uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

uOttawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Nevin Metwly
AUTEUR DE LA THÈSE / AUTHOR OF THESIS

M.A.Sc. (Electrical and Computer Engineering)
GRADE / DEGREE

School of Information Technology and Engineering
FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

Common Application Programming Interface Architecture Bridge for Connecting Heterogeneous Home Computing Middleware

TITRE DE LA THÈSE / TITLE OF THESIS

Hussein Mouftah
DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

A. Elsaddik

C.H. Lung

G. Oarham

Gary W. Slater
Le Doyen de la Faculté des études supérieures et postdoctorales / Dean of the Faculty of Graduate and Postdoctoral Studies

Common Application Programming Interface Architecture Bridge for Connecting Heterogeneous Home Computing Middleware

Nevin Metwly

Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
In partial fulfillment of the requirements
For the M.A.Sc degree in
Electrical and Computer Engineering

School of Information Technology and Engineering
Faculty of Engineering
University of Ottawa

Your file Votre référence
ISBN: 978-0-494-65514-6
Our file Notre référence
ISBN: 978-0-494-65514-6

Abstract

Microprocessors are now embedded in various equipments such as home appliances and digital audio/video devices, providing varieties of services. An integration of these services and creation of new ones is always needed to fulfill ongoing users' demands.

There are a number of issues limiting the expandability of the intelligent home networks. One of these issues is that most middleware for home computing networks are not compatible with each other. To solve this problem of diversification of middleware and to ensure interoperability among them, the one-to-one bridge has been developed, which connects two specific middleware varieties and performs protocol conversion; also various one-to-any home network middleware bridges have lately been introduced.

In this thesis, a proposal for any-to-any bridge connecting home computing middleware is introduced. It enables any appliance under any middleware to communicate and control any other appliances under different middleware. Also, new middleware can be easily integrated with the proposed architecture.

Acknowledgements

Sincere thanks to my supervisor, Professor Hussein Mouftah, for his outstanding guidance, support and encouragement throughout the course of my masters at Ottawa University. His boundless enthusiasm, dedication to excellence, and infinite patience will always be points of inspiration for me.

I would like to thank all the professors who taught me during my course of study, for creating a dynamic environment for research and study.

Words alone can never express my love, appreciation, and gratitude for my dear mother for encouraging me, for my husband Khaled for his continuous support, and my precious children Maryam, Mai and Kareem for being patient. Without their support and encouragement this work would not have been possible.

Contents

List of Figures

List of Tables

List of Acronyms

CAL	Common Application Language
CAPIA	Common Application Programming Interface Architecture
CG	Command Generator
CLI	Common Layer Interface
CP	Command Parser
DCM	Device Control Module
DHCP	Dynamic host configuration protocol
DHE	Digital Home Environment
DLNA	Digital Living Network Alliance
DNS	Domain Name System
EHS	European Home System
EHSA	European Home Systems Association
FCM	Functional Component Module
FT	Fault Tolerance
HAVi	Home Audio-Video interoperability
HIGA	Home IMS GAteway
HNB	Home Network Broker
HNB	Home network broker
HNCP	Home Network Control Protocol
HOMI	HOme network Middleware for Interoperability
IMS	IP Multimedia Subsystem
IR	Infrared
LFD	Legacy FireWire Devices

NHAs	Networked Home Appliances
OMG	Object Management Group
OSGi	Open Service Gateway initiative
PCM	Protocol Conversion Manager
PLC	Power Line Communication
PSTR	Primary-Shadow TMO Replication
RF	Radio Frequency
RG	Residential Gateway
RMI	Remote Method Invoke
SDPs	Service Discovery Protocols
SIP	Session Initiation Protocol
SLP	Service Location Protocol
SOAP	Simple Object Access Protocol
SSDP	Simple Service Discovery Protocol
TMO	Time-triggered Message-triggered Object
UDT	Universal Device Template
UHNM	Universal Home Network Middleware
UMB	Universal Middleware Bridge
UMB-A	Universal Middleware Bridge-Adaptor
UMB-C	Universal Middleware Bridge-Core
UPnP	Universal Plug 'n' Play
VHN	Versatile Home Network
VSG	Virtual Service Gateway
VSR	Virtual Service Repository
XML	Extensible Markup Language

Chapter 1: Introduction

1.1. Background

A smart House or Home can be defined as "A dwelling incorporating a communications network that connects the key electrical appliances and services, and allows them to be remotely controlled, monitored or accessed" [JIA04].

A working model of the Smart House environment, as shown in the CENELEC Workshop Smart House Final Report [CEN03] has three separable interest areas: the Smart Home itself and its in-house networks and applications (i.e. the clients); the access point to the Smart Home, often referred to as the residential gateway; and provision of services in a standardized way to the Smart Home and related access networks.

1.1.1. Smart Home Components

A Smart Home must contain four components: (1) an internal network, (2) intelligent control, (3) home automation, and (4) middleware. An internal network is the network inside the house; it is the basis of a Smart Home. It can be wire, cable or wireless. Intelligent control is the use of gateways to manage the system's communication. Home automation refers to products and appliances that interact and can be controlled by one another through the middleware to create integrated services.

1. **Smart Home networks:** Smart Home network technology [JIA04] exists in three main areas: (1) powerline (e.g. X10, Em Powerline, etc.) (2) bus line (e.g. EIB, Cebus, Lonwork, Batibus EHS, etc.) and (3) radio frequency (RF) (e.g. Bluetooth, 802.11, etc.)

   Power line: These systems are made of devices that can be connected directly to the main power supply. The major problems with these systems are related to interference and power cuts.

   Bus line: Smart Homes use a separate 12-volt cable (twisted pair) to transmit data to devices, and this cable runs parallel to the traditional main cable. The use of this cable implies that the devices are independent of conventional power supplies. Bus line has to date been proven to be the most effect and reliable form of Smart Home network as it can be configured to prevent devices malfunctioning during power cuts. The two-way protocols also allow the systems themselves to monitor devices without recourse to external computerized systems [JIA04].

   Radio frequency (RF) and infrared (IR): RF is becoming increasingly popular. Most manufacturers of Smart Home technology have a RF product range. These products have tended to be less reliable due to interference problems and short-range identification issues [JIA04].

   Bluetooth (IEEE 802.15): Bluetooth is the well-known technology used in mobile telephones and other hardware like printers and digital cameras. It is a set of design protocols for systems that allow radio frequency control. Bluetooth enables devices to be connected while within a short distance [JIA04].

Zigbee (IEEE 802.15.4): Zigbee is a wireless standard of data transmission allowing machine to machine communication (M2M). Zigbee is demotics-oriented with a very low electric power consumption, which makes it an asset when used in smart electronics.

Zigbee, UWB, Blue tooth, RFID are the most popular wireless future home network technologies [JIA04].

2. **Residential gateway:** A residential gateway (RG) is a device that is the interface between the broadband access network and the home network. All the devices are connected to the RG via wired and wireless networks, which in turn receive and send instructions from a remote device over the Internet as well as from the home area network [ANA07].
3. **Appliances/devices:** There are four types of home network services: home entertainment services, home automation services, and home office services. Accordingly, appliances are classified as the following [JIA04] :
   - The A/V (Audio/Video) appliance category includes audio and video appliances common to home entertainment services. Most of these appliances communicate with each other over IEEE 1394 networks which are based on HAVi middleware, or over wireless LANs which are based on UPnP middleware.
   - The control appliance category includes actuator, sensors, and white goods common to home automation services. Most of these appliances communicate with each other over the power line and are based on LonWorks middleware [ECH10].

- The information appliance category includes appliances common to home office services. Most of these appliances communicate each other over Ethernet or wireless LAN, and are based on UPnP or Jini middleware.
- Others: pagers, PDAs, cell phones, wearable devices, augmented "things that think," and radio frequency identification devices (RFID).

4. **Middleware technologies:** As multiple physical media and lower-layer standards and protocols are expected, interoperability can be provided only at the higher layers of the protocol stack. "Middleware technologies aim to provide a minimum common set of interfaces and functions that will hide devices' or resources' heterogeneity and enable different resource types to communicate transparently" [ODR01].

Various middleware technologies have already evolved, such as Home Audio-video Interoperability (HAVi) [HAV10], Universal Plug 'n' Play (UPnP) [UPN10], and Jini [JIN10].

1.2. Motivation

From the background information and the smart home overview, one can conclude that a Smart Home network can make use of diverse technologies with respect to all of its components (Smart Home networks, RG, appliances and middleware).

The advancement of technologies to enable communication with networked appliances is providing new integrated services, which enable the user to effectively centralize the management and services in a house. This can be done by controlling and monitoring networked appliances from the home, including entertainment systems, lights, and HVAC facilities (heating/ventilation/air-conditioning) - adjusting

temperature or humidity to meet the home owner's requirements, for example. Controlling and monitoring home networked appliances can be done remotely over the Internet. This has numerous contributions in securing a high quality of living conditions, especially for the elderly and people with disabilities. In addition, home automation can be responsible for indoor energy management and supervision based on the instructions of household owners.

Currently, various specifications of devices and communication methods coexist, raising solutions that are restricted to a specific technology or service, which makes it challenging to secure connectivity between devices or between makers. Consumers will not accept intelligent home appliances, nor can this market really flourish, unless the environment for providing such an integrated solution service is established.

Middleware can be defined as "The software technologies, architectures and systems that promise seamless interconnection to devices with different hardware and network characteristics" [ODR01]. New middleware has been proposed whenever new requirements are found, and many varieties of home computing middleware will be developed in the future. Such middleware diversification causes two significant problems. The first is diminished interoperability, which is the ability for devices in the home to discover, configure, and control the capabilities of peer devices which use different middleware. The second is diminished function for a service which uses several kinds of middleware.

1.3. Objective

Our main objective is to help in making a Smart Home environment less expensive, easy to deploy, and accessible to consumers in a manner that will ensure a high standard of living, especially for the elderly and people with disabilities.

We are proposing architecture to solve the problem of interoperability between heterogeneous middleware, which is considered a key element for success of digital and Smart Homes.

There are three approaches to ensure interoperability among the various kinds of home network middleware. The first is a one-to-one protocol, such as the HAVi to UPnP bridge at Thomson that provides the interoperability between UPnP and HAVi [DEN02]. Another approach is a one-to-any protocol bridge. Our proposed architecture will take the any-to-any approach.

1.4. Thesis Contributions

This dissertation focuses on interoperability among heterogeneous home computing middleware, especially the one-to-any approaches:

1. Design an architecture for an any-to-any middleware bridge, called the Common Application Programming Interface Architecture Bridge (CAPIA), to facilitate the communication between different appliances under different middleware. The proposed architecture will help solve the interoperability issue of the Smart Home; it will function without the need to modify the appliance and will not require user interference in reprogramming scenarios.
2. Model and design of interaction algorithm between different modules of the proposed CAPIA.

3. Models designed for CAPIA and for the different one-to-any home middleware bridges will be presented. These models have been simulated using Arena software [ROC05] to compare the total time of the process under differing loads.

1.5. Thesis Organization

The thesis is organized as follows: Chapter 2 presents a comprehensive study on home computing middleware while focusing on the interoperability issues among different middleware, and presents different approaches in solving these issues, especially with regard to one-to-any bridges. Chapter 3 presents the proposed Common Application Programming Interface Architecture Bridge (CAPIA), which will facilitate the communication between different appliances under different middleware. A complete description of the functionality of each module in the architecture is given, followed by a scenario for setting up a connection between two devices under different middleware, showing the flow of messages in the CAPIA bridge. Chapter 4 presents a model for CAPIA and how the process was simulated using Arena software under different loads. Models are provided for two of the one-to-any bridges that are studied in Chapter 2 and simulated using the same simulation software Arena environment as CAPIA. Results are collected and a comparison is made between the proposed CAPIA architecture and the other two bridges. Finally, Chapter 5 provides concluding remarks and discusses future work.

Chapter 2: Middleware Technologies

2.1. Introduction

As discussed in the background of Chapter 1, home networks can operate under diverse environments, which raise issues with networking intelligent home appliances. These issues can be technical in terms of both efficiency and the user's operation perspective, and can be organized into three categories: compatibility, scalability, and connectivity.

Compatibility ensures that the operation of devices and systems that comprise an entire network of intelligent home appliances is functional even when one of them is replaced upon upgrading by the same or different provider [STU05].

Scalability ensures that already implemented devices or systems can be enhanced through the improved functionality of their components, or by being combined with a new device or system in order to improve overall functionality. It is necessary to ensure successive future upgrades [STU05].

Connectivity guarantees that different devices, systems, or networks can be connected to each other so that they can be used in an integrated manner. In order to provide services or content on an intelligent home appliance network, various intelligent home appliances need to be connected to the network as a whole. Therefore, it is crucial that connectivity between devices, systems, and networks, regardless of differences in manufacturers or types of models, is always secured in a stable manner since various specifications of devices and communication methods coexist at the moment [STU05].

From the user's perspective, intelligent home appliance networks must represent what users want to use or what they can safely use. The cost burden of these networks, their information security, and the usability of devices are also important points.

When we consider the cost burden, the unit price of intelligent home appliances should be restricted within a range that is not so much higher than the unit price of the existing consumer electronics. The cost burden should be reduced by using existing in-home wiring or a wireless LAN [STU05].

Ease-of-use is another important factor. Currently available consumer electronics do not offer much operational connectivity between devices. As operations become more complicated intelligent home appliance networks, it is extremely important to ensure ease-of-use, since users of intelligent home appliances are not necessarily familiar with operating such devices [STU05].

The biggest challenge for business operators is how to build business models for intelligent home appliance networks. Business operators may cover enterprises and organizations in various industries, including home electronics makers, telecommunication carriers, content/service providers, and energy suppliers. Business models should differ between business operators and allow free competition.

What is most hoped for in networking intelligent home appliances is widespread standardization and connectivity, i.e. the environment where everyone can connect devices easily.

The intelligent home appliance product market is efficient only when the product offers an integrated service solution rather than providing the same service with discrete components. Consumers will not accept intelligent home appliances, nor will

this market truly prosper, unless the environment for providing such integrated solution services is established. Embedded appliances have limited resources and functions; they cannot provide effective services without cooperating with other appliances. Therefore, we need middleware that provides a high-level abstraction and zero-configuration, and it must also make it possible to integrate new services without great effort.

New middleware is proposed whenever new requirements for compatibility are discovered, which means that many new varieties of home middleware will appear in the future.

Such middleware diversification causes two difficult problems [TOK02]:

1. Less interoperability of services that use different middleware, where interoperability here is defined as the ability for devices in home to discover, configure, and control the capabilities of peer devices. For example, a UPnP-based client cannot access a Jini-based service without significant difficulty.
2. Less deployability for services which use several kinds of middleware. For example, if we want to deploy a service which uses both Jini and UPnP, it is forced to use both clients, not simply one.

2.2. Various Middleware Technologies

Various middleware technologies have already evolved. This section will discuss a selection of standardization efforts by vendors.

The Smart Home components are shown in Fig. 2.1. The figure also shows where the middleware layer resides, and it is usually under the application as shown.

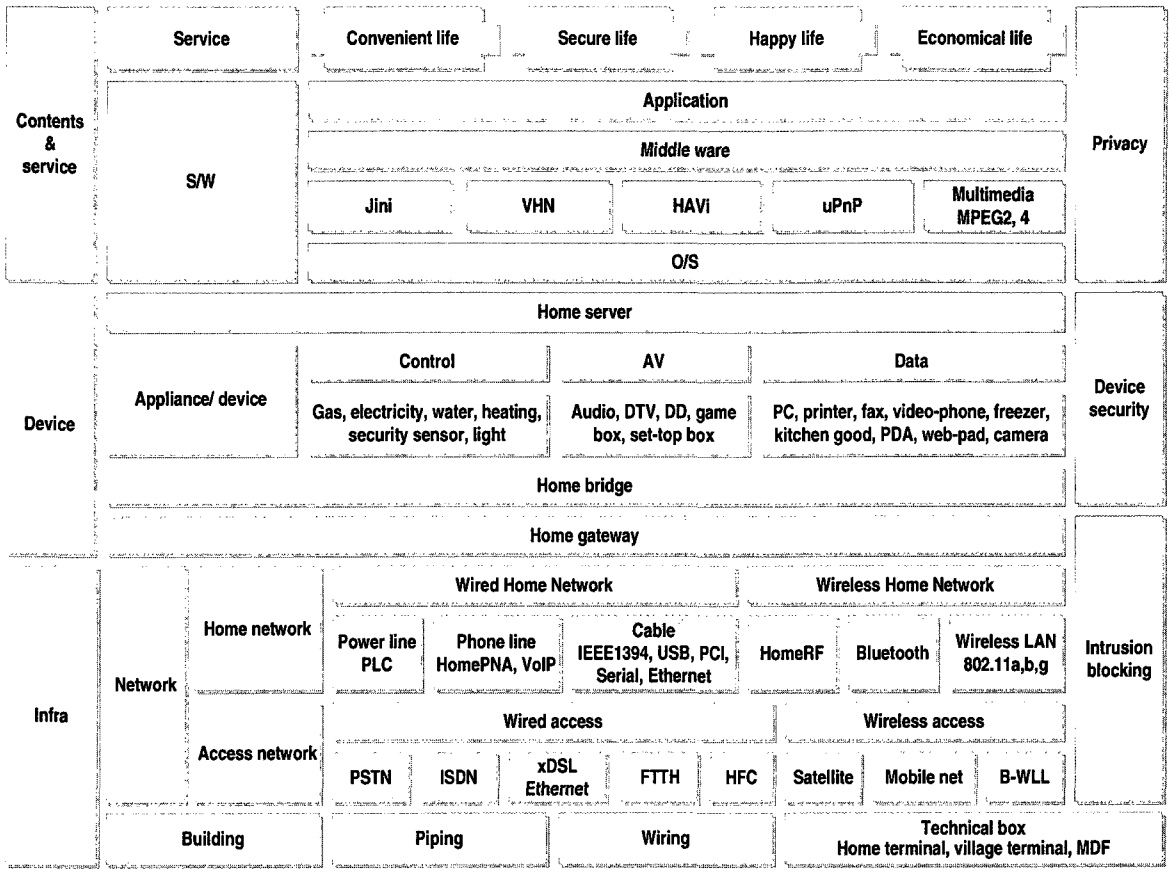

Fig. 2.1: Home network, home gateway, Devices & Appliances, and the Middleware

2.2.1. Home Audio-video Interoperability

Home Audio-Video Interoperability (HAVi) is a higher-layer standard aiming to allow all manner of digital consumer electronics and home appliances to communicate with each other [MIL01].

The main objective of HAVi is to provide a lightweight, open, platform-independent home networking software specification, for seamless interoperability among home-entertainment products. It has been designed and optimized to meet the particular demands of digital audio and video. HAVi has selected the IEEE 1394 bus over other physical and transmission protocols to interconnect A/V devices, because it supports multiple isochronous communication channels [ODR01] and is a high-speed network.

HAVi has various APIs to control digital AV devices and content, but it doesn't have APIs to control information appliances like PDAs or cellular phones [TOK02].

2.2.2. Universal Plug 'n' Play

Universal Plug 'n' Play (UPnP) is a higher–layer protocol stack that aims to extend the simplicity and auto configuration features of device PnP to the entire network, enabling discovery and control of networked devices and services.

UPnP is built over the standard IP, HTTP, and extensible markup language (XML). It enables a device to join a network dynamically, obtain an IP address, convey its capabilities, and learn about the presence and capabilities of other devices. Devices can automatically communicate with each other directly without additional configuration [ODR01].

UPnP can be used over most physical media including radio frequency (RF, wireless), phone line, power line, coaxial, Ethernet, and IEEE 1394.

UPnP is a pure IP network where UPnP devices are connected; other technologies such as HAVi could be accessed via UPnP Bridge.

2.2.3. Jini

Jini provides a software layer for dynamic service discovery and joining, enabling services to be added or removed, and also enabling the cooperation of various computer devices such as embedded devices, handheld devices and PCs. Jini calls this cooperation "federation." The federation of devices integrates various functions provided by Jini-based devices. Jini runs on the Java framework [TOK02].

The heart of a Jini system is lookup services, which is a simple process that allows a Jini client to find potential service providers. "Services are found and resolved by a lookup service. The lookup service is the central bootstrapping mechanism for the system and provides the major point of contact between the system and users of the system. In precise terms, a lookup service maps interfaces indicating the functionality provided by a service to sets of objects that implement the service. In addition, descriptive entries associated with a service allow more fine-grained selection of services based on properties understandable to people." [JIN06].

Jini can be used only on certain types of network, such as TCP/IP [TOK02].

2.2.4. Versatile Home Network

Versatile Home Network (VHN) [ODR01] is an interoperability standard which defines a digital in-home intranet that connects previously incompatible component or cluster networks. A component or cluster network is a homogeneous network of devices that reside together on a physical network, such as a lighting control network or an A/V entertainment network.

The VHN standard uses a long-distance version of IEEE 1394 as the digital backbone and IPs for interworking. VHN uses TCP/IP and UDP/IP for protocol stacks as the base for communication, DHCP to allocate address, and DNS for device name space. Streaming data is supported by the Object Management Group (OMG) enhanced with A/V commands and HAVi.

VHN uses XML queries for device to device control and HNB (Home Network Broker) and Interface Repository (IR) to maintain device configuration information

and device states, which allow simple network maintenance. HTML is used for the GUI interface.

VHN and UPnP share many similarities, and there is interoperability and compatibility between them. However, VHN goes beyond UPnP by specifying a physical and transport layer using IEEE 1394, QoS for isochronous message transport, and a common interface for RGs that connect devices in the home to outside access networks.

2.2.5. KONNEX

Three companies serving different areas and applications in the home network environment merged together and were given the name KONNEX [ODR01]. These three companies are the European Installation Bus Association, Batibus Club international, and EHSA.

The aim of KONNEX is to provide a technical basis for convergence into a single common system supported by relevant industrial companies.

2.3. Interoperability Among Different Home Network Middleware

It is clear that home networks operate under a wide range of diverse technologies; each has its own protocols, various specifications of devices and communication methods, services, etc., which makes it difficult to secure connectivity between devices or makers. Different manufactures are taking individual action, but a single middleware cannot integrate all services.

In order for the intelligent home to become accepted and affordable, the use and the deployment of services of home appliances should be possible without special knowledge of diversification of network forms and middleware. Therefore, the interoperability of heterogeneous middleware is the key element for the success of the digital home.

As an example of such a network, imagine a Smart Home network that consists of several types of networks and middleware - a HAVi-based IEEE1394 network connecting a digital TV and VCR, a Jini-based Ethernet network connecting a dishwasher, a washing machine and a dryer, and a UPnP-based Ethernet network that connects a printer, a computer, and a scanner. What the user wants is to control the TV, the VCR, the dishwasher and dryer from a PC without needing to be conscious of heterogeneous forms of network and middleware. The service discovery and the protocol conversion between Jini , HAVi and UPnP should be managed outside of the user's awareness [TOK02].

There are issues that must be addressed to insure interoperability among heterogeneous middleware; how can devices adopting different middleware find each other transparently as each middleware utilizes different service discovery mechanisms? How can services adopting different middleware still invoke each other?. "A service invocation mechanism of Jini relies on Java RMI (Remote Method Invoke) which uses Java byte code. UPnP uses SOAP (Simple Object Access Protocol) to invoke a service transferring XML text stream. Problems caused by the difference between service invocation mechanisms must be solved to provide interoperability between heterogeneous middleware. In order to solve the problem,

syntax elements of middleware (e.g., method name, the order of arguments, the type size of return value and arguments) should be adjusted."[KIM06] It is also necessary to convert calling methods according to the service invoke mechanisms of each middleware.

2.4. Middleware Interoperability Protocols

There are several approaches to ensure interoperability among the various forms of home network middleware.

1. One-to-one protocol
2. One-to-any bridge protocol

2.4.1. One- to-one Protocol

The HAVi to UPnP Bridge at Thomson provides interoperability between UPnP and HAVi. It minimizes the modification of existing middleware to use the HTTP protocol, but a HAVi device must be extended to support HTTP [MOO03A].

Fig. 2.2 shows different one-to-one bridges.

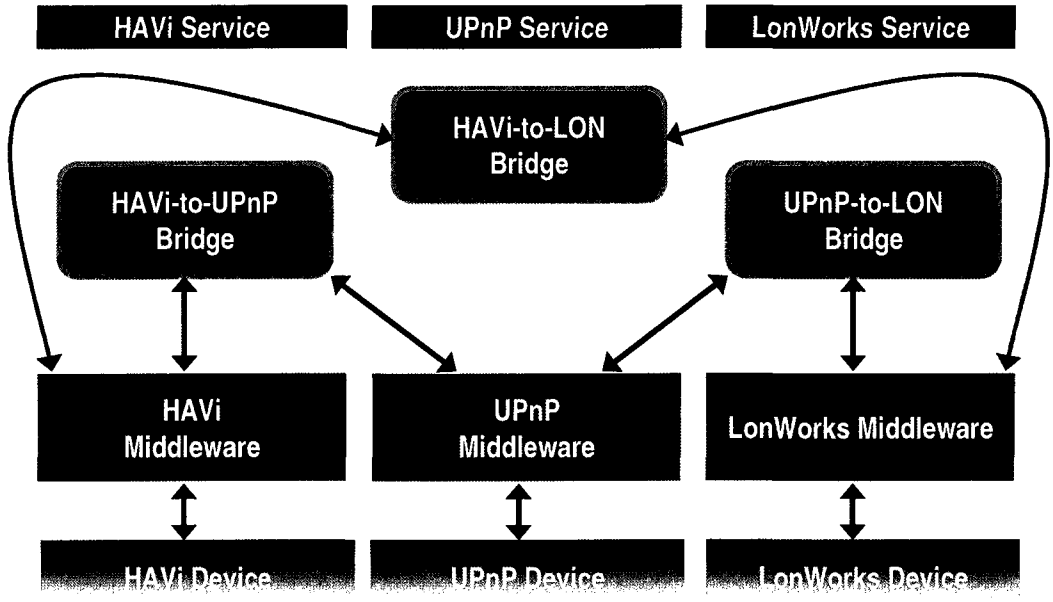

Fig. 2.2: One-to-one bridges

The following section will give a brief overview of the IEEE1394 to UPnP bridge.

2.4.1.1. IEEE1394 to UPnP Software Bridge Structure

With the growth of IP coverage, UPnP, which is a TCP/IP-based home network middleware, attempts to cover all kind of devices connected to the home network. However, IEEE1394 devices do not have any component of the UPnP stack including TCP/IP. These kinds of devices are usually referred to as "legacy IEEE1394 devices" or "legacy FireWire devices" (LFD). To enable the UPnP to control LFDs, a bridge structure is needed to represent an LFD as a UPnP device. Additionally, data format conversion is necessary to support interoperability between devices which use different media formats [POL05].

Fig. 2.3 illustrates the IEEE1394 to UPnP bridge structure.

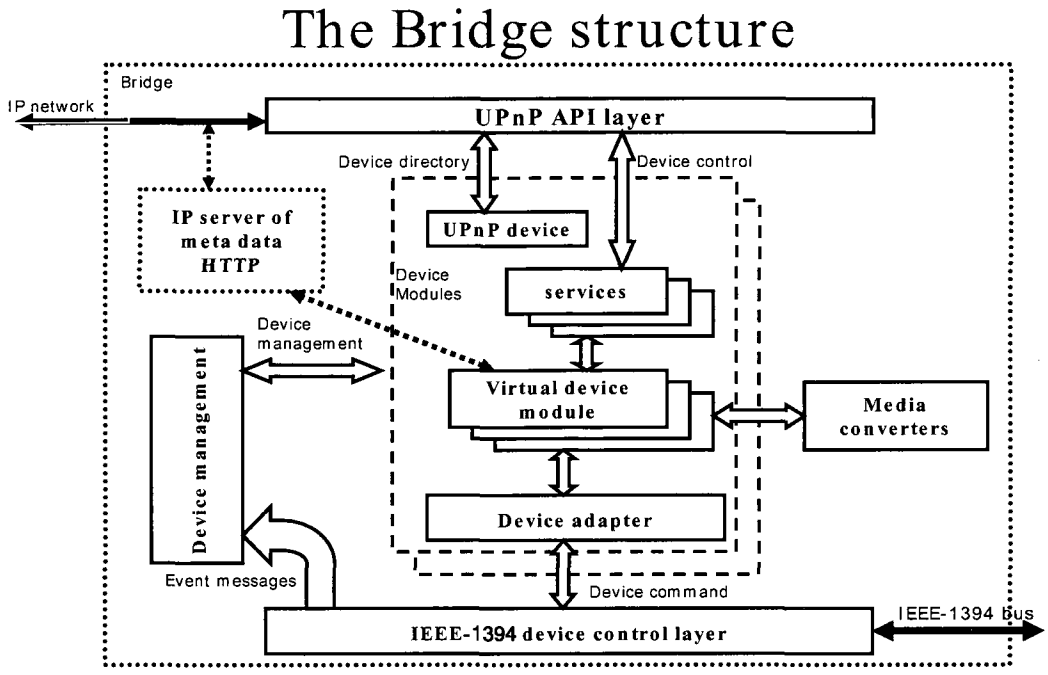

Fig. 2.3: The IEEE1394 to UPnP bridge structure [POL05]

- **Bridge Composition**

The bridge is composed of:

- The IEEE1394 device control layer, which is a low-level interface to the IEEE1394 bus and bus drivers.
- The device manager, which processesIEEE1394 bus event messages and manages device modules. It provides system resource control.
- The device module, which includes several modules that transfer messages between UPnP and IEEE1394.

Each IEEE1394 device type requires its own device module. This part consists of a few layers:

1. UPnP device, which provides its device interface information to the UPnP network, which in turn includes UPnP notification message sending and access to device and service information.
2. UPnP services, which are an interface for device control. A device can have several services for each logical device.
3. Virtual device module, which is an abstract layer between the LFD device and services. This layer provides the uniform interface for a real device and the transparent interface for data format conversion.
4. Device adaptor, which is the driver for the IEEE1394 device.

- Media converters, which are a set of format converters for static and streamed media data.
- HTTP server, which is a part of the UPnP device. It can provide HTTP access to static or streamed media data.
- UPnP API layer, which is an API for the UPnP stack.

- **Working Algorithm**

When an IEEE1394 device is connected to the Device manager, it gathers the device information and initializes related Device Module. The UPnP device in the initialized Device Module broadcasts the UPnP advertisement message to the UPnP network. From now Control Point (CP) can discover UPnP device and control it through services. For example CP can control low–level device parameters or check and set output media format. Virtual device module layer provides uniform interface from services to device.

The user interacts with the Control Point's U1 to locate and select the desired devices without knowing about device communication detail. Control Point should check data format and transfer protocol compatibility between source and target devices.

There are some limitations to this bridge which can be summarized as the following:

- The bridge adds delays to data transferring. This delay consists of bridge processing delays, format conversion delays, data transferring delays.
- The environment requiring real-time processing is only environment that can be effected by the delay.

Transferring huge volume of media data through bridge and data format conversion can require great and complex computing resources. This will require strong CPU or special hardware solution for media conversion.

2.4.1.2. General Limitations for One to One Bridge

This type of bridge solves some problems with the diversification of middleware, but most problems remain for the following reasons:

1. One-to-one bridges have a limit on scalability and there will always be a need to develop a single bridge that connects two specific middleware one-to-one when new middleware is constantly in development. The eventual complexity of the system will be too high: (n*(n-1)) / 2 where n is the total number of middleware [LEE05].
2. Sometimes the appliance must be extended for interoperability, [MOO03A] e.g HAVi and Jini by Thomson.

3. It can become possible to create more than one virtual device for the same device, due to device identification weakness [KIM08].
4. Failure to update the context (entities' information) of the device between the bridges is always a concern [KIM08].

2.4.2. One - to any Conversion

Integrating all services over the world with a single, perfect middleware is impossible [TOK02]. However, several one-to-any middleware bridges have been designed to integrate heterogeneous middleware and to create an environment in which services based on any middleware can connect to any other service transparently, and where new middleware can be effortlessly connected to this framework.

There are four home computing middleware bridges to be introduced; these bridges are either one-to-any or any-to-any bridges:

1. Multi Middleware Bridge (MMB)
2. Universal Middleware Bridge (UMB)
3. A framework for connecting home computing middleware at Waseda University.
4. HOmenetwork Middleware for Interoperability (HOMI)

2.4.2.1. Multi-Middleware Bridge for Supporting Interoperability in Home Network Environments

The Multi-Middleware Bridge (MMB) [KIM08] approach is simply cooperating with these one-to-one bridges by using the bridge repository as shown in Fig. 2.4.

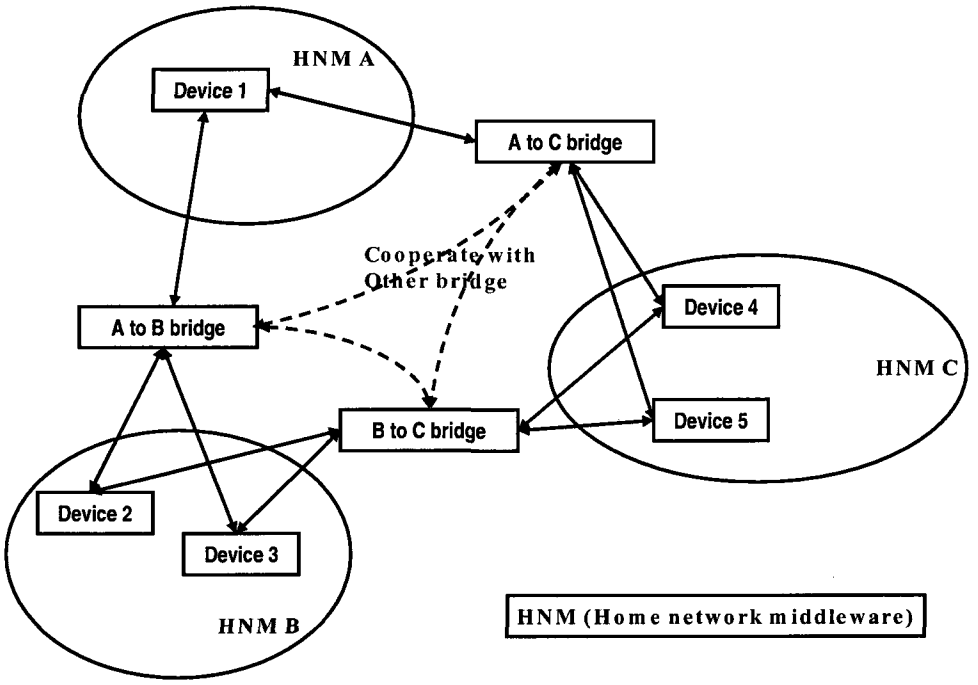

Fig. 2.4: Network topology of proposed bridge architecture

- **MMB Bridge System Components**

Each home network middleware has its devices, the bridge system joining two middleware support interoperability for such devices. The MMB bridge system cooperates with other bridges using the bridge repository [KIM08]. These bridge systems consist of: [KIM08].

1. A device interface for communication with devices using home network middleware such as UPnP and Jini.
2. A repository manager that writes data received from the device interface.
3. A monitoring service that checks integrity of data and monitors whether the repository is working.
4. A communication manager that communicates with the other repositories and update servers used to update contents in the static repository.

The bridge repository is the most important component in the MMB scheme. It has information about the device context mapping table and an organization of bridges and virtual devices. Fig. 2.5 shows the structure of individual bridge systems while Fig. 2.6 shows the structure of bridge repository.

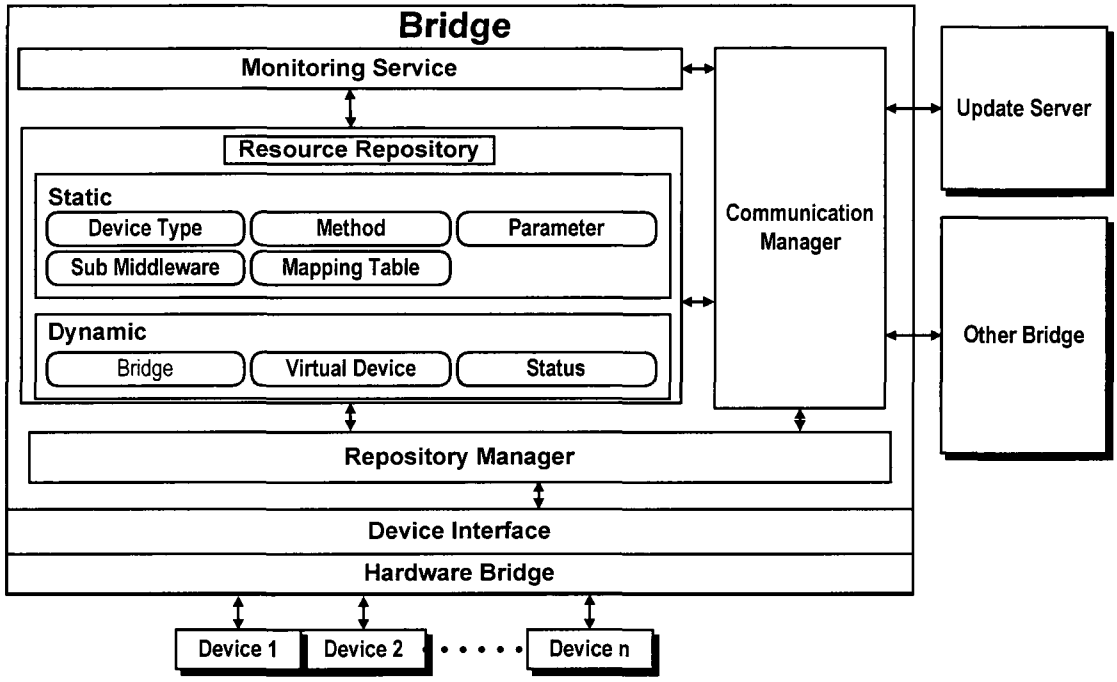

Fig. 2.5: Individual bridge systems [KIM08]

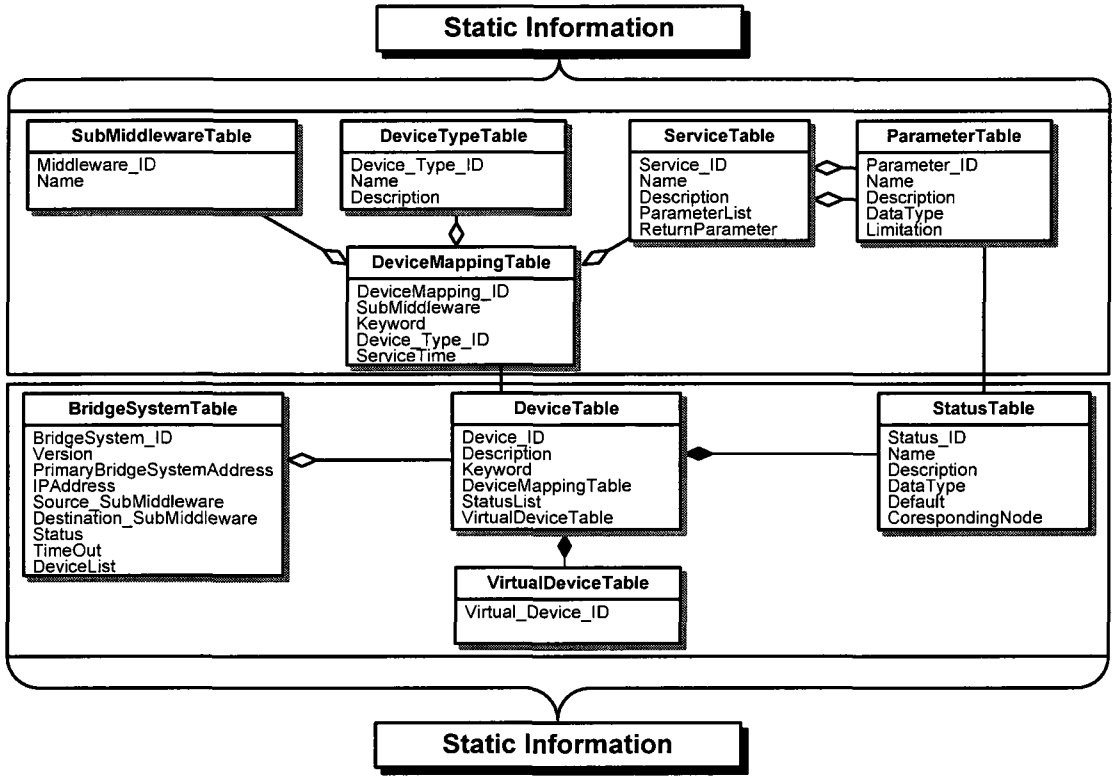

Fig. 2.6: Structure of bridge repository [KIM08]

The bridge repository structure in Fig. 2.6. is modeled by UML (Unified Modeling Language); the following arrow ——◇ represents aggregation which means ("has a"), for example DeviceMappingTable has a DeviceTypeTable and at the same time DeviceTypeTable has a meaning without DeviceMappingTable. The ServiceTable, has several parameters from ParameterTable.

The arrow ——◆ represents composition which means ("is a part of"), for example virtualdevicetable is part of DeviceTable and has no meaning without the DeviceTable.

Bridge repository consists of dynamic information and static information.

- **Static Information**

Static information [KIM08] is abstract information about middleware and device mapping. The standard of middleware changing static information is updated by communication with the update server via the communication manager.

- The middleware table has information about home network middleware.
- DeviceTypeTable has information about home network device classification such as video, lights, and air conditioning.
- ServiceTable has information about service in such device.
- ParameterTable contains the parameters used in service interaction.
- DeviceMappingTable matches home network devices to DeviceTable and Middleware Table using keywords. For example, in UPnP, UDN and UUID are the keywords.

- **Dynamic Information**

The dynamic information [KIM08] is a record of other bridge systems, home network devices, and virtual devices.

This information changes dynamically when the device connects to networks or devices invoke the service.

- BridgeSystemTable holds information about bridge systems in home network environments.
- VirtualDeviceTable contains information about virtual devices copying real home network devices in such a bridge system.

- A virtual device is an interoperable communications broker with respect to communication with real devices via another middleware. The virtual device contains the device's context, and this data synchronizes with other virtual devices in other bridge systems.

The static information described above matches with the dynamic information by means of the Device Mapping Table.

MMB has three algorithms to support interoperability [KIM08]:

1. Connection phase of the bridge system: when a new bridge connects to the network, this bridge discovers the primary bridge, which will send dynamic information data about this network to the new bridge. Next, the primary bridge sends the bridge description of the received bridge to other bridges. If the new bridge contains any other information, the primary bridge updates their static information. This connection phase was not included in the model created in chapter 4.
2. Connection phase for a home network device: when a new device joins the network, the bridge system that supports the appropriate middleware includes this device's discovery phase process. The bridge system finds another bridge system that manages the same middleware area, and negotiates control of the device that is requesting the service. This negotiation prevents the problem of generating duplicate virtual devices. Next, the bridge system creates the virtual device. This virtual device processes the discovery phase according to their middleware mechanism.

The connection phase of home network devices is illustrated in Fig. 2.7.

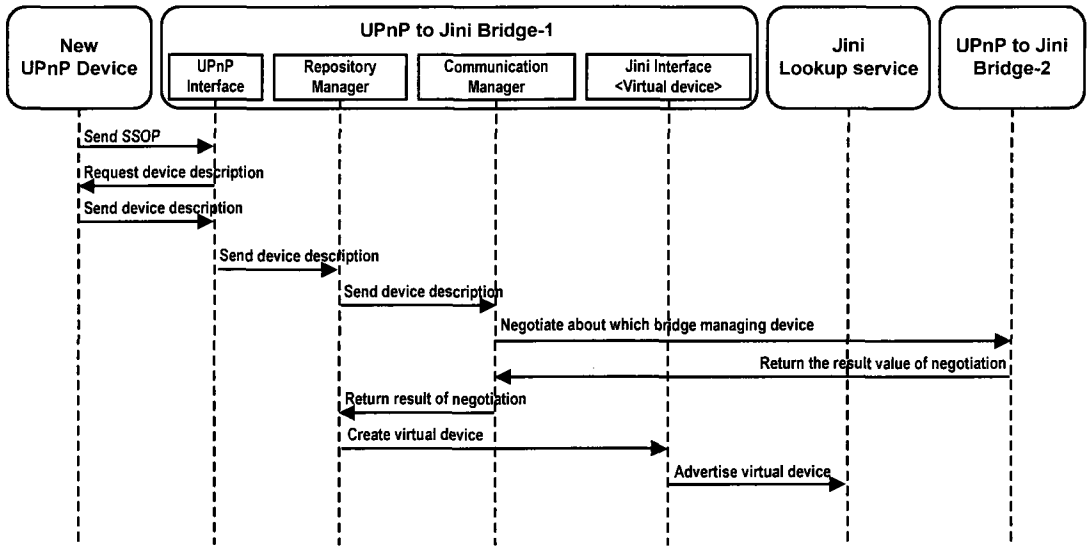

Fig. 2.7: Connection phase of home network device [KIM08]

3. Interaction phase between home network devices [KIM08] as shown in Fig. 2.8.

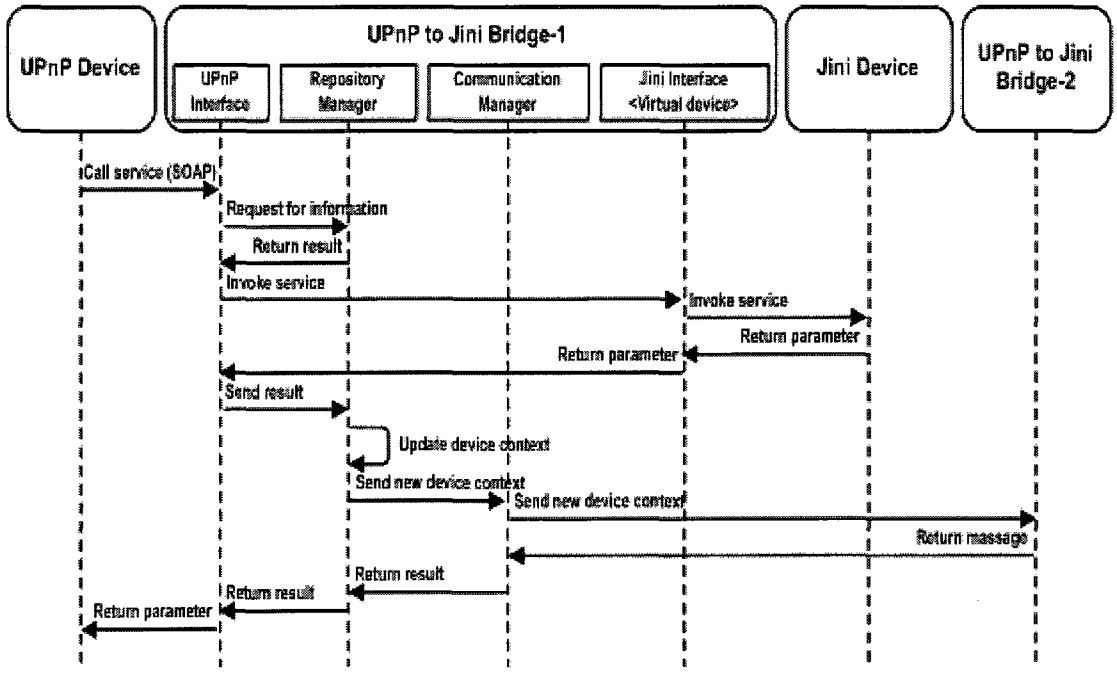

Fig. 2.8: Interaction phase between home network devices [KIM08]

The UPnP device recognizes Jini devices via the virtual device in the UPnP interface.

UPnP devices invoke service from Jini devices as shown in Fig. 2.8. This occurs through the UPnP and Jini Interfaces, and then the UPnP interface updates the device context in the repository manager, which in turn sends it to other repository managers.

It has been observed that MMB architecture manages to avoid creating multiple virtual devices from the same device, and always updates other repositories in their distributed environment with any changes in the joined device's contexts. Also, the MMB distributed environment has more than one-to-one bridge of the same networks (i.e managing the same middleware).

These two points can create complexity for the overall system and lead to more message exchanges and more time to finalize invoking service between two devices in two different forms of middleware.

From the design architecture, one can note the modification to the existing one-to-one bridges as shown in Fig. 2.5. This means less possibility of deploying the architecture, as it will increase the complexity of the whole system. This architecture is not yet implemented, nor is its performance analysis studied in the scope of quantitative analysis [KIM08].

2.4.3. Universal Middleware Bridge for Device – Interoperability in Heterogeneous Home Network Middleware

The Universal Middleware Bridge (UMB) [LEE05] [KIM08] "dynamically maps physical devices in all different middleware domains into virtually abstracted devices

in the UMB domain, and enables all home devices overlaid on heterogeneous networks to be seen as virtually the same physical devices in the same middleware domain, as well as to detect and control each other." [LEE05].

The UMB Bridge also contains the UMB message, which is a protocol defined in the Extensible Markup Language (XML). Components of the UMB system use the UMB message to communicate with each other; the UMB message is also used as an intermediate message for sending a middleware-dependent message between two different middleware networks [KIM07A].

Heterogeneous middleware creates a number of problems, including the problem of protocol conversion. The UMB is focused on overcoming this problem.

Fig. 2.9 shows the architecture of the UMB

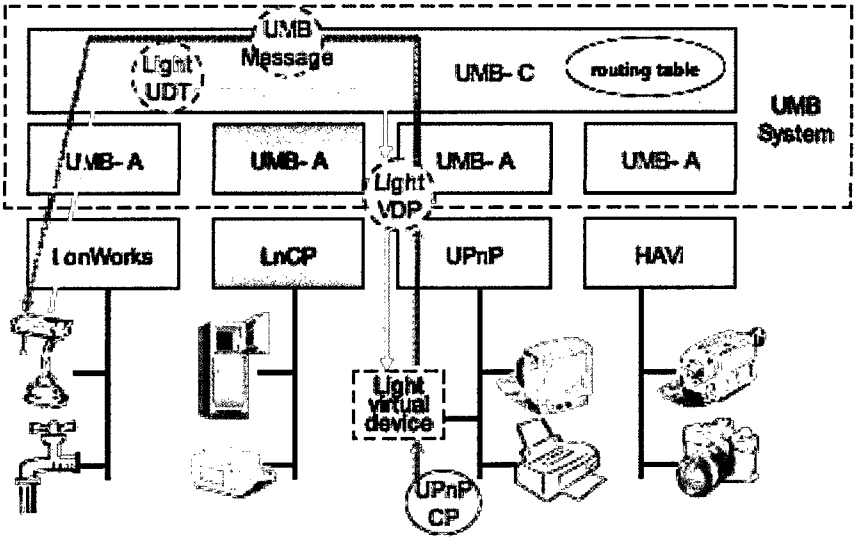

Fig. 2.9: The Architecture of Universal Middleware Bridge [LEE05]

The architecture of the UMB consists of the UMB Core (UMB-C) and UMB Adaptor (UMB-A); every middleware network should have a UMB-A to be connected to the UMB system.

The structure of the UMB-C is illustrated in Fig. 2.10.

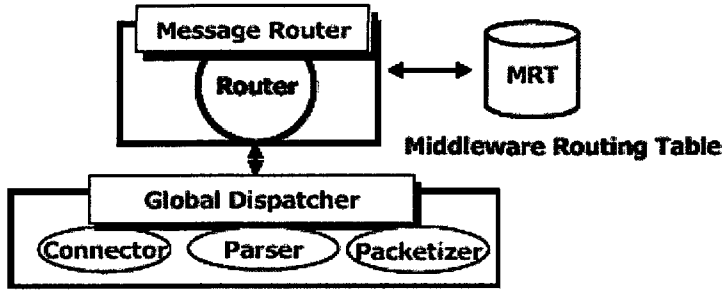

Fig. 2.10: The Structure of UMB Core (UMB-C)[LEE05]

The main functions of the UMB-C are as follows:

1. Managing UMB-As and providing a communication channel for UMB-As.
2. Modifying the middleware routing tables (MRT) and routing UMB messages to the UMB-As or any other destination

The structure of UMB-A is illustrated in Fig. 2.11.

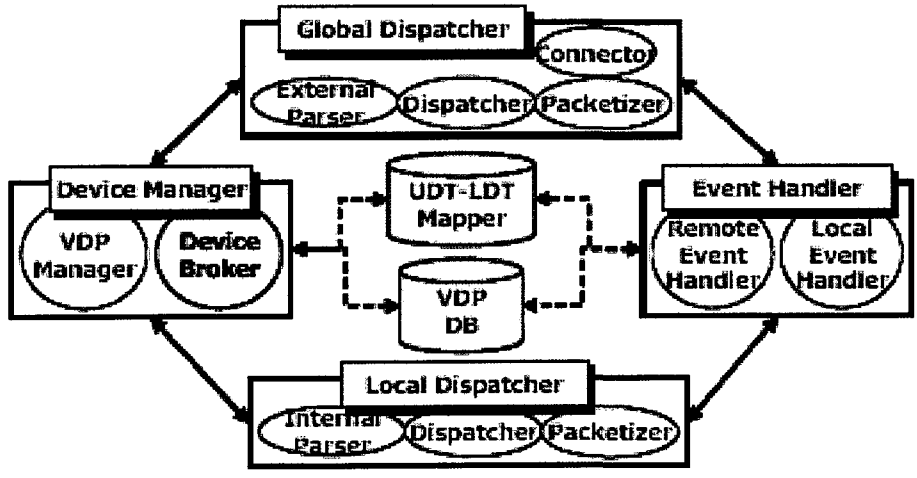

Fig. 2.11: The Structure of UMB Adaptor (UMB-A) [LEE05]

Main functions of UMA-A include:

A. Communicating with other UMB-As.

B. Managing and controlling devices in the middleware network.

C. Converting devices and messages between a middleware network and the UMB system.

2.4.3.1. The Device Conversion in a UMB-A

There are two steps for device conversion in the UMB –A:

1. When the device is plugged in with specific middleware, the UMB Adaptor overlaid on that middleware converts the physical device into the virtually abstracted one described by Universal Device Template (UDT). The Device Manager (DM) of the UMB Adaptor matches each local device's function, which is expressed by the Local Device Template (LDT) with appropriate UDT by the UDT-to-LDT Mappers [LEE05].

   The UDT is designed as an intermediate device structure so that the structure of an original device can be maintained. "The UDT defines device types and function types supported by the UMB system, as well as which types of functions should be included in each device type in extensible markup language (XML) "[KIM07A].

   The Global Dispatcher (GD) of the UMB Adaptor creates and sends a Universal Metadata Message containing the translated UDT to the UMB Core to be routed using the MRT.

   The Event Handler (EH) of the UMB Adaptor sends and receives the events generated by changing the device's status.

2. If the UMB Adaptor, which contains any other middleware, receives the exposed virtual device with UDT through the UMB Core, it creates the

Virtual Device Proxy (VDP) database, which is a S/W module in UMB-A that has a structure defined by the home network middleware, such as UPnP and announces this locally to connected devices [LEE05].

When a new device type is defined in home network middleware, it is necessary to modify the mapping rules. The mapping rules can be modified and applied dynamically, since the mapping rules are described in XML [KIM07A].

2.4.3.2. UMB Protocol Operation

The UMB protocol shown in Fig. 2.12 and Fig. 2.13 and message flow [KIM07A] has been the basis for the design of the UMB model in chapter 4 and can be summarized as follows:

1. When a device is connected to a middleware network, the UMB-A gets that device information, including its device type and function types. "The UMB-A scans the device-mapping table in order to determine a proper entry, and determines the UDT device type that corresponds to that of the newly connected device. The UMB-A also scans the function-mapping table to convert functions of the device into UDT function types. If the middleware does not have the concept mapped to the function of the UDT, the UMB-A then creates a virtual function" [KIM07A].
2. After these conversions of the device type and the function type, the structure of the middleware device is completely converted into the structure of a UDT device.
3. When a UMB-A receives a device plug-in message with the converted UDT device structure, it converts the UDT device into a device that has a structure

and properties that can be understood by the home network middleware according to the same mechanism described above.

The result of the device conversion is termed the VDP. In the UMB system where all devices in this middleware interact with this VDP and send messages to it, the message sent is then translated to UDT, then routed to the right UMB-A, where the message is converted to the actual device's format..

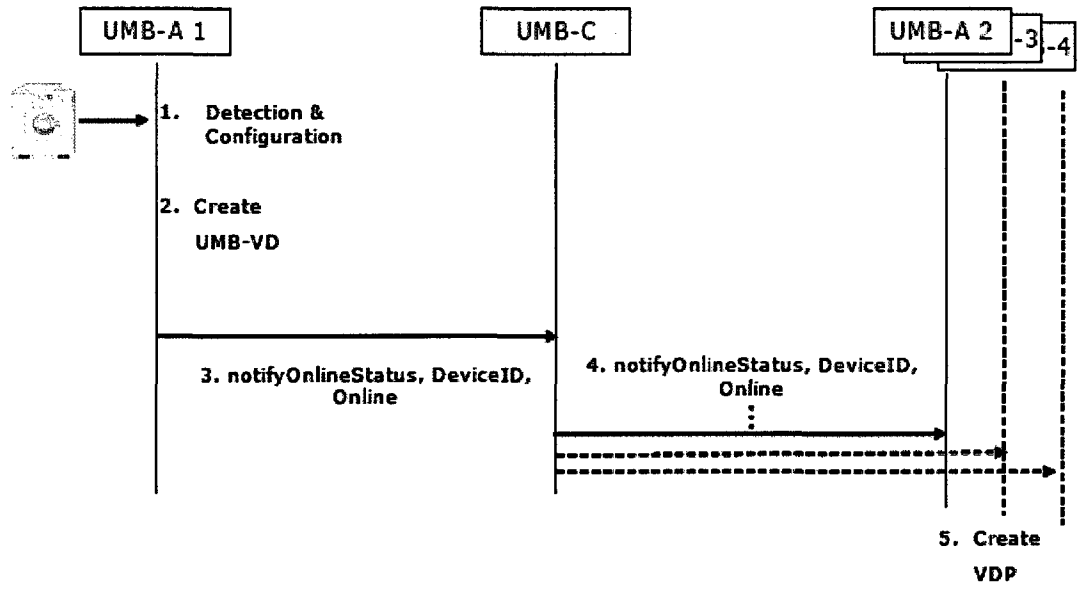

Fig. 2.12: Flow when a new device is plugged in [LEE05]

If the processing mentioned above is done properly, the physical light device on LonWorks Middleware shown in Fig. 2.13 would be registered as a virtual UPnP light device using UPnP middleware, and the UPnP control point should have exactly the same physical UPnP light device experience [LEE05].

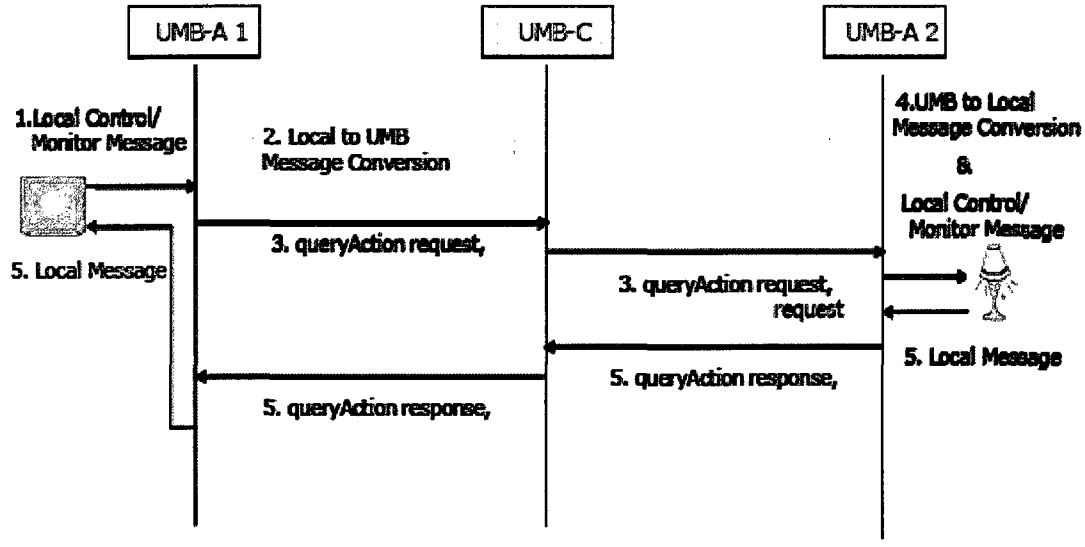

Fig. 2.13: Flow when a device is controlled and monitored [LEE05]

New middleware or appliances will always appear one after another, and the conversion mechanisms will be able to handle them by adding some XML files. This advantage of the conversion mechanisms makes the UMB system extensible [KIM07A].

2.4.3.3. UMB Mapping Rules

The translation between each type of middleware message and the UMB message is performed based on certain mapping rules for the commands and parameters. These mapping rules are described in the XML and are designed with the following three assumptions [KIM07A]:

1. A control message is composed of a command and parameter. Terms such as a 'command' and a 'parameter', can be replaced by other terms such as action, control-signal, and so on. In the case of the UMB system, an 'action' is the same as a 'command'.

2. A command in one type of middleware can be replaced by a different command in another type of middleware.
3. The number of parameters and the syntax of the parameters may be different for each type of middleware. However, the required information and the semantics of the parameters do not differ in principal.

The UMB system has been tested, and the results allegedly showed that the time cost for the device conversion and the message translation were very low [KIM07A]. We disagree with these conclusions, as more testing should be done under different loads to prove this point and form our models and simulation results; it showed that the UMB with its meta-data message in the middle might cause a delay. If faults occur in the UMB core, the appliances managed by different middleware systems will lose their connections with each other [MOO08]. To solve this problem, a fault tolerance mechanism is introduced based on the application of a real-time object replication scheme which is called the PSTR (Primary-Shadow TMO Replication). The PSTR scheme provides an approach for designing real-time fault tolerance capabilities into the TMO-structured (time-triggered messaged-triggered objects) application. To use the PSTR scheme in building fault tolerance capabilities into the UMB core, the restructure of the UMB framework as a TMO network is necessary.

A PSTR template was presented as a skeleton of a desirable fully general template. It works fine for simple TMO cases but needs to be enhanced in future research [MOO08].

2.4.4. Framework for Connecting Home Computing Middleware

The framework for connecting home computing middleware at Waseda University [TOK02] enables any appliances under any middleware control to communicate with any other appliances.

It uses and deploys service from home appliances without special knowledge of diversification of networks forms or middleware [MOO03A].

The basic framework concept for connecting home computing middleware is that; each middleware network has Virtual Service Gateway, Protocol Conversion Manager, and Virtual Service repository as shown in Fig. 2.14.

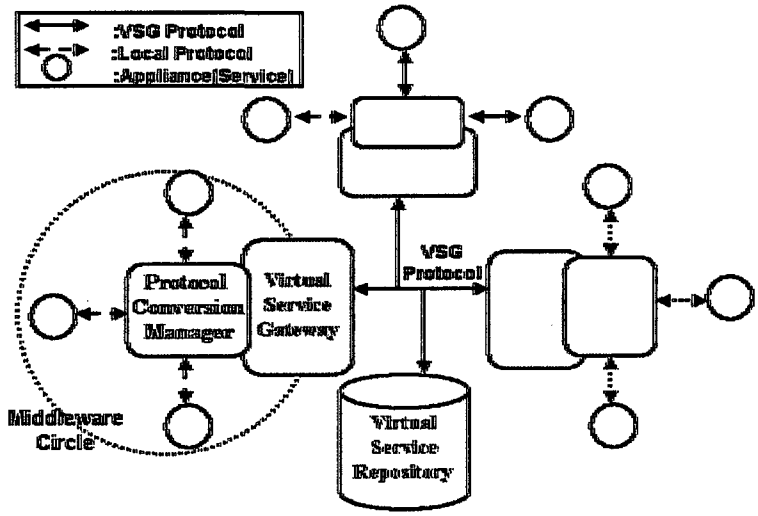

Fig. 2.14: Connecting Middleware [TOK02]

2.4.4.1. Virtual Service Gateway

A Virtual Service Gateway (VSG) which connects one middleware to another middleware uses a certain protocol to decide the type of service information, such as

interface, location and data. The prototype of the framework is implemented with SOAP (Simple Object Access Protocol).

2.4.4.2. Protocol Conversion Manager

The Protocol Conversion Manager (PCM) converts the local middleware protocol to the protocol of VSG, and also converts VSG into a local middleware component. The PCM has two proxy modules (as shown in Fig. 2.15):

1. The Server Proxy module (SP) which provides the interfaces of remote services to the local services.
2. The Client Proxy module (CP) which converts the interfaces of local services into the VSG services.

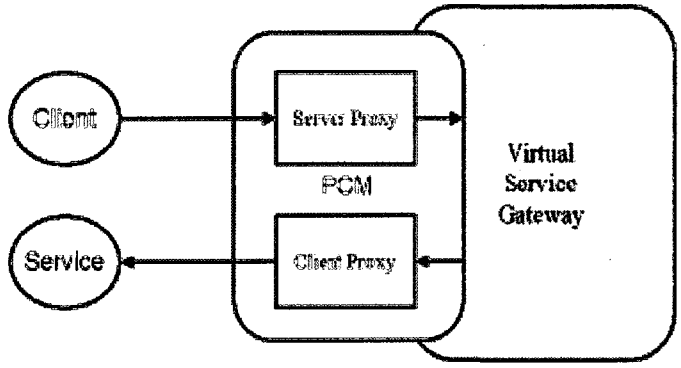

Fig. 2.15: Proxy Modules [TOK02]

3. "Virtual Service Repository, which records locations and functions of services, is a virtual database that has a lot of information about heterogeneous services such as service locations and service contexts. The VSG and the PCM use this component to detect services or aware contexts" [TOK02].

The prototype of the framework is implemented with Java, Linux and SOAP. Because it is an XML/HTTP-based protocol, it is simple and easy for implementation, and light-weight for networked system. Fig. 2.16 shows the protocol conversion between Jini and X10.

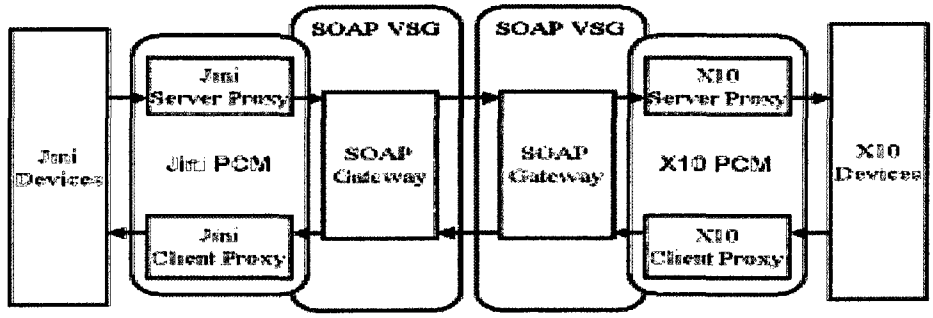

Fig. 2.16: Protocol Conversion between Jini and X10 [TOK02]

Some difficulties were discovered during the experiments, such as multimedia data conversion and dynamic service activation, which is due to the limitation of HTTP.

"HTTP is inherently a client/server protocol, which does not map well to asynchronous notification scenarios. Additionally, current HTTP must run over TCP, and a TCP stack is large and complex. This can be an issue in small devices or appliances with stringent memory and processing requirements" [TOK02].

SIP provides end-to-end security and can carry a flexible payload. These features are suitable to connect end to-end services. Also, SIP supports asynchronous calls. SIP may be more suitable than other protocols such as HTTP for service integration. But the problem is the low popularity of the SIP [TOK02].

There is a resemblance between the ideas of this approach and UMA; for this approach the authors used a Virtual Service Gateway for each middleware, while UMA uses a "UMA-A" adapter for each middleware. The protocol conversion

manager in this approach is almost the same idea as the UMB meta message in the UMB system, which is considered an intermediate language between different middleware. CP and SP in this approach resemble the remote event handler and the local event handler in UMB approach.

This approach is relatively simple but cannot automatically support dynamic service activation by combining any functions of any appliances and the device's discovery scheme, without user intervention [MOO03A].

2.4.5. Home Network Middleware for Interoperability

Home network Middleware for Interoperability HOMI [KIM06] allows users to build and organize their scenarios through the interfaces. It is also trying to solve semantic problems caused by the difference of heterogeneous middleware interfaces.

HOMI consists of five main modules as shown in Fig. 2.17:

1. Agent Manager: When each middleware Agent discovers services in a network, after analyzing service names, interface names, argument names, and the device status, it converts them to a Common Description format, and then sends them to Agent Manager.

   Agent Manager also defines other templates that agents follow. Because these templates are all defined by XML, they are independent of any particular language or OS. Agents and Agent Manager are reliable and stable because they communicate through TCP/IP.

2. Standard Method Manager: Standard Method Manager notifies each Agent of standard methods related to each middleware. The Agent creates a service

if it is one of services provided by HOMI and defined by standard methods, and notifies its network.

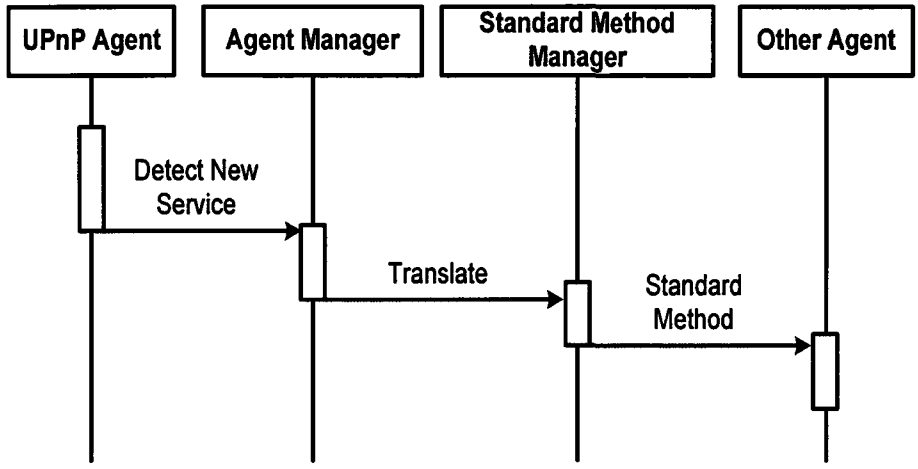

Fig. 2.17: Transmission Diagram of Standard Method

3. State Manager: For each middleware, each Agent in HOMI monitors the status of appliances under its control and notifies the State Manager of the HOMI server of the result.
4. HOMIL Analyzer: An event driven script language called HOMIL (HOMI Language), this language is handy and easy to use when constructing scenarios. Scenarios constructed with HOMIL are sent to HOMI after being converted to an XML format through the HOMIL parser.
5. Context/Event Manager: Context Managers and Event Managers always operate in pairs.

   After classifying scenarios according to contexts, the Context Manager transmits them to the Event Manager. The Event Manager manages scenarios for each context with queues and triggers an appropriate scenario when the conditions in the context of the scenario are satisfied.

The result of the executed scenario changes the status of the device, and the table of state variables of State Manager is updated due to the changed status. This change might generate an event to trigger the next scenario. Fig. 2.18 illustrates the HOMI Architecture.

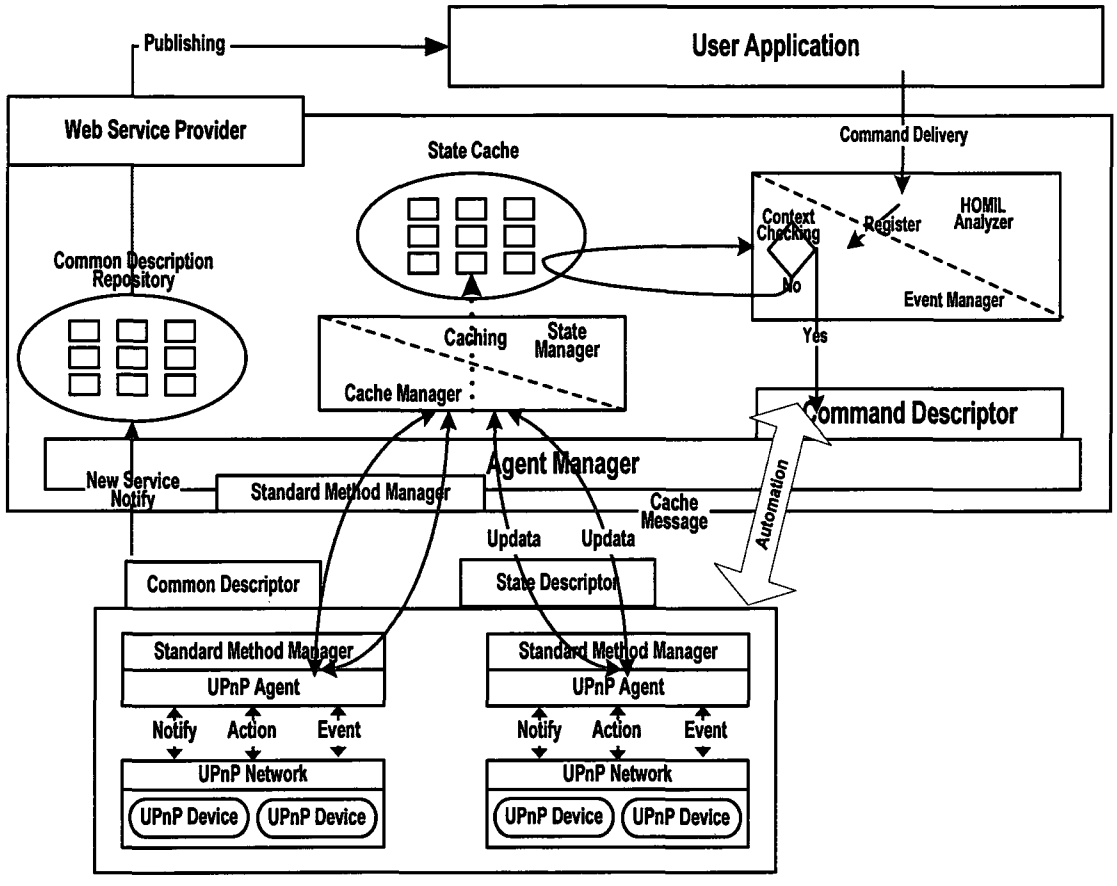

Fig. 2.18: HOMI Architecture

HOMI can distribute each Agent into various servers in order to solve the bottleneck problems encountered in a centralized architecture. Also, all protocols have an open architecture using XML and provide utility classes and APIs based on C++ and JAVA to assist in developing agents for newly defined middleware. HOMI has a load balance function using agent manager [KIM08].

HOMI needs a lot of user interaction and program development to introduce more scenarios and new APIs for developers to develop new middleware agents.

Scenario builders which support users in conveniently constructing scenarios, and new APIs for developers to quickly develop new Agents for middleware, are needed [KIM06].

"Fault tolerance of the HOMI server using a centralized mechanism is a major issue. A fault tolerance system which can operate when the HOMI server does not operate by recognizing high-performance appliances under home network and distributing important services into high-performance appliances should be investigated in the future"[KIM06].

2.5. Summary

We conducted a comprehensive analysis of the several one-to-any bridges, and from this analysis we will highlight some points:

T. Nakajima's framework from Waseda University proposed a virtual overlay network that has a one-to-any protocol conversion approach. It is a simple approach but it has some limitations as it cannot automatically support dynamic service activation without user intervention [LEE05]. It can globally manage the entire home, but applications or appliances should be modified in order to enable interoperability.

The UMB approach enables interoperability without any modification of appliances under heterogeneous middleware and globally manages the entire home network.

UMB provides interoperability by utilizing the unified protocol which integrates all protocols in home networks with a UDT (Universal Device Template); this is a one-to-any bridge [KIM08].

UMB automatically converts a protocol which is specific to a certain middleware into other middleware-specific protocols by mapping the middleware-specific protocol into meta-protocol defined by UMB via a UDT-LDT mapping relation; this scheme can efficiently support many middleware. However, the design of meta-middleware merging all existing middleware is difficult. In case of Waseda's approach, a middleware-specific protocol is also converted into other middleware-specific protocols automatically via a java-assistance tool. [LEE05].

HOMI utilizes two managers: the agent manager and the context manager. Each manager helps to reduce message traffics and provide home automation, respectively. Fault tolerance for a HOMI server using a centralized mechanism is a major issue. UMB and HOMI are weak in the area of supporting distributed traffic [KIM08].

MMB cooperates with these one-to-one bridges by using a bridge repository, and based on their approach, a change to the one-to-one bridge is required

We have observed that MMB architecture manages to avoid creating double virtual devices from the same device, and always updates other repositories in their distributed environment with any changes in the joined device contexts. The MMB distributed environment also contains more than one-to-one bridges of the same networks (i.e. managing the same middleware). These two points can create complexity for the overall system and lead to more message exchanges and more time required to finalize invoking service between two devices in two different middleware.

Chapter 3: Common Application Programming Interface Architecture Bridge

3.1. Introduction

In order for appliances to be able to interoperate in a network environment, they must have at least the following components [KIM06]:

1. Services Description, which is a service specification that can be understood by human users.
2. Service (Action) which is a function performed by the service
3. A state where appliances must have their own states and provide mechanisms that notify the connected devices of its status.

It is possible for an appliance to interoperate with other appliances as long as this kind of mechanism is provided either from outside the appliance by adding external systems or from inside it.

In this thesis, a proposal for a Common Application Programming Interface Architecture Bridge (CAPIA) is made; it will facilitate the communication between different appliances under different middleware without the modification of either the appliance or the user interface to program scenarios as in the case of HOMI architecture.

3.2. Common Application Programming Interface Architecture

CAPIA is reliable and stable because the communication between devices under different middleware occurs through TCP/IP, which is a reliable protocol. Moreover, the proposed CAPIA can be either centralized or distributed depending on the home network and the number of devices connected to the home network, and the load expected. Fig. 3.1 demonstrates the CAPIA Bridge connecting different devices under different middleware.

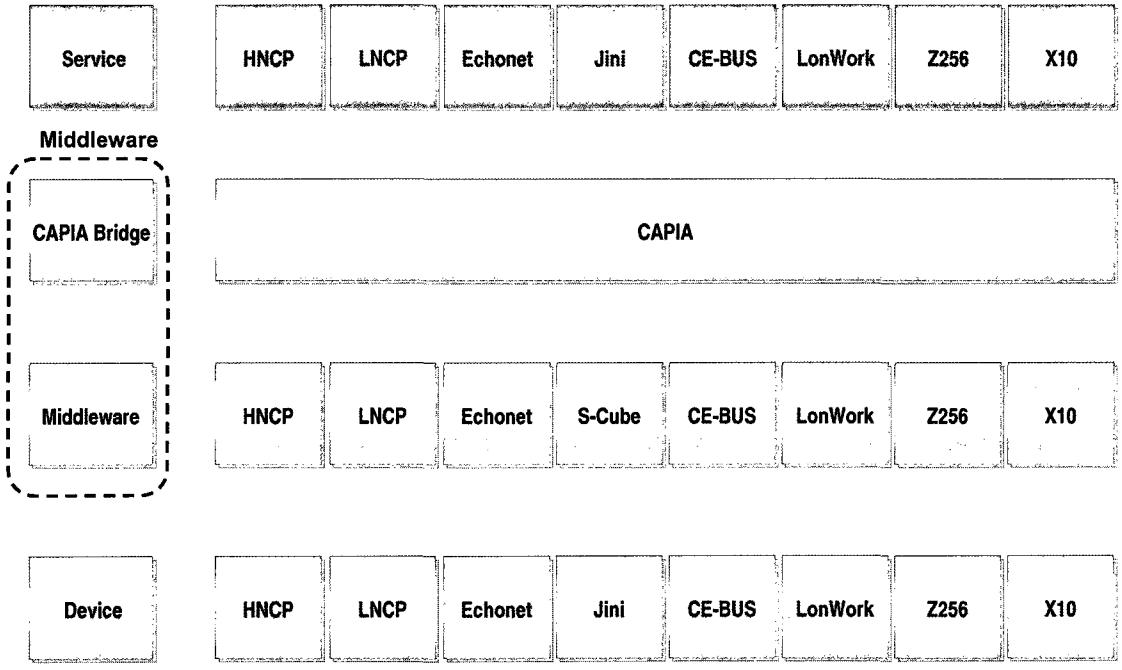

Fig. 3.1: CAPIA Bridge connecting different devices under different middleware

The coming sections will show the architecture of the CAPIA Bridge, followed by a step-by-step descriptive introduction to show how devices using UPnP middleware will connect with a Device in a Jini middleware.

A simulation model is designed which uses Arena software simulation tool to simulate the CAPIA Bridge, where a task's total time is recorded and compared with

MMB and UMB models, under different loads. Fig. 3.2 shows the centralized architecture of the proposed bridge.

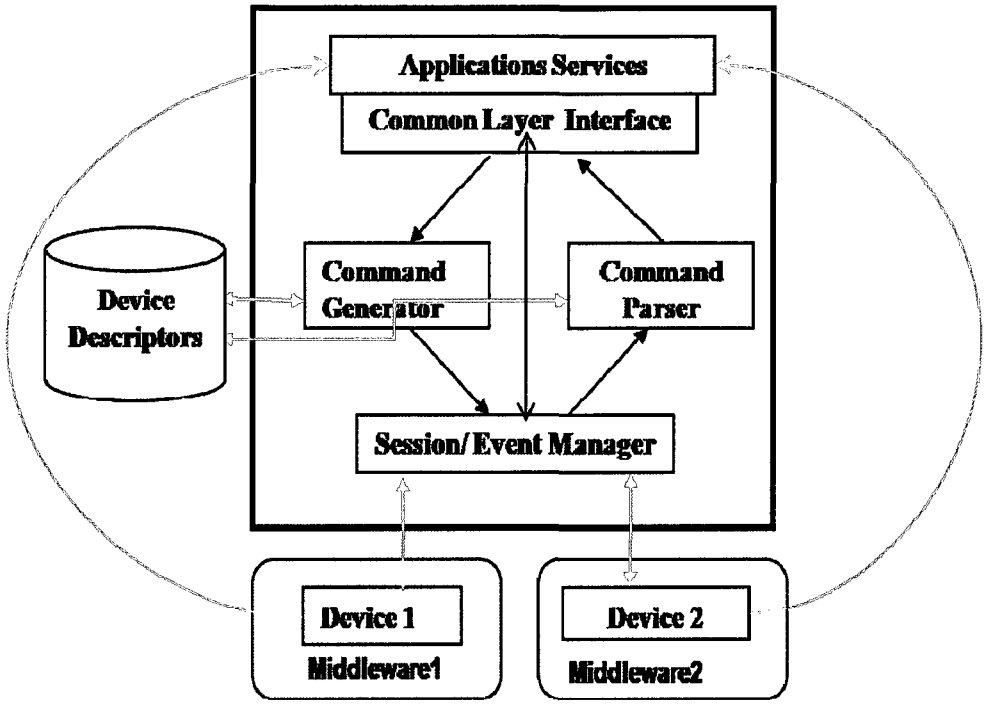

Fig. 3.2: Centralized CAPIA Bridge

3.3. CAPIA Bridge Components

The proposed CAPIA Bridge consists of five main components:

1. Application Interface
    a. Application Services
    b. Common Layer Interface
2. Command generator
3. Command parser
4. Session manager
5. Device Descriptors

3.3.1. Application Interface

The CAPIA Application Interface consists of application services, and a common layer interface.

3.3.1.1. Application Services

Several Service Discovery Protocols (SDPs) are proposed as the part of the coordination architectures that ensures device interaction with the ultimate aim of simplicity, seamlessness and scalability of the device inter-operability.

In a service discovery world, a printer for example invokes service discovery to find available devices, evaluates their characteristics, chooses a device, and initiates a print job. A PDA in desperate need of additional storage searches for a remote file storage service, off-loads some of its content to free the available memory, roams away, and then returns to claim the files.

Virtually anything can be a client or a service-client that needs things and services to be provided to them. Connecting the needy clients and available services is the aim of the service discovery.

These technologies directly attack the "I don't know where you are" and "I don't know how to talk to you" issues of client/server. There are many competing, incompatible service discovery technologies [HEL02].

Emerging SDPs include Jini, Universal Plug and Play, and Salutation [LEE02]. Appendix A contains a table that compares these discovery protocols.

CAPIA enables different communicating devices with their different discovery protocols to register themselves. Each is assigned a unique address; this address can have the first letter of the middleware, first letter of the device and an address. For

example, a HAVi camera can be assigned HC001, and a UPnP camera can be assigned UC002, and so on. The device addresses along with their services are saved in device directory. This area will be investigated further in future research; we might also propose a unique global address for each device without considering the device type and middleware.

Each device will discover the available devices and identify the one from which it requires service to be invoked.

Our proposed architecture allows each device to use the entire functionality provided by the CAPIA, and to identify the following parameters:

1. The device type/address, the device middleware, and the device status.
2. The receiver device/address to be controlled, its middleware, and the service requested.

The Application services then fill the table Sender/Receiver in the Common layer interface.

3.3.1.2. Common Layer Interface

The common layer interface (CLI) consists of Sender/Receiver tables, Common Command tables, a device controller and backup memory. CLI has grouped all the common functions and services for different middleware and devices into three different common groups (three main tables) according to the different home services:

1. Home entertainment tables that handle VCRs, TVs, cameras, etc.
2. Control appliance tables that have commands related to "white" goods (refrigerators, washing machines, microwaves, etc.)

3. Information data tables, which have the common commands related to PCs, printers, and scanners.

When the common services/commands are grouped the search for a particular service and conversion time from a certain middleware format to a different middleware format will be quicker. For example, suppose a UPnP device wants to turn on a HAVi device. Since this is a common command, the common layer interface will search the home entertainment table for the equivalent of a UPnP turn-on command in the HAVi format, retrieve this command, and fill the Sender/Receiver table as shown in Table 3.1.

The distinctive functions/services are uniquely handled through the Command Generator, the Command Parser, and the Device Descriptors as will be explained later. This approach makes the conversion faster and leads to less future development from the developers' side.

Table 3.1: Sender / Receiver table in the Common Layer Interface

Sender					Receiver				
Middleware	Device/ Address	Command/ Service	Status	Session#	Middleware	Device/ Address	Command/Service	Status	Session#
UPnP	Monitor/ UM001	Turn ON	ON		HAVi	Video/H V001		OFF	

The common layer interface also communicates with other common layer interfaces in the distributed environment through the device controller. The purpose of this communication is to negotiate who is going to manage the incoming service request. Every device controller checks the available cache memory in the command generator and in the command parser, and the server that reports the greatest cache

memory will handle the new request. However if a server already has a session with the appliance requesting service and has sufficient cache memory in both the command generator and command parser, then this server will have priority over the other servers. When the receiver device (HAVi device) requests a service from the sender device (UPnP), then the same procedure mentioned above will be repeated, but there will be a session opened between these two devices, and the application services will fill the Sender/Receiver tables, since in this case the HAVi device will be a sender device.

The distributed environment of CAPIA has an advantage in fault tolerance; if a server is not operating due to any fault of its components, then other servers can take over, and continue the sessions that are active. Once the Sender/Receiver tables in CLI are completed or have changes in any of their fields (e.g. status of a device is changed), then the tables should be compressed and backed up and saved periodically in CLI backup memory. Messages like the "keep alive" messages should be exchanged periodically between the servers, as well as a copy of the updated Sender/Receiver table.

If fault is detected in one of the servers, then the other servers negotiate which one will take over according to their available cache memory as mentioned before, retrieve the compressed Sender/ Receiver tables of the two active sessions, and resume the interrupted service. The user should not feel any interruption of service since this can take place in less than a millisecond.

3.3.2. Command Generator

The command generator (CG) is responsible for generating the command to be sent to the device connected to the different middleware. Usually this command is one of the unique commands that were not found in the common commands tables in CLI. The following summarizes the operation of the command generator:

- When an application requests a specific command for a selected device, the command generator checks the device descriptor.
- The CG imports the specific table that contains this device's commands, and imports its translation to the receiver device middleware.
- The CG saves it in the command generator cache (CG Cache)
- The CG extracts the translation of the requested command that will be executed by the receiver middleware and device.
- The CG fills the Sender/Receiver table with the translated command
- The CG passes the command to the session manager.

3.3.3. Device Descriptors

The device descriptors will have conversion tables between different middleware containing device types, service commands, parameters, etc. They are constructed with XML. The command generator and the command parser will use these tables to convert commands between different middleware.

The device structure is different from one middleware to another, as shown in the Table 3.2 [KIM07A].

Table 3.2: Device structure of home network middleware

Middleware	Device structure
HAVi	DCM FCM Command
LnCP	Device Command
LonWorks	Object Command
UPnP	Root device Embedded device Service Action

The device structure of HAVi has a device control module (DCM), a functional component module (FCM) and a command module. The device structure of UPnP is a root device that can have several embedded devices, and those embedded devices can have recursive embedded devices. The root device is a representative device, and the embedded device is an accessory device in the root device [KIM07A].

The device descriptor will consist of conversion tables for different devices from one middleware to another middleware control format and device structure. Since all the conversion will be done by XML, any upgrade to a device or any new device /middleware can be integrated to CAPIA by simply adding the conversion tables in the device descriptor.

3.3.4. Session Manager

The session manager is responsible of handling the different sessions opened between different communicating devices. CAPIA can handle multiple sessions with the device.

A session will be opened between the two communicating devices, and the session manager will update the Session/Event table. The Session/Event manager will then route the translated command to the receiver device using all the information stored in the Common Layer Interface (Sender / Receiver) tables. When the receiver device changes status, it updates the session/event manager, which in turn will fill the device status column in the session/event table as shown in Table 3.3.

Table 3.3: Session / Event table in session manager

Session /Event Table					
Sender Device	Sender Device Status	Receiver Device	Receiver Device Status	Session #	Session Active
Monitor/UM001	ON	HV001	ON	1	1

The session manager updates the common layer interface (Sender/Receiver table) with the session number and state of the device as shown in Table 3.4.

Once the Sender/Receiver table is completed, then it should be compressed and saved in the CLI backup memory. CLIs of different CAPIA servers exchange the "keep alive" messages with the compressed tables, and once the Sender/Receiver table is updated due to change of status or termination of a session, then it should be

compressed and backed up in the backup memory (i.e. it will overwrite the previously backed up table).

When a device requests service from a device that already has a session number, the status of this device will be known and this will speed up further service requests from that device and show its availability.

Table 3.4: Updated table in the Common Layer Interface

Sender					Receiver				
Middleware	Device/Address	Command /Service	Status	Session#	Middleware	Device/Address	Command/Service	Status	Session#
UPnP	Monitor/UM001	Turn ON	ON	1	HAVi	Video/HV001	Turn ON	ON	1

3.3.5. The Command Parser

The command parser is responsible for parsing the receiver device's replies and commands.

The command parser rule is almost the same as the Command Generator rule but in the reverse direction, i.e. when the receiver device in the receiver middleware wants to communicate with the sender device in the sender middleware, the same steps explained above will take place.

First, the service requested will register itself in the application services, where it can discover the registered devices; then, the application services will fill the Sender/Receiver tables and discover that there is a session already open between these two devices. The CLI will check if the service requested is a common command

or not, and if the service is common, it will be imported and fill the Sender/Receiver table as shown in Table 3.5.

Table 3.5: Sender/Receiver table with new service request in the CLI

Sender					Receiver				
Middleware	Device/Address	Command /Service	Status	Session#	Middleware	Device/Address	Command/Service	Status	Session#
UPnP	Monitor/UM001	Turn ON	ON	1	HAVi	Video/HV001	Turn ON	ON	1
HAVi	Video/HV001	Display record	Play		UPnP	Monitor/UM001	Display record		

In case the command is a unique command, the CLI will pass the Sender/Receiver information to the command parser, which will perform the following steps:

1. Check the device descriptor.
2. Import the table that contains this device's commands as well as its translation to the receiver middleware.
3. Save it in the Command Parser Cache (CP Cache) .
4. Extract the translation of the requested command that will be executed by the receiver middleware and device.
5. Fill the Sender/Receiver table with the translated command.
6. Pass it to the session manager.
7. The session manager will assign session # and route the command to the receiver and update the sender/receiver table as shown in Table 3.6.

Table 3.6: Sender/Receiver table in the Common Layer Interface

Sender					Receiver				
Middleware	Device/Address	Command/ Service	Status	Session#	Middleware	Device/Address	Command/ Service	Status	Session#
UPnP	Monitor/UM001	Turn ON	ON	1	HAVi	Video/HV001	Turn ON	ON	1
HAVi	Video/HV001	Display record	Play	2	UPnP	Monitor/UM001	Display record	Displaying	2

When the UPnP monitor and the HAVi video want to communicate, all the commands and their translations concerning these two devices will be saved in the CG Cache and the CP Cache, which will make future communication between these two devices faster. In this case, the command generator and command parser will skip the step where they access the Device Descriptor and search for the matching table for the service requested and its middleware.

3.4. The Operation CAPIA

The operation of the CAPIA Bridge can occur in either a centralized or distributed mode.

3.4.1. Centralized Bridge

If CAPIA is configured to be centralized, then when a device requests a service from the CAPIA, it registers itself in application services where the following actions take place:

1. Check the device directory, analysis of service requested, interface names, argument names, and the device status.

2. Addition to the Sender/Receiver table with the right arguments.
3. CLI verification of the common commands section.

If the service is a common service, it gets the equivalence (translation) of the command and sends it to the Sender/Receiver table in the common layer interface.

In case the service is a unique service, after filling the table in the common layer interface with the available information, it sends the information to the Command Generator, where it imports the required translated table from the Device Descriptor. These templates are all defined by XML; they are independent of a particular language or OS.

3.4.2. Distributed Architecture:

In the case of distributed architecture, separate CAPIA will be installed in different servers as shown in following Fig. 3.3. The CAPIA supports intercommunication through TCP/IP, which makes it reliable. The physical connection between CAPIA can be through Ethernet inside the house. The distributed approach solves the overhead problem which may occur in centralized architecture in case of heavy home network loads or any faults occurring in the server.

When a device in the distributed environment requires a service, it sends its request to the CAPIA servers, the server that first receive the request will check the device directory, if this device is new and not assigned an address, it will notify the other servers, and assign an address to it, update the device directory then send the updated directory to the other servers. In the common layer interface, the device controllers on different servers will check their cache memories and convey them to each other. The CAPIA with the greatest cache memory will handle this service.

The server that has a session going with a device will have priority for a subsequent session requested by the same device, if it has enough cache memory.

After the servers negotiate with each other, a server will be selected to manage the service request; steps 1 to 3 described for the centralized bridge will be repeated in the selected server.

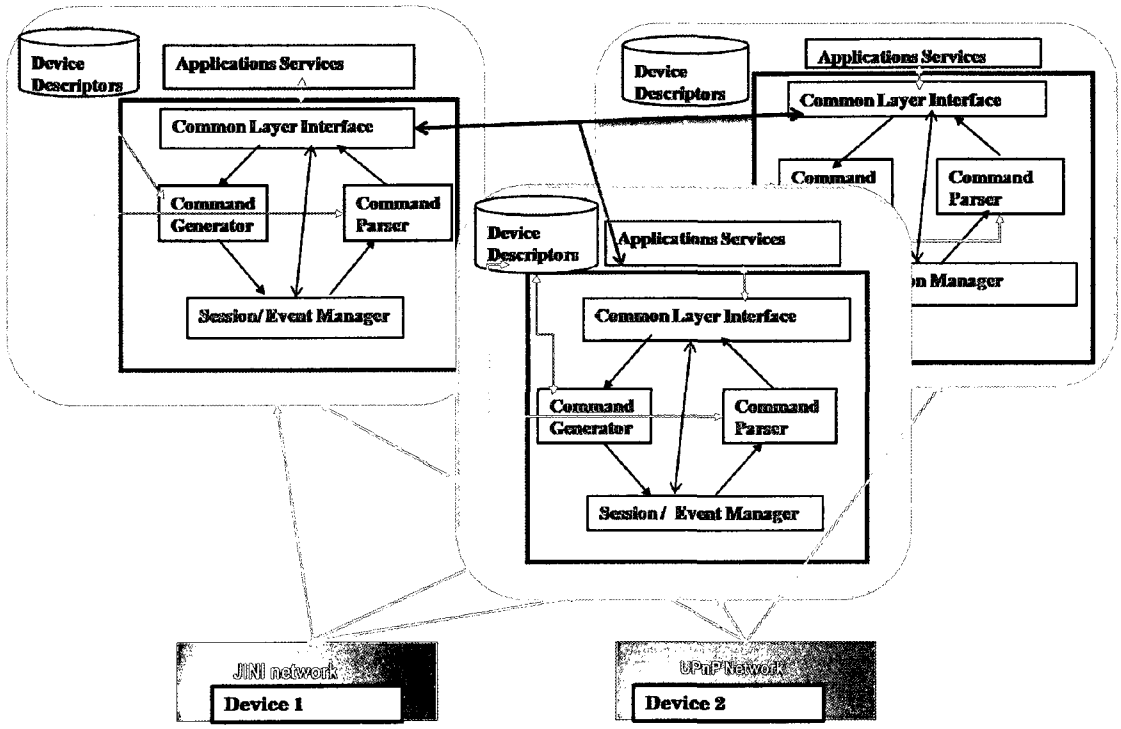

Fig. 3.3: Distributed CAPIA Bridge

3.4.3. Steps for Connecting UPnP Device with Jini Device

The following section is a step by step operation of the distributed CAPIA, clarifying how a device under UPnP middleware communicates with a device under Jini middleware.

Device 1 registers itself (or its network) and has a service requested from the application services.

The interface will check if it is a common command or a unique command. If the command is a common command, then it will get the command from the common commands tables, which include three tables according to the different home services:

1. Home entertainment tables that handle VCRs, TVs, cameras, etc.
2. Control appliance tables that handle commands related to "white" appliances (refrigerators, washing machines, microwaves, etc.)
3. Information data tables, which handle the common commands related to PCs, printers and scanners.

These three common tables will have the common commands used and their translations to different middleware. The Common Layer Interface (CLI) will get the translated command corresponding to the service requested and complete the rest of the information.

1. Sender information (device name, middleware, translated command, status)
2. Receiver information (middleware, device, and translated command)

The CLI will then pass the table to the Session/Event Manager.

Fig. 3.4 shows in detail an overview of the functionality of CAPIA architecture.

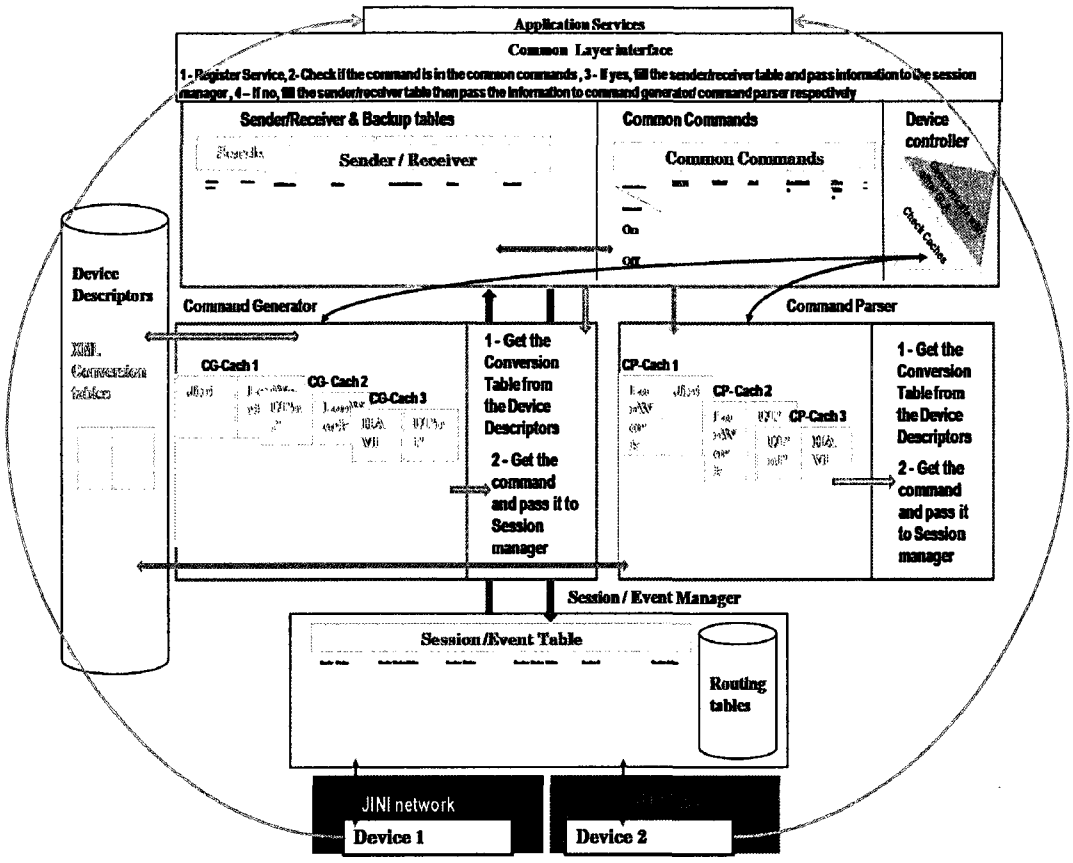

Fig. 3.4: Overview of the functionality of CAPIA Bridge

If the received command is a unique command, then the following steps will take place:

1. The CLI will pass the request to the Command Generator.
2. The command generator will search for the table that contains the translated command, then it will import the table from the Device Descriptor and save it in the cache memory (CG – Cache); after that, the command generator will extract the required translated command from the saved table in the CG-Cache memory.
3. The Command Generator will complete the Sender/Receiver table with the translated command and pass it to the CAPIA session manager.

4. In the session manager, a session will be opened containing the following information: sender device, sender device status, receiver device, receiver device status, session number and session activity.
5. The session manager will route the command to the receiver device, which will receive the translated command according to its middleware format.
6. The receiver device will update the session manager with its status after executing the command.
7. The session manager will update the tables in the common layer interface with the session number and the receiver device status.
8. When Device 2 requests to send a command to Device 1, the sender will be a receiver (Device 1) and the receiver will be a sender (Device 2)
9. Device 2 sends the requested service/command to the application services, where the interface will discover that there is a session number. The CLI checks if the command is in the common commands tables, and if the command is found the CLI will extract it and pass everything to the session manager, which assigns a session number and routes the message to Device 1.
10. In case the command is unique, the common layer interface will pass the message to the Command Parser, which will import the conversion middleware table and save it in the CP-Cache. Then the CP extracts the command translation, fills the Sender/Receiver tables and passes it to the session manager, which will assign a session number and route the command to Device 1.

Fig. 3.5 shows the protocol used for setting up a connection between Device 1 in the UPnP middleware and Device 2 in the Jini middleware for the proposed architecture.

This scenario is for a distributed CAPIA where the services requested were unique services.

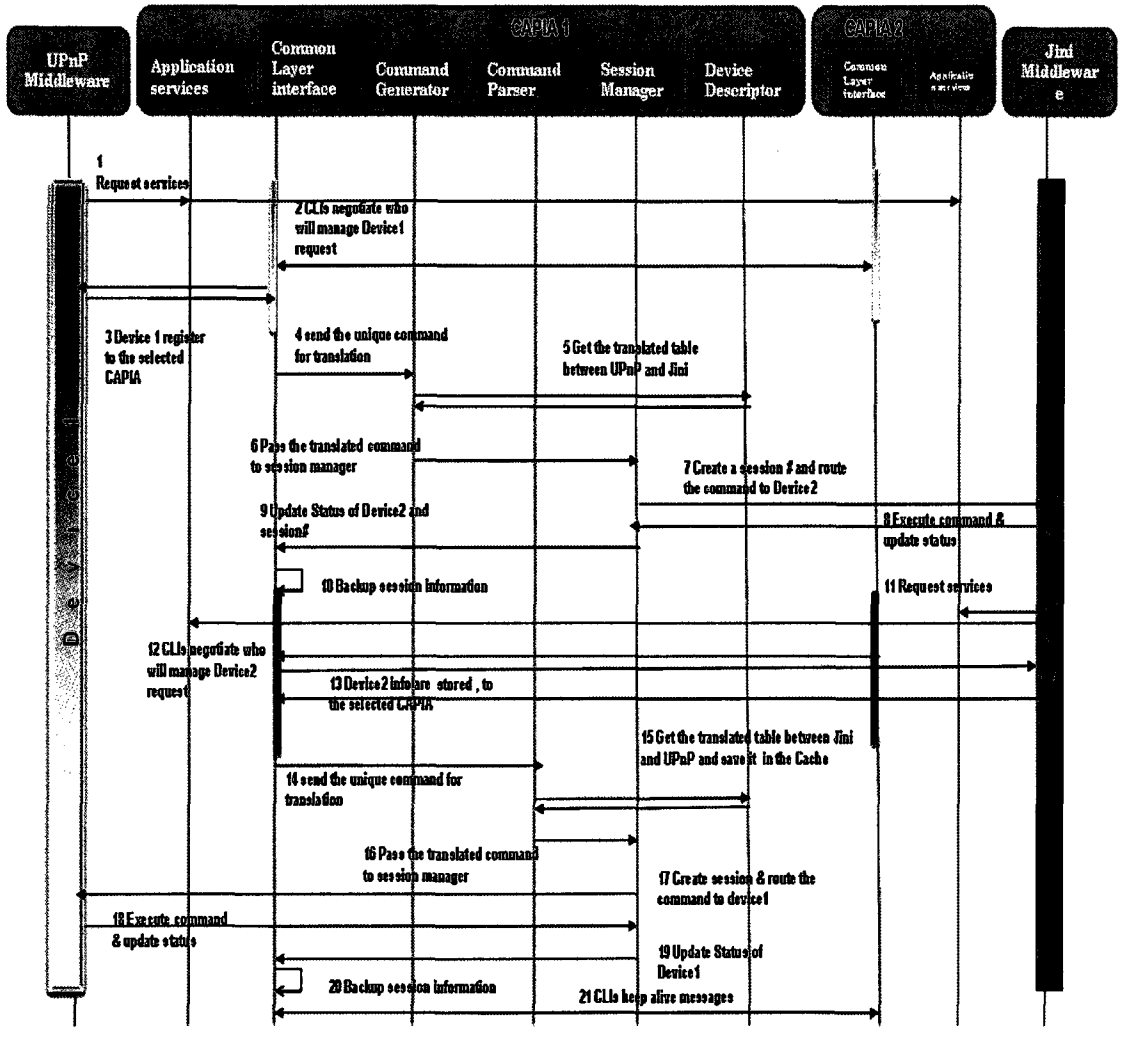

Fig. 3.5: Connection and Interaction algorithm between home network devices

The protocol and message transfer between CAPIA blocks are summarized as follows:

1. Device 1 registers itself, check device directory and requests to turn on Device 2 on the Jini network.

2. CLIs between different CAPIA bridges negotiate which one will manage Device 1's request.
3. Device 1 registers with the selected CAPIA.
4. The unique command is sent to the command generator for translation.
5. The translated table from Jini to UPnP is received for Device 2.
6. The translated command is passed to the Session Manager.
7. The session number is created and routed to Device 2
8. Device 2 executes the command, changes its status and updates it with the Session Manger
9. The status of Device 2 is updated in the CLI and session number.
10. The session information is backed up in the backup memory at the CLI.
11. Device 2 requests services from Device 1.
12. CLIs negotiate which one will manage Device 2's request
13. There is already a session between Device 1 and Device 2, so CAPIA1 will handle this service request.
14. The unique command is sent for translation to the Command Parser.
15. The translated table between Jini and UPnP is retrieved and saved in the CP-Cache.
16. The translated command is passed to the Session Manager.
17. A session is created and the command is routed to Device 1.
18. Device 1 executes the command and changes its status and sends that updated status to the Session Manager.
19. The Session Manager updates the status of Device 1 in the CLI.

20. The session information is backed up.
21. The "keep alive" message, and compressed sender/receiver tables are exchanged between the CLIs.

3.5. Summary

In this chapter we have introduced the CAPIA Bridge and explained the functionality of each component of the proposed architecture.

In this proposed architecture, CAPIA is covering the device context changes among the bridges, keeping the status of the devices up to date. It is also handling the protocol conversion between different middleware without having its own intermediate language unlike UMB, and this is done without the intervention of the user. This will lead to faster conversion, and it makes the CAPIA design simpler than other one-to-any bridges.

As new middleware or appliances appear, our conversion mechanisms can easily support them by adding a few XML files, which makes the CAPIA Bridge extensible.

The distributed environment of CAPIA helps with load balance and fault tolerance, which will guarantee entire system's reliability.

Chapter 4: Performance Evaluation

4.1. Introduction

We are proposing the CAPIA bridge to solve the interoperability between heterogeneous computing home middleware. In order to test CAPIA's performance, a model has been designed for the distributed CAPIA, and other models have been designed for UMB and MMB. We have chosen to design a model for UMB as there are several papers published for this bridge, and the protocol used in this bridge is described; another reason is that; Waseda University approach and UMB approach have a lot of similarities in their design, and there is only one paper for "A framework for connecting home computing middleware". We are designing a model for MMB, because it is presenting a new approach to solve the interoperability between heterogeneous home middleware. We are simulating the three models, measuring the time for the command to be executed in each model, and comparing them at the end of this chapter. A unit time was used for each task in the models modules.

The simulation used Arena software to simulate the process of the three bridges, and the following assumptions were used for all three models:

- Only one service request processed at a time
- A service was requested constantly and not randomly

- Requests were repeated a maximum of four times (one request at a time) with a difference of seven minutes between each service request.
- The resources allocated are executing each task of the simulation in .1 second except if the resource is dealing with Cache memory then it executes the command in 0.06 second.

To be sure of the stability of the process, we have run the simulation under different arrival times, i.e. we made the service arrive at the model every two minutes four times. We repeated the simulation under the same assumptions, except for the service inter-arrival times we used, every three minutes, then every four minutes, and so on, until reaching ten minutes. At each simulation run, the total time of the process is measured producing the same result outputs for all the above mentioned inter-arrival times. The results are logical, since the total process time of the model should be fixed even if we change the inter arrival-time between the services, so we have chosen seven minutes as the inter-arrival time between the incoming services in the simulation with fixed enter-arrival time; also the simulation was run under random enter- arrival time with different loads to test the behavior of the three bridges as will be shown later in this chapter.

4.2. CAPIA Model

The CAPIA model was designed as shown in Fig. 4.1. The scenario used is as follows: a UPnP device is sending a command to a Jini device, and at the same time a Jini device is sending a command to a UPnP device via the CAPIA bridge.

The following resources were allocated to different CAPIA modules:

- Common Layer interface:

  - The servers negotiate which CAPIA who will control the service request
  - Fills the service table
  - Gets the common command
  - Sends Filled table to session manager
  - Backs up Sender/Receiver table
  - Exchanges keep alive messages with the back up tables
- Command Generator:
  - Extracts translated table
  - Stores the extracted table
  - Fills the service table
  - Sends the information to session manager
- Command Parser:
  - Extracts translated table
  - Fills the service table
  - Stores the extracted table
  - Sends the information to session manager
- Session Manager:
  - Creates a session
  - Routes the command to the device
  - Updates Device Status
  - Updates common layer
- Device Descriptor:
  - Table searcher

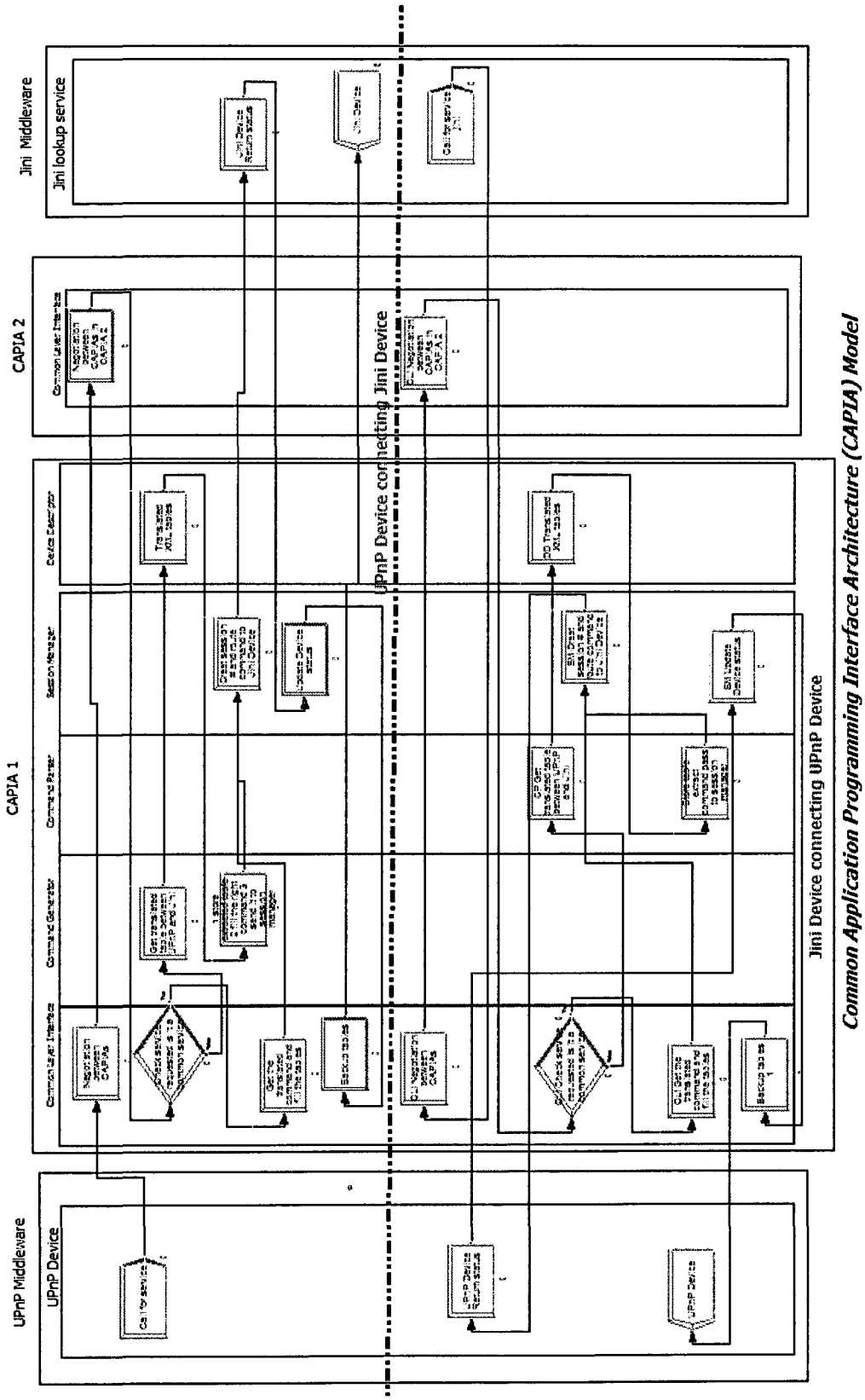

Fig. 4.1: CAPIA Model

The utilization of the resources in the CAPIA Model was measured and a comparison between the scheduled utilization is shown in Fig. 4.2 and Fig. 4.3.

Usage

Scheduled Utilization

	Value
Back up tables	0.00018152
CG fill the service table	0.00007923
CP Extract translated table	0.00005759
CP fill the service table	0.00008649
CP Send the information to session manager	0.00008649
CP Store the extracted table	0.00008649
Create a session	0.00030742
Execute command	0.00013496
Extract translated table	0.00005705
Fill the tables	0.00020047
Get the common command	0.00020047
jini Execute command	0.00013721
jini Return status	0.00013721
Negotiate 1 which CAPIA will control the service request	0.00019115
Negotiate which CAPIA who will control the service request	0.00019246
Return status	0.00013496
Route the command to device	0.00030742
Send filled table to session manager	0.00020047
Send the infromation to session manager	0.00007923
Session Manager Update status	0.00027897
Store the extracted table	0.00007923
Table searcher	0.00009225
update Common layer	0.00027897

Fig. 4.2: The utilization of the resources in CAPIA Model

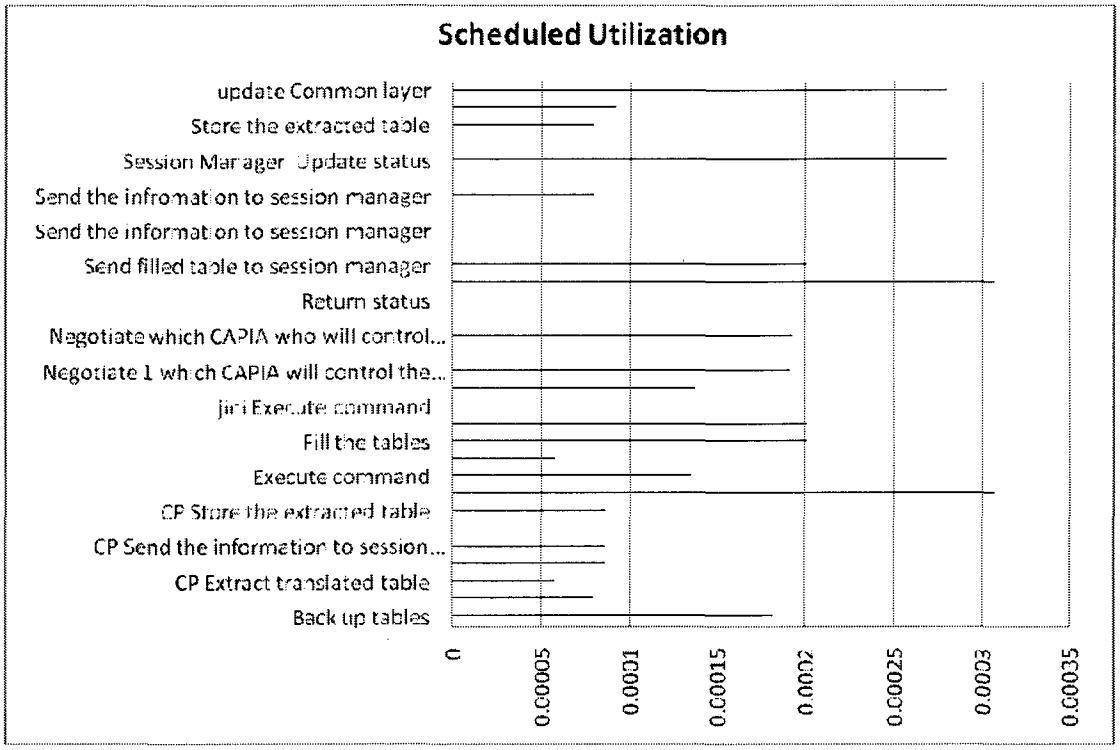

Fig. 4.3: Scheduled utilization for CAPIA resources in minutes

A waiting time was noticed and measured for queues in the device controller when the CLI was negotiating who will manage the incoming service request, when extracting the translated command from the tables, and when filling out the Sender/Receiver tables.

The total time of the simulation was then measured, which is the total time that the command takes from reaching CAPIA till it leaves the bridge.

Fig. 4.4 shows the waiting times in the queues, where ϵ is a very small number almost = 0

Queue

Time

Waiting Time

	Average
1 store extracted table 2 fill the right command 3 send it to session manager.Queue	є
Backup tables 1.Queue	є
Backup tables.Queue	є
CLI Get the translated command and fill the tables.Queue	0.00108341
CLI Negotiation between CAPIAs in CAPIA 2.Queue	0.00006971
CLI Negotiation between CAPIAs.Queue	0.00094601
CP Get translated table between UPnP and Jini.Queue	є
Creat session # and route command to Jini Device.Queue	0.00048540
DD Translated XML tables.Queue	є
Get the translated command and fill the tables.Queue	є
Get translated table between UPnP and Jini.Queue	є
Jini Device Return status.Queue	є
Negotiation between CAPIAs in CAPIA 2.Queue	0.00006621
Negotiation between CAPIAs.Queue	0.00077164
SM Creat session # and route command to Jini Device.Queue	є
SM Update Device status.Queue	є
Store table extract command pass to session manager.Queue	є
Translated XML tables.Queue	є
Update Device status.Queue	0.00005321
UPnP Device Return status.Queue	є

Fig. 4.4: Waiting time in Queues (minutes)

Fig. 4.5 shows the total time of simulation, which is 0.01971164 minutes, and ϵ is a very small number almost = 0.

Entity

Time

VA Time

	Average
Service Command Jini	0.01815422
Service Command UPnP	0.01778757

NVA Time

	Average
Service Command Jini	ϵ
Service Command UPnP	ϵ

Wait Time

	Average
Service Command Jini	0.00155742
Service Command UPnP	0.00137647

Transfer Time

	Average
Service Command Jini	ϵ
Service Command UPnP	ϵ

Other Time

	Average
Service Command Jini	ϵ
Service Command UPnP	ϵ

Total Time

	Average
Service Command Jini	0.01971164
Service Command UPnP	0.01916404

Fig. 4.5: Total time in minutes for the command to be executed in CAPIA model

The simulation was run under different loads: two service requests arrive for the application services at the same time, then three service requests arrive at the same time, and so on until eight service requests arrive at the same time. The collected results were saved in Table 4.1 (at the end of this chapter).

4.3. MMB Model

A model was designed for MMB, as shown in Fig. 4.6, and was simulated using Arena software. The protocol for MMB which was described in Chapter 2 was the basis for designing the MMB model.

In this model, a device from UPnP is trying to invoke service from a device under Jini middleware. The connection phase of the new device and the advertisement of its virtual device in the Jini interface first take place, then the interaction phase between the devices the virtual UPnP device and the Jini device.

The simulation was run under the same assumptions that were mentioned in the introduction.

The following resources were allocated to different MMB modules:

- Device Interface:
  - Requests device description from the connecting device
  - Sends device description
  - Invokes service to virtual device
  - Returns results and parameters to device
- Repository Manager:
  - Writes data from device interface (Device Mapping)
  - Creates virtual device

- Returns results to device interface
- Updates device context, and send it to communication manager

- Communication Manager:
  - Negotiates with other bridges on which one will control the device
  - Returns the result of the negotiation to repository manager and updates the server
  - Sends device context received from the repository manager to other communication manager in another bridge
  - Returns changes and results received from the communication manager of other bridge to the repository manager
- Device interface virtual device
  - Advertises new virtual device to other bridges
  - Invokes service to the device
  - Returns parameters received from Jini to device interface

The utilization of resources in the model was studied and a comparison between the scheduled utilization was measured and shown in Fig. 4.7.

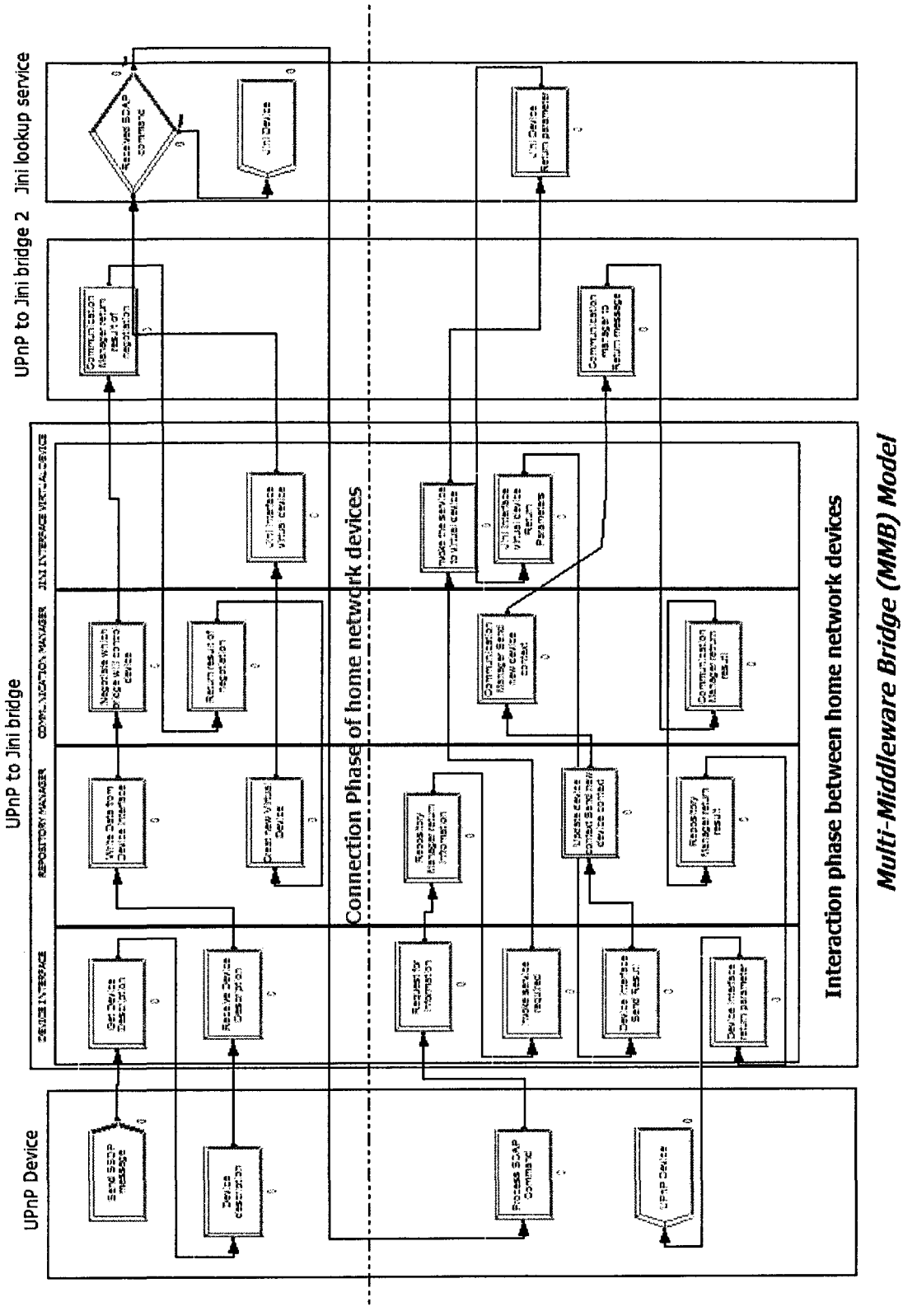

Fig. 4.6: MMB Model

Resource

Usage

Scheduled Utilization

	Value
Advertise NEW virtual device	0.00034949
Communicate with other repository	0.00157286
Creat virtual Device	0.00031978
Invoke Service	0.00071558
Process SOAP Command	0.00026968
Request device description	0.00067760
Return message	0.00031944
Return Parameter	0.00096257
Return result od negotiation	0.00070181
Return result to Device interface	0.00036602
Send Device description	0.00101437
Send new device context	0.00078821
Send Result	0.00101714
Upade Server	0.00157286
Update device context	0.00044429
Write data received from Device interface	0.00043435

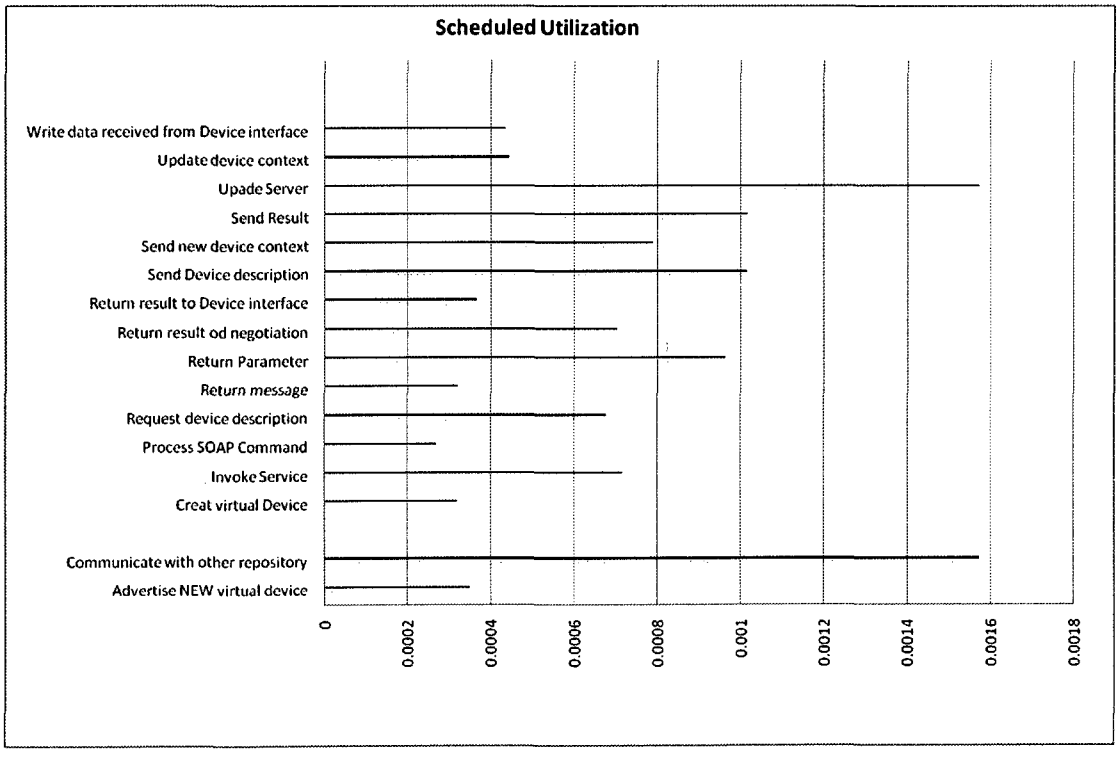

Fig. 4.7: Total Scheduled utilization for MMB resources in minutes

The total time for simulated model was .04404713 minutes, and ϵ is very small almost = 0 as shown in Fig. 4.8 where the Discovery command is the SSDP message and the SOAP call as shown in the Model.

Entity

Time

VA Time	Average
Device Command	0.04404713
NVA Time	Average
Device Command	ϵ
Wait Time	Average
Device Command	ϵ
Transfer Time	Average
Device Command	ϵ
Other Time	Average
Device Command	ϵ
Total Time	Average
Device Command	0.04404713

Fig. 4.8: Total time in minutes for the command to be executed in MMB model

4.4. UMB Model

A model was designed for UMB as shown in Fig. 4.9; this model was simulated using arena software under the same simulation environment as CAPIA and MMB.

The UMB protocol and message flow explained in Chapter 2 was the basis for the design of the UMB model.

The scenario used is the following: a UPnP device sends a command to a Jini device and at the same time a Jini device sends a command to a UPnP device via the UMB Bridge. In this case, the Jini device will have to wait until the UPnP virtual device is advertised to the Jini middleware.

A condition was used in the simulation wherein the Jini device could not invoke service from the UPnP device until a virtual UPnP device is advertised. The Jini device then can invoke the service requested and communicate with the virtually advertised UPnP device.

The following resources were allocated to different UMB modules:

- UMB-A
  - Message converter to UMB by Global Dispatcher
  - SSCP discovery and query list
  - Event handler
  - Virtual function creation
  - Converts LDT function to UDT function
  - Converts device to UDT
  - Converts any middleware message to UMB message and vice versa
  - Maps the UDT function to a different middleware function
  - Finds match and converts UDT device to VDP
- UMB-C
  - Contains routing tables
  - Routes messages between UMB-As

- Routes UDT

The design of the UMB model was not complicated compared to the design of MMB model. It has been suggested that having its own message format as an intermediary between different middleware does not significantly affect the total time of the interaction between different devices under different middleware; we disagree with this notion, as the simulation showed that it increases the process time. On the other hand, we do not believe that this delay is critical, as it should not affect the home environment in general, although it could prove to be an issue in the more demanding industrial sector.

The utilization of resources in the model was studied and a comparison between the scheduled utilization is measured and shown in Fig. 4.10.

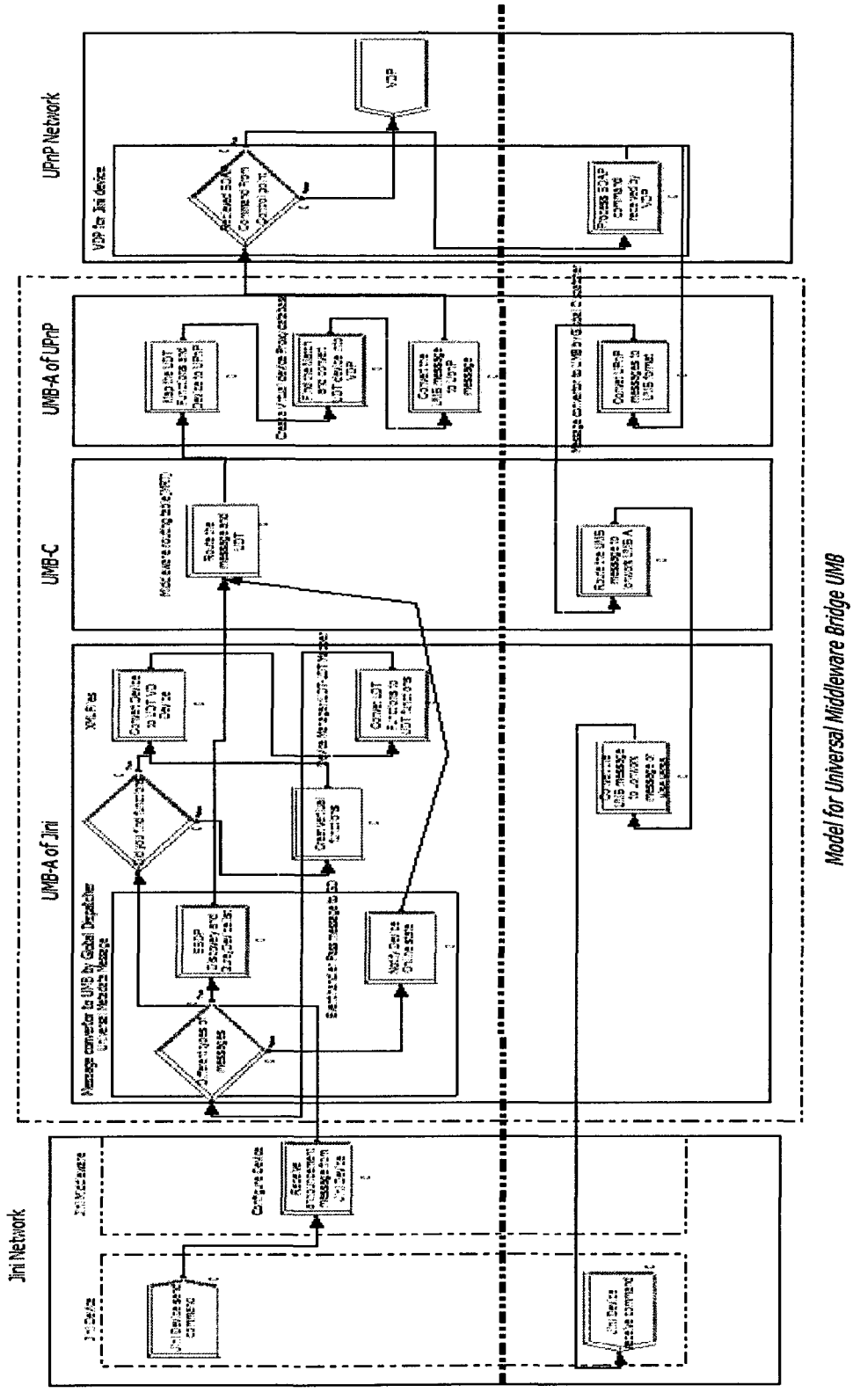

Fig. 4.9: UMB Model

Resource

Usage

Scheduled Utilization	Value
Configure Devices	0.00014846
Control messages	0.00019835
Control messages 1	0.00018513
Device Convertor from UDT to VDP	0.00015941
Device Manager	0.00008879
Device Mapper	0.00010583
Device to UDT device	0.00008806
Discover device	0.00014846
Event handler	0.00004045
Find the Match	0.00015941
Global Dispatcher	0.00010061
Message convertor	0.00032620
Message convertor 1	0.00029842
Message router	0.00023113
Monitor messages	0.00019835
Monitor Messages 1	0.00018513
Route UDT	0.00015908
VDP process soap command	0.00009088
Virtual function creation	0.00002061

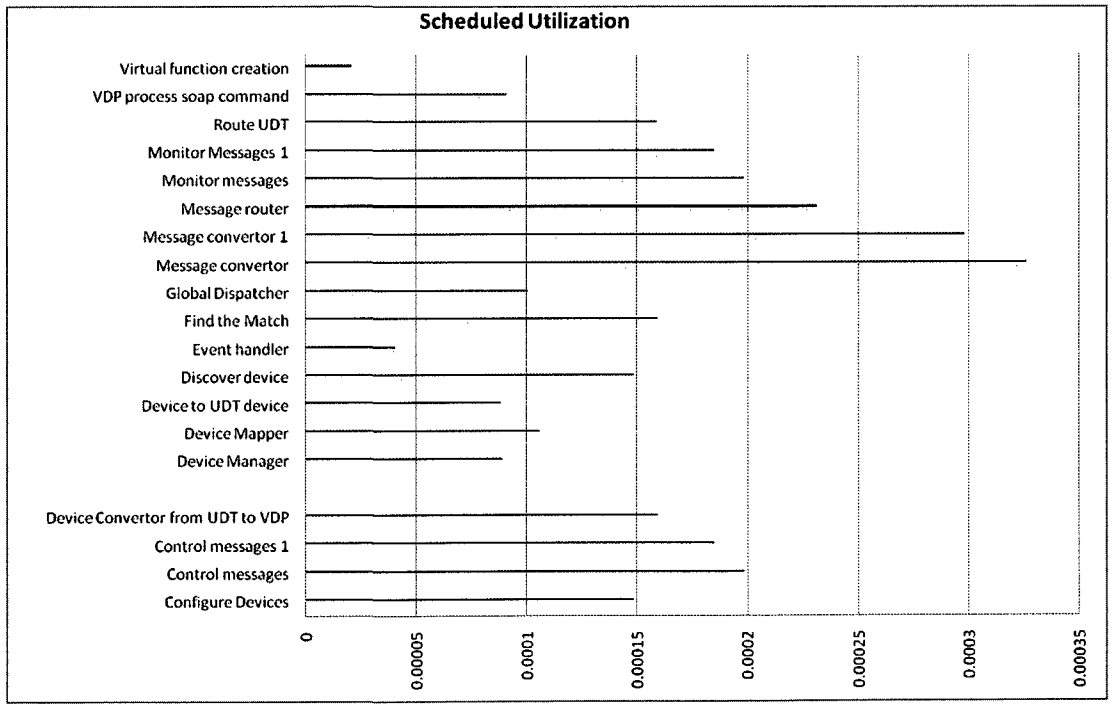

Fig. 4.10: Total Scheduled utilization for UMB resources in minutes

The total time for simulated model was 0.02804035 minutes ; Є is a very small number almost = 0 as shown in the Fig. 4.11, where the Device command is the command sent by the Jini device and the SOAP call for service command sent by the UPnP device (control point) as shown in the Model.

Entity

Time

VA Time	Average
Device Command	0.02804035
NVA Time	Average
Device Command	Є
Wait Time	Average
Device Command	Є
Transfer Time	Average
Device Command	Є
Other Time	Average
Device Command	Є
Total Time	Average
Device Command	0.02804035

Fig. 4.11: Total time in minutes for the command to be executed in UMB model

4.5. Simulation of CAPIA, UMB, and MMB Under Different Loads

In this section we will study the behavior of the 3 models under different loads, with fixed inter arrival time between the services. Also the 3 models will be tested under different loads with random inter arrival time between the services.

4.5.1. Simulation with Fixed Inter-arrival Times Between Services

The loads that were studied include the following: only one service request processed at a time, repeated four times with the difference of seven minutes between each consecutive service. Two services request processed at the same time, and three to eight service requests at a time.

We have fixed the inter arrival time to 7 min. between each service/s arrival. After running the simulations for CAPIA, UMB, and MMB, the results were collected and a comparison made concerning the total time for two devices in two different middleware controlling each other and communicating.

The results for CAPIA and UMB were close, although CAPIA is the fastest among the three simulated bridges for single requests as well as simultaneous multiple requests.

The average total times for the simulations of the three bridges are recorded in Table 4.1. and the results are shown in the graph in Fig. 4.12.

Table 4.1: Simulation results for CAPIA, UMB & MMB under different loads

Number of service requested simultaneously with max arrivals 4	CAPIA total process time in minutes.	UMB total process time in minutes.	MMB total process time in minutes.
1	0.01971164	0.02841648	0.04251851
2	0.02253668	0.03149252	0.0475369
3	0.02411951	0.03473812	0.05501119
4	0.02731695	0.03738485	0.05858193
5	0.03081165	0.04062178	0.06368766
6	0.03348212	0.04508547	0.06881829
7	0.03655785	0.04666092	0.0729037
8	0.03962923	0.05143283	0.07891501

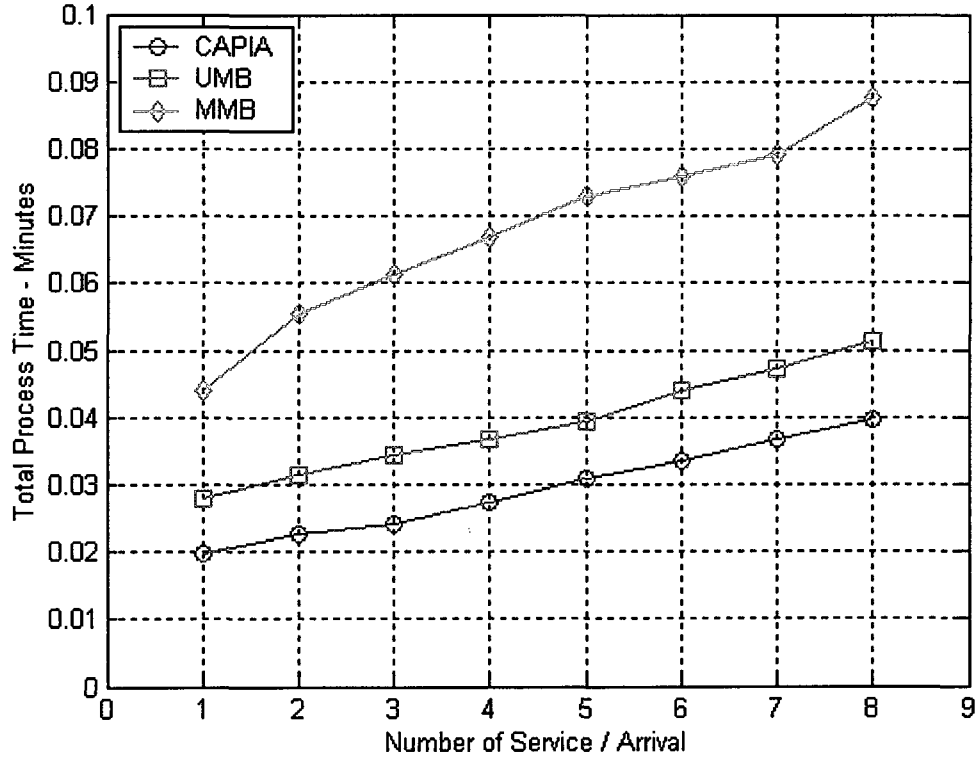

Fig. 4.12: Total process time for CAPIA, UMB, and MMB under different loads

4.5.2. Simulation with Random Inter-arrival Times Between Services

We ran the simulation for the three models with random inter-arrival times, in order to test the stability of the models. The following parameters were used with each model:

1. Number of services per arrival = 1
2. Maximum number of arrivals = 4
3. Inter-arrival time is random using the Poisson distributed function POIS (mean value), where the mean value is 0.05, .5, 1, 2, 3, and so on up to 19.
4. Steps 2 and 3 were then repeated with different loads, when the number of services/arrival =2, then 3, and finally 4.

In the following section, we will show the average total time of the process in the CAPIA, UMB and MMB bridges with different random inter-arrival times for various services and different loads (services/arrival).

We have observed that with an inter-arrival time of POIS(0.05), the average time for the process was higher than when it was POIS(1, 4,.....19). This difference in the process time is logical, as services will have to wait longer in the queues in order to be processed when inter-arrival time is shorter. On the other hand, when the inter-arrival time is longer, then the service will be processed before the other service reaches the modules. This was observed in the behavior of MMB and UMB models as well.

- **CAPIA Model with Random Inter-arrival Time Between Services**

Fig. 4.13 shows the relation between the total process time and the random inter-arrival time for CAPIA model.

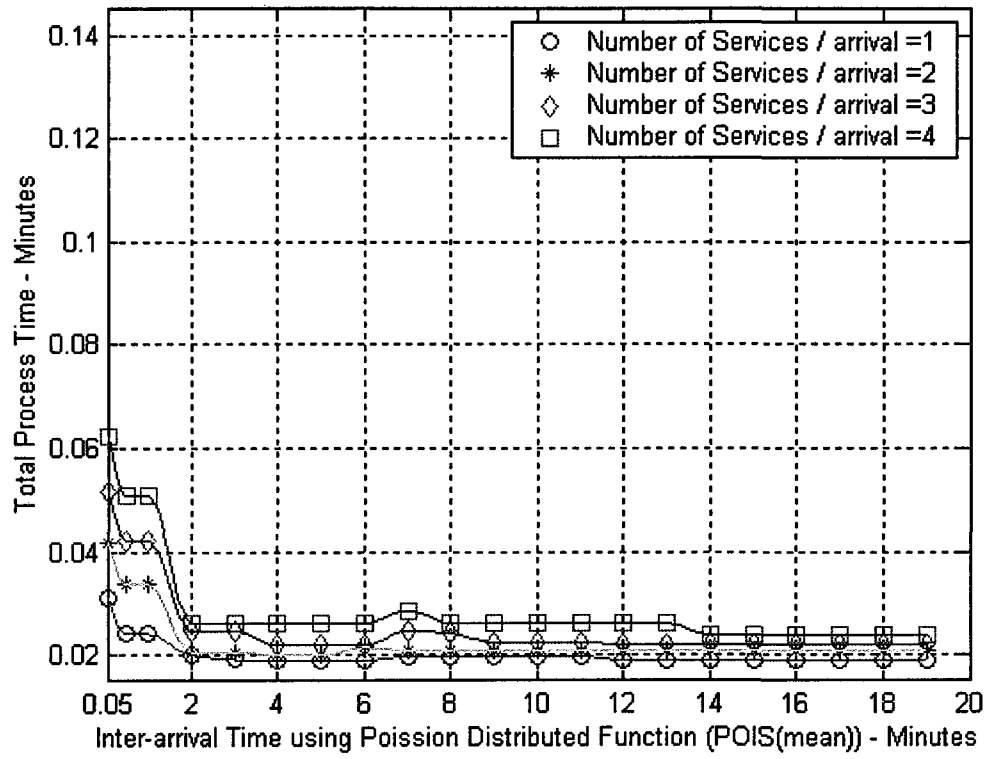

Fig. 4.13: CAPIA total process time vs. the random inter-arrival time

- **UMB Model with Random Inter-arrival Time Between Services**

Fig. 4.14 shows the relation between the total process time and the random inter-arrival time for UMB model.

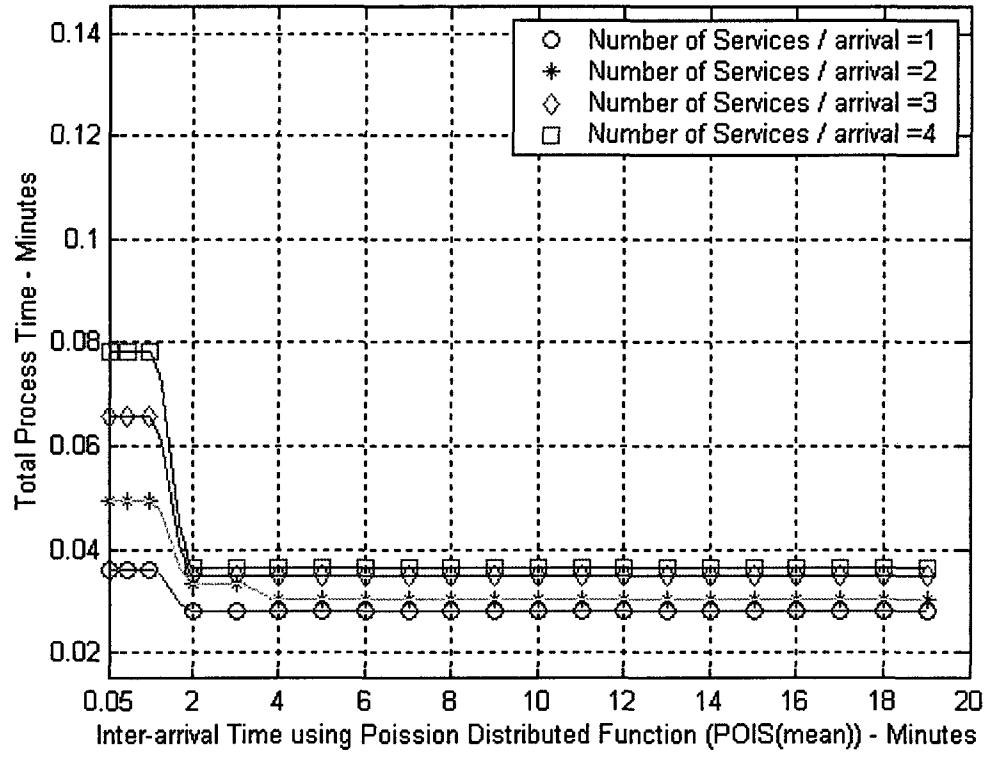

Fig. 4.14: UMB total process time vs. the random inter-arrival time

- **MMB Model with Random Inter-arrival Time Between Services**

Fig. 4.15 shows the relation between the total process time and the random inter-arrival time for UMB model.

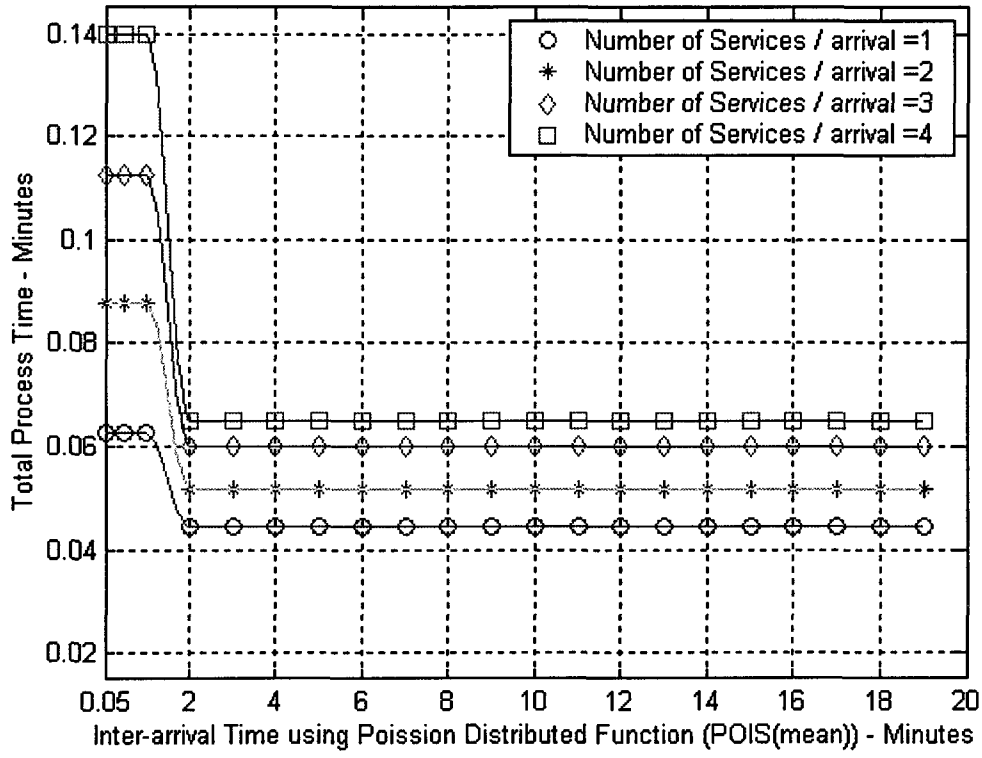

Fig. 4.15: MMB total process time vs. the random inter-arrival time

From the previous results of random simulation for the three models, Fig 4.16 is showing the CAPIA, UMB, and MMB models total process time in minutes vs. the random inter-arrival time for 4 services/ arrival

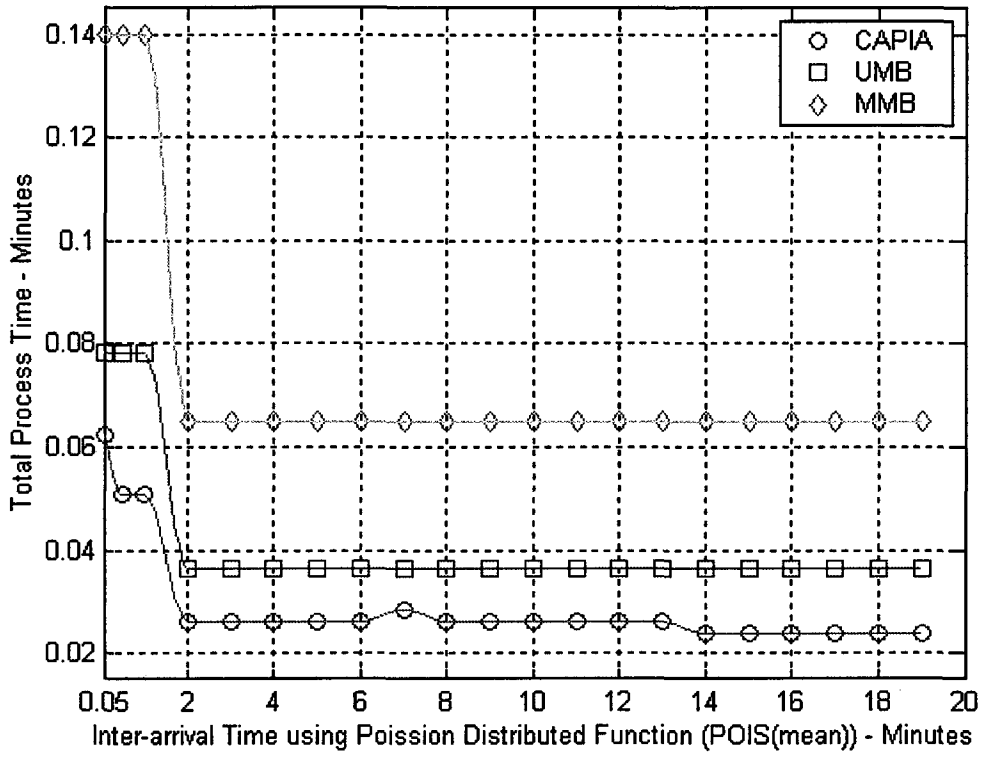

Fig. 4.16: CAPIA, UMB, and MMB models total process time vs. the random inter-arrival time

The total process times for the MMB and UMB are higher than that of the CAPIA bridge, as they have to advertise a virtual device before the connection can take place between the two devices in different middleware. Also, the MMB has a distributed environment where the bridges negotiate who is going to handle the incoming service, and to avoid creating multiple virtual devices from the same device, MMB always updates any changes in the joined devices; this leads to extra waiting and processing times.

After conducting the comprehensive analysis through the survey in chapter 2 and the simulation in this chapter, we are holding a comparison between the three models and is shown in Table 4.2. For the resource discovery in the three models, our analysis is as the following: CAPIA has a Centralized/distributed environment and a device directory in the application interface where all the devices are registered and can be discovered by other devices, at the same time CAPIA distributed architecture, the servers are exchanging the updated device directory between them, thus the CAPIA is providing a Hybrid resource discovery mechanism that is not strictly peer to peer or server based mechanisms.

UMB is characterized in their research by being hybrid in the resource discovery, for several reasons such as; the devices connected to a certain UMB-A are converted to devices defined by UDT, and then converted to virtual devices in other different middleware, also the QueryDeviceList message requests all devices list on specific middleware; Then NotifyOnlineStatus message notifies the UMB system event to all UMB Adapters whenever specific UMB Adapter or Device is Plugged in/out. The MMB can be considered a server based, since each one to one bridge in their architecture is responsible for the resources between its specific two middleware, and all of the one to one bridges are updating the context and the change of the devices status in one server.

Table 4.2: Comparison between CAPIA, UMB, and MMB

	CAPIA	UMB	MMB
Interpretability under heterogeneity	Any - to - any conversion based on XML files	One –to – any conversion based on defined XML meta middleware	Any-to -any middleware
Modification of legacy appliances	No	No	No
Resource management	Global	Global	NA
Modification of current one to one bridges	No	No	Yes
Resource discovery	Hybrid	Hybrid	Server based
Query Level	Flexible	Flexible	NA
Bridge topology	Centralized/ Distributed	Centralized	Distributed
Dynamic binding with device	Yes	Yes	Yes
Load Balance	Yes	No	Yes
Creating virtual device at the different middleware	No	Yes	Yes
Speed	Based on our simulation , the fastest among the three bridges	Based on our simulation Faster than MMB, and slower than CAPIA	Based on our simulation ,Fast but slower than CAPIA and UMB
Fault tolerance	Supported	Supported/ ongoing research	NA

4.6. Summary

Models were designed for CAPIA, UMB, and MMB and were simulated using arena software. The design of the UMB model was simple compared to the design of MMB model. It was suggested that although the UMB has its own message format for intermediate message between different middleware, this does not significantly effect the total time of the interaction between different devices under different middleware. We disagree with this; the simulation showed that it increased the

process time. The simulation was run under the same assumptions for the three models, and under different loads, the results showed that CAPIA has the fastest process time, followed by UMB. The slowest model was MMB. Also, in CAPIA, any device can communicate with another device in a different middleware without waiting for it to be virtually advertised, which is not true for either UMB or MMB.

We have changed the conditions of the simulation as follows: the service arrives to the model randomly under the Poisson distribution function POIS (mean value), with mean values starting from 0.05 till 19; those inter arrival times were repeated under different loads, where the number of services/arrivals increases from 1 to 4. All three simulated models showed logical output results. The two bridges UMB and MMB must advertise the virtual device to different middleware, so that other devices using that middleware can communicate with it; this is not the case for the CAPIA bridge, and this key aspect makes our proposed architecture faster even with random inter-arrival times.

Chapter 5: Conclusions and Future Research

5.1. Conclusions

Interoperability among heterogeneous middleware is crucial to the successful operation of a Smart Home. Most of the middleware that has been developed does not guarantee interoperability among home appliances. Various home network middleware bridges have been introduced to support interoperability, due to the heterogeneity of home network middleware.

For these middleware bridges, there are several approaches. The one-to-one approach connects two specific forms of middleware and performs protocol conversion, while the one-to-any approach integrates heterogeneous middleware and ensures interoperability between them. In this dissertation, we are proposing an any-to-any CAPIA bridge, which is designed to solve interoperability problems in home networks in which several types of heterogeneous home network middleware exist.

Our proposed system is centralized/distributed middleware bridge architecture; it can operate either centralized as a standalone server or as a distributed architecture, where more than one server will support CAPIA. This depends on how big the home network is, and how many appliances are connected to it.

Our proposed CAPIA supports fault tolerance in the case of the distributed environment, such that if one of the servers is not operating at the home network, the

user will not feel the interruption of service since the devices will continue their session with the remaining operating servers.

CAPIA supports load balance in the distributed environment, and can globally manage resources in the Smart Home. The conversion tables between different devices and services under different middleware are all defined by the use of XML; they are independent of any particular language or OS. If new middleware appears in the future, the proposed architecture will integrate it just by adding the new XML conversion tables, which means that there will be no modifications required either for the appliances (which is the case for HOMi) nor for the middleware used in the home networks (which is the case for MMB).

The CAPIA approach provides interoperability via any-to-any protocol conversion based on XML conversion tables. Therefore the cost of implementation is lower than that of one-to-one bridges. In addition, CAPIA's cost for interpretability is less than that of UMB, since there will be no need for a special intermediary translation language.

The distinctive functions and services are uniquely handled through the command generator, the command parser and the device descriptors, while the common commands are saved in a common command table which can be easily accessed. This approach makes the conversion process faster.

CAPIA does not create virtual devices from the real devices, it deliver the service command to the device translated to its own middleware format allowing any device to communicate with the other device in a different middleware without waiting for it to be virtually advertised as in the case of UMB and MMB.

We designed a model for CAPIA and other models for UMB and MMB. The three models were simulated using Arena software under identical assumptions, and the process times for all three were measured. The results of the simulation showed that CAPIA is the fastest bridge of the three.

5.2. Future Research

We have presented CAPIA from a design perspective, and the structure of CAPIA has been explained; however, each component of CAPIA can be an area for future research.

A performance characterization for the three bridges with different simulation conditions needs to be done; e.g., the maximum number of the service request arrivals can be changed to different values than currently used. Also, we can change the service request arrival-time from a uniform to a random process. Although we have already tried some of these conditions as mentioned before, we can still carry out further useful analysis in the future.

More comprehensive studies have to be carried out on different components of CAPIA; one of the main areas to be investigated is how to assign a unique address to each device when it registers itself to the applications services in the application interface. A newly unique global address is to be assigned to each device while considering the device type and middleware, allowing each device to choose the device it wants to control and communicate with during this stage.

Another area for future research is to study different service invocation protocols and the conversion between them. Handling multimedia streams for multimedia applications should be considered while implementing CAPIA.

CAPIA is designed to support fault tolerance in the distributed environment, and future research should be conducted on how to maintain the keep-alive message and the exchange of backup compressed Sender/ Receiver tables between the servers, as well as how the operating server will take over the session in case a fault accrues in one of the servers.

More research can be conducted on how the different CAPIA modules exchange messages, from the CLI to the CG and CP, and also from Session/Event manager to CLI and vice versa.

A challenging subject for future research is the implementation of CAPIA and the construction of a test bed to test the functionality of the proposed architecture in real-time.

Bibliography

[ALL03] J. Allard, V. Chinta, S. Gundala, and G. G. Richard III, "Jini Meets UPnP: An Architecture for Jini/UPnP Interoperability", Proceedings of Symposium on Applications and the Internet, Jan. 2003, pp. 268-275.

[ANA07] V. Anand, S.;"A DLNA framework for NEXT GEN mobile terminals connecting IMS Networks for human-centered Digital Home Environment" , International Conference on IP Multimedia Subsystem Architecture and Applications, Dec. 2007, pp.1 – 5.

[BEL03] P. Bellavista, A. Corradi, R. Montanari, C. Stefanelli, " Dynamic binding in mobile applications", IEEE Internet Computing, March 2003, pp 34-42.

[CEN03] CENELEC Work Shop, "Smart House Final Report", July 2003.

[CHA00] D. Chakraborty , H. Chen, "Service Discovery in the Future for Mobile Commerce", ACM's First Electronic Publication, Nov. 2000.

[CHE04] W.CHEN, "Home Networking Basis, Transmission Environment and Wired/Wireless Protocols", Prentice Hall, 2004.

[CHO05] K.Choi, Y.Park, Y.Ahn, K.Jung, S.Hong " Design of Interoperable Module between Two Home Network Middlewares",International Journal of Future Generation Communication and Networking IJFGCN, Vol. 1, No. 1, 2005.pp. 7-14.

[DEN02] M. Denis , C. Valerie, D. Christophe, " Methods for bridging a HAVi sub-network and a UPnP subnetwork and device for implementing said methods," Thomson Multimedia, European Patent Application, 2002.

[DYN07]Mu Dynamics, "Simple Service Discovery Protocol, SSDP Protocol brief", 2007, www.mudynamics.com/resources/collaterals/SSDP-ProtocolBrief.pdf, last access was on February 21, 2010.

[ECH10] Echelon Corporation, "Lonworks technology overview," http://www.echelon.com/communities/energycontrol/developers/lonworks/ , Last accessed March 2010.

[EDW06] W.K.Edwards," Discovery systems in ubiquitous computing", Pervasive computing IEEE, Vol. 5, Issue 2, 2006, pp. 70-77.

[GOL99] Y. Goland, T. Cai, P. Leach, and Y. Gu." Simple Service Discovery Protocol/1.0", IETF Internet Draft, October 1999.

[GUT99] E. Guttman, C. Perkins, J. Veizades, M. Day "Service Location Protocol, Version 2",RFC 2608, Network Working Group, June 1999.

[HAV10] HAVi home page, http://www.havi.org, Last accessed March 2010.

[HEL02] S. Helal," Standards for service discovery and delivery", Pervasive Computing, IEEE, Vol. 1, Issue 3, July-Sept. 2002 , pp. 95 - 100 .

[HLA08] H. Hlavacs , R.Weidlich, T.Treutner, "Energy saving in future home environments", Wireless Days, WD '08, 2008 , pp. 1 - 5.

[ISH04] H.Ishikawa, Y. Ogata, K. Adachi, T. Nakajima, " Building smart appliance integration middleware on the OSGi framework", Proceedings. Seventh IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, May 2004, pp.139 – 146.

[JIA04] L. Jiang; D. Liu; B. Yang;"Smart Home research, " Proceedings of 2004 International Conference on Machine Learning and Cybernetics Vol. 2, Aug. 2004 pp. 659 – 663.

[JIN06] Jini, "Category: Jini specifications", http://www.jini.org/wiki/Category:Jini_Specifications, last access was on February 21, 2010.

[JIN10] JINI home pages, http://www.sun.com/jini, Last accessed March 2010.

[KIM06] M. C. Kim and S. J. Kim, "A Scenario-Based User- Oriented Integrated Architecture for Supporting Interoperability Among Heterogeneous Home Network Middlewares," ICCSA 2006, Lecture Notes in Computer Science, Vol. 3983, 2006, pp. 669-678.

[KIM07A] D. Kim; C. Lee; J. Park; K. Moon; K. Lim; " Scalable Message Translation Mechanism for the Environment of Heterogeneous Middleware," IEEE Transactions on Consumer Electronics, Vol. 53, Issue 1, Feb. 2007, pp.108 – 113.

[KIM07B] D. Kim; C. Lee; J. Park; K. Moon; K. Lim;"The Conversion of a Device And a Control Message in The Universal Middleware Bridge", International Conference on Consumer Electronics, ICCE , Digest of Technical Papers, Jan. 2007, pp.1 – 2.

[KIM08] Y. Kim; H.Jang; J. Choi; C. Song; Y. Eom;"Design of a Multi-middleware Bridge for Supporting Interoperability in Home Network Environments",

International Conference on Advanced Language Processing and Web Information Technology ALPIT '08., July 2008, pp.477 - 482 .

[LEE01] K. Lee et al., "A new control protocol for home appliances-LnCP", Proceedings of ISIE2001, Vol. 1, 2001, pp.286-291.

[LEE02] C. Lee and S.Helal "Protocols for service discovery in dynamic and mobile networks", International Journal of Computer Research , Computer and Information Science and Engineering dept. University of Florida, Vol. 11, Issue 1,2002, pp. 1-12.

[LEE05] C. Lee; K. Moon; "Design of a universal middleware bridge for device interoperability in heterogeneous home network middleware", International Conference on Consumer Electronics ICCE, Digest of Technical Papers. Jan. 2005, pp. 371 – 372.

[LEE07] H. Lee, S.J Kim " A Standard Method-Based User-Oriented Integrated Architecture for Supporting Interoperability among Heterogeneous Home Network Middlewares ", International Journal of Smart Home Vol. 1, No. 1, Jan. 2007.

[MAR01] P. Marshall, "Home networking: a TV perspective," Electronics & Communication Engineering Journal Vol. 13, Issue 5, Oct. 2001, pp.209 - 212.

[MIL01] B. Miller, T. Nixon, C. Tai, ;M. Wood, "Home networking with Universal Plug and Play", IEEE Communications Magazine, Vol. 39, Issue 12, Dec. 2001, pp.104 - 109.

[MOO03A] K. Moon; Y. Lee; Y. Son;" Universal home network middleware (UHNM) guaranteeing seamless interoperability among the heterogeneous home network middleware for future home networks," , IEEE International Conference on Consumer Electronics, ICCE, June 2003, pp.306 – 307.

[MOO03B] K. Moon; Y. Lee; Y. Son;" Universal home network middleware guaranteeing seamless interoperability among the heterogeneous home network middleware" , IEEE Transactions on Consumer Electronics, Vol. 49, Issue 3, Aug. 2003, pp. 546 – 553.

[MOO08] K. Moon; J. Park; K. K.H.; L. Zheng; Q. Zhou "Fault-Tolerance in Universal Middleware Bridge" 11th IEEE International Symposium on Object Oriented Real-Time Distributed Computing (ISORC), May 2008, pp.471 – 477.

[MOY01] S. Moyer, D. Marples, S. Tsang. "A Protocol for Wide-Area Secure Networked Appliance Communication", IEEE Communication Magazine, Aug. 2001.

[MYO06] K.Myoung; J. Lee; D. Kim; W. Kwon "Home network control protocol for networked home appliances," IEEE Transactions on Consumer Electronics, vol. 52, Issue 3, Aug. 2006, pp. 802 – 810.

[NGO07] L. Ngo," Service-oriented architecture for home networks", TKK T-110.5190 Seminar on Internetworking, March 2007.

[ODR01] G. O'Driscoll; "The Essential Guide to Home Networking Technologies", Prentice-Hall, 2001.

[OH03] J. Oh; J. Park; G. Jung; S. Kang; "CORBA based core middleware architecture supporting seamless interoperability between standard home network middlewares", IEEE Transactions on Consumer Electronics, Vol. 49, Issue 3, Aug. 2003, pp.581 - 586.

[PER08] T. Perumal, A.R. Ramli, C.Y. Leong, S. Mansor, K. Samsudin, "Interoperability among heterogeneous systems in smart home environment", IEEE International Conference on Signal Image Technology and Internet Based Systems, (SITIS '08), 2008, pp. 177–186.

[PER09] T. Perumal, A. Ramli, C.Leong, K. Samsudin, S. Mansor, "Middleware for heterogeneous subsystems interoperability in intelligent buildings", 2009, Automation in Construction, Corrected Proof, Dec. 2009.

[POL05] Y. Poltavets; J. Park; D. Kim; 'IEEE1394 to UPnP software bridge structure", International Conference on Consumer Electronics, ICCE. 2005 Digest of Technical Papers, Jan. 2005, pp.375 – 376.

[PON03] R. Ponnekanti , A. Fox. "Application Service Interoperation without Standardized Service Interfaces" Proceedings of the First IEEE International Conference on Pervasive Computing and Communications, March 2003, pp. 30-40.

[RED06] Reddy, R.; "Robotics and Intelligent Systems in Support of Society", Intelligent Systems, IEEE Vol. 21, Issue 3, Jan.-Feb. 2006, pp.24 – 31.

[REI02] Reilly, D.; Taleb-Bendiab, A.; "An Jini-based infrastructure for networked appliance management and adaptation", Liverpool. Proceedings IEEE 5th International Workshop on Networked Appliances, 2002, pp. 161 – 167.

[RIC06] Ricquebourg, V.; Menga, D.; Durand, D.; Marhic, B.; Delahoche, L.; Loge, C.;" The Smart Home Concept: our immediate future," 2006 1ST IEEE International Conference on E-Learning in Industrial Electronics, Dec. 2006, pp.23 – 28.

[ROC05] Rockwell Software Inc., Arena version 10.00.00 – CPR7, Copyrights ©1983-2005, Academic version.

[ROS01] Rose, B.;" Home networks: a standards perspective," Communications Magazine, IEEE Vol. 39, Issue 12, Dec. 2001, pp.78 – 85.

[RUI07] C.Rui; H.Yi-bin; H.Zhang-qin; Z. Yong; L. Hui;" Framework for Local Ambient Intelligence Space: The AmI-Space Project," Computer Software and Applications Conference, 2007. COMPSAC 2007, 31st Annual International Vol. 2, July 2007, pp. 95 – 100.

[STU05] Study Group on Intelligent Home Appliance Networking, " Interim Report", July 2005.

[TOK02] Tokunaga, E.; Ishikawa, H.; Kurahashi, M.; Morimoto, Y.; Nakajima, T. " A framework for connecting home computing middleware", Proceedings. 22nd International Conference on Distributed Computing Systems Workshops, July 2002, pp.765 – 770.

[UPN10] UPnP home page, http://www.upnp.org, Last accessed March 2010.

[W3C01] World Wide Web Consortium, "Simple Object Access Protocol", http://www.w3.org/TR/SOAP/, last accessed February 16, 2010.

[W3C02] World Wide Web Consortium, "Extensible Markup Language (XML)", http://www.w3.org/XML , last accessed February 16, 2010.

[WAC02] K. Wacks, "Home Systems Standards: Achievements and Challenges", IEEE Communication Magazine, April 2002, pp 152 – 159.

[ZAH03] T.Zahariadis "Home Networking, Technologies and Standards", Artech House Inc., 2003.

Appendix A: Summary of Major Service Discovery Protocol

The following table summarizes the major service discovery protocols [LEE02].

	Jini	UPnP	Salutation	SLP
Home Page	www.sun.com/jini	www.upnp.org	www.salutation.org	www.svrloc.org
Main Entities	Lookup Service, Client , Service	Control Point, Devices (Services)	Salutation Manager, Transport Manager, Client, Server	Directory Agent, Service Agent, User Agent
Service Repository	Lookup Service	No	A set of SLMs (Salutation Managers)	DA (Directory Agent)
Service Announcement	Discovery/Join protocol	Advertisement (ssdp:alive)	Registering with local Salutation Manager	Service registration
Service Discovery	Query to lookup service	Contact control point, or listen to advertisement	Query to local SLM and cooperation among SLMs	Contact DA, or multicast to SAs
Access to Service	Service proxy object based on RMI	Invoking action to the service (SOAP), Query for variable state	Service Session management	Service type (service protocol) for the discovered service
Service Description and Scoping	Interface type and attribute matching	Description in XML	FU (Functional Unit) and attributes within it	Service type And attribute matching (fairly powerful matching)
Service Registration Lifetime	Leasing	CACHE-CONTROL header in alive message	No	Lifetime in service registration
Service Group	Group	No	No	Scope
Event Notification	Remote Events	Service publishes event(GENA) when state variable changes	Availability Checking (periodic & automatic)	No
Others	Java-centric architecture	Automatic con-figuration (AutoIP)	Transport independence	Authentication security feature

Appendix B: Definitions

• Smart Home

"A Smart Home is a dwelling incorporating a communications network that connects the key electrical appliances and services; it allows them to be remotely controlled, monitored or accessed", Definition by Intertek [JIA04].

A home, which is smart, must contain three elements, which are internal network, intelligent control and home automation. Internal network is the basis of a Smart Home, and it can be wire, cable and wireless. Intelligent control means gateways to manage the systems. Home automation means products to interact together and be controlled by one another.

• SOAP

Originally defined as Simple Object Access Protocol, is a protocol specification for exchanging structured information in the implementation of Web Services. It relies on Extensible Markup Language (XML) as its message format, and usually relies on other Application Layer protocols (most notably Remote Procedure Call (RPC) and HTTP for message negotiation and transmission. SOAP can form the foundation layer of a web services protocol stack, providing a basic messaging framework upon which web services can be built. This XML based protocol consists of three parts: an envelope - which defines what is in the message and how to process it - a set of

encoding rules for expressing instances of application-defined data types, and a convention for representing procedure calls and responses [Wikipedia].

- **Simple Service Discovery Protocol**

Simple Service Discovery Protocol (SSDP) is a Protocol to allow network clients to discover network services such as printers or external disk drives with little user interaction. SSDP is supported by Microsoft Windows operating systems. The advantage of using SSDP is that devices can advertise their capabilities and enable other machines to access those services without any need for the remote users to know anything about the network address or name or need to manage any network infrastructure in order to facilitate accessing other devices [DYN07].

- **Jini Lookup Service**

"Services are found and resolved by a *lookup service*. The lookup service is the central bootstrapping mechanism for the system and provides the major point of contact between the system and users of the system. In precise terms, a lookup service maps interfaces indicating the functionality provided by a service to sets of objects that implement the service. In addition, descriptive entries associated with a service allow more fine-grained selection of services based on properties understandable to people." [JIN06].

- **The Session Initiation Protocol**

The Session Initiation Protocol (SIP) is a signaling protocol, widely used for controlling multimedia communication sessions such as voice and video calls over

Internet Protocol (IP). Other feasible application examples include video conferencing, streaming multimedia distribution, instant messaging, presence information and online games. The protocol can be used for creating, modifying and terminating two-party (unicast) or multiparty (multicast) sessions consisting of one or several media streams. The modification can involve changing addresses or ports, inviting more participants, adding or deleting media streams, etc.

- **Service Integration**

The service integration is making a new service from more than one Service cooperating with each other [TOK02].

- **Universal Description, Discovery and Integration**

Universal Description, Discovery and Integration (UDDI) is a platform-independent, Extensible Markup Language (XML)-based registry for businesses worldwide to list themselves on the Internet. UDDI was originally proposed as a core Web service standard. It is designed to be interrogated by Simple Object Access Protocol (SOAP) messages and to provide access to Web Services Description Language (WSDL) documents describing the protocol bindings and message formats required to interact with the web services listed in its directory [Wikipedia].

- **The Context**

It is all information which can be used to characterize an entity [RIC06].

- **Extensible Markup Language**

Extensible Markup Language (XML) is a simple, very flexible text format derived from SGML (ISO 8879). Originally designed to meet the challenges of large-scale

electronic publishing, XML is also playing an increasingly important role in the exchange of a wide variety of data on the Web and elsewhere.

- **The Service Location Protocol**

The Service Location Protocol (SLP) is a service discovery protocol that allows computers and other devices to find services in a local area network without prior configuration [Wikipedia].

The Service Location Protocol provides a scalable framework for the discovery and selection of network services. Using this protocol, computers using the Internet need little or no static configuration of network services for network based applications. This is especially important as computers become more portable, and users less tolerant or able to fulfill the demands of network system administration [GUT99].

- **Salutation: A light Weight Network Protocol Independent Service Discovery Protocol**

Salutation is a service discovery and session management protocol developed by leading information technology companies it is an open standard independent of operating systems, communication protocols, and hardware platforms. Salutation was created to solve the problems of service discovery and utilization among a broad set of appliances and equipment in an environment of widespread connectivity and mobility. The architecture provides applications with different services that are scattered all throughout the network. It also contains functions to find out capabilities of remote services. Salutation provides features for an application to establish interoperable sessions with any remote service [HEL02].

- **Interoperability**

Interoperability is defined as the ability for devices in home to discover, configure, and control the capabilities of peer devices.

- **Middleware**

Middleware is the "glue" between software components or between software and the network or it is the slash in Client/Server.

Also middleware can be defined as the software technologies, architectures and systems that promise seamless interconnection to devices with different hardware and network characteristics are often called middleware

- **Heterogeneity**

Different interfaces and architectures of diverse components exist in a specific domain.