and elderly, remote access and monitoring of home appliances, and vehicular networks.

The market for such technology is set to expand. The population of the world is aging and will become more dependent on technology to assist them around the home. Also, young people are becoming more technology savvy and are becoming more dependent on technology to save time.

1.2.1 Home Networks and Automation

The ability to control home appliances based on disparate communication and discovery technologies and using different user devices and modalities could prove to be hugely beneficial to users, particularly disabled and elderly persons. Using a single user device to communicate multi-modally with home appliances based on different technologies could bring untold advantages to such people. Communication with networked appliances is set to become more intuitive as advances are made in user interaction technologies, such as, hand gesture and voice interaction. Thus, any potential solution should accommodate the deployment of new user interface technologies without having an effect on the existing system.

Enabling appliances to cooperate and communicate could also potentially bring advantages, allowing multiple appliances to coordinate together on a particular task to create an ad-hoc virtual appliance. Again, this technology is bound to become more sophisticated as advances are made in the fields of ubiquitous computing and artificial intelligence.

1.2.2 Remote Access and Monitoring over the Internet

In the future, the ability to monitor and control appliances remotely over the Internet could become one of those conveniences that we wonder how we managed without. As an example, imagine being away from your home and getting a call on your mobile phone from a repairman who is standing outside the front door of your home and needs to get into your house to do some repair work. Using your mobile phone, it is possible to check the identity of the repair man from a discreet camera above the door and remotely open the door to let the repair man into the house.

Other possible scenarios include the ability to monitor and control appliances such as heaters, lights, bath-tubs etc. over the Internet. This has obvious advantages in that it allows users to configure their home remotely from work or the car, to be ready for them when they arrive home. Remote monitoring could also be used by people to monitor their elderly relatives. For example, by checking the status of the kettle in an elderly relative's home from your office, it is possible to tell if your relative has used the kettle that morning. If they have not, then something may be wrong and they may need assistance.

1.2.3 Vehicular Networks

Vehicular Networks are rapidly becoming standard in new vehicles. Vehicle systems being integrated include navigation systems, telephony systems, audio systems, entertainments systems, air conditioning systems and security systems. Auto companies at the forefront of these technologies, in particular BMW, have developed systems that allow users to interact multi-modally with the systems in the vehicle. The possibility of integrating the disparate technologies of the different systems and allowing users to interact multi-modally with these systems has the potential to bring unimagined innovations to this domain. Also, vehicles today are increasingly reliant on microcomputers to control systems such as braking, suspension, and fuel injection. The ability to remotely monitor these components in vehicles will further enhance the safety and reliability of vehicles in the future. Remote monitoring of vehicles could also be useful for the fleet management and other such industries.

1.3 Dissertation Outline

The layout of the remainder of the dissertation is as follows:

Chapter 2 presents both the functional and non-functional requirements for communicating multi-modally with networked appliances.

Chapter 3 presents the current state-of-the-art technologies in the domain that attempt to meet the requirements defined in the previous chapter. This includes a discussion on communication and discovery technologies, wide-area access protocols, user interface technologies and services gateways.

Chapter 4 presents the proposed architecture using the most recent state-of-the-art technologies to address the problem of communicating multi-modally with networked appliances in a heterogeneous environment.

Chapter 5 presents a proof-of-concept implementation of the proposed architecture, followed by an evaluation of the architecture and implementation.

Chapter 6 presents a conclusion and possible future work that has the potential to further improve the proposed architecture.

Chapter 2

Requirements and Specification

As in any software development project, before deciding on the appropriate technologies or designing the architecture, it is necessary to gather and clearly define the requirements of what needs to be built.

The purpose of the technologies and architectures being researched in this dissertation is to facilitate multi-modal communication with networked appliances. It should be possible to discover and interact with appliances irrespective of the underlying technology of the appliance. This should be done transparently to the user. It should also be possible to interact with appliances using a preferred interaction mode, both for the user, the device and the current situation. This multi-modality should also be transparent to the user. Another requirement of this project is that users should be able to interact with appliances remotely over the Internet. This should be as simple for the user as interacting with appliances within the local network that the appliances operate.

The requirements for this project are detailed under two headings: -functional requirements and non-functional requirements.

2.1 Functional Requirements

The functional requirements for this dissertation fall under three major categories:

- Heterogeneous discovery and communication
- Interacting with appliances remotely over the Internet
- Multi-modal interaction with networked appliances

2.1.1 Heterogeneous Discovery and Communication

In a home network or a vehicle network, there are numerous disparate technologies in operation. There are a number of different hardware and transport mediums in operation (e.g. 802.3, 802.11, Bluetooth, infrared, wired, X.10 using the powerline etc.). Also, there are a number different discovery and communication protocols in use (e.g. Jini, UPnP, Salutation, HAVi etc.). And finally, the exact characteristics and capabilities of the devices and appliances in the network will all be different. To fully exploit the capabilities offered by networked appliances, the complex diversity of networking infrastructures and device technologies must be coordinated. The requirements to enable heterogeneous communication that is independent of any particular technology are presented concisely in [3]:

"It must be possible to work with different in-domain networking technologies transparently. This requirement applies both to the physical networking and the application networking technologies."

Networked appliances should be auto-configuring as much as possible. There should be minimal user interaction, but advanced users should still be allowed to configure their appliances if they so wish.

2.1.2 Interacting with Appliances over the Internet

In order for the true value of networked appliances to be realised, it is necessary to offer remote access over the Internet. Networked appliances should be accessible to the user when away from their local environment. This should require no more effort on the part of the user than interacting with the same appliances within the local environment. The same mechanisms that exist for interacting with networked appliances within the domain should exist outside the domain. It should be possible to browse and search for particular available networked appliances within a remote domain. Naturally, some of the appliances and services that may be available to a local user should not be made available to a remote user. Any potential wide-area solution should cater for this. The requirements for communicating remotely with networked appliances are addressed extensively in the IETF Internet Draft [4].

Much of the requirements listed below are taken from this draft. The draft divides the requirements for wide-area access to networked appliances into the following sections:

- Naming and Addressing requirements
- Communication protocol requirements
- Communication mode requirements

Naming and Addressing Requirements

- A networked appliance must be assigned a globally unique name so that it can be referred to unambiguously.
- There must be support for classification of addresses and selection between multiple instances. For example, it must be possible to search for all air conditioning units, or allow for refinement of a search for a particular air conditioning unit in a particular room.
- The addressing scheme must use UTF-8 for character representation.

Wide-area Communication Protocol Requirements

- Any potential communication protocol must provide a flexible payload capability which will allow the transport of commands to, and responses from, individual networked appliances or classes of networked appliances.

- The communication protocol must provide reliability against all forms of communication errors. If an error is unrecoverable, the communication protocol should be capable of signalling this to participating entities.
- Since most communication with networked appliances is in real-time, the protocol must support efficient messaging for control. Messages should be as short as possible and be delivered as quickly as possible.
- Event notification is very important for interaction with networked appliances (e.g. notification that your washing machine is over a certain temperature), consequently, the protocol must have support for event subscription and notification.
- The communication protocol must be capable of encapsulating various appliance characteristics. For example, some appliances may respond to a user command immediately whereas other appliances may not respond or may respond after a non-determinate amount of time.

Communication Mode Requirements

There are a number of different modes for interacting with networked appliances. Support for the following interaction modes are required:

- Control - It must be possible to send control commands to appliances, for example, set the air conditioner temperature to 20 degrees.
- Queries - The communication protocol should facilitate the querying of the state of networked appliances, for example, what is the temperature of the air conditioner?
- Asynchronous events - Notification is very important for communicating with networked appliances. The protocol should allow users to subscribe to receive notifications when events happen, for example, notify a user's mobile phone when the house burgular alarm is set off.
- Discovery - A protocol should allow users to look for appliances to meet particular requirements, for example, find an appliance that allows that will make me a cup of coffee.

- Media Streaming Sessions - Any potential wide-area protocol should support media streaming sessions, for example, to view a baby sitter cam.

2.1.3 Multi-modal interaction with networked appliances

Interaction with networked appliances requires users to be able to access information and issue commands from a wide variety of user devices, with some devices designed to suit the user's specific usage requirements. The increasing rise in the computational power available to these user devices coupled with advancements in the fields of ubiquitous computing and artificial intelligence will allow client devices to offer evermore sophisticated modalities and functions on ever more miniaturised devices. Users will come to demand multi-modal interaction on a single device, or multiple devices, in order to maximise their interaction with networked appliances in eyes-free and hands-free environments (e.g. driving a car) [5]. Studies have shown how context independent interfaces and disability interfaces are closely related. For example, the problem of a user needing to be eyes-free and hands-free whilst driving a car resembles the problems with low-vision issues for disabled users [5]. It is necessary for there to be a continuity in the interface metaphors and the "look-and-feel" of the different user-interfaces, even though a user may be interacting entirely visually at one point, for example, in the home or office or entirely aurally at another point, for example, while driving a car [5]. The interfaces should be straight-forward and easy to learn so that it is not neccesary to have to re-learn when switching between modalities. Also, new functions and capabilities should be easy to master as they are introduced.

This ability to interact with appliances via a multiplicity of modalities, each designed to suit the user's specific needs and abilities at any given time, means that these interactions will exploit all available input and output modalities to maximise the usefulness of networked appliances. Probably the most intuitive and effective method of interaction is based on what users are already familiar with in their day-to-day human interaction - speech. Once speech technology develops sufficiently, it is likely to lead to a revolution as significant as graphical user-interfaces in the 1980s and the Internet in the 1990s.

To allow for multi-modal interaction, user-interfaces should be made available in formats that can be rendered by the user devices. Rather than a user-interface designer providing code for all the different possible user devices and modalities, using a generic user-interface format offers a much more efficient and sensible approach. The generic user-interface format can then be either transcoded to the target specific format on the server, or transported to the client to be transcoded locally. The requirements for multi-modal communication with networked appliances and for this generic user interface format are as follows [6, 7]:

- There should be a natural separation of user-interface and non-user-interface code.
- The generic interface representation should be completely independent of the technology and protocols used for transport over the network.
- The interface representation should not be limited to one specific modality, but it should allow mapping to other modalities. It should be possible to render the representation in the device specific format, or in a personalised format, e.g. for disabled users.
- The interface representation should be extensible. New features can be added without affecting existing features.
- The interface representation should promote a high degree of usability for persons with disabilities. Accessiblity for disabled persons may require user interface technology such as voice, braille etc.
- The interface representation should independent from the target format as much as possible.
- It should be possible to use the rendered interface to interact with networked appliances both locally within the domain and remotely over the Internet.
- The server and the user devices should be capable of run-time content negotiation to enable dynamic customised user interface delivery to the user devices.

- It should be possible for clients to cache the user interface representation. This being either the generic format or the actual target-specific user interface code. This is done so that user devices do not need to download the user-interface representation every time they communicate with the same appliance.

Synchronized multi-modal user interfaces

A hot area of research at the moment is coordinated, synchronized multi-modal interaction that works across a multiplicity of modalites and user devices. This allows a user to switch between modalites while interacting with a networked appliance, for example, interacting visually with an appliance and then switch, mid-task, to speech interaction. The requirements for synchronizing multi-modal user-interfaces that work across a multiplicity of modalities and user devices, all accessing the same information base, are outlined in [8]. Applied to the domain of networked appliances, the requirements are:

- A potential user should be able to interact in parallel with the same networked appliances via a multiplicity of user devices and networked appliances.
- A unified, synchronized and coordinated view of the networked appliances should be present across the user devices.
- There should be synchronized interaction history across the user devices.
- Interacting with appliances should be uniform and behaviour independent of the user device or modality.
- There should be tight synchronization across multiple parallel modalities.
- There should be coordination of the user interfaces, behaviours and services.
- Mechanisms should be in place to achieve synchronized interaction in a distributed environment.
- Updates to the underlying status of a networked appliance via any given user device or interface needs to be immediately reflected in all available views (user interfaces/modalities).

- User interfaces must support dynamic and often unpredictable switches across modalities.

2.2 Non-Functional Requirements

Due to the dynamic nature of a home or vehicular network, with a wide variety of networked appliances and user devices joining and leaving, it is necessary for a networked appliance architecture to be robust and capable of dealing with change and failure on many levels. It is also necessary to deal with security and real-time communication issues. It is vital to meet these non-functional requirements for there to be any major rise in the use of networked appliances by the general population.

2.2.1 Adaptation to Failure

It is expected in a pervasive environment such as a networked appliance network that connectivity will not be continuously guaranteed. Any potential solution must display a fair reaction to this situation. Communication must be best effort and be capable of recovering from failures.

2.2.2 Consistency

What a user sees in his or her user device should conform to what is happening in the real world. There is no point in having the ability to monitor a networked appliance if the system does not give a true picture of the real state of the networked appliance. The consequences of this are not difficult to imagine, for example, the house is on fire, and the user's device is not aware that the networked fire alarm has gone off. Also, there should be a consistency between all user devices within the system. This means that messages passed within the system should be ordered properly. This is relatively easy to achieve since all messages sent to networked appliances go through the gateway server, which can act as a sequencer for the messages sent by the user devices.

2.2.3 Security

The importance of security to networked appliances is obvious. A broad range of security threats exist for networked appliances. Ease of use, location transparency, multiple user-interfaces and mobile terminals raise many security concerns with regards to networked appliances. If the system is connected to actuators of any kind, a compromised networked appliance has the potential to spoil food, injure its user, or set fire to a house [9] and naturally one does not want his or her front door to be opened by an unauthorised person. Also, information such as the existence of certain networked appliances, their locations within the home, the times and frequencies an owner accesses these appliances, may all be sensitive information.

The traditional taxonomy of security threats identifies three main classes; confidentiality, integrity and availablility [9]. Confidentiality is violated when unauthorized users gain access to private information; integrity is violated when unauthorized users modify information or systems; and availability is violated when an entity in the system is prevented from performing its intended function. Protection against all these security threats relies on a distinction between authorised and unauthorised users. A failure in authentication can easily lead to violations of confidentiality, integrity and availability.[9] The security considerations for communicating with networked appliances will be addressed under these three headings.

Confidentiality

When people first think about security issues for pervasive computing, they think of eavesdropping as a consequence of wireless communication. This concern is not really an issue of major concern however. Once the difficult problem of authenticating the users and sharing key material has been done, there are mature and robust symmetric ciphers such as AES (Advanced Encryption Standard) for protecting the communication channel's confidentiality [10].

For wide-area access to networked appliances over the Internet, where user's may not have access to a shared secret key, there are robust asymmetric key algorithms available. An alternative solution, outlined in [11], argues that communication with networked appliances usually involves communication between a relatively small set of communicating entities. At one end is the gateway server. On the other end will be the relatively small number of various user devices. These include the users' mobile phones, PDAs, laptops and so on. It is conceded that occasionally a user may borrow another person's mobile phone, or similar user device, to interact with the home network. While this is possible, it is more likely that a user will be using his own user devices most of the time. This characteristic can be exploited to allow the use of shared secrets on the user devices. Symmetric-key algorithms are generally much faster to execute than asymmetric key algorithms. This would be an obvious advantage on resource constrained user devices, such as mobile phones and PDAs.

It is much more difficult, however, to protect the confidentiality of the user devices themselves. Today, the information held on user devices is not very valuable but as innovations are made in communicating with networked appliances, it is likely that information about user's personal activities and preferences will be stored on the user devices. This opens up the issue of abuse of personal privacy if the confidentiality of the devices is compromised.

Thus, it is important to protect the confidentiality of the data held in the user devices that interact with the networked appliances. In a networked appliance environment, use of a particular service must be restricted to authorised users only and authenticity of the user must be verified in a reliable way before the user is granted access. If a user's authenticity is not verified, it is possible for many threatening scenarios to occur. Another issue of ensuring confidentiality is protecting within the user device any long-term keys used to encrypt the commands being sent to the networked appliance or gateway server. It is also important to have a procedure to recover from a user device theft when the thief has possession of the user's password.

Another issue affecting confidentiality that needs to be addressed is traffic analysis, which could be used as a surveillance tool [10]. Even though the commands being sent to the appliances are encrypted, the form of the transaction may still remain observable. Also, it may be possible to determine a user's location based on this traffic analysis, which again is a violation of privacy.

Integrity

The basic integrity problem is to ensure that messages from the user device to the networked appliance (or gateway server) are not corrupted by a malicious third party. This is similar to the confidentiality problem already addressed in that, once the hard work of authentication and key distribution is done, the problem of integrity is easy to address using well known cryptographic mechanisms, such as message authentication codes [10]. The most difficult issue for integrity, therefore, is again not with the transmission of the messages but with the user devices themselves. The most effective way to address integrity is to ensure that the user devices and the services offered by the networked appliances are tamper-proof.

Availability

The classic attack on availability is the denial-of-service attack. Some networked appliances may be particularly vulnerable to denial-of-service attacks because of their limited power supply, for example, battery operated appliances. Because of this limited power supply it may be necessary for some networked appliances to go on stand-by to conserve energy. An effective attack on such appliances would be to continuously send messages to the networked appliance, thereby keeping it awake, until it runs out of battery and dies. Authentication combats this to some extent, but the problem arises when networked appliances serve queries from unknowns. It is very unlikely that a networked appliance will receive commands or queries from unknowns since most communication with networked appliances will be done through a gateway server, but it is still necessary to be aware that this may be a possiblity.

Another problem can occur when an attacker forces multiple valid user devices into cooperating in an attack. This is called the DDOS (Distributed Denial-of-Service) attack. With the DDOS attack, malicious attack software is installed on the user device and left dormant until a sufficient number of user devices have been infected. At that point all the infected user devices are ordered to attack a particular target (i.e. a networked appliance) simultaneously. Even if the attack consists of nothing more than sending empty messages to the networked appliance, its replicated nature will flood the network [9]. Once attacked, there is very little a networked appliance can do other than disconnecting. Networked appliances are particularly attractive for DDOS attacks because the appliances are permanently connected to the network and will usually not have a knowledgable system administrator looking after them and noticing security breaches [9].

2.2.4 Open Standards

It is necessary to base all the protocols and formats used in the system on open standards. This includes the communication protocol, the deployment of the gateway server, and the format of the generic user-interfaces. This is an obvious requirement in order to make the proposed solution as interoperable, flexible, and as future-proof as possible. Usage of open standards leads to seamless integration of technologies, protocols and devices throughout the environment.

2.2.5 Performance

Since communication with networked appliances is going to be done in real time, it is neccessary that all messages sent to networked appliances are sent as quickly and as efficiently as possible. This requirement encourages the use of a lightweight and flexible protocol for wide-area communication. The protocol should work equally well in connection-oriented mode (over TCP) and in connectionless mode(over UDP).

2.2.6 Resource Constraints

Many of the networked appliances, and also many of the user devices, may have limited memory and processing power. Thus, it is necessary to keep the processing done on both the networked appliances and the user devices to a minimum. Most of the intensive processing should be done on the gateway server. This includes the transcoding of the generic user-interface format to the target-specific format. It should still be possible for thick clients, with more resources, to do their own transcoding and offer richer functionality if they have the capabilities to do so.

2.3 Specification

The specification provides more detail on the issues described in the requirements. It describes the behaviour of the architecture as seen by an external observer. This is neccessary to ensure that there is no ambiguity about the features and behaviour of the system before the design phase.

In communicating with networked appliances, there are three different entities that are capable of performing actions. They are; a local user within the domain, a remote user accessing the domain over the Internet, and finally the gateway server. The gateway could be a lightweight server in a vehicle or a residential gateway in a home. It is used as a gateway to the Internet and is also used to federate the disparate technologies that are operating within the domain. The gateway also stores, or generates, the generic user-interface descriptions used to facilitate multi-modal communication with networked appliances. Depending on the client device capabilities, it may also be the responsibility of the gateway server to transcode the generic user-interface format to the target-specific format.

The use case diagram for the local user is shown in Figure 2.1, the remote user in Figure 2.2 and the gateway server in Figure 2.3. Some of the use case descriptions are presented in Appendix A.

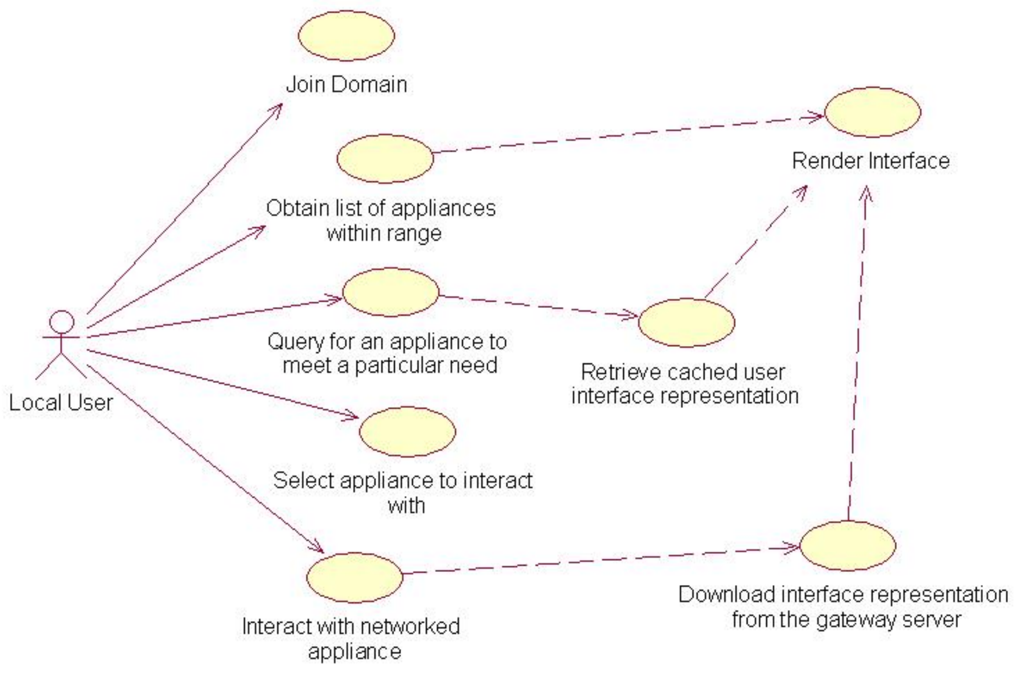
Join Domain
Render Interface
Obtain list of appliances within range
Query for an appliance to meet a particular need
Retrieve cached user interface representation
Local User
Select appliance to interact with
Interact with networked appliance
Download interface representation from the gateway server

Figure 2.1: Use case diagram for a local user within the domain

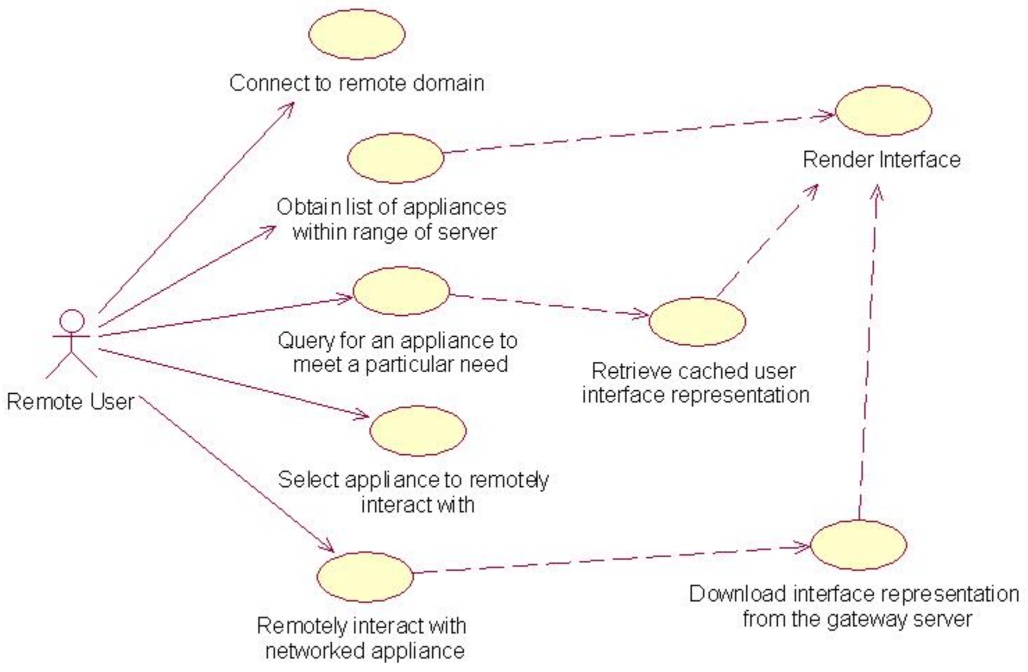
Connect to remote domain
Render Interface
Obtain list of appliances within range of server
Query for an appliance to meet a particular need
Retrieve cached user interface representation
Remote User
Select appliance to remotely interact with
Download interface representation from the gateway server
Remotely interact with networked appliance

Figure 2.2: Use case diagram for a remote user accessing networked appliances over the Internet

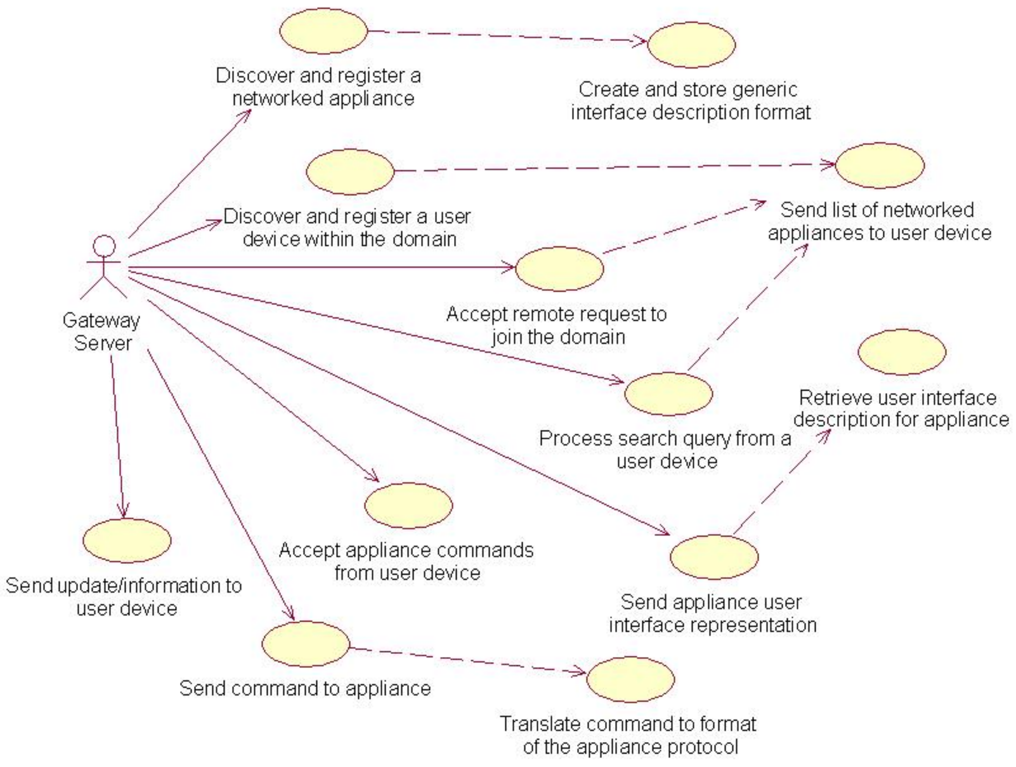
Discover and register a networked appliance
Create and store generic interface description format
Discover and register a user device within the domain
Send list of networked appliances to user device
Gateway Server
Accept remote request to join the domain
Retrieve user interface description for appliance
Process search query from a user device
Accept appliance commands from user device
Send update/information to user device
Send appliance user interface representation
Send command to appliance
Translate command to format of the appliance protocol

Figure 2.3: Use case diagram for the gateway server

Chapter 3

Enabling Technologies and State-of-the-Art

The number of networked environments, as well as the number of networked appliances and devices within the environment, are increasing at an accelerated pace. Commonplace home appliances, such as air conditioners, refrigerators, coffee pots etc., as well as vehicle systems, are being networked together. This trend, coupled with the profileration of a variety of user devices is making the home and vehicle network ever more complex. This results in a myriad of communication standards and protocols, many of them proprietary, which do not interoperate easily with each other. This expanding list of technologies includes hardware and transport technologies such as IEEE 802.3, IEEE 802.11, Bluetooth, Infrared and X.10 using the powerline. Coupled with this there is also a wide array of discovery and communication protocols such as, Jini, UPnP, HAVi and many more. The state-of-the-art in these technologies is discussed, as are methods of federating the disparate technologies to enable communication using a multitude of protocols and technologies.

As discussed in the requirements, to fully exploit the opportunities presented by networked appliances, it is neccessary to have wide-area access. Up until quite recently, despite the fact that there was a plethora of technologies available for accessing networked appliances within the network, there was not adequate support for accessing networked appliances remotely over the Internet. In recent times however, some research in the area of remote access to networked appliances has been done. Based on an analysis of this research, it seems that most prominent solutions for remote access to networked appliances over the Internet are based on either HTTP (Hyper-Text Transfer Protocol) or SIP (Session Initiation Protocol), both of which run over the Internet Protocol(IP). The use of SIP for remotely interacting with networked appliances provides certain advantages, and an extended version of SIP has been proposed, specifically designed for communicating with networked appliances. The specifics of this extended version of SIP are presented.

The difficulties of having a wide variety of user devices, each with a specific modality, was outlined in the previous chapter. A possible solution to this is to use a generic user-interface representation to encode the user interface and then transcode this generic representation to the target-specific format. The state-of-the-art in generic user interface languages and technologies including, XIML, UIML and DISL is presented in this chapter.

Finally, the deployment of the gateway server needs to be decided. Recently, the gateway has evolved from a simple hub, providing internet services to a few networked PCs, into a services gateway [12]. A services gateway is a platform on to which applications and services can be dynamically deployed and managed. It has also become the responsibility of the services gateway to federate the disparate technologies in operation within the environment. The Open Services Gateway Inititive (OSGi) is a specfication for such a services gateway and is explained in detail.

3.1 Hardware and Transport Medium Technologies

The physical connection between the networked appliances and the gateway server can be implemented in various ways, either by using a wire or some type of wireless signal. It is likely that a variety of wired and wireless mediums will operate together within an environment to fulfill the needs of a broad range of applications. For example, some applications such as audio and video will require high bandwidth, whereas other applications such as controlling a coffee pot will require low bandwith. In general, wired technologies support higher bit rates over longer distances when compared to wireless technologies and would be suited to applications like video-streaming[13]. Wireless technologies are generally more suited to low bit-rate services requiring a high degree of mobility. In [13], the physical media that home networks operate over is organised into three broad groups: structured wiring, existing wiring, and wireless.

- **Structured wiring** requires the installation of new cabling in the walls. Both the cabling and its installation are defined by standards (e.g. 802.3 for Ethernet). The cabling is typically unshielded twisted pair (UTP) or fibre optics).
- **Existing wiring** uses the existing infrastructure of electrical, telephone, or other wiring already installed within the home.
- **Wireless** avoids the use of any wired infrastructure by sending signals through the air.

3.1.1 Structured Wiring

It is recognised in [14], that installing new structure wiring is a safe way to deploy new services in the home environment, as these technologies have been tested in the entreprise and business sectors. The structured wiring technogies that deserve particular consideration are Ethernet, Universal Serial Bus (USB), and IEEE 1394.

Ethernet

Ethernet is formally standarised as IEEE 802.3. It is a relatively mature technology and its installation and configuration are simple and familiar. The Ethernet standard is also supported by a large number of vendors, so Ethernet devices, network interface cards (NIC) and wiring are widely available. For this very reason, it is an attractive solution for networking appliances and other such devices. However, the main problem with using Ethernet in a home environment is that it requires the setting up of a new infrastructure which can be both time consuming and expensive.

USB (Universal Serial Bus)

The main advantage of USB for home networks is its ability for "hot-swapping" (i.e. plugging and unplugging devices without interrupting the operation of other devices) of multiple peripherals in a daisy-chain architecture [14]. Connecting USB devices is more or less an automatic process, since most modern PCs and operating systems provide support for USB connections. Moreover, in many cases, it is unnecessary for a USB device to have a power source, as the power can be automatically delivered from the PC. USB provides both asynchronous data transfer and isochronous audio/video streaming channels which promise transfer rates of up to 460-480 Mbit/s, and covers the requirements of bandwith devices such as cameras [14]. However, it is unlikely that USB will achieve dominance in home networks, since it requires the networked device to be located a short distance from the gateway server.

IEEE 1394

IEEE 1394 (also known as Firewire or i.Link) was initially proposed by Apple Computer in the 1990s, as a PC and digital video serial bus interface. It provides isochronous real-time data services and high transfer rates. Though originally designed for entertainment applications, it has turned into an emerging standard that targets all multimedia applications. IEEE 1394 does not specify a physical medium and mostly uses copper or plastic optical fibre (POF). The main advantage of IEEE 1394 for networked appliances is that it implements an easy to set-up, hot-plugging scalable architecture that does not require terminators or device IDs. It allows for a mix of transfer speeds, enabling the interconnection of devices with different speed capabilities and costs. Currently, a number of standardisation bodies and committees, including DVB (Digital Video Broadcasting), DVC (Digital VCR Conference), the Consumer Electronic Association (CEA) and the Versatile Home Network (VHN) have adopted IEEE 1394 as their default interface for broadband, low-cost, digital communications. [14]

3.1.2 Existing wiring

Since structured wiring can be quite expensive to install, many companies are devloping technologies based on the existing infrastructure already within the home [13]. The most prominent solutions for using existing wiring within a home are X10, LonWorks, HomePlug and CEBus over the powerline and HomePNA over the phoneline.

X.10

X.10 provides a solution for controlling simple home appliances. It utilizes existing electrical wiring in buildings and, hence, does not require any additional cabling. The basic command set of X.10 includes commands for switching power on and off, adjusting brightness level and querying statuses with optional numerical parameters. X.10 has 256 possible addresses within a domain. X.10 specifies different modules for controlling different appliances. For example, a lamp module is specified for controlling lamps. This module is capable of switching a lamp on or off and dimming the brightness of the lamp. More complex modules are required for controlling more sophisticated appliances. A infrared and radio frequency based transport are also defined for X.10. One major problem with X.10 is slow data rates of around 20 bit/s. This shortcoming means that X.10 is, in practice, restricted to very simple operations on very simple networked appliances (e.g. turning a coffee pot on etc.). [15]

HomePlug

The HomePlug Powerline Alliance was formed to build a cost effective technology that uses the existing powerline and offers Ethernet standard connectivity rates. The alliance consists of 12 founding members and over 80 participating companies. The current HomePlug standard allows for speeds up to 14 Mbit/s. In August 2005, a new standard which allows for speeds greater than 100 Mbit/s was approved by the HomePlug Powerline Alliance. This standard is still backward compatible with HomePlug 1.0. [15]

LonWorks

LonWorks is a networking platform created by the Echelon corporation specifically to address the unique performance, reliability, installation and maintenance needs of control applications. LonWorks can use the powerline, twisted pair, radio frequency, infrared, fibre optics and coaxial cable. It is a low bandwidth protocol and is particularly popular for the automation of various functions within buildings such as lighting and heating and ventilation systems. [15]

CEBus

Consumer Electronic Bus is a communications standard for in home networks developed by the Electronics Industry Association (EIA) and the Consumer Electronics Manufacturers Association. Like LonWorks, the standard does not just apply to communication over the powerline but also coaxial cable, radio frequency and Infrared. A disadvantage of CEBus is that there are relatively few products available and the high cost of those products. [15]

HomePNA

HomePNA is being defined by the Home Phoneline Networking Association whose aim is to promote and standardise technologies for home phone-line networking, and to ensure compatibility between home networking products[14]. Since HomePNA takes advantage of the existing home phone wiring, this enables an immediate market for HomePNA devices. HomePNA 2.0 provides 10 Mbit/s rates.

3.1.3 Wireless Infrastructure

Wireless connectivity is extremely desirable for communicating with networked appliances, because it allows unhindered mobility and access. If the networked appliance architectures are to benefit all intended audiences, particularly disabled and elderly users, then wireless access is a necessity. Today, there are a broad range of technologies available to construct a wireless infrastructure. It is unlikely that any one technology will win out, since each of the technolgies is designed for different situations. It is much more likely that a home network or a vehicle network will consist of a number of complementary technologies, all part of the overall solution. Some of the prominent candidates for communication with networked appliances are presented.

IEEE 802.11

IEEE 802.11, also known as Wi-Fi, is an obvious candidate for communicating wirelessly with networked appliances. The 802.11 family of Ethernet standards are used to standardise wireless LANs. It is a mature, robust technology that has been deployed for some time in corporate settings. It operates in the unlicenced band of 2.4 and 5 GHz. IEEE 802.11 systems operating within the 2.4GHz range provide data rates up to 2 Mbps. Enhancements to the 802.11 standards have created faster versions of the specification. The most significant is the IEEE 802.11b specification, which achieves data rates of 5.5 and 11 Mbps by using CCK (code keying modulation). [16]

Bluetooth

Also known as IEEE 802.15.1, Bluetooth has been suggested as a potential candidate for wirelessly connecting networked appliances [14, 16, 17]. Bluetooth is under the direction of the Bluetooth Special Interest Group. This group defines profiles for use with Bluetooth, for example, the Hands-Free Profile that allows hands free kits in cars to communicate with mobile phones. Bluetooth has a range of up to 10 metres, operates in the 2.4GHz unlicenced band, and provides rates of up to 10 Mbit/s. Bluetooth operates by having up to 7 slave devices communicating with a master device. This arrangement of up to 8 devices is known as a piconet. It is possible to link piconets together to form a scatternet, allowing for a more flexible configuration. Because of the limited bandwith, Bluetooth is unlikely to be used for audio and video transmission. Also, because of the limited range, it is likely to be restricted to only certain applications within the home.

Infrared

In [17], Infrared communication, as standardised by the Infrared Data Association (IrDA), is examined as a potential candidate for networking embedded computers and networked appliances. At data rates of up to 4 Mbps, it seems like an appropriate solution for communication with networked appliances. It is pointed out however, that there are a number of interoperability issues due to the many operating modes that the standard tries to make available. Another major drawback to using Infrared to communicate with networked appliances is that Infrared requires, in most cases, line-of-sight operation. This could be be viewed as an advantage, though, for appliances that require explicit selection, for example, by pointing directly at the appliance to interact with it.

IEEE 802.15.3

IEEE 802.15.3, also known as UWB (Ultra Wide Band), is designed to offer an wide bandwidth over a limited area. It is targeted specifically at high-definition video and high-fidelity audio and offers a wide bandwidth to such applications (3.1 - 10.6 GHz). The specifiction dictates a data rate of 55 Mbps or more. The standard is optimised for short-range transmissions limited to 10 metres. This enables appliances to have very low-power requirements (approx 200uW. This is an obvious advantage for battery operated appliances.

IEEE 802.15.4

IEEE 802.15.4 is an emerging standard for low-power wireless monitoring and control. It has the potential to scale to many devices and has a typical range of 10-75 metres. ZigBee is the network and application layer on top of IEEE 802.15.4 and is standardised by the ZigBee Alliance. It is specifically aimed at devices with low data rates and low-power consumption. Again, like IEEE 802.15.3, it is aimed at very low-power devices, but it has much lower data rates. Possible applications within the home could include low data rate, battery operated devices, like smoke detectors.

3.2 Discovery and Communication Technologies

End-users are not too interested in the underlying protocols used to communicate with networked appliances. What end-users are really interested in are the services provided by appliances and devices. Ideally, networked appliances should be capable of advertising services and user devices should be able to search for particular services. The lack of network administrators in the home or car requires that the networks, and their services, are capable of doing this automatically. This task of service advertisement and discovery is addressed by service discovery protocols such as Jini, UPnP, SLP (Service Location Protocol), and Salutation. In general, service discovery protocols work by advertising the available service and supplying information about the capabilities and details about the services offered. Clients may then discover a particular service and register to use that service. Common service and discovery protocols include Jini, UPnP, SLP (Service Location Protocol), SDP (Service Description Protocol) and Salutation. Two of the more prominent discovery protocols used for communicating with networked appliances, Jini and UPnP, are discussed below.

3.2.1 Jini

Jini-based technology, developed by Sun Microsystems, addresses the problems of distributed systems by defining mechanisms to support the federation of machines or programs into a single, dynamic distributed system. Jini provides mechanisms for service construction, advertisement, lookup, communication, and use in a distributed system [18]. Each Jini enabled device provides a service by publishing its own interfaces, which other devices can then discover and access the service provided by the device. Each device in the Jini network is represented as a Java object and the object's interface advertises the services that the device has to offer. Discovery and lookup services allow Jini clients to obtain references to the remote objects advertised by the devices. The clients can then invoke the methods of the remote objects using Java RMI (Remote Method Invocation). Examples of services operating within the home domain might be air conditioners, coffee pots etc. Examples of clients would be user-devices that wish to use this service.

Lookup Service

The lookup service acts as a broker between the service and the client. Services register their proxy objects with a lookup service using discovery and join protocols. Discovery occurs when a service is looking for a lookup service to register its service. Join occurs when a service has located a lookup service and joins it. Clients can then use the lookup protocol to find the service offered. Clients see a service as a Java interface with methods that the client can invoke to execute a particular service. The lookup service maps the Java interfaces seen by the clients to a set of RMI stubs. The client can then download this RMI stub and communicate back with the server using RMI. This enables a client to use the service without knowing anything about it. [18]

Leasing

To improve fault tolerance within a Jini network, Jini grants access to services on a lease basis. It is likely that devices or appliances may leave the domain or suddenly fail without having a chance to deregister themselves. To protect against this, a service is granted for a negotiated time period between the client and the service provider. The lease must be renewed before its expiration if a service user wishes to continue using the service. Otherwise, the resources associated with the services are released. [18]

Jini Surrogate Architecture

Jini also allows non-Jini devices to join a Jini network. This is useful for devices that may lack the resources to run their own JVM (Java Virtual Machine) which is likely to be the case for many networked appliances. A surrogate host is used is used to act on behalf of the non-Jini device. The surrogate host performs the discovery and lookup tasks, thereby allowing the non-Jini device to participate within the Jini federation. All future communications between the non-Jini device and the other Jini-enabled devices takes place through the surrogate proxy [19].

3.2.2 UPnP (Universal Plug and Play)

UPnP is a service and discovery protocol that exploits existing open standards, such as IP, TCP, UDP, HTTP, SOAP and XML. In UPnP, a device can dynamically join a network, obtain an IP address, provide its capabilities upon request and learn about the presence and capabilities of other devices. All of this is done automatically. The following are the characteristics of UPnP that make it attractive for networked appliances [20]:

- **Zero-configuration** - Once an UPnP device is plugged into the network it will automatically configure itself, plug itself into the network, acquire an IP address, and announce its presence.

- **Standards based** - UPnP is built on standards such as IP, TCP, UDP, HTTP, SOAP and XML. This enables UPnP to be a very interoperable protocol.
- **Platform independent** - Because UPnP is standards based it can be implemented using any programming language, on any operating system or hardware platform.
- **Transport layer independence** - UPnP can be used over a wide variety of underlying physical mediums providing it has an IP stack.

UPnP Services

A UPnP service provides actions that can be invoked by clients. The state of a service is modelled by state variables. It is possible for clients to subscribe to be notified of changes to these state variables. A UPnP enabled device may contain a number of services. The UPnP forum is establishing standards for UPnP devices and services, for example lighting controls.

Control Point

A UPnP control point is capable of discovering and controlling other devices. Control points are analagous to clients in Jini. It is possible for UPnP control points to also provide services.

UPnP Communication

The operation of UPnP can be explained in five steps [18]:

1. **Addressing** - IP addresses are used to uniquely identify every UPnP device and control point. DHCP (Dynamic Host Configuration Protocol) is used to provide IP addresses. It is necessary for all UPnP devices to have DHCP clients. When a device first connects to the UPnP network, it contacts the DHCP server for

an IP address. If the DHCP server fails, then Auto-IP will be used to assign an address.

2. **Discovery** - UPnP uses Simple Service Discovery Protocol (SSDP) for service discovery. SSDP is built on top of HTTPU (HTTP over unicast) and HTTPMU (HTTP over multicast). When a device is added to the network, the device advertises its services to the control points on the network, by sending out a multicast message. Similarly, when a control point is added to the network, a SSDP discovery message is multicasted, in order to look for devices of interest. Any devices that match the search criteria specified by the control point must respond to it with a unicast message.

3. **Description** - For the control point to learn more about the capabilities of the device it needs to know the capabilities of the device. The advertisement message sent by the device contains a URL that points to an XML file describing the features and capabilities of the device and the services it offers. The control point can retrieve this XML file to inspect the features of the device and the services it offers to see if it fits its purposes. This XML description allows complex, powerful desciption of services, as opposed to Jini's interface API offering. The XML file contains URLs for control and eventing. For each service, the description includes a list of commands or actions to which the service responds. The description also includes a list of state variables which model the state of the services at run time.

4. **Control** - Once a control point has the XML device description document, it can control the device. A control URL for each of the services offered by the device is provided in the XML description document. Control messages are expressed in XML using the Simple Object Access Protocol (SOAP). HTTP is used as the underlying protocol to send the messages.

5. **Event Notification** - Services can publish updates when the state variables which model the state of the services change. It is possible for control points to subscribe to receive these updates. For example, a user device within a home network could subscribe to receive a notification when the state variable defining the temperature of an air conditioner changes. When a change occurs in a state variable that a control point is subscribed to, the control point will receive

an event message containing the new value of the state variable. These event messages are expressed in XML and are formatted using the General Event Notification Architecture (GENA). GENA is defined to send and receive messages using HTTP over TCP/IP and using HTTPMU over UDP/IP.

3.3 Wide-area Communication Technologies

As discussed in the previous chapter, the problems that need to be addressed by a potential wide-area solution to communicating with networked appliances are [21]:

- limited address capabilities.
- handling of mobility.
- security, in particular preventing unauthorised access to home appliances.
- resource limitations in mobile terminals and networked appliances.
- simple protocol, so as not make networked appliances complex and expensive.

HTTP (Hyper-text transfer protocol) and SIP (Session Initiation Protocol) are the two most common solutions for communicating with networked appliances over the Internet. There are a number of reasons why SIP, with some extensions, might be particulary attractive for communicating with networked appliances.

3.3.1 HTTP (Hyper Text Transfer Protocol)

HTTP is a well known protocol used for communication with web browsers. Many solutions to accessing networked appliances use HTTP for this very reason. It is a relatively mature and widespread technology, easy to implement, and has some security features, for example SSL (Secure Sockets Layer). However, there are a number of reasons why HTTP is not suitable for communication with networked appliances [22]. Networked appliances could potentially be mobile and are also likely to generate asynchronous notifications. HTTP does not provide good support for mobility or notifications. Also, another disadvantage of using HTTP to communicate with networked appliances is that HTTP must run over TCP, which could be a liability in some situations.

3.3.2 SIP (Session Initiation Protocol)

SIP is a call setup and management protocol for multi-media communications. It is the most popular protocol used for Voice over IP solutions. SIP is an application protocol and is independent of the underlying transport protocol and only requires unreliable datagram service, as it provides its own reliability mechanism. It usually runs over UDP or TCP, but it could just as easily run over many other transport protocols. SIP also provides a suite of security services which include denial-of-service prevention, authentication, integrity and encryption.

SIP [23] facilitates the creation of sessions allowing the exchange of data between participants in the session. These sessions include Voice over IP telephone calls and other multimedia content. It is also possible for SIP to invite participants to already existing sessions. SIP entities are identified by their SIP URI (Uniform Resource Identifier). SIP URIs have a similar format to email addresses. SIP is based on a HTTP like request/response transaction model. Each transaction consists of a request that invokes a response from the server. SIP clients (for example SIP-enabled softphones) send messages to SIP URIs. All messages sent by the SIP client go through a SIP server. A SIP server is an intermediate device that recieves SIP requests and then forwards the requests on the clients behalf. A basic transaction begins by a client sending an INVITE request to the SIP server. The **INVITE** request includes the address of the caller (in the From header field) and the address of the intended callee (in the To header field). The SIP server then sends the **INVITE** message on to the proxy server acting on behalf of the intended callee. This server then passes the message on to the callee. Once the callee has received an **INVITE** message, the two clients negotiate a session through their respective proxy servers. The session is negotiated by the sending of a series of very simple messages and acknowledgements. Once the session has been established, a media session begins using the format agreed upon in the session setup phase. At the end of the session, a **BYE** message is sent to terminate the session. To demonstrate the simplicity of SIP messages, an **INVITE** message taken from the SIP RFC [23] is shown below.

```
INVITE sip:bob@biloxi.com SIP/2.0
Via:  SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bK776asdhds
Max-Forwards:  70
To:  Bob <sip:bob@biloxi.com>
From:  Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710@pc33.atlanta.com
CSeq:  314159 INVITE
Contact:  <sip:alice@pc33.atlanta.com>
Content-Type:  application/sdp
Content-Length:  142
```

Extended SIP

An extended version of SIP, specifically designed for communicating with networked appliances, has been suggested [22]. Rather than design a new protocol from scratch, the designers decided that reusing the infrastructure of an existing protocol and adding a few extensions would be more appropriate. The new methods added to SIP are **DO**, **SUBSCRIBE** and **NOTIFY**. Messages or requests for networked appliances are carried in the body of the **DO** request, and are delivered to the home environment [24]. Since event notification forms a very important part of networked appliance communication, **SUBSCRIBE** and **NOTIFY** methods were added [25]. **SUBSCRIBE** enables user devices, or other networked appliances, to subscribe to certain events within the network, and **NOTIFY** allows appliances to notify subscribers of events occurring. The advantages of using SIP for communication with networked appliances over other protocols, or over designing a completely new protocol, are presented in [26]. Succinctly, these advantages are:

- **Scalability** - SIP is a very scalable protocol. It works equally well in LAN and WAN conditions. SIP can also work over a variety of transport protocols, including UDP and TCP.
- **Simplicity** - SIP allows service providers to rapidly deploy new services easily. It is a very lightweight protocol and it is text based. This simplicity means that

it can be implemented on a small footprint, an attractive feature for networked appliances. It is also well integrated with the IP (Internet Protocol) family of protocols.

- **Flexibility** - It is very simple to add extensions to support new features. The protocol is defined in such a way that any provider can easily define extensions to the existing grammar to add features which may not exist in the core SIP specification yet. This is useful for accommodating novel applications. Most importantly, SIP specifies mechanisms ensuring that any new extension does not break an existing SIP aware node that may be in route but does not understand the new extension.

- **Registration** - In SIP, it is not necessary that a calling device needs to know exactly where to locate the called device or appliance. An appliance can register its current location with its registrar. The exact location will be resolved by the proxy in conjunction with the registrar.

- **Personal Mobility** - Related to the above point, SIP provides the concept of personal mobility at no extra cost.

- **Security** - SIP provides both authentication and encryption using schemes such as PGP (Pretty Good Privacy) to provide end-to-end security. This has obvious advantages since security is very important for networked appliances.

- **Transport Independence** - SIP is agnostic about the underlying lower layer transport protocol. This essentially means that SIP messages can traverse through various heterogeneous networks.

- **Event notification** - SIP has been extended to introduce **SUBSCRIBE** and **NOTIFY** methods for communicating with networked appliances. This enables entities to subscribe to certain events and to be notified when they occur. This is one of the more attractive features of using SIP to communicate with networked appliances.

- **Addressing** - The URI (Uniform Resource Identifer) addressing scheme used by SIP is extremely generic and flexible. It can encompass a wide range of addressing requirements, which may well be the case for communicating with networked appliances.

- **Integration with existing SIP service mechanisms** - Using SIP enables networked appliances to exploit the rich infrastructure that SIP provides.

- **Session based and non-session based communication** - SIP allows communication using both session based and non-session based transactions, which is ideal for communicating with networked appliances providing different types of services.

- **Business models** - Finally, since the extended version of SIP is the only protocol specifically designed for communicating with networked appliances, it would introduce the possiblities of different service models for networked appliances. This is similar to what HTTP has done for the world-wide-web. For example, Electrolux are already investigating a "pay-per-use" washing machine based on remote monitoring of use over the Internet.

Solutions for interacting with appliances using the extended version of SIP and utilising the new **DO**, **SUBSCRIBE** and **NOTIFY** methods are presented [27, 28, 21]. Two other extensions, **LOCK** and **UNLOCK** are suggested in [27] to allow appliances to interact with each other without interference. With the **LOCK** and **UNLOCK** extensions, it is possible for services to gain exclusive control of an appliance. For example, a burglar alarm service may want a door to remain locked and not allow any other service to open it.

The format that is to be carried as the body of the new SIP messages (i.e. the SIP payload) is presented in [2]. This format is intended to be carried as the body of the **SIP**, **DO**, **SUBSCRIBE** and **NOTIFY** messages. An XML based format, Device Message Protocol (DMP) is suggested. An XML-based format is an ideal solution to such a problem. XML is based on industry standards and is independent of the transport protocol that is used to carry the message. This is important for communicating with networked appliances, since a goal of any solution is to be as future-proof and interoperable as possible. Using an XML-based format as the payload means that any protocol that supports MIME types, and not just SIP, may also be valid for carrying

DMP messages. For example, it may be possible for HTTP to carry DMP messages by setting the MIME type to "text/dmp" [2]. Another advantage of using XML is that it is relatively easy to process. DMP captures all the requirements for communicating with networked appliances - conveying information pertaining to control, query and event subscription and notification. Protocol bridges are required to convert between DMP and the various technologies that exist within the domain (e.g. X10, UPnP etc.). This translation is done by the gateway server.

A solution for applying SIP Security to Networked appliances is presented in [11]. The paper recognises that SIP supports basic and digest authentications, which follows a challenge-response paradigm. The solution outlined also suggests using shared secrets in communication with networked appliances. As discussed in the previous chapter, it is likely that communication with networked appliances will involve a relatively small number of communicating entities. Most of the time, a user will be using his or her own user devices to communicate with the same networked appliances. This characteristic can be exploited to allow the use of shared secret keys. This is an advantage because symmetric key security is typically more efficient than public key encryption for the same level of security. At the same time however, it is pointed out, that because of the relatively frequent communications with networked appliances, it is necessary to update the symmetric key in a timely fashion. Otherwise, attackers may be able to determine the key for an analysis of the large amount of information that is being passed. The use of Secure Remote Password (SRP) [29] is suggested as the session key agreement algorithm in communicating with networked appliance. An expiry time of 24 hours is suggested as the expiry time for the session key, since it would be demanding, and unpractical, to have to set up a session key for every single session. It is beyond the scope of this disseration to go into details on the security features, but basically the security solution presented in [11] provides protection for different users accessing networked appliances and for the same user roaming across multiple user devices.

3.4 User Interface technologies

As discussed previously, it is expected that there will a myriad of devices, modalites and user interfaces operating within a home or vehicle network. Some of the potential user-interface technologies that may be in operation within a networked appliance environment include:

- Java Swing and Visual C++ for desktop PC applications.
- Mark-up languages such as WML (Wireless mark-up language) for mobile devices, and HTML (Hypertext mark-up language) for web browsers.
- VoiceXML, SpeechML and other such languages for voice-enabled user devices.

It is unreasonable to expect users to exclusively use a single modality or device, so solutions need to be offered that cater for multiple modalities. It is also unreasonable to expect user-interface developers to develop numerous different user-interfaces for every potential user device. This would require developers to be experts in all of the user-interface technologies and languages used to communicate with networked appliances which is almost impossible. The solution to these problems is to build interfaces with a single, generic language, free of assumptions about the user devices and the interface technologies that they use, as much as possible citeAPPLIANCEINDEPENDENT. Some of the more prominent generic user-interface languages are presented next.

3.4.1 XUL (XML User Interface Language)

XUL (XML User Interface Language) [30] pronounced "zool" is an XML-based language for expressing graphical user interfaces. XUL supports Mozilla applications like Mozilla Firefox and Mozilla Thunderbird. It has also been used to express GUIs for applications such as spreadsheet editors, calculators, calendars and other desktop applications. The main benefit of XUL is that it offers a simple and portable definition of common widgets. In XUL, each widget in the GUI is declared with one or more XUL tags. XUL provides a very simple and easy-to-use way for declaring GUIs. For example, to create a scrollbar, the `<scrollbar>` tag is used. A drawback of XUL is that it can only describe graphical user interfaces and is not scalable to other modalities. This drawback probably excludes XUL from being used for communicating with networked appliances, since it is expected that there will be voice and other such modalities in operation.

3.4.2 XIML (eXtensible Interface Markup Language)

XIML (eXtensible Interface Markup Language) is an XML-based solution for common representation for interaction data [31]. Its initial development took place at the research laboratories of RedWhale Software. XIML is an organised collection of interface elements that are categorized into five basic components: user tasks, domain objects, user types, presentation elements and dialog elements. A relation in XIML is a definition or a statement that links any two or more XIML elements either within the one component or across components. An attribute in XIML is a feature or property of an element that can be assigned a value. XIML is sufficiently expressive to define entire interface definitions for a number of applications. XIML achieves this by making a very strict separation between the definition of the user-interface and the rendering of that interface on the target device. It is possible to render an interface for a PDA, and an interface for a PC from the same XIML representation. XIML also supports the reconfiguration of the layout of a user-interface. This is based on knowledge about the user interface, also captured in the XIML representation. It is also possible to specify personalised information in the XIML representation, for example, disabled users preferences. [32, 33]

At the moment, XIML is probably the most advanced user-interface specification language. However, currently, there is not yet a public tool freely available to transcode XIML representations to target specific representations. Also, several parts of XIML are still under development.

3.4.3 UIML (User-Interface Mark-up Language)

UIML [34] is the most widely used generic user-interface language. It was designed at the Virginia Polytechnical institute. A UIML document is divided into five main sections: structure, style, content, behaviour, style and peers. The first four sections specify the content and behaviour of the user-interface. The last section, the peers section, describes how the components are mapped to widgets and constructs in the target-specific platform. A skeleton of a UIML document is shown in 3.1:

```
<uiml>
  <interface>
        <structure>   ...</structure>
        <style>       ...</style>
        <content>     ...</content>
        <behavior>    ...</behavior>
    </interface>
    <peers>
  </peers>
</uiml>
```

Figure 3.1: Basic structure of a UIML document

- **structure** - The `<structure>` tag contains a list of `<part>` elements describing some abstract part of the user interface.
- **style** - The `<style>` tag contains a list of `<property>` elements, which give presentation properties of the parts, for example, blue background.
- **content** - The `<content>` tag contains the text, images, and other content that belongs to the user-interface. This is especially useful when creating user-interfaces that might be used in multiple languages. For example, separate content would be needed for a user interface that is in both German and English.
- **behaviour** - The `<behaviour>` tag contains a set of rules to define how the User Interface reacts to stimulus. Each rule consists of a `<condition>` tag and an `<action>` tag. The rule fires by executing the `<action>` when the `<condition>` is satisified.
- **peers** - The `<peers>` section maps the generic user-interface description to a concrete toolkit and to concrete method calls. For example, it would be necessary to have a peers section for Java Swing, HTML and VoiceXML. Each of the peers sections would describe how the generic description would be mapped to each target specific platform. The `<peers>` section names a vocabulary file. The vocabulary file contains mappings from the part names given in the UIML document to the target-specific platforms, for example, a JLabel in Java Swing. A vocabulary for describing UIML descriptions for VoiceXML, HTML, WML and Java Swing is

provided by Harmonia Inc [35]. The UI developer uses the platform specific tags or class names as `<part>` names in the `<structure>` element. Harmonia also provides a tool for rendering UIML to these target specific platforms.

UIML offers a fairly sophisticated method of user-interface representation with a high degree of modularisation [36]. The problem with UIML however, is that it is not completely independent from the target- specific platform. Also, the behavioural part of UIML is not well developed and does not give sufficient means to specify real interactive, state-oriented user interfaces [36]. There is currently much research being done in the UIML community to address the problem of UIML being too tightly coupled to the target specific representation.

A generic UIML vocabulary, capable of mapping to and from UIML representations to multiple platforms that share common layout capabilities, is suggested [37]. Platforms that share common layout capabilites are known as a family of devices. For example, HTML and Java Swing would belong to the same family since they share the same layout capabilities. The vocabulary proposed is not sufficient to build user interfaces for widely varying platforms, such as VoiceXML, handhelds and desktops, but it is capable of describing user interfaces that belong to the same family. The objectives of the vocabulary are to be powerful enough to accommodate a family of devices and to be generic enough to be used without having expertise in all the various platforms and toolkits within the family. Currently, one generic vocabulary has been defined, GenericJH, which maps both to Java Swing and HTML 4.0. Generic properties are specified as generic class names, for example, `G:TopContainer` would refer to a `JFrame` in Java Swing and would map to a page heading in HTML. Similarly, a `G:Button` would refer to a `JButton` in Java Swing and a HTML button. Many of the properties available in Java Swing are not be available in HTML and vice versa. To cater for this, it is possible for the developer to fine tune the UI to a specific platform. In the generic vocabulary, these property names are prefixed by a `J:` or `H:` for mapping to Java Swing or HTML only. The generic UIML file would contain three `<style>` elements. One would be cross-platform, one for HTML and one for Java Swing. When the UI is rendered, the renderer will choose exactly one `<style>` element. For example, the Java Swing renderer will only choose the `<style>` for Java Swing. The selected platform specific `<style>` element specifies in its source attribute the name of the common, shared `<style>` element shared by both the HTML and Java Swing style elements. Thus, it is possible for a Java or a HTML renderer to utilize both the generic `<style>` element and the platform specific `<style>` element. [37] Generic vocabularies are useful for specifying interfaces for families of devices but not interfaces across families of devices. This drawback is likely to limit the adoption of UIML generic vocabularies.

More sophisticated solutions suggest a UIML oriented language which mainly describes the dialog model, with only hints to its appearance, in order to provide a maximum degree of generality. The Multi-modal Interaction (MMI) framework defines an architecture that uses such a language to cater for combined audio, speech, handwriting, and keyboard interaction [36]. This language is the Dialog and Interface Specification Language (DISL) and is based on an UIML subset. The salient features of DISL are presented next.

DISL (Dialog and Interface Specification Language)

Since UIML definitions are too much platform-related, DISL attempts to identify the smallest set of UIML user-interface elements for general applications as the smallest denominator over several platforms. The smallest denominator is needed to ensure that a user-interface can be rendered on all limited devices and be used with different modalities. [36] It should be noted though, that DISL is a essentially a different language to UIML and cannot be processed by a UIML processor. DISL incorporates the Object-Oriented Dialog Specification Notation (ODSN) [38] which has been developed to model complex state space for advanced human-computer interaction. ODSN models the user interaction as different objects, which communicate by exchanging events. Each object is described by the definition of hierarchical states, user events, and transition rules. Each rule has a condition and body where the condition may range over sets of states and sets of user events. The body is executed when the specified events occur and the object is in the specified state. This execution of the body may change a state. [39] A serialized form of DISL that allows faster processing and a smaller memory footprint has been designed especially for communication with limited resource mobile devices [39]. DISL processors have to process a relatively complex tree structure, whereas a S-DISL interpreter just has to process a list of elements. A

XSLT transformation is required to flatten the tree structure used in DISL into a S-DISL representation. Tools for processing and manipulating DISL are not yet freely available.

3.4.4 Context-Aware and Synchronized User-Interfaces

For the true benefit of multi-modal communication with networked appliances to be realised, it should be possible to automatically customise the modalities for users based on the user's preferences, abilities, and current situation. A profile based concept for multi-modal interaction in intelligent environments is presented in [40]. Information on the user's preferences/abilities, the device's properties and the current situation is stored in dynamically modifiable profiles. These adaptive profiles can be queried by the system, which enable it to decide on the appropriate devices with the appropriate interaction modalities for an individual user. The best suited device could mean to identify the device within closest range of the user, or the device that the user prefers. Sensors could be used to provide information to the profiles, for example, the user is currently driving. The location profile of that particular user would then be dynamically updated with this information. The user-interface representation is stored as UIML on the server side. When a user requests interaction with an appliance, the server sends the UIML description to a transcoding process. This process checks the location profile of the user, to determine the current situation of that user and sees that the user is currently driving. When an email arrives, and even if the system knows through the personal profile that the user likes to read emails on the dashboard display, the email is not displayed on the display, as the driver's attention needs to be kept on the road. Instead of transcoding the UIML representation to a visual user-interface display, the transcoder employs a rule-set that describes the transcoding from UIML to a speech interface. The user then interacts with the system by issuing spoken commands. Profiles can be used to describe the location and situation of users, the capabilities of devices and user preferences. The selection process for the proper user-interface and modality may require the evaluation of many different profiles and properties at the same time. To deal with the non-crisp values provided by this situation, fuzzy rules are suggested to determine the best modality [40].

Allowing users to seamlessly switch between modalites in the midst of a task would be an attractive feature of communicating with networked appliances. Today, users juggle between mobile phones, laptops and PDAs. A solution is presented in [8], to the problem of synchronizing user interfaces across a range of modalities. The paper proposes a solution to present a unified, synchronized view of information across the various user devices and modalities that access the information. This solution for information access is just as applicable for interaction with networked appliances. The paper reasons that interaction will be based on what users are already familiar with in their everyday interaction with people, where interaction is modeled as a conversation amongst the various participants in the conversation. It should be noted here that the term conversation is used to define more than just speech interaction and is used to encompass all forms of interaction, including graphical interfaces, voice or any of the other possible interaction modes. A conversation is a similar concept to a dialog. The most important concept in the proposed solution is the ability to synchronise interaction history across a variety of user devices. This means that when a switch between modalities occurs in the middle of a task, it is possible for the new modality to seamlessly pick up where the last one left off. For this to happen, user interaction with a specific device needs to be reflected across all available devices. The Model View Controller (MVC) design pattern is adapted to conversational interactions, where a single information source (the model) resides on the server and is viewed via different devices (the view) and also manipulated via different devices (the controller). The resulting Conversational MVC (CMVC) is the key underlying principle of multi-modal conversational interaction. The device that the user is interacting with at any given time is called the *active* device. The other devices that are not presently active are called *listeners*. Each of the available *listeners* needs to be primed to carry on where an *active* device leaves off. For this to occur, all the devices need to have a unified view of the state of a conversation, which is achieved using the CMVC. A single universal information representation, encoded in XML, is used to encapsulate the current state of a conversation. XML is chosen because, as outlined previously, it is an industry standard and it is possible to apply modality-specific transformations to the representation using XSLT or other such methods. User interaction with the currently *active* device is mapped back to this single universal information representation, continually being updated as changes are made to the state of the conversation. This manipulation of the same underlying model by various controllers (i.e. devices) provides a synchonized view for all the devices. Each view can be considered as a transformation of the underlying modality-independent XML representation. These transformations provide a natural mapping amongst the various views, since any portion of the view can be mapped back to underlying modality-independent XML representation. This portion consequently maps back to the corresponding view in a different modality by applying the appropriate transformation rules for that modality. This solution works, as there is always a model of the current conversation/dialog independent of the rendering modality. This acts as a repository of the current dialog state, the dialog flow and the whole conversation history and context. [8]

3.5 Gateway Server

As outlined previously, it is envisioned that the networked home and vehicle of the future will have a wide range of technologies and protocols in operation (e.g. UPnP, Jini, X10 etc) that need to be integrated. It is also envisioned that home and vehicle networks will have a diversified range of services on offer such as, audio/video, monitoring, networked appliance control and many more. The services gateway has evolved to meet these requirements for a platform on to which services can be dynamically delivered and managed. The services gateway also facilites the integration of the disparate technologies in operation within the network. Basically, the services gateway acts as a mediator between the end user and the networked appliances. Obviously, it is desirable for this platform to be standardised in some way so that applications and services are not tied to any particular product. The most prominent standards solution, with seemingly overwhelming industry support, is the Open Services Gateway Initiative (OSGi) Service Platform defined by the OSGi Alliance [1].

3.5.1 OSGi Services Platform

The OSGi Service Platform consists of two pieces: the OSGi framework and a set of standard service definitions. The OSGi framework is a lightweight execution environment for dynamically downloadable services. It provides a component model, a service registry, and support for service deployment. The standard service definitions specifies a number of useful, optional services, such as a logging service, a HTTP service and a Device Access architecture. The framework and the service definitions together form the foundation of a platform for building service-oriented applications for communication with networked appliances . The OSGi framework allows for just-in-time service delivery whereby a service can be downloaded over the network when the service is needed. The service may then be used just once and discarded, or it may be kept persisently on the server. The framework is also capable of updating existing services dynamically from a remote location. This is useful for software used to interact with networked appliances, since it is likely that new versions of the software will be frequent. The framework also provides a service and discovery mechanism with which a component or service in the framework can consult a service directory to obtain and use a service within the framework. The service directory, or registry, contains service descriptions published by service providers. The service registry allows service requesters to discover and bind published services. This service-orientation is a key feature of the OSGi framework allowing for a very dynamic networked environment in which a wide variety of services may be offered. OSGi technology plays a complementary role to other technologies operating with the home or vehicle, such as Jini, UPnP and HAVi. [41, 42]

OSGi Framework

The OSGi framework is defined to run on top of the JVM (Java Virtual Machine). Together, the operating system and the JVM make up the foundation for the execution environment for the gateway. The advantage of specifying Java include platform independence, the ability to dynamically load code, a rich and extensible object-oriented programming language and a built-in-security architecture [43]. The framework provides the execution environment for the services. Within the framework, a service is represented as a Java interface. The interface says what the service does and not how it does it. This separation of interface and implementation ensures that the service interface to the users of the service remains the same, while it is possible to have many implementations for the same service. It is also possible to change the implementation of a service while the interface seen by the users remains the same. Service implementations are delivered and deployed to the framework in a packaging form known as a bundle. A diagrammatic representation of the OSGi framework is illustrated in Figure 3.2.

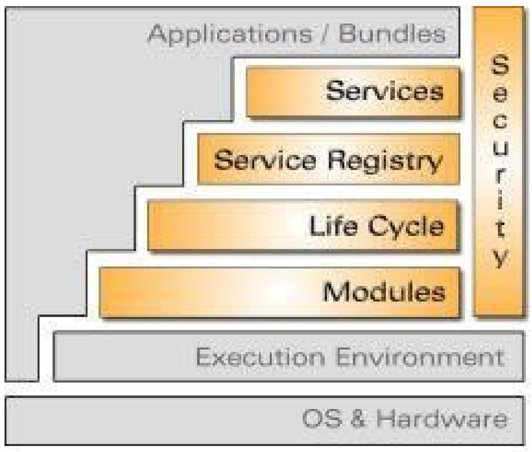

Figure 3.2: OSGi Framework taken from [1]

A bundle is a functional unit with life cycle operations and class loading capabilities. More specifically, a bundle corresponds to a JAR (Java Archive) file that contains class files, resources and a deployment manifest. The manifest is a standard entry in a JAR file and includes meta-information about the JAR itself. An optional set of manifest headers have been defined in the OSGi Service Gateway Specification. These headers inform the framework of the external Java packages that the bundle depends on, and any packages that the bundle is willing to share with other bundles. One other piece of fundamental information contained in the deployment manifest is the name of the bundle's activator class, which is used to perform customised operations at the time when the bundle is started and stopped [44]. A bundle is self-contained with its own class loader. This means that code within one bundle cannot refer to classes inside another bundle, unless of course, the bundle has made its code available by exporting a package in the manifest. This is how bundles achieve insulation from one another. A bundle has one of six states on the framework during its lifetime, INSTALLED, RESOLVED, STARTING, ACTIVE, STOPPING, and UNINSTALLED. This is illustrated in Figure 3.3. Once a bundle has been installed, the framework must resolve a bundle before it can be started. This involves the framework checking the deployment manifest of the framework for external packages that the bundle depends on. If the dependencies exist then the bundle is resolved and ready to be started. When a bundle is instructed to start, it implicitly moves into the STARTING state temporarily, for a brief period, while the bundle is starting up. If the activator class is successful in starting up the bundle, the bundle moves into the ACTIVE state, and the bundle is started, ready for its services to be used by other bundles within the framework. When a bundle is to be stopped, the framework takes up the responsiblity of tidying up after the bundle. The framework unregisters and releases any services provided by the bundle, notifies interested listeners that the bundle is being stopped and moves the bundle back into the RESOLVED state. The framework can generate three kinds of events that interested listeners can receive event notifications for; a `ServiceEvent` reports the registration, unregistration, or changes for services, a `BundleEvent` reports changes in a bundle's life cycle and a `FrameworkEvent` reports that the framework has encountered errors. [44, 43]

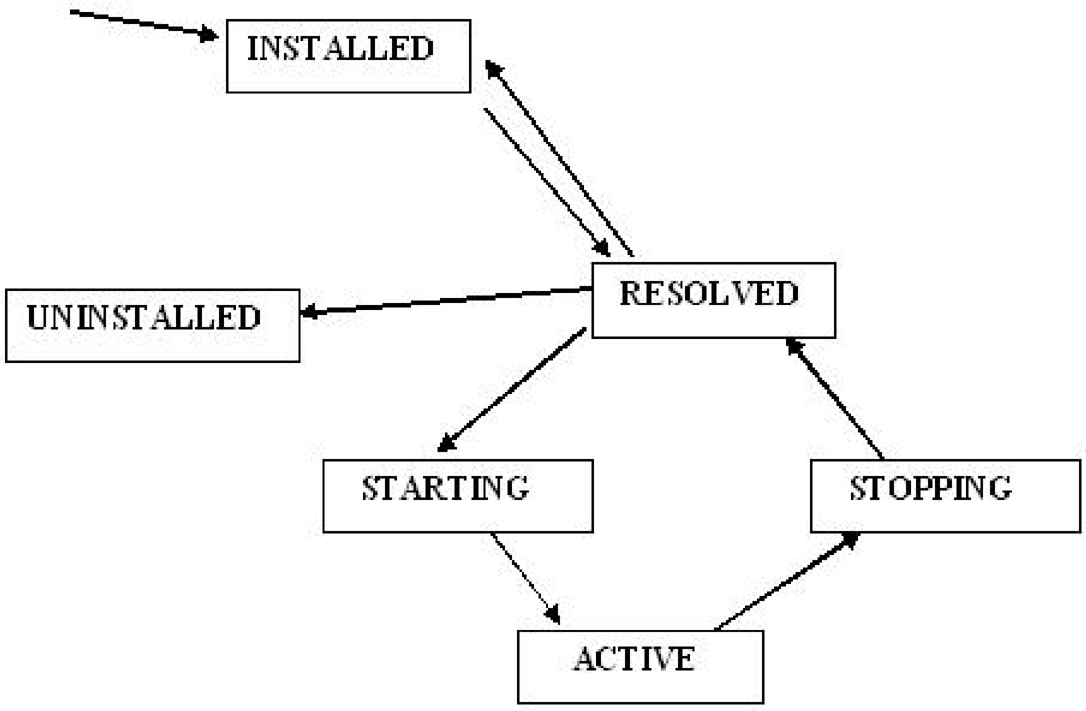

Figure 3.3: Lifecyle of a bundle

Bundles can make services available by registering, or publishing, the services with the service registry. This registration is done by the activator class of the bundle when the bundle is started up. As part of registering a service, the registering bundle may attach a set of properties in the form of attribute-value pairs to the service. These properties can be used to distinguish between multiple providers of the same service. When another bundle is looking for a service, it uses the service interface name and an optional Lightweight Directory Access Protocol (LDAP) query selection filter over the service properties. When a bundle is deactivated the activator class unregisters the services that it provides, and it releases the services of other bundles, that it is using. It is therefore necessary for bundles that use services to monitor the availability of those services. When a service departs the framework, as a result of a bundle being deactivated, the bundle using that service must release references to the departing service and take any necessary corrective actions, such as finding an alternative provider for the same service. [44, 43, 45]

OSGi Services

A number of basic OSGi services have been defined by the OSGi Alliance. The original focus of the OSGi specifications was on services gateways but it has found uses beyond this. For example, the Eclipse IDE uses OSGi as the underlying runtime for their plugin architecture. OSGi is still mostly used however, as a services gateway for homes and vehicles. Specifications for basic services such as logging and HTTP are provided by the OSGi Alliance. OSGi Expert Groups are responsible for defining the specifications for other specific services that may be offered. An Expert Group usually consists of a group of organisations or companies. For example, BMW are part of the Expert Group defining the specifications for services for vehicular networks. One of the more important specification as regards communication with networked appliances is the Device Access Specification (DAS), defined by the Device Expert Group (DEG). DAS allows devices to be discovered, and their services advertised by the framework so that they can be made available to other devices and services.

DAS works as follows [44]: It is detected, with the aid of base (operating system) drivers that a device is connected, and a **device service** is registered to represent the functionality of the device. For example, if a web cam has just connected to the USB port, a `USBService` would be registered. Once a device service has been registered, a device refinement process takes place to find a better abstraction for the new device service. Continuing the web cam example, the **device manager** within the OSGi framework asks the existing device **driver services** to refine (provide a better abstraction for) the web cam connected to the USB port. **Driver services** understand the specifics of certain types of devices and if better abstractions are available. If a **driver service** knows about web cams connected to the parallel port, then it will respond positively to be considered by the **device manager**. At the same time that the **device manager** is asking the driver services for better abstractions, it also asks **Driver Locator** services to find and download new driver services, from an external location, to participate in the device refinement process. Once all the potential driver services have been found, the **device manager** queries them to determine which one can refine the device the best. Simple matching criteria is used to determine this. The **device manager** selects the best driver service to refine the device, and instructs that driver to register its device services as a refinement for the device. The process then repeats itself for the new **device service**, until there are no further refinements. What the Device Access Specification aims to achieve is to map the physical devices, which are dynamic and flexible, to services in the OSGi framework, which are equally dynamic and flexible.

The Device Expert Group (DEG) has also defined a number of services that map an external protocol to an OSGi service. DEG has recently defined a UPnP service that can map UPnP devices as services to the OSGi framework and can map OSGi services to the UPnP network. DEG has also recently defined a Jini specification that enables access to Jini services and devices from within the OSGi framework and also allows members of a Jini federation to use OSGi services. Basically, these specifications work on an import/export model. Services registered within the OSGi framework are exported out of the framework to be used by different device protocols. Similarly, devices and services provided by specific device protocols are imported into the OSGi framework, to be used by OSGi entities.

The OSGi Forum Core Platform Expert Group (CPEG) is currently working on defining a SIP service for communication with networked appliances. This service will be based on the extended version of SIP for communicating with networked appliances presented earlier. Recent research has been done [46] on how this new OSGi SIP service specification could be used. The obvious scenario is wide-area communication. A less obvious scenario, also outlined in [46], is to allow OSGi device and service application-layer mobility using SIP. An OSGi service can be exported as a SIP proxy service. This service then gains the mobility feature of SIP, and can then move and register with the SIP service of another SIP-enabled OSGi gateway while maintaining its SIP service identity. This makes OSGi services importable into other OSGi frameworks.

3.5.2 OSGi Research Efforts

A significant amount of research effort has been put into investigating OSGi as a solution for communicating with networked appliances in a heterogeneous environment [47, 48]. Most solutions present a similar solution by using the Device Access Specification (DAS) import/export model to achieve heterogenity. There has been very little research done on the more interesting applicaton of OSGi to achieve multi-modality or context-awareness. The limited amount of available research is discussed below.

OSGi and Context-Awareness

Most solutions for adding context-aware functionality to OSGi compliant-gateways propose the use of some type of context-inference engine deployed as an OSGi bundle. The context inference engine could be used to customise the user-interface according to the user's preference and situation [49]. This customisation information would then be passed on to a user-interface contructor that builds the interface based on this information. The context inference engine would contain a series of rules that are applied to the user-interface. It gathers information from the environment based on input from sensors embedded in the environment (current location, noise levels etc). Information can also be gathered from user profiles containing preferences, abilities and other such information. The use of video surveillance to perform object tracking and human behaviour analysis is also suggested as a method of gathering information about the environment [50]. The video surveillance service transforms the raw video data into high-level descriptions of scenes and events that characterise a specific scenario. Audio analysis could also be used to detect abnormal audio events from raw audio signals. In [51], a solution is presented for realising context-aware automotive services. An ontology-based context model is employed throughout the entire process of sensing, interpreting, managing and exchanging context information. The Web Onotology Language (OWL) is used as the modelling language to implement the context model. Context-aware services then trigger actions when a specific event happens using basic **IF-THEN** rules. For example, if a phone call comes in and the driver is driving at high speed, then the context-aware service puts the call straight through to voice mail (in order not to distract the attention of the driver).

3.6 State-of-the-Art Conclusion

Clearly there are a broad range of technologies to choose from when designing an architecture for multi-modal communication with networked appliances. Many of these technologies compliment one another rather than compete with one another. It is clear that no one particular user interface technology, transport protocol, or discovery and communication protocol is likely to win out as the exclusive solution for communicating with networked appliances. Thus, it is neccessary for a potential solution be as flexible and dynamic as possible. A discussion of the technology choices, based on the state-of-the-art just presented, takes place in the next chapter. This chapter presents a proposed architectural solution for communicating multi-modally with networked appliances in a heterogeneous environment and is based on the most appropriate state-of-the-art technologies chosen from the technologies presented in this chapter.

Chapter 4

Proposed Architecture

Based on an analysis of the requirements of the project and the state-of-the-art, an architecture for communicating multi-modally with networked appliances in a heterogeneous environment is now proposed. The architecture is designed to meet the requirements as closely as possible and is based on available state-of-the-art technology. The architecture is specifically designed to be as flexible as possible since it is realised that home and vehicle networks are dynamic environments and are likely to become more dynamic in the future with innovations in the services offered. The proposed architecture will be described under four headings. Firstly, the general physical layout of the components will be presented. Secondly, the technologies and protocols that have been decided as most appropriate for meeting the design issues will be justifed and explained. Thirdly, a detailed software architecture and design will be presented. Finally, considering the extensibility of the proposed architecture, possible additional components to the architecture will be discussed.

4.1 Physical placement of components

As discussed in the specification, there are three entities in the system capable of performing actions; a local client user, a remote client user in an external network (e.g. the Internet) and the gateway server. The client users access the networked appliances through the gateway server. The user device could be a laptop, PDA, mobile phone etc. The concept of external networks interacting with the internal network through the residential gateway is illustrated in Figure 4.1. There are numerous different wired and wireless networks and protocols in operation within the internal network. It is the responsibility of the gateway server to federate these technologies.

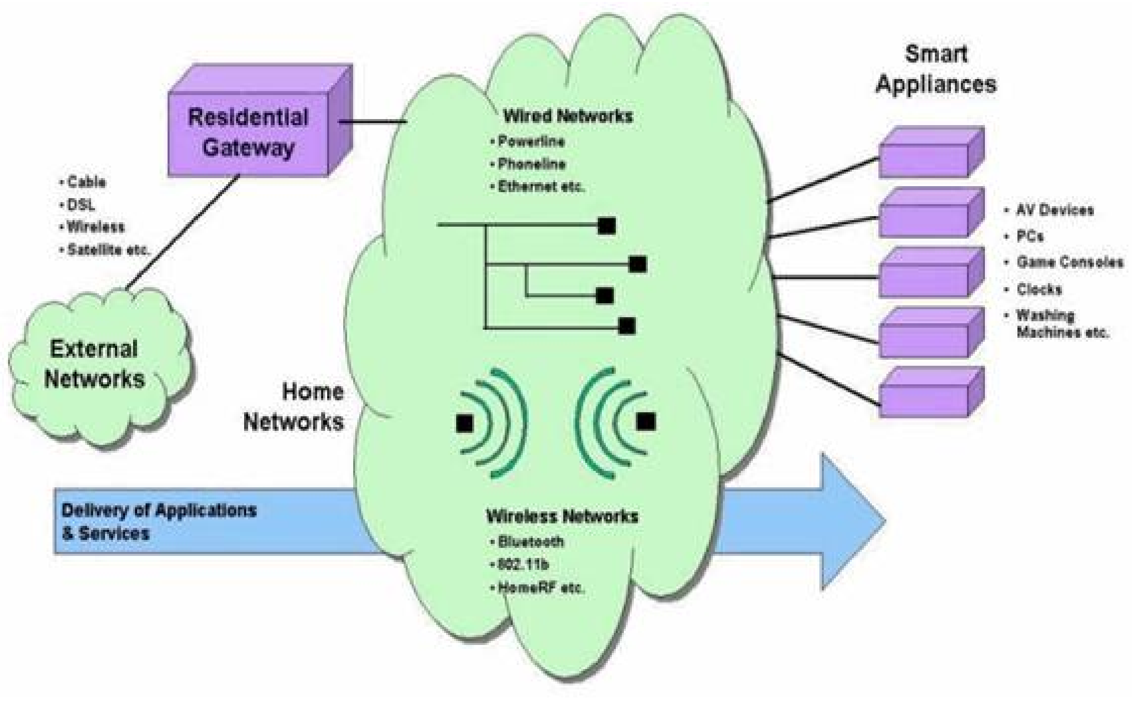

Figure 4.1: External networks interacting with internal networks (image taken from http://www.ida.gov.sg)

4.2 Technologies and Protocols

The previous chapter presented the state-of-the-art technologies currently available for communicating with networked appliances. In this chapter, the state-of-the-art technologies deemed most appropriate for communicating multi-modally with networked appliances, both locally and remotely, will be chosen. The technology choices that need to be made are: the protocol used for wide-area communication, the deployment of the gateway server, the hardware and transport protocols, the discovery and communication protocols, and how multi-modal communication is to be realised.

4.2.1 Protocol used for Wide-Area Communication

There are a few existing wide-area protocols that could be considered for communication with networked appppliances [22]. SMTP (Simple Mail Transfer Protocol) seems a likely candidate on first inspection, however, it does not support events and media sessions and can sometimes exhibit very high latency. Since, a networked appliance could be considered a managed object, SNMP (Simple Network Management Protocol) seems like a good candidate. SNMP also supports events and notifications which is a big advantage for communicating with networked appliances. The problem with SNMP however is that it only supports network-layer addressing, and networked appliances need to support application layer addressing. Furthermore, SNMP does not support multimedia sessions. [22]

Since there is no wide-area protocol currently available specifically designed for communicating with networked appliances, there may be motivation for designing a completely new protocol from scratch. The new protocol would be designed exclusively for communication with networked appliances. This protocol would be unlikely to resemble any wide-area protocol currently in existence [22]. There are a number of reasons why designing a new protocol from scratch is not appropriate. Firstly, a new protocol would be unable to reuse any existing infrastructure. Secondly, introducing yet another protocol may add to the confusion generated by the myriad of protocols and technologies currently in operation for in-domain communication with networked appliances. Thirdly, a new protocol would require widespread adoption to justify the development. This can be difficult to achieve since it may require organisations and users to set up a new infrastructure or change an existing infrastructure. Lastly, if a new protocol was designed it would probably look different from the extended version of SIP, which was modified for communicating with networked appliances, but it would still be likely to share many of the same characteristics of the extended SIP [22]. The benefits of being able to reuse the existing SIP infrastructure for communicating with networked appliances, provides sufficient motivation for not creating yet another protocol.

The extended version of SIP offers an ideal solution to communicating with networked appliances. There are reasons, however, why HTTP might still be considered for communicating with networked appliances. Many users are comfortable with web browser technology, so, it makes sense for users to be able to interact with their networked appliances through a browser. Despite the fact that HTTP has no support for notifications, it could still be useful for checking the status of networked appliances by sending HTTP `GET` requests and sending simple commands to networked appliances, by sending HTTP `PUT` requests. The request can be processed by servlets or CGI (Common Gateway Interface) scripts on the gateway to be passed on to the networked appliance. This would suffice for many types of users and would be no more complicated for them than surfing the web. Despite user-interfaces becoming more user-friendly and easy to learn, some users are still reluctant to change from what they know best. This is one of the main reasons why HTTP should be chosen to be a part of the solution to any networked appliance architecture.

The extended version of SIP should also be chosen as part of the solution to the problem of communicating remotely with networked appliances. It is the only wide-area protocol currently available that is specifically designed or in this case, modified, for communicating with networked appliances. This is enough justification for its selection. It was designed specifically with the requirements for communicating with networked appliances in mind. DMP (Device Messaging Protocol) has been specifically designed as the payload of the extended SIP [2], so this should be availed of to carry the body of the SIP messages. This data format captures all the requirements for communicating with networked appliances. Protocol bridges are required between DMP and the protocol or API used by the gateway server.

4.2.2 Deployment of the Gateway Server

Since user devices communicating with networked appliances are unlikely to be aware of every protocol used by the appliances within their range, it is necessary to have a gateway server to federate the disparate technologies. The gateway server also accepts wide-area requests from remote users and provides user-interfaces for the user devices. The choice of gateway server is an important decision, since all of the communication between user devices and networked appliances is done through the gateway server.

The gateway server needs to be standards based and capable of dynamically changing the state of services (installing, stopping, starting etc.) without requiring a reboot of the server. A proprietary solution is undesirable since this locks applications and services into specific platforms. Application servers such as J2EE and .NET could be considered. The problem with using these platforms in the home or vehicle is that they have high resource consumption and have limited support for dynamic deployment and dependency management. Applications servers are much more suited to corporate settings. The only standardised services gateway currently available is the OSGi platform. The OSGi platform is probably the most rational choice for a services gateway within a home or vehicle. OSGi-compliant service gateways support dynamic component deployment, component dependency management, and component lifetime management. The only major drawback in using an OSGi solution is that a JVM (Java Virtual Machine) must be present on the gateway server which increases the footprint required to run applications on the gateway server. A possible solution to this might be, for the Java community, to specify a small footprint JVM for use on service gateways. This would require the backing of the Java standardization body which would probably take a lot of time. The advantages of using Java far outweigh its disadvantages however. Java is platform independent and is a rich and extensible, widely used, object-oriented language. It is capable of dynamically downloading code, and has an extensive security architecture that can guard against the downloaded code.

The OSGi framework would most likely run on a home PC, a set-top box or other such device. It is also possible for the framework to run on a user's mobile device, such as, a PDA. Thus, it would be possible for the services gateway to move across domains with the user. This is desirable for situations where a user is authorised to use appliances across a number of domains and a gateway server may not be available on a desktop computer or set-top box within the domain.

4.2.3 Hardware and Transport Protocols

The most convenient way of communicating in a networked appliance environment is wirelessly. This is also the most likely way that users will interact with networked appliances in the home and vehicle of the future. IEEE 802.11, Bluetooth, the IEEE 802.15 standards and other wireless hardware and transport protocols will complement one another within the environment. At the same time, there may be a need to communicate through a wired link, for example, if a higher bit-rate is required over a longer distance. Thus, any solution to communicating with networked appliances will need to accomodate a wide range of hardware and transport protocols.

4.2.4 Discovery and Communication Protocols

It is clear, for the meantime anyway, that no one particular discovery and communication protocol is going to be used for communication with networked appliances. Like the hardware and transport protocols, there is going to be a broad range of discovery and communication technologies in operation within the environment. Of all the service and discovery protocols available, UPnP seems to be one of the more attractive options. UPnP is based on industry standards, and because its description is in an XML file, devices are able to provide a rich description of the capabilities they offer. In contrast, Jini has a simple interface API description and requires devices to have a JVM. Also, Jini is not meant directly for networked appliances, whereas UPnP is. But again, each of the discovery and protocols were designed with different audiences and purposes in mind, so it is likely that protocols such as UPnP, Jini, HAVi, Salutation, SLP and others are going to be deployed together to complement one another within the domain. Since the OSGi framework has been decided as method of deployment for the services gateway, the discovery and communication protocols used by the networked appliances need to be translated to OSGi method calls. As discussed in the previous chapter, DAS (Device Access Specification), as defined by the DEG (Device Expert Group) provides a solution specifically for this. DAS also allows for the dynamic discovery of drivers at runtime, which allows for even more flexibility. The DEG is working on specifing design patterns for OSGi discovery APIs to form a template that can enable rapid development of additional discovery APIs. For example, as explained previously, it is possible to import an UPnP or Jini service into the OSGi framework so that it appears as a valid OSGi entity and makes the service fully accessible by other OSGi entities. Similarly, it is possible to export registered OSGi services so that they become discoverable using native discovery techniques. For example, a Jini service could be discovered by an UPnP control point through the OSGi framework. The OSGi Device Access Specification is used in this architecture to enable heterogeneous appliance discovery, registration and communication.

4.2.5 Multi-Modal Interaction

There is likely to be at least a half a dozen user devices operating wihtin a domain at any one time. Users may have PDAs, mobile phones, wearable computers, laptops and desktop computers. As innovations are made, more and more interaction modes will become available. Interaction with numerous different devices by the same user will become part of the user's daily routine. Interaction modes may be personalised according to the users situation and location, to provide a more intelligent environment for the user. In all these situations, the user-interface for interacting with a particular applicance operating within the domain will not be known in advance by the user device. Also, the characteristics and features of the user device will not be known by the gateway server. The gateway server should be able to provide user-interfaces for all the networked appliances that it has access to. It can do this by either storing a static representation of the user-interface on the server or by generating the user-interface when it is required. The ability to generate user-interfaces at run-time is particularly attractive for situation-dependent interfaces. Obviously, because of this diversity, a solution to multi-modal communication with networked appliances needs to be based on open standards. An ideal solution, as shown in the previous chapter, would be to provide a generic description which allows for transformation to any possible target specific user-interface. An XML-based representation would be the most obvious, since it has gained widespread acceptance and is relatively easy to process. Having the developer provide his or her own XML based generic description and providing XSLT stylesheets to transform those representations would be inadequate. This would require other developers to adhere to that developers standards every time they wanted to deploy an additional networked appliance user-interface on the gateway server. Instead, it is necessary for the generic user-interface language used to be standardised as much as possible. XIML is the most advanced generic user-interface language currently available but is still under development. Similarly, DISL offers a method of specifying user-interfaces at a high level of abstraction, but again it is still under development and tools are not freely available.

Currently, UIML is the only practical choice as a generic user-interface language. It is well-specified and tools are freely available to manipulate UIML representation. Its major limitation is the fact that it is still too tightly coupled to the target-specific representation. Harmonia and uiml.org have developed different component vocabularies for a number of target user-interface platforms (Java Swing, WML, VoiceXML, HTML) containing target language specific information [7]. However this approach only works when the target languages are known in advance. In the case of future home environments, it is virtually impossible to know what the user-interfaces for future user devices might be. As shown, there is much research being done at the moment by the UIML community to solve this problem. Solutions suggest to divide the styles of interaction into parallel interaction (such as GUIs) and serial interaction (speech dialogs) and provide separate support for both styles of interaction [7]. This solution is similar to providing support for separate families of devices, which was outlined earlier. The proposals are not really adequate for communicating multi-modally with networked appliances since it is highly likely that parallel interaction and serial interaction will both be in operation together.

Thus, the current UIML specification does not completely meet the requirements for communication with networked appliances. The most ideal solution in the pipeline is probably XIML, but until it is further developed and tools are freely available, current solutions will have to make do with UIML. Once XIML has been fully developed, converting from an UIML solution to an XIML solution should not require too much effort since XIML is more generic than UIML and would require less descriptions to achieve the same functionality. The UIML transcoders would also need to be replaced with XIML transcoders. Harmonia Inc. defines vocabularies for HTML, Java Swing, WML and VoiceXML and provides tools for transcoding these UIML representations. These tools are used in the architecture of this system.

4.3 Software Architecture and Design

A high-level architectural solution using the chosen technologies is illustrated in Figure 4.2. The architecture, as shown, consists of seven components: `SIP Service`, `HTTP Service` for wide-area interaction, `User Device Adapters` providing access points to various types of user devices within the domain, `User Device Registry` for authenticating users, `Generic User Interface Manager` for providing the generic user-interface description, `Appliance Registry` for providing access to networked appliances and `Appliance Drivers` for communicating with networked appliances in their native protocol. It is designed to be implemented on an OSGi-compliant services gateway and to meet the requirements outlined for communicating with networked appliances as much as possible. Each component in the diagram is a bundle within the OSGi framework. That is, each component is a self-contained JAR file that can be deployed at run-time. The bundles rely on services provided by other bundles within the framework. The OSGi framework automatically manages the dependencies between these bundles. Services are provided by the bundles in the form of interface APIs.

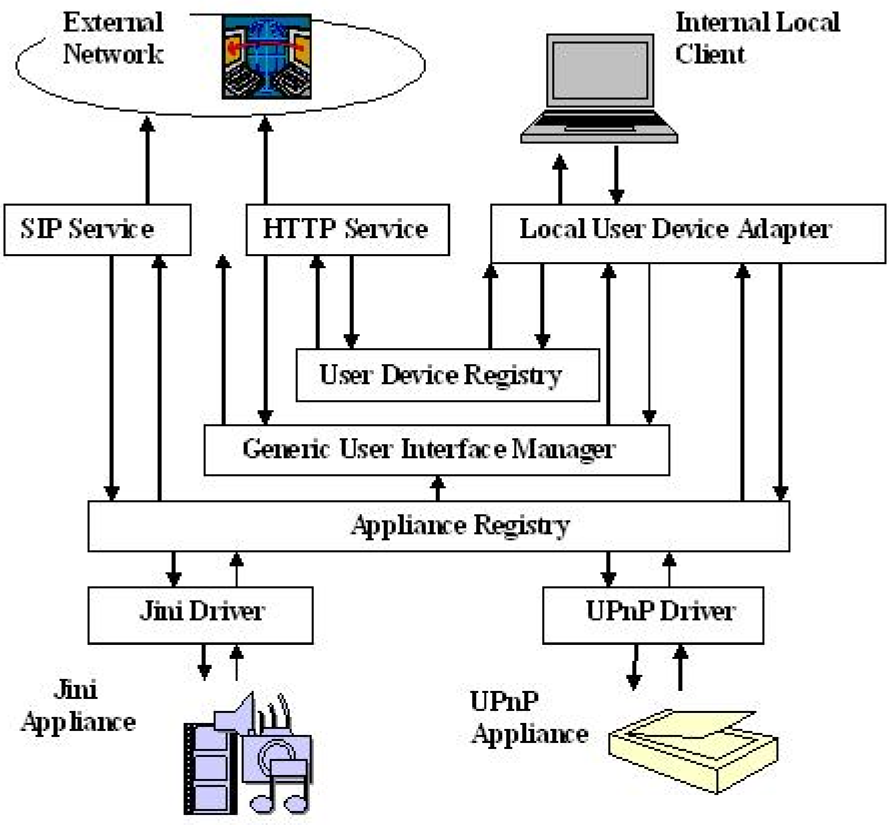

Figure 4.2: Proposed Architecture

4.3.1 `SIP Service`

The choice of a `SIP Service` in the architecture is based on the assumption that the extended version of SIP will gain acceptance as the method for communicating remotely with networked appliances. The OSGi Forum Core Platform Expert Group (CPEG) is currently working on the details of the SIP Service API for services gateway. This work is expected to be ratified by the OSGi Alliance in the near future. The `SIP Service` component accepts SIP messages from remote users and translates the payload of the SIP message from the DMP format into OSGi method invocations. It also sends messages, generated within the OSGi framework back to SIP enabled user devices. It is envisioned that most SIP user devices will be remote devices rather than local devices, but `SIP Service` operates equally well in remote or local conditions. `SIP Service` depends on the `User Device Registry` for authenticating users, the `Generic User Interface Manager` for providing the generic user-interface descriptions and the `Appliance Registry` for interacting with the networked appliances. The `SIP Service` transcodes the generic user-interface format into the target-specific format or returns the generic user interface description to be transcoded locally by the client.

4.3.2 `HTTP Service`

The OSGi Alliance has defined a `HTTP service` that can act as a lightweight HTTP server. Having a HTTP server on the gateway allows users to interact with the networked appliances through one of the most universal user-interfaces, a web browser. The `HTTP service` is among other things, a servlet runner. Bundles can provide servlets on the gateway which can be made available through the HTTP service. The dynamic update facility of the OSGi service platform makes the `HTTP Service` a very attractive web server that can be updated with new servlets and resources without requiring a restart. Resources such as images, static HTML pages and other files can also be made available through the `HTTP Service`. Servlets are capable of generating dynamic content and invoking methods within the framework. A command from a web browser to control a networked appliance is sent to a servlet registered with the `HTTP Service`. This invokes the `doPost()` method of the servlet. Within the body of the `doPost()` method, the servlet invokes a method in the `Appliance Registry` component to send the command to the networked appliance. This component, similar to

SIP Service depends on the User Device Registry, the Generic User Interface Manager and the Appliance Registry components.

4.3.3 User Device Adapters

The User Device Adapter bundles act as access points for the user devices operating within the domain that the networked appliances operate. Multiple User Device Adapters are made available for the different types of user devices and protocols that are in operation within the network. The OSGi framework is ideal for this since user device adapters can be downloaded from the Internet and deployed dynamically without requiring a restart of the server. The adapters translate the protocol specific messages sent from the user devices into OSGi method calls to be invoked in the other components. As the User Device Adapters act as access points, similar to the SIP Service and the HTTP Service, they also depend on the User Device Registry, the Generic User Interface Manager and the Appliance Registry.

4.3.4 User Device Registry

The component could be considered the access control point of the system, as it provides both authentication and authorisation of both internal and external users. Authentication is required every time that a user tries to connect to the domain from outside. Also, authentication is required for internal users, since many of the user devices will be attempting to connect wirelessly to the services gateway. Only users accepted by the User Device Registry can interact with the networked appliances. Once accepted, the User Device Registry allows the SIP Service, HTTP Service and the User Device Adapters to set up sessions with the user devices.

4.3.5 Generic User Interface Manager

The Generic User Interface Manager provides generic user-interface descriptions for all the networked appliances available within the domain. The generic user-interface descriptions can either be stored statically on the server, generated dynamically, or retrieved from an external location such as a remote repository maintained by the manufacturer of the networked appliance [7]. When the generic user-interface description is fetched by either the `SIP Service`, `HTTP Service` or the `Local User Device Adapters`, it is either transcoded by these components, or it is returned to the client device still encoded in the generic description to be transcoded locally by the client. The client caches the user-interface representation so it is not necessary to download the user-interface when it repeatedly interacts with the same appliance. In this architecture, UIML will be used as the generic user-interface description format, and UIML transcoders from Harmonia Inc. will be used to transcode the user-interfaces into target-specific representations. XIML or DISL representations and transcoders could be added fairly easily when they become available. The `Generic User Interface Manager` may also take responsiblity for updating versions of the user-interface representations. This could be done either by regularly checking the appliance manufacturer's user-interface remote repository, or by a human providing a new user-interface description to the server.

4.3.6 Appliance Registry

The `Appliance Registry` has knowledge of every registered appliance and service within the domain. It stores the names and addresses of the networked appliances, and provides an interface between the appliance drivers and the other components in the architecture. The `SIP Service` component, the `HTTP Service` component and the `User Device Adapter` components interact with networked appliances through the `Appliance Registry`. The `Appliance Registry` also allows user devices to subscribe to events fired by networked appliances. `Appliance Registry` relies on the `Appliance Drivers` to interact with the networked appliances. The `Appliance Registry` obtains a reference to every active `Appliance Driver` in the framework.

4.3.7 Appliance Drivers

The `Appliance Drivers` bundles translate between the appliance-specific protocol used by the networked appliances and the OSGi method calls. Different appliances need different drivers bundles. For example, UPnP appliances would need a UPnP driver bundle, and Jini appliances would need a Jini driver bundle. These bundles adhere to the DAS (Device Access Specification) as defined by the Device Expert Group (DEG).

4.4 Sequence Diagrams

To demonstrate the interactions occurring within the architecture, two sequence diagrams are presented. Figure 4.3 shows a SIP-enabled client connecting to the `SIP Service`, being authenticated by the `User Device Registry`, and the `Generic User Interface Manager` providing the generic user interface to the `SIP Service` to be transcoded and returned to the client. A SIP-enabled client subscribing to and receiving an UPnP event is also shown in Figure 4.3. Note that the **DO**, **SUBSCRIBE**, and **NOTIFY** messages of the extended SIP are shown. The payload of these messages is the DMP (Device Message Protocol) proposed as the body of the new SIP messages. A local user sending a command to a Jini appliance is also shown in Figure 4.4. The messages sent between the local user devices and the `Local User Device Adapter` are in the format used by the local user device. It can be seen from the diagrams that the method calls within the architecture are the same regardless of the type of user that is connected.

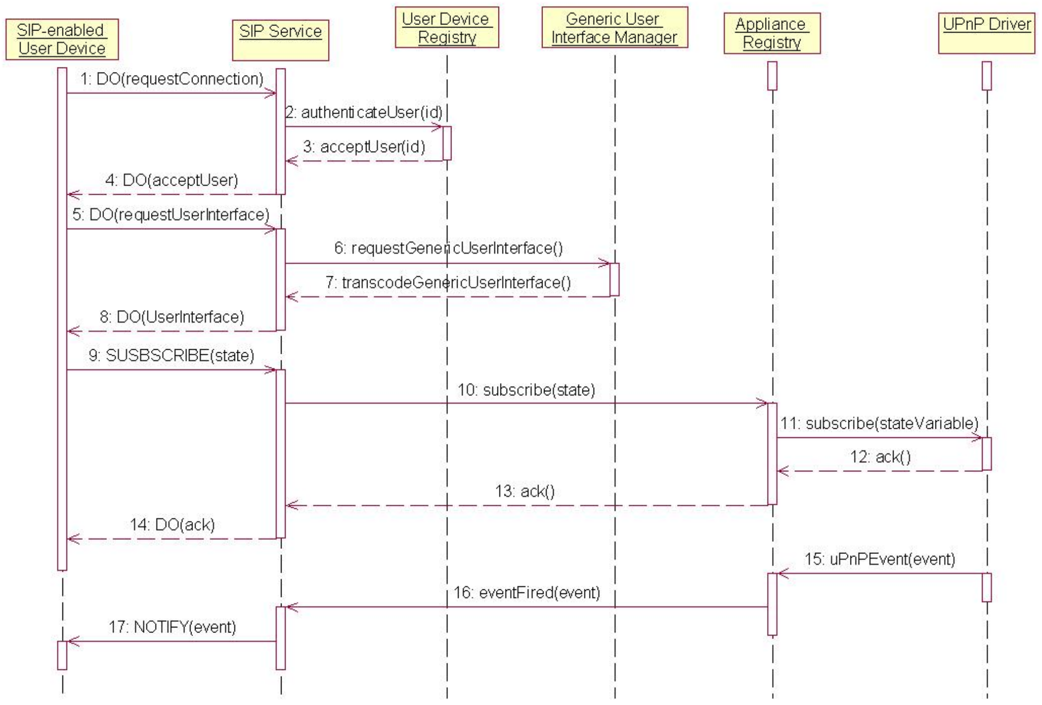

Figure 4.3: A SIP user subscribing to an UPnP state variable and being notified of an UPnP event.

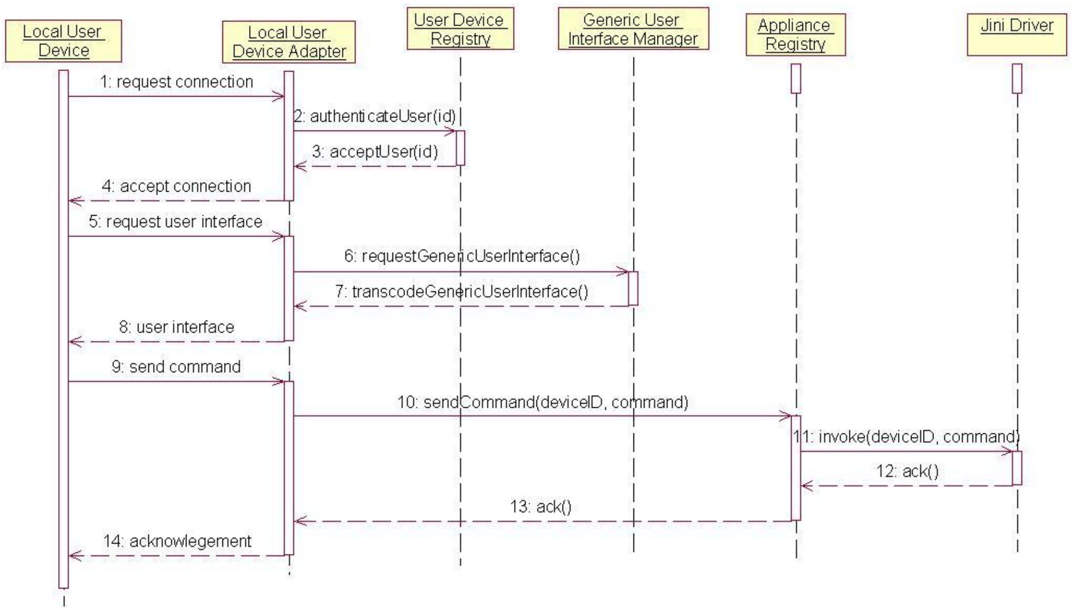

Figure 4.4: A local user sending a command to a networked appliance.

4.5 Additional Functionality

The appliance interaction architecture presented and the technologies it is based on are designed to be as dynamic and flexible as possible. This facilitates the deployment of additional components and functionality without interfering with existing components. One of the more obvious additional features to the architecture would be a context-awareness feature to provide for a more intelligent environment. Context-awareness is especially useful in a rapidly changing environment. Context-aware functionality could be added to an OSGi framework by providing a context sensing bundle to determine the location, situation etc. of the user and a context manager bundle to process this information. The context sensing bundle could sense the context of the user through sensors or using the video and audio analysis technique presented by [50]. A Profiles Manager bundle could also be added, storing user preferences, capabilities and other such information. Both the `Context Manager` and the `Profiles Manager` could be utilised by the `SIP Service`, `HTTP Service` and the `Local User Device Adapters` to customise the modality and the information that is sent to the user. The architecture with a new `Context Sensors` bundle and a `Context Manager` bundle is shown in Figure 4.5.

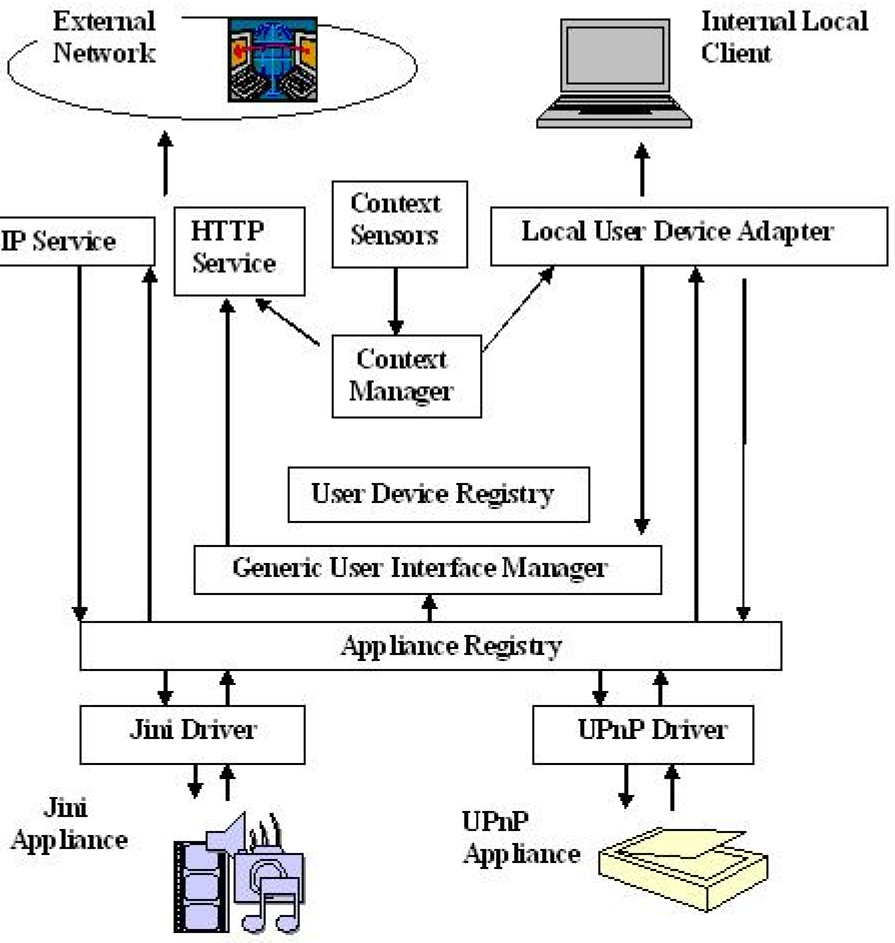

Figure 4.5: Extended architecture with context awareness functionality.

Another additional component that could be added to the system would be a bundle that manages the downloading of new `User Device Adapters`. This would be similar in concept to the device manager and driver locator bundles in the Device Access Specification. For example, if a new user device tries to connect to the services gateway and there is not a `Local User Device Adapter` available to translate the device specific format into OSGi method calls, the `User Device Adapter Locator` would start off a process to find a user device adapter from a remote location.

In actuality, because of the flexibility and extensibility of the OSGi framework, there is no limit to the number of functions and features that can be added to the architecture. This is probably one of the most attractive features; both of the architecture and the technologies used to realise the architecture.

Chapter 5

Implementation and Evaluation

A proof-of-concept implementation of the architecture presented in the previous chapter was developed to analyse the issues that may arise from such an implementation. The implementation was built using the currently available state-of-the-art technologies and standards and reusing as many existing software components as possible. Since the architecture is an OSGi based solution designed to be deployed on an OSGi-compliant server, a decision needed to be made on which OSGi implementation to use. This choice is not really an issue since all the OSGi implementations are based on the same standard. Bundles developed and tested on one implementation should operate the exact same on other implementations. There are two commonly used, freely available OSGi implementations; the Oscar and Knopflerfish OSGi frameworks. Oscar provides a bundle repository (the Oscar Bundle Repository or OBR) which acts as a repository for OSGi bundles [52]. This enables reuse of existing bundles and services. It is also possible to dynamically deploy OBR bundles into an executing OSGi framework. For example, the DAS `device manager` can instruct `driver locators` to look in the OBR for appliance drivers at run-time. The OBR provides a substantial number of services, including HTTP, Logging and recently a UPnP base driver. The bundles in the OBR can be deployed successfully to any OSGi-compliant services gateway and not just Oscar. Based on preliminary experiments carried out before the implementation, it was decided that Oscar is stable, has a relatively easy-to-use command-line interface, is fully compliant with the OSGi specification, and would be used during the implementation stage of the project. The final implementation was tested on both the Oscar and

Knopflerfish OSGi frameworks, giving the exact same behaviour on both.

5.1 Scenario

To prove the intended functionality of the architecture, it was necessary to demonstrate an implementation that allows multi-device access to networked appliances based on different communication technologies. The scenario devised was to allow a web browser client and a Java Swing client to communicate with networked appliances based on different technologies. The web browser and Java Swing client can interact with two simulated UPnP appliances, a clock and an air conditioning unit. A service to grab an image from an USB Web Cam was also provided. A UPnP subscription and notification mechanism was also implemented; allowing the Java client to subscribe to changes in the clock time and the temperature of the air conditioner.

5.2 Implementation Details

To realise the above scenario, the shaded components of the architecture shown in Figure 5.1 needed to be provided.

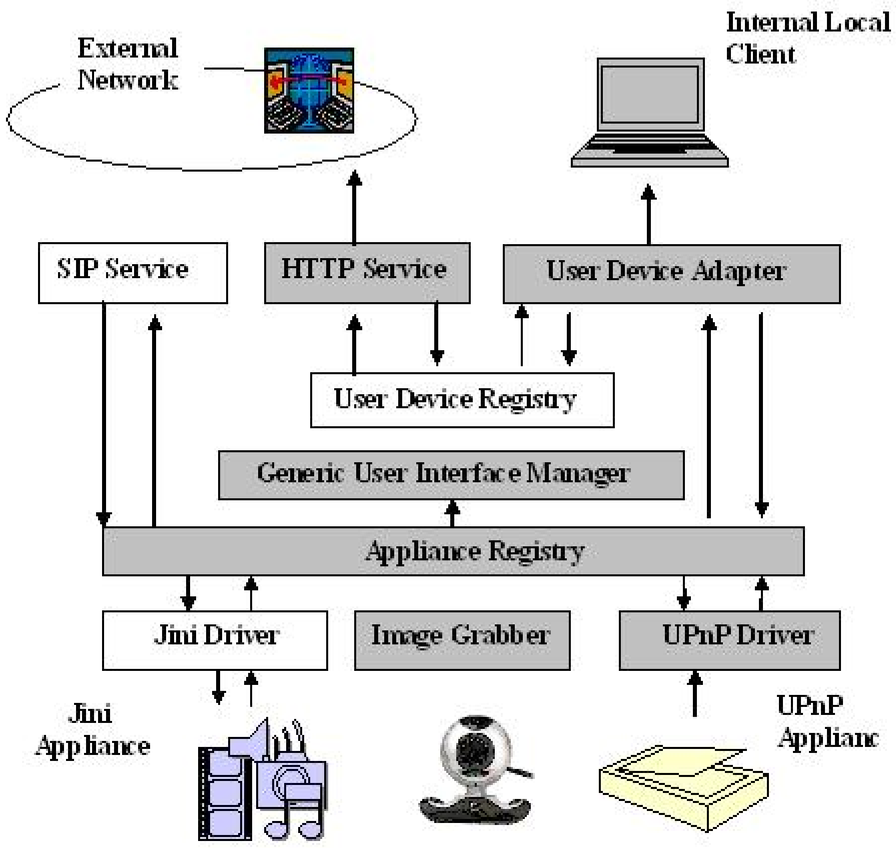

Figure 5.1: Deployed components of the architecture

Each of the components are deployed as bundles on the OSGi framework. All the code and the resources needed to run the bundles are contained in the bundle JAR files. The `HTTP Service` was needed as a servlet container for a `Appliance Servlet` bundle to provide access for the web browser, the `Local User Device Adapter` was implemented as a `Java Sockets Adapter` and was needed to provide access to the Java client, the `Generic User Interface Manger` provided the generic user interface descriptions, the `Appliance Registry` provided access to the different appliances based on different technologies and the `Appliance Drivers` were used to communicate directly with the neworked appliances. The bundles provide services to one another in the form of interface APIs with each bundle implementing its own service.

Before going into detail on how each bundle was developed, the workings of the OSGi framework APIs need to be explained. Most bundles are designed to provide a service, but it is also possible to design bundles that do not provide any service of their own and simply use the services of other bundles. In this implementation though, most of the bundles make services available to the other bundles. The process of developing an OSGi service bundle is as follows: [44]

1. Design the service interface API.
2. Implement the service.
3. Provide a bundle activator to register the service when the bundle starts up and unregister the service when the bundle stops. The bundle activator may also do other operations when the bundle starts up and stops, such as binding to other services when the bundle starts up and closing a socket when the bundle stops.
4. Declare the packages exported by the bundle in the `Export-Package` manifest header. The package that is to be exported is the package that contains the service interface that is being provided. The packages that the bundle relies on are declared in the `Import-Package` manifest header.
5. The final step is to compile the classes and pack everything into a bundle JAR file. The bundle is then ready to be deployed on the OSGi framework.

The `org.osgi.framework` APIs are used to develop and deploy bundles. The bundle activator implements the `public void start(BundleContext context)` and `public void stop(BundleContext context)` methods of the `org.osgi.framework.BundleActivator` interface. Each bundle has what is known as a bundle context that allows the bundle to access the functionality of the OSGi framework. The `start` and `stop` methods are passed `org.osgi.framework.BundleContext` objects to interact with the framework on behalf of the bundle. The bundle activator uses this object to register and unregister services on behalf of the bundle and also to obtain services registered by other bundles in the services registry of the framework. The operation of these methods and APIs will become clear from the implementation discussions below.

The bundle manifest provides the framework with information about how to deploy the bundle. The properties in the manifest file include the name of the bundle activator, the import packages that the bundle needs to use and the export packages that the bundle is willing to provide. For example, every bundle deployed on the OSGi framework needs to import the `org.osgi.framework` package. Thus, every bundle manifest will have the following property in its manifest file:

```
Import-Package:  org.osgi.framework
```

Additional packages can be imported by separating each package name with a comma. Similarly, packages can be exported by the bundle by declaring an `Export-Package`. It also is possible to specify the minimum version numbers of the packages that the bundle wishes to import or export. In the example below, the bundle wishes to import at least version 1.0 of the `org.osgi.service.http` package. If a version lower than 1.0 is available, then the package will not be imported by the bundle. If a specification version is not specified than the default is version 0.0, meaning any version will do.

```
Import-Package:  org.osgi.service.http; specification-version=1.0
```

A discussion of the implementation of each of the components (bundles) will now take place. The services offered and used by each bundle will be explained along with any implementation issues. To give the reader a feel for how the `org.osgi.framework` packages are used, some code will be displayed for the first component discussed; the `Appliance Registry`. The manifest files, and the code to implement the activators for bundles are shown in Appendix B.

Appliance Registry

The other bundles in the architecture rely on the `Appliance Registry` to interact with the networked appliances. The `Appliance Registry` is aware of all the networked appliances in the domain and has references to the `Appliance Drivers` used to communicate with them. It provides a service to the other bundles, allowing them to interact with the networked appliances through the `Appliance Registry` service API. To provide its service API to the other bundles, it has the following `Export-Package` and `Export-Service` headers as entries in its manifest file:

```
Export-Package:  AdvancedApplianceRegistry.Registry
Export-Service:  AdvancedApplianceRegistry.Registry.ApplianceRegistry
```

To allow other bundles to dynamically bind to the services provided by the `Appliance Registry`, it must register its services with the framework's service registry. It does this in the `public void start(BundleContext context)` of its `Activator` class. The code for this method is shown in Figure 5.2:

```
public void start(BundleContext context) throws Exception {
  System.out.println("Appliance␣Registry␣is␣starting␣up....");
  Activator.context = context;
  applianceRegistry = new ApplianceRegistryImpl();
  ServiceRegistration serviceRegistration = context.registerService
    (ApplianceRegistry.class.getName(),applianceRegistry, null);
}
```

Figure 5.2: Activator's start method

Any other bundle wishing to use this service must first obtain a reference to the service from the framework's service registry. For example, the Servlet that the browser interacts with and the Java Sockets Adapter both obtain a reference to the services provided by the `Appliance Registry`. This is done in the activator class of these bundles and is achieved using the following code shown in Figure 5.3:

```
ServiceReference reference = context.getServiceReference(
    "AdvancedApplianceRegistry.Registry.ApplianceRegistry");
```

Figure 5.3: Obtaining a reference to services offered by the Appliance Registry

Once the client bundles have obtained a reference to the service offered by the `Appliance Registry`, they are free to invoke methods provided by the `ApplianceRegistry` interface API. To do this however, the client bundles need to have the following header entry in their manifest files:

```
Import-Package:  AdvancedApplianceRegistry.Registry
```

A specification-version is not defined so clients are happy with any version of the package. Now that the client bundles have a reference to the `Appliance Registry` they can interact with the networked appliances by invoking methods in the `Appliance Registry` interface API. The `Appliance Registry` provides the interface API shown in Figure 5.4.

```
public interface ApplianceRegistry{

  public void invokeCommand(String applianceID,
            String commandID, String commandValue);

  public Object getValue(String applianceID, String valueID);

  public void subscribe(String applianceID, String stateID);
}
```

Figure 5.4: Interface provided by the Appliance Registry

To interact with a networked appliance, a client bundle simply invokes one of the methods of the `Appliance Registry` service retrieved from the services registry. As an example, Figure 5.5 shows code for a client obtaining the time from the UPnP enabled networked clock.

```
ServiceReference reference = context.getServiceReference(
      "AdvancedApplianceRegistry.Registry.ApplianceRegistry");

if(reference != null){
  ApplianceRegistry applianceRegistry = (ApplianceRegistry)context.
                          getService(reference);
  String time = applianceRegistry.getValue("upnpClock1", "time");
}
```

Figure 5.5: Client bundle getting the time from the networked clock

The `Appliance Registry` in this implementation provides access to two simulated UPnP appliances as well as a USB web cam. The `Appliance Registry` interacts with the UPnP appliances by acting as a UPnP control point and talking directly to the UPnP base driver. The details of how this was implemented are presented next.

As stated earlier, a UPnP base driver bundle was recently made available on the OBR (Oscar Bundle Repository). The UPnP base driver adheres to the Device Access Specification defined by the OSGi Alliance. To interact with UPnP appliances through the OSGi framework, this bundle needs to be installed and started on the framework. Any bundle interacting with the UPnP base driver needs to use the UPnP APIs specified by the OSGi Alliance, to interact with the UPnP base driver. A class was developed and packaged with the `Appliance Registry` bundle to act as a UPnP Control Point; `UPnPApplianceRegistry`. This class has the functionality to subscribe to UPnP events generated on the network. The UPnP clock and the UPnP air conditioner, both generate UPnP events. The clock sets off an event when the time is updated, and the air conditioner sets off an event when the temperature of the air conditioner is changed. The `UPnPApplianceRegistry` class also has functionality to update the temperature of the air conditioner. The simulated clock appliance is an open source implementation provided by Domoware Software [53] and the code for the simulated air conditioning unit was derived from the code for this clock. The UPnP control point bundle that was developed in this implementation was also derived from an open source UPnP control point implementation from Domoware Software.

When the UPnP base driver is started on the framework, it multicasts out a search request using SSDP (Simple Service Discovery Protocol) to look for any UPnP appliances that are attached to the network. Any UPnP appliances or simulated UPnP appliances respond directly to this request. On receipt of this response, the UPnP base driver registers the newly discovered appliance within the OSGi framework as an implementation of the `org.osgi.service.upnp.UPnPDevice` interface. The services offered by the UPnP appliance, described in the XML description file, are contained in a `java.util.Dictionary` object within the `org.osgi.service.upnp.UPnPDevice`. For example, the device type of the clock is represented as
"urn:schemas-upnp-org:device:clock:1" and the timer service offered by the clock is represented as "urn:schemas-upnp-org:service:timer:1". Similarly, the air conditioner provides a temperature service. The control point can then search for specific service descriptions and use that service. The timer service defines a time state variable and the temperature service provides a temperature state variable. The `UPnPApplianceRegistry` control point subscribes to receive notifications when these state variables change. It does this by registering as a `UPnPEventListener` within the OSGi service registry and implementing the `public void notifyUPnPEvent(String deviceId,`
`String serviceId, Dictionary events);` method of this interface. This method is called by a `UPnPDevice` when an event that the `UPnPEventListener` is subscribed to has fired. When a change is made to a state variable, this method is called on all the `UPnPEventListener` implementations subscribed to recieve notifications when the

state variable changes. For example, when a change is made to the temperature of the air conditioner, the `UPnPDevice` representing the air conditioner, obtains a reference to the `UPnPEventListeners` from the OSGi service registry and calls the `public void notifyUPnPEvent(String deviceID, String serviceID, Dictionary events)` method on this object. The `UPnPEventListener` that the method has been invoked on can then extract the air conditioner temperature from the `Dictionary` object that represents the event generated by providing the code shown in Figure 5.6 within the implementation of the `public void notifyUPnPEvent(String deviceID, String serviceID, Dictionary events)` method.

```
public void notifyUPnPEvent(String deviceId,
        String serviceId, Dictionary events) {
  if (deviceId.indexOf("Airconditioner") != -1){
    airconTemp = (String) events.get("Temperature");
  }
}
```

Figure 5.6: Implementation of notifyUPnPEvent method by a UPnPEventListener

This code checks to see if the appliance that generated the event is the air conditioner by checking if the device has an ID of "Airconditioner". If it does then the value generated by the "Temperature" event is obtained and set.

A bundle was also developed to grab images from an USB webcam; `ImageGrabber` using the JMF (Java Media Framework). The code for this was based on code available at http://www-adele.imag.fr/ donsez/dev/osgi/webcamproducer/readme.html. The very simple interface for `ImageGrabber` is shown in Figure 5.7

```
public interface ImageGrabber {
  public java.awt.Image getImage();
}
```

Figure 5.7: ImageGrabber interface

The `ImageGrabberService` is registered in the OSGi framework using the code in Figure 5.8.

```
ServiceRegistration imageGrabberService = context.
  registerService(
    ImageGrabber.class.getName(),
    imageGrabber,
    null
    );
```

Figure 5.8: Registering the Image Grabber service

The `Appliance Registry` obtains a reference to the `ImageGrabber` service from the OSGi service registry, and invokes the `getImage()` method to obtain a `java.awt.Image` object, representing an image grabbed from the web cam. The code for this is shown in Figure 5.9. This code is run within the `public Object getValue(String applianceID, String valueID)` method when the `Appliance Servlet` and the `Java Sockets Adapter` bundles request an image from the web.

```
try{
  imageGrabberReference = context.getServiceReference(
    "ImageGrabberService.ImageGrabber");

  imageGrabber = (ImageGrabber)context.getService
              (imageGrabberReference);
}catch(Exception e){System.out.println(e);}

if(imageGrabberReference != null){
  //grab the image from the webcam
  Image imageFromWebCam = imageGrabber.getImage();
  return imageFromWebCam;
}
```

Figure 5.9: Grabbing an image

Generic User Interface Manager

The Generic User Interface Manager that was implemented is a relatively simple component that stores generic user-interface representations for interacting with the networked appliances. This bundle provides a simple UIML documents based on the Harmonia Inc's HTML vocabulary. The fact that only a few documents are stored almost negates the need for this component. However, this is a proof-of-concept implementation, and the Generic User Interface Manager is provided to demonstrate how a large number of generic user-interface descriptions would be stored and provided. The UIML documents based on Harmonia's HTML vocabulary are returned to the `Appliance Servlet` on request. The `Appliance Servlet` then invokes Harmonia's UIML-to-HTML transocoding tool to transform the representation from UIML into HTML. The HTML representation is then returned to the browser client.

Appliance Servlet

The `Appliance Servlet` allows a web browser client to interact with the networked appliances within the domain. `Appliance Servlet` is developed using the standard `javax.servlet` API. For the servlet to be made available to remote web clients, it must be registered with the `HTTP Service`, a lightweight web server available on the OBR. To do this, the `Appliance Servlet` obtains a reference to the `HTTP Service` from the OSGi service registry. Once the reference to `HTTP Service` has been obtained, the `Appliance Servlet` is registered under the alias, `/networkedAppliances`. The code for this is shown in Appendix B. The client browsers use this alias to interact with the networked appliances. The `doGet` and `doPost` methods in the servlet process HTTP GET and HTTP POST requests respectively. To obtain the time of the networked clock and the temperature of the air conditioner, the client browser sends a GET request to the `/networkedAppliances` alias. To change the temperature of the air conditioner, the temperature is changed in a drop-down combo box, and a form with the new value of the temperature is sent to the servlet in the HTTP POST request. The `Appliance Servlet` uses its reference to the `Appliance Registry` obtained from the OSGi service registry to change the temperature of the air conditioner.

Java Sockets Adapter

The final component to be discussed is the `Java Sockets Adapter`. This bundle provides access for a Java client communicating through the Java sockets API. Like the `Appliance Servlet`, it obtains a reference to, and invokes methods on, the `Appliance Registry` bundle to interact with the appliances. It is possible for Java clients to subscribe to UPnP events generated by the appliances (the time and the temperature changes) through this adapter. This is achieved by the `Appliance Registry` invoking the `public void notifyEvent(String serviceID, String value)` method provided by the Java Sockets Adapter. This method is invoked every time the `Appliance Registry` receives a notification of an UPnP event. The Java Sockets Adapter bundle exports a package to allow the `Appliance Registry` to invoke this method. Every time this method is invoked the `Java Sockets Adapter` sends the notification back to the Java client. Communication between the Java client and the Java Sockets Adapter is achieved by the passing of a series of serialized objects. The Java client sends `RequestToJoinMessage`, `RequestToSubscribeMessage`, `RequestToUnSubscribeMessage` and `ApplianceCommandMessage` messages. The `Java Sockets Adapter` sends `ApplianceListMessage`, `RefuseRequestMessage`, `NotifyMessage`, and `UpdateMessage` messages. These objects all implement the `java.io.Serializable` interface. The objects contain attributes relating to information about the messages and methods to set and get this information. The code for the `Java Sockets Adapter` activator is shown in Appendix B.

5.3 Evaluation of the architecture

The proof-of-concept implementation proved the viability of the architecture reasonably well. No major issues that might act as a disincentive for using the proposed architecture arose from the implementation. One problem that might be an issue is the lack of reusable OSGi driver bundles currently available. This will become less of an issue in the future, however, as there is currently much work underway to develop driver bundles to map between OSGi and specific protocols, such as Jini, HAVi and other communication protocols. The developers of the Knopflerfish OSGi framework are currently working on a Jini base driver that conforms to the DAS specifications for mapping beteen Jini and OSGi.

As was acknowledged in the state-of-the-art, UIML does not fully meet the requirements as the generic interface representation language for communicating with networked appliances. During the implementation, this became even more apparent. Using the vocabularies made available by Harmonia, it was still necessary to have knowledge of the target-specific platforms which almost defeats the purpose of having a generic user-interface representation. Also, Java Swing user-interfaces specified using UIML were too basic, and this approach was abandoned in favour of hand-coding the user-interface. It is very unlikely that the current UIML specification will gain acceptance, but some of the more advanced generic user-interface options such as, XIML and DISL, which are still currently under development might take off. These solutions could be easily deployed on an implementation of the proposed architecture.

The performance of the architecture was also analysed. The latency of the commands and requests sent through the network was almost instantaneous in most situations. Also, the time taken for messages to be passed through the framework (i.e. methods to be invoked within the OSGi framework) was almost instantaneous. The implementation was tested with numerous web browser clients and Java clients simultaneously connected to the gateway server. The Java clients were all subscribed to receive UPnP notifications. There was no noticable difference in performance with this set-up. A large number of bundles were also installed and started on the OSGi framework to determine how well it would cope with such a situation. Such a scenario may occur where there are a large number of user devices and networked appliances based on different technologies connected to the domain. Again, there was no noticable degradation in performance with this set-up. Finally, considering that the gateway server should be capable of running for months on end, the implemented architecture was left running on the server for a number of days and again, as expected, there was no significant degradation in performance.

Chapter 6

Conclusion and Future Work

The appliance interaction architecture proposed facilitates multi-modal communication with networked appliances and is designed to be as flexible and future-proof as possible. It achieves this flexiblity from being based on open, rather than proprietary, standards. While it is not absolutely certain that the technologies chosen will remain the de-facto standards in the future, it is likely that they will be strong candidates for any solution that provides communication with networked appliances. The choice of how the services gateway should be deployed is one the more important choices since it dictates what other technologies and protocols can be used within the environment. Proprietary solutions do not offer much value since it is expected that appliances, services and applications for communicating with networked appliances will be made available by a broad range of companies and organisations. Thus, it is desirable that users are not locked into using one particular group of services or applications. The standardised OSGi services gateway specification has achieved broad industry support from many of the major, highly influential, technology companies. Companies such as BMW and IBM are using OSGi solutions in important projects. For example, BMW uses an OSGi compliant services-gateway in their latest 5-Series and 7-Series models. For this very reason, it is very likely that the OSGi will remain, and possibly gain strength, as the de-facto standard for services gateways. With OSGi becoming widespread, the choice of what other technologies to use within the home or vehicle becomes less of an issue as the component-oriented model that OSGi provides allows the dynamic deployment of new drivers and adapters at run-time. It is left up to the developer to decide what communication protocols are most appropriate, and then use an existing OSGi driver bundle or develop a new driver bundle to allow communication using the chosen protocol.

6.1 Future Work

The most interesting future service that could be deployed and tested on the architecture would be the SIP Service, once it has been fully specified and ratified by the OSGi Alliance. Using the extended version of SIP, specifically designed for wide-area communication with networked appliances, would provide much richer functionality for remote clients. The SIP Service is set to be ratified in the near future, and a SIP Service implementation could be plugged in to the architecture easily. Also, as the DEG (Device Expert Group) specifies more appliance drivers for the different discovery and communication technologies, there will be a broader range of driver bundles available for communicating with appliances. Again, these components can be easily plugged in to the architecture.

As pointed out earlier, components providing context-awareness features would offer a lot more functionality to the architecture. This is a huge area of research encompassing a range of fields, requiring expertise in Neural Networks, Fuzzy Logic and other areas. It is worth the time and effort to research this area, however, since the true benefit of the architecture proposed would be realised for certain users, particularly the disabled and elderly, by advances made in this area.

Appendix A

Use Case Diagrams

USE CASE	**JOIN DOMAIN**
GOAL IN CONTEXT	User Device joins the domain
Preconditions	The user device has the proper client program to communicate with the gateway server.
	The user device is announcing its presence to the gateway server.
	The user has knowledge of the password to join the domain.
	The gateway server understands the communication protocol used by the user device.
Success End Condition	The user device has successfully joined the domain and the user is ready to interact with the networked appliances in the domain.
Failed End Condition	The user device cannot join the domain or the user has his/her password rejected.
Primary, Secondary Actors	User device, gateway server.
Trigger	A user device launches the application to communicate with the networked appliances
DESCRIPTION	
Step 1	The user launches the networked appliance communication program on their user device.
Step 2	The user device announces its presence to the gateway server.
Step 3	The gateway server discovers the user device and challenges the user to provide a password.
Step 4	The user provides the password.
Step 5	The user device successfully joins the domain and waits to receive a list of appliances within range.

USE CASE	DOWNLOAD USER INTERFACE REPRESENTATION FROM GATE-WAY SERVER
GOAL IN CONTEXT	A generic user interface representation, or a target-specific user interface representation is downloaded from the gateway server.
Preconditions	The user has selected an appliance from the list to interact with.
	The relevant user interface is not stored locally on the client device.
	The generic user interface reprentation, or the target-specific representation is stored on the gateway server.
Success End Condition	The correct user interface for the selected appliance is downloaded to the client device from the gateway server.
Failed End Condition	Either the user interface representation is not downloaded or an incorrect user interface representation is downloaded from the server.
Primary, Secondary Actors	User device, gateway server.
Trigger	The user device program sends a request to the server for the user interface
DESCRIPTION	
Step 1	A request is sent to the gateway server for the user interface representation for the selected appliance
Step 2	The request is processed by the server and the server sends the interface description to the client device.
Step 3	The client device receives the user interface representation, from the server and is ready to render it.

USE CASE	**INTERACT WITH NETWORKED APPLIANCE**
GOAL IN CONTEXT	The sends requests/commands to a selected networked appliance.
Preconditions	The user device has joined/connected to the domain.
	The user interface for the appliance has been rendered on the client device.
Success End Condition	The request/command sent by the user is successfully carried out. Information is returned to the user device if required.
Failed End Condition	The request/command is not carried out or an incorrect command is carried out.
Primary, Secondary Actors	User device, gateway server.
Trigger	The use selects a command from the user interface of the appliance.
DESCRIPTION	
Step 1	The user selects a command from the interface.
Step 2	The command is sent to the gateway server.
Step 3	The gateway server processes the command, translates it to the protocol used by the selected appliance, and sends it on to that appliance.
Step 4	The command is carried out by the appliance and/or information is returned to the client device through the gateway server
Non-Functional Requirements	
Fault Tolerance	If a device cannot carry out the requested action, then this information must be returned to the client device.
Performance	The translation done by the server must be as efficient as possible.

USE CASE	**CONNECT TO A REMOTE DOMAIN**
GOAL IN CONTEXT	A user connects to a domain remotely over the Internet
Preconditions	The user device has the proper client program to communicate with the gateway server.
	The user has access to an Internet connection.
	The user has the address of the remote gateway server.
	The user has the proper authentication (i.e. username and password) to connect to the remote domain.
Success End Condition	The user device has connected to the remote gateway server and the user is ready to interact with the networked appliances in the domain.
Failed End Condition	The user has not connected to the remote gateway server.
Primary, Secondary Actors	User device, gateway server.
Trigger	The user enters an address of a remote gateway server to connect to, and commands the client device to connect to the server over the Internet.
DESCRIPTION	
Step 1	User selects the option to connect to a remote gateway server by providing the address of the remote gateway server.
Step 2	The user device connects to the remote gateway server and is ready to receive a list of appliances to interact with.
Non-Functional Requirements Fault Tolerance	 If the remote gateway server is down, then the client device should be made aware of this.
Performance	The messages sent over the network must be as small as possible.

Appendix B

Implementation

```
Bundle-Name: Appliance Registry
Bundle-Activator: AdvancedApplianceRegistry.Registry.Activator
Import-Package: org.osgi.framework, org.osgi.service.upnp,
UserDevices.SocketsAdapter, ImageGrabberService
Import-Service: UserDevices.SocketsAdapter.SocketsOSGiAdapter,
ImageGrabberService.ImageGrabber
Export-Package: AdvancedApplianceRegistry.Registry
Export-Service: AdvancedApplianceRegistry.Registry.ApplianceRegistry
```

Figure B.1: Manifest entries for the Appliance Registry

```
package AdvancedApplianceRegistry.Registry;

import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;
import org.osgi.framework.ServiceRegistration;
import org.osgi.service.upnp.UPnPDevice;

public class Activator implements BundleActivator {

  static BundleContext context = null;
  private ApplianceRegistry applianceRegistry = null;

  public void start(BundleContext context) throws Exception {
    System.out.println("Appliance␣Registry␣is␣starting␣up....");
    Activator.context = context;
    applianceRegistry = new ApplianceRegistryImpl();
    ServiceRegistration serviceRegistration = context.
            registerService(ApplianceRegistry.class.getName(),
                    applianceRegistry, null);
  }


  public void stop(BundleContext context) throws Exception {
    System.out.println("Appliance␣Registry␣shutting␣down...");
    Activator.context = null;
    applianceRegistry.shutDown();
  }
}
```

Figure B.2: Activator for the Appliance Registry Bundle

```
Bundle-Name: ApplianceServlet
Bundle-SymbolicName: ApplianceServlet
Bundle-Activator: Activator
Import-Package: AdvancedApplianceRegistry.Registry,
  javax.servlet, javax.servlet.http, org.osgi.service.http
Import-Service: AdvancedApplianceRegistry.Registry.ApplianceRegistry
```

Figure B.3: Manifest entries for Appliance Servlet

```
import java.net.*;
import java.io.IOException;
import javax.servlet.*;
import javax.servlet.http.*;
import org.osgi.framework.*;
import org.osgi.service.http.*;
import AdvancedApplianceRegistry.Registry.*;

public class Activator implements BundleActivator{
  private HttpService http;
  final static String SERVLET_ALIAS = "/networkedAppliances";
  final static String HTML_ALIAS = "/htmlFiles";
  ApplianceRegistry applianceRegistry = null;

  public void start(BundleContext context)
    throws ServletException, NamespaceException
  {
    ServiceReference ref = context.getServiceReference(
        "org.osgi.service.http.HttpService");
    http = (HttpService)context.getService(ref);
    HttpContext hc = new HttpContext(){
      public String getMimeType(String name){
        return null;
      }
      public boolean handleSecurity(HttpServletRequest req,
          HttpServletResponse resp)throws IOException
      {
        return true;
      }
      public URL getResource(String name){
        URL u = this.getClass().getResource(name);
        return u;
      }
    };
    ServiceReference reference = context.getServiceReference(
      "AdvancedApplianceRegistry.Registry.ApplianceRegistry");
    if(reference != null){
      applianceRegistry = (ApplianceRegistry)context.
               getService(reference);}
    ApplianceServlet servlet = new ApplianceServlet(context);
    http.registerResources(HTML_ALIAS, "/htmlFiles", hc);
    http.registerServlet(SERVLET_ALIAS, servlet, null, hc);
  }
  public void stop(BundleContext context) {
    if(http != null){
      http.unregister(SERVLET_ALIAS);}
  }
}
```

Figure B.4: Activator for Appliance Servlet

```
Bundle-Name: Sockets OSGi Adapter
Bundle-SymbolicName: Sockets OSGi Adapter
Bundle-Activator: UserDevice.SocketsDevice.Activator
Import-Package: AdvancedApplianceRegistry.Registry
Import-Service: AdvancedApplianceRegistry.Registry.ApplianceRegistry
Export-Package: UserDevice.SocketsDevice
Export-Service: UserDevice.SocketsDevice.SocketsOSGiAdapter
```

Figure B.5: Manifest entries for Java Sockets Adapter

```
package UserDevice.SocketsDevice;

import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;
import org.osgi.framework.ServiceRegistration;
import org.osgi.service.upnp.UPnPDevice;
import AdvancedApplianceRegistry.Registry.*;

public class Activator implements BundleActivator {

  static BundleContext context = null;
  private SocketsOSGiAdapter socketsOSGiAdapter = null;
  ApplianceRegistry applianceRegistry = null;

  public void start(BundleContext context) throws Exception {
    System.out.println("Sockets␣OSGi␣Adapter␣is␣starting␣up....");
    Activator.context = context;
    try{
      socketsOSGiAdapter = new SocketsOSGiAdapter(context);
    }catch(Exception e){System.out.println(e);}

    ServiceRegistration serviceRegistration = context.
                registerService(SocketsOSGiAdapter.class.getName(),
                    socketsOSGiAdapter, null);

    ServiceReference reference = context.getServiceReference(
          "AdvancedApplianceRegistry.Registry.ApplianceRegistry");

    if(reference != null){
      applianceRegistry = (ApplianceRegistry)context.getService(reference);

    }

  }


  public void stop(BundleContext context) throws Exception {
    System.out.println("Sockets␣OSGi␣Adapter␣is␣shutting␣down...");
    socketsOSGiAdapter.closeSocket();
    Activator.context = null;
  }
}
```

Figure B.6: Activator for Java Sockets Adapter bundle

Bibliography

[1] Osgi alliance. http://www.osgi.org.

[2] Khurana S. et al. Xml based wide area communication with networked appliances. *IEEE Conference on Wired and Wireless Communication*, 2004.

[3] S Moyer et al. Service portability of networked appliances. *IEEE Communications Magazine*, pages 13–19, 2002.

[4] S Tsang. Requirements for networked appliances: Wide-area access, control, and interworking. *IETF Draft*, September 2001.

[5] Trace research and development centre - university of wisconsin at madison, usa. http://trace.wisc.edu.

[6] Abrams M. et al. Uiml: An appliance-independent xml user interface language. *WWW8/Computer Networks*, 1999.

[7] J Plomp. Uiml in future home environments. *Submitted to the first UIML conference held in Paris*, January 2001.

[8] SH Maes et al. Multi-modal interaction in the age of information appliances. *IEEE International Conference on Multimedia and Expo*, 2000.

[9] Stajano F. et al. Security issues for internet appliances. *IEEE Symposium on Applications and the Internet (SAINT) Workshops*, 2002.

[10] Anderson et al. The resurrecting duckling: Security issues for ubiquitous computing. *Supplement to Computer Magazine*, pages 22–26, May 2002.

[11] Chan T. On applying sip security to networked appliances. *IEEE 4th International Workshop on*, pages 31–40, 2002.

[12] Bull P. et al. Residential gateways. *BT Technology Journal*, 20:73–80, April 2002.

[13] Teger S et al. End-user perspectives on home networking. *IEEE Communications Magazine*, 40:114–119, April 2002.

[14] Zahariadis Th. et al. A comparison of competing broadband in-home technologies. *Electronics and Communication Engineering Journal*, pages 133–142, August 2002.

[15] Wikipedia. *http://www.wikipedia.org.*

[16] Vaxevanakis K. et al. A review on wireless home network technologies. *Mobile Computing and Communications Review*, 7:59–68, April 2003.

[17] Gaetano Borriello. Embedded computation meets the world wide web. *Communications of the ACM*, 43(5):59–66, 2000.

[18] Choonhwa Lee. Protocols for service discovery in dynamic and mobile networks. *International Journal of Computer Research*, 11:1–12, 2002.

[19] Reilly D. et al. A jini-based infrastructure for networked appliance management and adaptation. *IEEE 5th Annual Workshop on Networked Appliances*, pages 161–167, Oct 2002.

[20] Wu Lan et al. Service discovery for personal networks. Master's thesis, University of Stuttgart, December 2004.

[21] Moyer S. et al. Service portability of networked appliances. *IEEE Communications Magazine*, pages 13–19, 2002.

[22] Moyer S. et al. A protocol for wide-area secure networked appliance communication. *IEEE Communications Magazine*, 39:52–59, October 2001.

[23] Rosenberg J. et al. Rfc 3261 - sip: Session initiation protocol. *Network Working Group*, June 2002.

[24] Moyer S. et al. Sip extensions for communicating with networked appliances. *IETF Draft*, 2002.

[25] A.Roach et al. Event notification in sip. *IETF Draft*, 2002.

[26] A Roychowdhury et al. Instant messaging and presence for sip enabled networked appliances. 2001.

[27] Marples D. et al. Feature interactions in services for internet personal appliances. *IEEE International Conference on Communications*, 2002.

[28] Tsang S. et al. Accessing networked appliances using the session initiation protocol. *IEEE Conference on Communication*, pages 1280–1285, 2001.

[29] Srpp. pages 97–111. Internet Society Network and Distributed System Security Symposium, Mar 1998.

[30] Mozilla xul. http://www.mozilla.org/projects/xul/.

[31] extensible interface markup language (ximl). http://www.ximl.org.

[32] Mayora-Ibarra O. et al. A visual programming environment for device independent generation of user interfaces. 2004.

[33] Angel Puerta et al. Ximl: A common represenation for interaction data. *ACM*, January 2002.

[34] User interface markup language (uiml). http://www.uiml.org.

[35] Harmonia inc. http://www.harmonia.com.

[36] Mueller W. et al. Interactive multimodal user interfaces for mobile devices. *IEEE Proceedings of the 37th Hawaii International Conference on System Sciences*, 2004.

[37] Shell E. et al. Building multi-platform user interfaces with uiml. *Proceedings of CADUI*, 2003.

[38] Schaefer R. et al. Object oriented specification with odsn. *In Proceedings of Hawaii International Conference on System Sciences*, 2002.

[39] Schaefer R. et al. Multimodal interactive user interfaces for mobile multi-device environments. *http://www-users.cs.umn.edu*, 2003.

[40] Mueller W. Adaptive profiles for multi-modal interaction in intelligent environments. *AI Moves to AI: Workshop on Artificial Intelligence, Information Access and Mobile Computing*, 2003.

[41] Hall R. et al. Challenges in building service-oriented applications for osgi. *IEEE Communications Magazine*, pages 144–149, May 2004.

[42] David Jordan. Java in the home: Osgi residential gateways. *Java Report*, pages 38–43, September 2001.

[43] Gong L. et al. A software architecture for open service gateways. *IEEE Internet Computing*, 2001.

[44] Gong Li. *Programming Open Service Gateways with Java Embedded Server.* Addison Wesley, 2001.

[45] Zhang D. et al. Open service residential gateway for smart homes. 2002.

[46] Bushmitch D. A sip-based device communication service for osgi framework. *IEEE*, 2004.

[47] Li X. et al. The design and implementation of home network system using osgi compliant middleware. *IEEE Transactions on Consumer Electronics*, 50(2):528–534, May 2004.

[48] Ditze M. et al. Service-based access to distributed embedded devices through the open service gateway. 2003.

[49] Ishikawa H. et al. Building smart appliance integration middleware on the osgi framework. *IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, 2004.

[50] Zhang D. et al. Osgi based service infrastructure for context aware connected homes.

[51] Hackbarth K et al. Osgi based service infrastructure for context aware automotive telematics. *IEEE Vehicular Technology Conference*, pages 2957–2961, 2004.

[52] Oscar bundle repository. *http://oscar-osgi.sourceforge.net/*.

[53] Domoware software. *http://domoware.isti.cnr.it/*.