

HOUS_e-KEEPER,
a vendor-independent architecture
for easy management of smart homes

Jean-Marc Seigneur,
Department of Computer Science,
Trinity College, Dublin.

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

2001

Declaration

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

Signed: _____

Jean-Marc Seigneur

The 14th of September 2001

Permission to lend and/or copy

I, the undersigned, agree that Trinity College Library may lend and/or copy this dissertation upon request.

Signed: _____

Jean-Marc Seigneur

The 14th of September 2001

Acknowledgements

First and foremost, I wish to thank my family for their support and patience in completing this dissertation. Also, I would like to thank my supervisor, Mr. Alexis Donnelly, for the advice and comments that I received. Thanks to the European Union. Finally, many thanks to those who helped me.

Abstract

Home-networking is gaining momentum. In a couple of months, Windows XP will be launched with the connected home experience as one of its core areas of interest. In the medium term at least, there will be more than one home networking “middleware” in a smart home, a home populated with smart, e.g., Web-enabled or network-enabled, devices. Nevertheless, the home system will have to present a global and complete view of smart devices in the home without excluding devices from one or more different high-level home networking technologies.

HOUSE-KEEPER proposes an open architecture into which home networking “middleware” can be “plugged”. In addition to being independent of the underlying high-level home networking technologies, it provides a means to monitor and control smart devices remotely over the Internet, as well as to easily grant access control and add some context information to them. These features are still in development for the two main high-level home networking technologies, Jini and UPnP. Moreover, HOUSE-KEEPER offers a real prototype of a smart home network accessible from the Internet for residential end-users. A lamp can be switched on from a Web page or the kettle for coffee water can be turned on from a WAP phone. The prototype demonstrates how it is possible to address the requirements of a smart home system thanks to the HOUSE-KEEPER architecture.

The beginning of this document presents the domain of home-networking and its state of the art review. Then, the HOUSE-KEEPER architecture is described. Finally, there is an analysis of what has been achieved and what has been brought to light.

Table of Contents

1.	INTRODUCTION.....	1
1.1.	Aims and objectives	1
1.2.	“Road map”	3
2.	DOMAIN.....	4
2.1.	Home Networking.....	4
2.1.1.	<i>Definitions</i>	4
2.1.2.	<i>An overview</i>	4
2.1.3.	<i>Technology survey</i>	8
2.2.	Research case studies	11
2.3.	Requirements	13
3.	REVIEW	17
3.1.	Home network “middleware”	17
3.1.1.	<i>Chai</i>	17
3.1.2.	<i>UPnP</i>	20
3.1.3.	<i>Jini</i>	21
3.2.	The missing link	24
3.2.1.	<i>Candidates</i>	25
3.2.1.1.	OSGi	25
3.2.1.2.	The Chai Place Manager.....	29
3.2.2.	<i>Is it possible to fulfil the requirements with the actual technologies?</i>	31
4.	HOUSE-KEEPER : THE NEW STEP.....	34
4.1.	The big picture	34
4.1.1.	<i>Assumptions</i>	34
4.1.2.	<i>Multi-tier architecture</i>	36
4.2.	Further detail	40
4.2.1.	<i>The choice of the server</i>	40
4.2.2.	<i>A service independent of the home networking technology</i>	41
4.2.3.	<i>An easy and vendor-independent declarative process for device context</i>	45
4.2.4.	<i>The answer to the multi-user security requirement</i>	47
4.2.5.	<i>The need-to-know issue</i>	50
4.2.6.	<i>More about the Jini proxy for Jini devices</i>	53
4.3.	Critique of the architecture.....	54
4.3.1.	<i>Place metaphor</i>	54
4.3.2.	<i>What has to be shipped?</i>	55
4.3.3.	<i>Limitations of the command line paradigm</i>	56
4.3.4.	<i>Is HOUSE-KEEPER a distributed system?</i>	57
4.3.5.	<i>Adaptation to the client capability</i>	58
4.4.	The prototype and its results	59
4.4.1.	<i>Test-bed</i>	59
4.4.2.	<i>“Use-cases”</i>	61
4.4.3.	<i>What did two implementations bring to light?</i>	64
5.	CONCLUSION	69
6.	REFERENCES.....	72
7.	WORLD WIDE WEB RESOURCES	76

Table of Figures

Figure I: High-level view of home networking architecture	8
Figure II: HP Chai platform overview	17
Figure III: HP ChaiServer components	18
Figure IV: Simplified architecture of an OSGi solution	26
Figure V: Components of an OSGi-based residential gateway	27
Figure VI: HP Cooltown Web Presence Manager architecture diagram	30
Figure VII: Back-end of the HOUSE-KEEPER architecture.....	36
Figure VIII: Front-end of the HOUSE-KEEPER architecture	38
Figure IX: Simplified technology stack of HOUSE-KEEPER	40
Figure X - HOUSE-KEEPER tightly coupled with Chai.....	42
Figure XI - HOUSE-KEEPER independent of the home networking technology	42
Figure XII: Subclasses of an abstract “device EJB” class	44
Figure XIII: Photo of the complete HOUSE-KEEPER test-bed	60
Figure XIV: Photo of the smart devices side of the HOUSE-KEEPER test-bed	60
Figure XV: Web page representing the place “home”	61
Figure XVI: The place “home” as displayed on a WAP phone	62
Figure XVII: Web content of the use case for “jm”	63
Figure XVIII: Web content of the use case for “alexis”	63
Figure XIX: Web representation of a Chai device	65
Figure XX: Web representation of a Jini device	66

1. Introduction

1.1. Aims and objectives

Currently, two main technologies compete to be the standard in home network middleware. The obvious reason is that the market is extremely significant, speaking of systems which will have the ubiquity of televisions, the upgradability of PCs (Personal Computers) and the broad possibilities of software applications. Cahners In-Stat Group [CISG00] expects more than a \$2 billion Internet residential gateway market segment by 2003.

Hence, Microsoft [MS] promotes UPnP [UPNP], a technology closely related to its set of products, and Sun Microsystems [SUN] continues to develop its Jini [JINI] technology, a Java-based [JAVA] solution for intelligent network infrastructure. In the following chapters, we will describe how these technologies provide all functionalities for federating those new Web-enabled, network-enabled daily appliances, called smart devices.

These technologies can discover devices that are plugged into the network and speak the same protocol. They also can search for devices of interest. A problem arises. How can a device from one vendor be discovered by middleware belonging to another vendor if the protocols are incompatible? Both of them will also provide different APIs (Application Program Interfaces) for publishing and receiving events at a programmer level, as well as different ways for defining and describing the devices and what they can do. Although APIs are convenient for programmers, end-users would prefer more user-friendly interfaces for using their daily appliances. Furthermore, the end-user would prefer to find all interesting devices by making a single query rather than making one query for the UPnP appliances and another for the Jini ones.

This dissertation presents a high-level service for home networking: the first end-user service for simply managing a smart home, locally or remotely over the Web, based on an open architecture, independent of the underlying home network technology. The following chapters will review the current solutions, many services have been created for programmers

to build applications for end-users more efficiently. Surprisingly, few services have been built for end-users. No such high-level service has been created to allow end-users to control and to monitor a smart home from the Internet yet. The prototype born from this dissertation is called HOUSE-KEEPER, pronounced “house-e-keeper” to imply an electronic house-keeper, as in the e of e-commerce. HOUSE-KEEPER uses the services offered by the current home-network middleware technologies to programmers to offer a real user-friendly service to end-users. The HOUSE-KEEPER service is real. The prototype can interact with real X10 [X10] devices wrapped in Jini or Chai [CHAI] pieces of software to switch on a real lamp remotely from the Internet or the coffee machine from a WAP phone [WAP]. The fact that it is controllable from the Internet transparently for different types of home networks is new. For example, as will be described, there are different on-going projects to find a way of using Jini devices from the Internet. One of them is the SOAPUDDI project [SOAPUDDI] of the Jini.org community [JC]. The HOUSE-KEEPER service addresses this issue. In fact, the HOUSE-KEEPER service provides a global and complete view of what is available in the home network. Its design is independent of home network middleware. As a matter of proof, the design has been prototyped with two different home network technologies, Chai and Jini. The HOUSE-KEEPER service also offers an easy-to-use declarative system of management of this home network rather than the programmatic one actually implemented. This way, it is possible to implement additional access control, even if the native device does not provide such access control.

The focus of this dissertation is on the home, not on corporate buildings or on public buildings. This is important because the requirements are different between corporate or public buildings and homes. By home, we understand a residence in which few people are living, such as a family with the husband, the wife and their children. Concerning the security requirement for example, the assumptions made do not fit very well in a corporate environment. This is the reason the service based on the overall architecture is called HOUSE-KEEPER. The service helps to manage different appliances in the house by storing the devices automatically and helping users to find them. The idea is that a future version of HOUSE-KEEPER could do the traditional job of the human house-keeper, who tidied the house and notified the landlord of the house when something happened and only when it was really needed. A simple example of this behaviour has been added in the prototype, but more intelligent behaviour could be implemented using the help of artificial intelligence technologies.

1.2. “Road map”

This document starts to set the scene of the home-networking domain. It gives some definitions and an overview of what residential users expect from a smart home. Then, a survey of the different technologies involved is done. A list of a few research projects related to smart places follows. At this stage, a summary of the requirements for home-networking and smart home systems is presented.

In chapter 3, technologies that are likely to address these requirements are reviewed. The analysis of these technologies demonstrates that they cannot by themselves answer the requirements. There is a kind of missing link. Advantages and disadvantages of two candidates, which could fill the gap, are discussed. However in the end it is apparent that a new solution is needed.

Chapter 4 describes this new solution, called HOUSE-KEEPER. The assumptions on which the design relies are explained. Afterwards, the high-level view of the architecture is given. Along with this description, the main design ideas and the benefits of the chosen technologies are presented. Some special issues addressed by the architecture are detailed. Difficulties found during implementation and their solutions are also explained. A discussion about other issues, which could have been dealt with differently, follows. Then, there is a description of the prototype and what the different implementations have brought to light.

Finally, the conclusion goes through each of the defined requirements in section 2.3 and summarizes what HOUSE-KEEPER offers for each of them. Of course, some work can still be done. For this reason, a list of future work ends the document.

2. **Domain**

This chapter sets the scene of home-networking, which is the domain of the HOUSE-KEEPER project. After some definition and an overview of what residential users expect from a smart home, there is a technology survey and a look at different research projects in smart places. This chapter ends up with the requirements of smart home systems.

2.1. **Home Networking**

2.1.1. **Definitions**

The CEA (Consumer Electronics Association) [CEAGTH] defines a home network as follows: “a home network facilitates communication among the appliances, home systems, entertainment products and information devices in a home, so they can work cooperatively and share information. This allows users to get information about the home's condition and remotely control home systems and appliances, as well as provide access to information and entertainment resources outside the home”.

From the International Engineering Consortium [IECHN], home networking is “the collection of elements that process, manage, transport, and store information, enabling the connection and integration of multiple computing, control, monitoring and communication devices in the home” (p.1).

2.1.2. **An overview**

This section is not technical. It gives an overview of why users could be interested in smart homes. It will help to define the HOUSE-KEEPER service, which targets real residential end-users. This is the normal process for finding out who the users are before building the software. To look at home networking market research is one way to do that and is required in order to build a service for a specific domain.

As [Glick99] outlines:

the home networking market is broken into five major categories: lighting and window treatments, security and access control, voice and data communications, environmental and energy management and finally, audio and video entertainment.

Home networking brings the power of distributed content to actual home automation technologies. [UCSQ01] underlines that “study after study indicates that sharing a common connection to the Internet remains the primary reason why people establish a home network” (p.4).

One question is whether the sales of home PCs will continue to increase or if the PC will be broken up into a variety of information appliances. A user research study [FDS00] combined with traditional market studies has shown there is “a real need for distributed computing and Internet access around the house” (p.4), but it is not known whether it will be done with PCs, appliances or a mixture of both. “What seems to be required is a shared multifunctional device, more focused on function than a PC but less restricted in function than an application-based appliance” (p.21). This notion of multiple users is important. We will speak more about the “Family Computer (FC)” rather than the “Personal Computer (PC)” (p.21). Teenagers motivate parents to surf, but parents want to control the children’s use of the PC, especially if it becomes ubiquitous. Access control for parents is essential. “There is a great demand for PC relocation” (p.15). Children would like their own PC in their bedroom. On the other hand, it is said that parents would prefer to leave the main PC in the place they chose initially and to have other devices around the house to access fun functionalities of their PC. There is also an “association between activities and locations” (p.17), e.g., scheduling in the kitchen. Due to contention that arises during peak PC usage time, e.g., after dinner, there is a need to access the PC remotely during daytime. The user may require access to the information stored in the home PC at any time. To a certain extent, the information contained in the home PC is part of the users. The home network will have to be monitored and controlled. HOUSE-KEEPER allows for the control of smart devices present in the home network. The need for daytime access means those smart devices will have to be controlled remotely. It is another facet of HOUSE-KEEPER to access the home network remotely from a Web page or a WAP (Wireless Application Protocol) [WAP] phone.

Another side of this question is if the home network controller will be integrated into the main PC or will be provided as a stand-alone box. The user research [FDS00] concluded there were “more comments about access and control than about usability and reliability” (p.20).

In [Glick99],

about 31 percent of those surveyed said they prefer to operate their electronics with a remote control, while 23 percent said they prefer to use a computer to do the job. But more affluent homeowners — those making \$60,000 or more per year — said they would prefer to use a computer to control their home network, while those with a yearly income of \$30,000 and less said they would rather use a remote control.

Others [IECHN] said the key drivers

for home networking are simplicity and reliability. If the process is not maintenance-free, easy to use, and quick to install, it will not likely be embraced. While consumers desire the sophisticated functionality of LAN — voice networking, shared Internet access, and smart device control — they do not wish to engineer or administer a complex system. They want plug-and-play functionality, believing that only experts require technological know-how and that — much like telephones — PCs should just work. (p.3)

What will drive networks in the home? It could be the PC, the TV or something else. [Glick99] states that “consumers demand reliability and choice. They do not want to be limited by a single connection to the network, especially not an unstable one” if we consider the PC solution. Residential end-users cannot become network administrators.

The final trend focuses on intelligent buildings where the home network system will transform the house automatically as a comfort zone. At the 2000 International CES (Consumer Electronics Show) [CES], representatives of some 80 different technology companies and associations met during the Home Networking Search Summit to discuss home networking concepts and their translation into the mainstream. They defined key features: easy-to-use, intuitive systems that are personalized to meet the demands of individuals' and families' lifestyles. [Elkin00] reports that “people, for the most part, want to have Internet connectivity, and they want to be able to make lifestyle improvements with

these control systems where they touch one button and it sets the house up for an evening at home”. While others would not mind getting updates on the home's condition via an email, consumers do not want to be spammed by their homes. This is one reason it is necessary for intelligent buildings to know how to reply to events automatically without disturbing the user. Users are interested in having access to the information they want and the ability to make use of it wherever, whenever and however suits them.

The next step in intelligent building is called the “aware home” [Sanders00]. With human-like perception, it could improve the quality of life for many, especially senior adults. The next generation of technologies for end-users will have computers that understand what people are doing, what they want and who people are, rather than needing a human being in charge of the remote control, for example. The first steps of home awareness do not require very advanced technologies as it has been said in the Xerox research [MBBMM98]:

A minimal impact ubiquitous computing interface, requires an environment aware of the context of its use and the likely intentions of its occupants. This might sound like we are chasing after the holy grail of artificial intelligence, but our experience has shown that context awareness can often be achieved through simple, yet effective, measures.

The UbiComp [Buxton95] view is that rather than adding complexity, technology should reduce it. How can ubiquitous sensing give computers a decision-making context like humans have? The next step will be full artificial intelligence, but research experiences at Xerox Parc have shown that simple decisions can be made with the measures of today. In the HOUSE-KEEPER service, a decision to send an email to the owner of a smart device is taken based on the context at the moment of the decision. This avoids spamming the HOUSE-KEEPER user's email. Take the example that a user is using the HOUSE-KEEPER service and a device is found to be out of order. If the user is the owner of the device, in real time a message will warn her or him that this specific device is out of order. Otherwise the user is not warned but an email is sent to the owner of the device saying this device has been found out of order by the current user. Hence smart spaces need to be aware of user context and support a wide range of interfaces. Reasonable access methods must be provided: search by word is better than the scrolling functionalities of a lot of devices, but another step will be to have vocabulary algorithms to search by concept. The notion of previous knowledge arises here. What happens in the house will have to be captured and made available for later access.

Thanks to this knowledge database, it will make it possible for the system to take decisions more easily. The history of what has happened will be needed to take decisions. The notion of history is related to the need of data persistence.

2.1.3. Technology survey

The diagram below describes a high-level view of a smart home network.

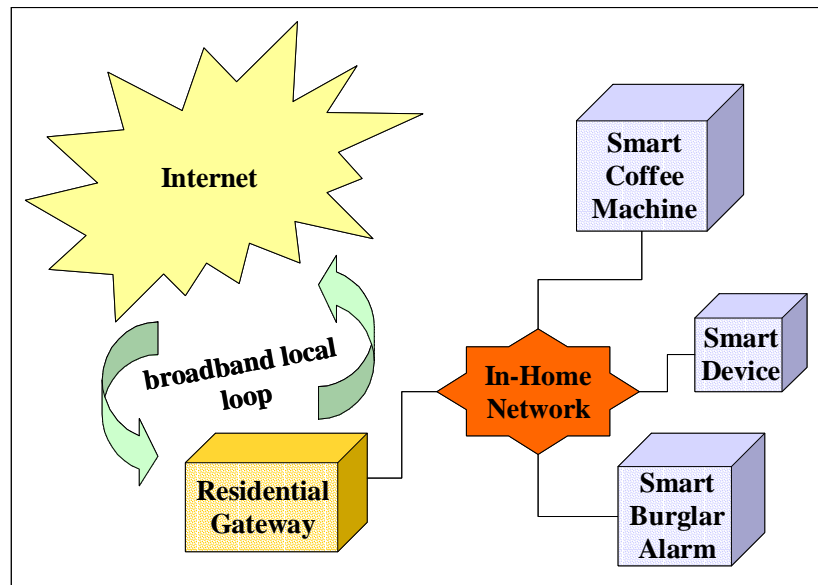


Figure I: High-level view of home networking architecture

We have an interaction between the Internet, a broadband local loop and the in-home network. In the in-home network, there is a residential gateway that links the outside with the network of smart devices or information appliances. Roughly, the broadband local loop will be provided by technologies giving a permanent connection such as xDSL [DSL], cable or satellite. The residential gateway will be the LAN (Local Area Network) –WAN (Wide Area Network) interface. An interesting candidate for this part should be the OSGi specification [OSGi00]. As will be explained later, this is a good candidate in the event that the residential gateway should become separated from any home PC. As is said in a paper on residential gateways [Meadows00], “the residential gateway is what makes the home network infinitely more useful”. The number of homes with a broadband connection is expected to grow rapidly. This is an important factor because broadband access is what makes home networking and residential gateways attractive.

The first obvious benefit of a home network will be to solve the problem of sharing the Internet connection and the diverse computer peripherals in a household with many computers. Later, all that can be plugged into the home network will be shared, and other benefits will come. Basically, there are four types of home networks:

- The traditional LAN
- The powerline
- The phonline
- The wireless

A point to bear in mind when reading about each type of home network is the transmission rate or speed of the network. Digital audio and video require faster networks. The transmission rate can commonly range from 5Mb/s to 10Mb/s for television quality MPEG2 [MPEG] video. There are many candidate technologies for the in-home network. It is possible to mix those different home networks to get the complete network. It is interesting to see those different candidates and their mixing as the creation of a “multi-layered home network” [Enikia99]. Obviously, the traditional wired Ethernet [ETH] is not a viable solution. Even though it is the corporate standard and has low cost for high speed, it requires that the house be rewired with category 5 UTP (Unshielded Twisted Pair) cable. High-speed powerline (HomePlug [HPLUG]) and phonline (HomePNA [HPNA]) technologies can act as a “backbone network” [Enikia99]. The wireless & RF (Radio Frequency) technologies (HomeRF [HRF], Bluetooth [BTOO], IEEE 802.11b [WETH] and Sharewave [SWAV]) could solve the ubiquity requirement and be considered as a “mobility network layer” [Enikia99]. Another layer could be seen as a “control network” [Enikia99] and should consist of the low-speed powerline automation technologies such as X10 [X10], CEBus [CEBUS] or Lonworks [LWOR].

This X10 [X10] powerline technology communicates between transmitters and receivers by sending and receiving signals over the powerline wiring. These signals involve short RF (Radio Frequency) bursts which represent digital information. The distance between the devices and controller should be less than 80m. The connection is made via home AC (Alternating Current) outlets, by plugging a module over a specific outlet. Modules can take an address from A to P with 16 numbers for each letter, e.g., A15 or E3. Thus there are 256 addresses available. It is also possible to set the same address for many devices to be able to

control them using one address for all. This should be considered a low-speed control network. It can be used in different countries in the world since it is available in different kinds of plug types. Some of the modules are two-way modules; they can be polled to know their states in addition to being controlled. The HOUSE-KEEPER service has used this technology to make real applications, such as dimming a desktop lamp from a Web browser.

One should also mention technologies such as infrared-based technology (IrDA DATA and IrDA CONTROL [IRDA]); an old European home automation network (Batibus [BATB]) and the formal merger of three leading European systems for home and building automation (EIB-Konnex [EIBK]); a Japanese evolutionary system which can remotely control and monitor almost any electric device (OpenPLANET [OPPL]); SCP (Simple Control Protocol) [SCP] related to UPnP for controlling small appliances not able to support a TCP/IP stack. Another one is HAVi, which is based on the IEEE 1394 standard also called FireWire [FIRW] by Apple [APPLE]. HAVi [HAVI], dedicated toward entertainment systems, is one of several technologies with high-level functionalities. It can make use of FireWire to detect new devices, and all HAVi devices on the network can make use of functionalities offered by any other HAVi device. In fact, higher level protocols wrap low-level hardware pieces in the home network.

There is a need to discover and federate the smart devices available. We are speaking more about middleware technologies. These technologies will be described in detail later since they have been used for building the HOUSE-KEEPER service. First of all, they rely on different service discovery protocols (SSDP (Simple Service Discovery Protocol) [SSDP], SLP (Service Location Protocol) [SLP] or Salutation [SALU]). As indicated in a paper on service discovery protocols [BR00], some of these technologies can be seen as a service discovery protocol. This is the case for UPnP or Jini. At this level of discovery, it could be better to compare them with SLP, the discovery protocol implemented in HP (Hewlett-Packard) [HP] Chai technology, the Bluetooth Service Discovery Protocol and the service discovery in Salutation. Furthermore, UPnP has its separate discovery protocol called SSDP. Nevertheless, in this document they are presented as a framework for a home network and they encompass more than a stand-alone discovery protocol. In fact, they are understood as a part of a much more complex environment: UPnP is a piece of the Microsoft “.NET” [DNET] architecture that uses SOAP (Simple Object Access Protocol) [SOAP], SSDP, SCP and soon C# [CSHA], while Jini brings Javaspaces [JSPAC] and many other services. They

provide other high-level services such as event services, device description and so forth. Section 3.1 will review the major technologies.

2.2. Research case studies

Below, some research projects related to smart places are described to provide an overview of previous work in the field.

- Voxi [Dübendorfer01]: The aim of this project was to create virtual counterparts for real-world objects. The researchers have built a general event-based system and an object-oriented representation. They have used Java and Jini for their implementation. Their future work is focused on developing a stronger solution for security and privacy.
- Mira [ACKMM00]: This project provides a distributed XML (eXtended Markup Language) [XML] multimedia repository. A framework of location functionality is added by means of Jini. There is a database. Data from sensors are encoded using XML. Javaspaces, a Jini service, is used for a communication pool for resource sharing.
- EasyLiving [BMKKS00]: This Microsoft research project has created a new middleware dedicated and optimised for the smart world. Their middleware is content and message oriented rather than method driven. XML messages are passed on asynchronously. The researchers use 3D metaphors of houses. There is some work on location functionality. In some parts, they would like to replace polling with an asynchronous event system and to increase the security. The middleware has a discovery service, whose design is based on a MVC (Model-View-Controller) design pattern to get the service abstraction. They have built this middleware in C++.
- iRoom [FJHW00]: This project is based on an infrastructure centric scenario. They have used RPC (Remote Procedure Call) over HTTP [BFF96]. Some servlets [SERV] are plugged into a Web server. Events are sent to an event heap. The overall system is method driven. Again there is a lack of security. It has been written in Java, helped by TSpaces [TSPA], an IBM [IBM] network middleware, which may be summarized as a network communication buffer with database capabilities.
- Cooltown [DC00]: This HP research project [COOLT] is a complete smart world solution. Recently, an open-source project has been started to explore this vision further. The solution is Web based, as well as content based. A simple security mechanism is available. The researchers use HTTP, Web servers and Java intensively. Their future work is to provide interoperability with the other technologies available. Roughly, each entity has a Web page. An entity can be a person, a place or a thing.
- Intelligent building [CCPS00]: In this project, they regard a building as a machine or a robot. They use Lonworks [LWOR] at the room level. Their future work would be to add fuzzy logic or advanced algorithms to their agents. In fact, they use the Internet protocol between their agents. Previously based on CORBA (Common

Object Request Broker Architecture) [CORBA], it is currently based on Jini and its Javaspaces service.

- Gaia [CGKKNMR00]: The aim is to provide an operating system for smart places. The researchers have thought about a Unified Object Bus over DCOM (Distributed Component Object Model) [DCOM], CORBA or EJB (Enterprise JavaBeans) [EJB]. The main design pattern will be MVC (Model-View-Controller). They have different schemes (Jini, CORBA and DCOM) and different networks (X10, Bluetooth, wired and wireless Ethernet). They have still to work on the implementation of the concept.
- The Context Toolkit [DAS99]: This toolkit is aimed to help in the creation of context-aware applications. It combines Java and XML over HTTP. Future work could be to have a more sophisticated event handler.
- RBMO [OJM97]: The objective of this project is to provide remote building monitoring and control. The implementation is CORBA based. The researchers give an interesting list of requirements for this type of application.
- The Adaptive House [Mozer99]: The project demonstrates that an event-based system is good for home automation. To a certain extent, the home could program itself. To the event-based system the researchers have added neural networks for prediction and control. They would like to get long-term statistical results.
- Domiscilica [MA97]: In this project, there is a centralized model of the home for controlling devices improved with computational power. There is a central database containing a model of the home. The researchers have created a Java toolkit for creating a user interface. Nevertheless, they have a lack of security and they would like a stronger event handler. They use sockets and Web servers. Multiple metaphors are provided for representing the smart home and interacting with it. Finally, they have placed an emphasis on location issues.
- AirJava [Mills99]: The aim of this project is to create a test-bed for creating smart places and to increase intelligence in buildings. The researcher highlights the issues for dealing with security and context. The main technologies used are Java, Jini and its Javaspaces service.
- BatTeleporting [HHSWW99]: The purpose of this project is to develop location-aware programming. The researchers say that context aware systems should know as much as the user about those aspects of the environment that are relevant to their application. Their future work should be to develop these prototypes into large-scale, fully-deployed services. They use a 3-tier model. They get persistent distributed objects using CORBA and Oracle [ORAC] databases. They have created a location-aware API.

In summary, it appears that Java is extensively present in smart building projects. This may be due to the fact it eases platform independent applications. Most of the projects listed here include Java. Jini [JINI] and JavaSpaces [JSPAC] are present in approximately 40% percent of the projects. It seems that they have moved from CORBA-based [CORBA] solutions to the latter solution since the start of research in smart building architectures. There is an overall drive toward using event-based systems. No projects have used either

UPnP [UPNP] and Cooltown technologies [COOLT] or OSGi [OSGI]. Most architectures are method driven but some projects claim the best solution may use a content-based approach by exchanging messages. In fact, XML [XML] has made its emergence in implementations for generating and exchanging standard messages. A lack of security has been noted in many projects. Another concern in the research area is the issue of locating people, things and places.

2.3. **Requirements**

As the overview, the technology survey and the research projects in the field have introduced, home networking comes with a few requirements. This section lists and describes these to bear them in mind. The need to fulfil the requirements guides the HOUSE-KEEPER architecture and helps to choose the best alternative.

- Multi-user: It is clear that the home is a place where different people live and interact. Furthermore, [FDS00] highlighted that there is a hierarchy in the multi-user environment of the home. For example, there is the head of the family and the family members. The parents could be considered administrators of the home network, having the last word concerning its use.
- Context-based: [DAS99] has shown that the context will play a major role in ubiquitous computing. In [DAS99], there is a requirement for allowing applications:
 - to access context information from distributed machines in the same way they access user input information from the local machine
 - to support execution on different platforms and the use of different programming languages
 - to support for the interpretation of context information
 - to support for the aggregation of context information
 - to support independence and persistence of context widgets
 - to support the storing of context history (p.7)
- Maintain history of the state of the environment: Several research projects, such as Voxi [Dübendorfer01] or The Context Toolkit [DAS99], have brought to light the need to store past information to be able to use it when required. [Coen99] proposes this principle: “throw everything you can at a problem”. Some information could become essential. If this information is stored, it is possible to “throw” it when required.
- Adaptation to device capability: This can be seen as necessary to adapt the content of a Web-like page, whether it be a Web browser running on a PC or a limited WAP phone display. Coming next, [V2] tries to define a standard protocol that will

enable IT (Information Technology) products to be more accommodating of the needs and preferences of the consumer by allowing for alternative user interfaces.

- Location awareness: Thanks to the next generation of mobile devices, it will become easier to locate people and things. [Dübendorfer01] mentions different types of positioning systems from infrared sensors to GPS (Global Positioning System).
- Security: Obviously this is the first non-functional requirement in home networking. As we have seen in section 2.2, most of the research projects on smart places are aware of the importance of security, but most of them have not dealt with the issue yet. However, from a security point of view, it is recommended in [MV00] to deal with security from the start of the project. Anyway, security has been taken into account from the beginning of the HOUSE-KEEPER project. Section 4.2.4 deals with this requirement.
- Asynchronous event-based model: [Meier00] explains that the right event service depends on the application. For example, [Barron99] proposes an event service to fit into the specific smart building infrastructure developed by the Intelligent Interfaces and Buildings group from Trinity College Dublin. iRoom [FJHW00] and the Adaptive House [MM98] research have explained why an event-based solution is needed to solve particular issues arisen in smart places. The latter demonstrated that an event-based segmentation provides a solution for controlling a complex system of lights around a house, reducing energy costs and maintaining user satisfaction. They have studied the area of adaptive control and home automation, a problem that appears intractable when cast in terms of a clock-based segmentation, but has an extremely simple solution when cast in terms of an event-based segmentation. From a research point of view, it is important to notice that to implement an event system is a high priority for many smart places research projects, e.g., EasyLiving [BMKKS00] or Domisilica [MA97]. Others have explicitly mentioned as an important requirement the presence of an event system, in Voxi [Dübendorfer01] and The Context Toolkit Kit [DAS99].
- Interoperability: The technology survey has given an idea about the broad range of technologies for creating smart homes. Concerning the physical network, there are three main solutions: the phoneline, the powerline and the wireless option. On the other hand, at the level of discovery and communication between smart devices, there are many on-going solutions such as Jini, UPnP, and HP Chai technologies. In fact, protocol specifications and standardization need future effort. Network technology for the home continues to evolve and revise itself at an increasing rate. The biggest problem for industries trying to develop smart devices for the home is interoperability, the ability for smart devices to work together to achieve a benefit for the residential user, without the residential user having to be a network engineer.
- Simplicity: The Cooltown research project [KBMBC00] thinks that a Web place master, a Webmaster for Web sites representing places, should not need to be more sophisticated than a Webmaster. Some well-known metaphors, such as the Web surfing experience or the traditional windows GUI (Graphical User Interface), should be used to ease the user learning phase.
- High level of abstraction: To build complex home systems, some APIs dealing with low-level issues to federate smart devices in the home network are needed. [CD00] underlines that discovery services should be available for programmers. They

should be easy to use. [KBMBC00] shows that a layered infrastructure helps to focus on what is needed. A programmer who has to create a GUI listing the different smart devices in the home should not have to know how these smart devices are found. In [BMKKS00], a service abstraction encourages the separation of the hardware device control, the internal logic and the user interface presentation.

- Relationships with other entities [CD00]: There will be interaction with different entities. The entities will be of different types and will represent various things, people, places, services, etc.
- “Open-ended entity description for programmatic search” [CD00]: Although standards organization such as UPnP standardize entity descriptions that will dictate what attributes can be searched, non-standard attributes should also be possible. This requirement reminds us that this is an evolving domain with tremendous unknown possibilities, in which change will occur even though standards have been chosen.
- AI (Artificial Intelligence): The notion of artificial intelligence will play an important role if we think about an “aware home” [Sanders00]. The users will not want to be flooded by a perpetual information stream. There is a requirement for the system to react to events occurring in the home network without the help of the user up to a certain extent.

In the RBMO [OJM97] project, the researchers discuss middleware requirements for remote monitoring and control applications. These requirements are extracted from experiences with remote building monitoring and operations of commercial buildings HVAC (Heating, Ventilating and Air Conditioning) systems. This domain is different from the home. For example, entertainment is important in the home domain, but not really in commercial buildings. Nevertheless it is interesting to notice several similarities between their requirements and those listed above.

Middleware requirements for remote building monitoring and control found in [OJM97]:

- event notification: asynchronous, persistent and (often) multicast
- fine grained time synchronization
- security
- naming and directory services
- mediation services
- unit and time conversion
- distributed configuration management
- dynamic substitution of components
- debugging support for distributed systems
- real-time ORBs (Object Request Brokers)

- performance enhancements in general (aggregation of measurements points) [OJM97]

The presence of ORBs points out that this project was done in 1997 when less work had been done in home network middleware such as Jini or UPnP. They want directory and naming services, which are related to the discovery and lookup services of Jini for example. The notion of real-time, synchronization and performance is more specific to the industrial domain than to the home one. There are again requirements for security and event services, as well as the idea of components.

Finally, as well as specific requirements for the home domain there are global requirements, which are worth looking at for choosing whether they are relevant for the HOUSE-KEEPER scenario.

Mobility is beyond the scope of the present project. The main objective is that devices be installed once in a specific location, then controlled and monitored. In other words, it was assumed that the devices would not often move from one place to another. This assumption is credible for heavy appliances like dishwashers, fridges and so on. The latest version of the prototype has been improved with a simple update process, which allows the context, in which the smart device is, to be changed and the view of what is available in a place to be updated. For example, if the dishwasher is out of order when a user looks at the location that contains the dishwasher, the user will not be able to interact with this appliance since it is down.

Scalability is one of the key requirements for global and worldwide infrastructures, as stated in the Cooltown project [KBMB00]. Is it required at a home level? A home should not get a billion entities. The scale does not seem to be crucial. The assumption is made that the HOUSE-KEEPER service is dedicated to a home level. It should be implemented in the home gateway of the next smart homes, dealing with the entities of one home and not more. The HOUSE-KEEPER service is not a worldwide service on its own.

3. Review

This chapter starts to review the different high-level home networking technologies. These technologies are Chai, Jini and UPnP. The analysis of these technologies brings to light that they cannot by themselves answer to the requirements defined in section 2.3. There is a missing link. Therefore two other technologies, OSGi and the Chai place manager, are reviewed to try to fill this gap. Finally, this chapter ends by summarizing the advantages and disadvantages of each current solution and concluding that a new solution is needed.

3.1. Home network “middleware”

3.1.1. Chai

HP Chai is a technology component of the HP Cooltown [COOLT] vision: a smart world studded with smart devices in which the Web can be accessed anytime and anywhere. The complete range of Chai components is shown below. Components displayed with a mosaic background have been used in the HOUSE-KEEPER architecture.

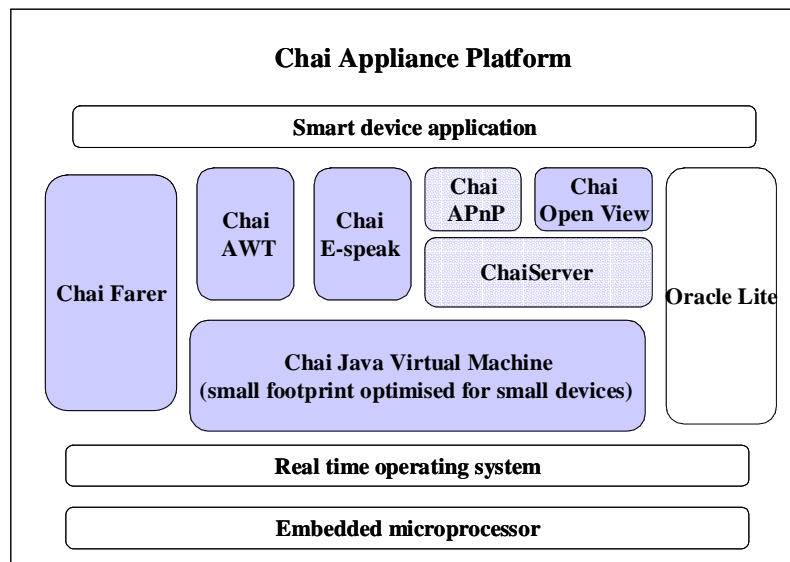


Figure II: HP Chai platform overview

Basically, everything has a URL (Uniform Resource Locator) in the HP smart world. A person, a device, a place, each has a Web page. It is a Web-based architecture. In fact, an appliance has to implement a special Web server called ChaiServer. Most of the components rely on Java. ChaiAWT is an equivalent of the Java AWT packages. ChaiFarer is a small footprint component, which allows the addition of a Web browser or a simple user interface to a smart device. Chai/E-speak helps smart entities to participate in e-services. A ChaiService or a Chailet is a specific service or action that can be called on a ChaiServer. For example, it can be a method for switching on a smart lamp if this lamp has a built-in ChaiServer. The features of the ChaiServer are listed below:

- HTTP 1.1 support
- Remote ChaiService loading, installation, and update
- Event notification and propagation
- Dynamic page composition of views on existing ChaiServices
- Dynamic data support via HTML (HyperText Markup Language) [HTML] streaming
- File-less operation mode
- GateKeeper framework for implementing any security mechanism (nevertheless the shipped security mechanism provides only an authentication mechanism)
- Programmatic access to ChaiServices

The diagrammatic view of those features is represented in the next figure:

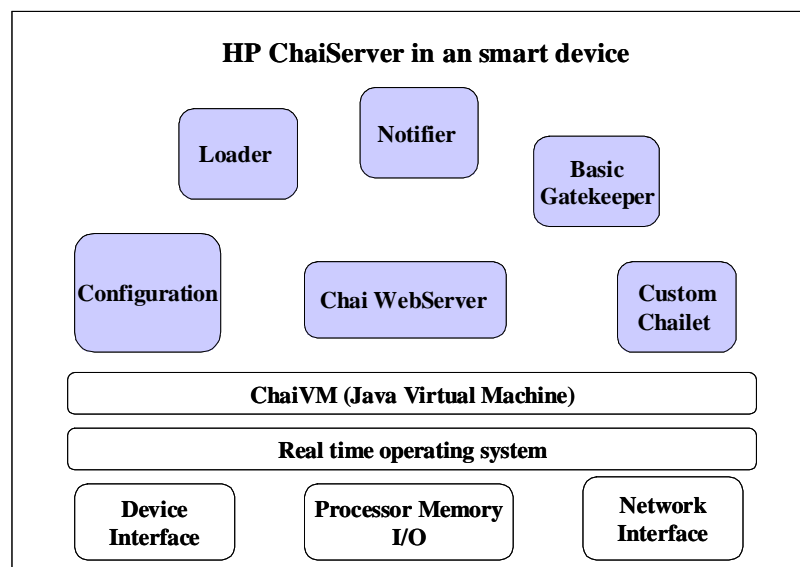


Figure III: HP ChaiServer components

The ChaiServer combined with ChaiAPnP (Chai Appliance Plug and Play) allows for automatic discovery of new devices and services. It is based on the draft version 8 of the Service Location Protocol from the IETF (Internet Engineering Task Force), which has been quite widely implemented in commercial products such as HP's JetSend [JETS] technology supporting amongst other : printers, digital cameras and scanners. As we will see, the Jini architecture principle for discovery is similar to that of SLP. Whereas a service request in SLP returns a service URL, the Jini object code offers direct access to the service using an interface known to the client. This code mobility replaces the necessity of pre-installing drivers on the client. Whereas Jini is dependent on Java, SLP is independent of a programming language. Furthermore, when SLP is combined with the ChaiLoader service, it permits moving code dynamically as well. It is possible to have a proxy for devices without the capability of implementing a Web server. Although there is an optimised JVM (Java Virtual Machine), developed by HP called ChaiVM, the HP ChaiServer can be used on any JVM. As an aside, since HTTP is used to control the devices, the control has a higher chance to pass through a firewall, since HTTP traffic is not normally blocked. To get persistence, the Chai platform has integrated a lightweight professional database, Oracle Lite [ORAC]. Another interesting feature is the ChaiOpenView component, which enables the network management of information appliances using the HP OpenView [HPOV] technology.

In any case, this architecture depends on the ChaiServer, a unique and proprietary Web server. The source code is freely available for development, but royalties are demanded for commercial use. This means that we cannot choose the Web server, and there is a risk of obsolescence of the technology if HP does not update the ChaiServer as fast as the speed of change in the Web domain. The overall architecture comes from a long research project done in the HP Labs. It is well layered, with well-defined components and the interoperability and scalability are promising thanks to the fact that it is based on Web standards. Nevertheless the extensibility will depend in part on the adoption of the architecture. At this stage, the Cooltown vision is not popular among software developers. In July 2001, HP made an open source project called Coolbase to broaden the community of Cooltown developers. One component of this project, the Cooltown WPM (Web Presence Manager), will be reviewed later in the document since it answers many of the requirements listed in section 2.3 and can be viewed as an alternative to specific aspects of the HOUSE-KEEPER architecture.

3.1.2. UPnP

UPnP (Universal Plug and Play) [UPNP] is an architecture for the pervasive peer-to-peer network connectivity of any type of device. UPnP needs no device drivers. It uses common protocols such as TCP/IP and “leverages” what is offered by the Web and the Internet. IP internetworking is one of its strongest features. The foundation for networking is IP addressing. UPnP devices can be implemented using any programming language, and on any operating system. The APIs to control a device are specified by the UPnP Forum, which seeks to develop standards for device protocols and XML-based device schemas for the purpose of enabling device-to-device interoperability in a scalable-networked environment. The UPnP consortium is acquiring momentum. At this date, more than 350 companies, including several big companies, have joined. UPnP is also part of Microsoft's latest technology called “.NET”. The “.NET” [DNET] architecture includes a set of new technologies. It is an opponent to the Java platform: UPnP versus Jini and C# [CSHA] versus the Java language.

Applications which themselves do not function according to the UPnP standard should also be integrable via an appropriate UPnP proxy. Concerning the fact that some devices cannot implement a TCP/IP stack, Microsoft presented the first version of “Simple Control Protocol” [SCP] in June 2000. This protocol will allow non-TCP/IP compatible devices, e.g., household appliances, to be integrated into a network.

The Discovery process relies on the Simple Service Discovery Protocol [SSDP]. The Remote Events feature is handled by the General Event Notification Architecture [GENA], basically notifying using HTTP [BFF96] over TCP/IP. Instead of downloading code, UPnP uses Simple Object Access Protocol [SOAP] to control remote objects. SOAP is a lightweight, XML-based (eXtensible Markup Language) [XML] protocol for exchange of information in a decentralised, distributed environment.

UPnP uses six steps:

- Addressing - first the device must get an IP address. This can be done in a managed network by a Dynamic Host Configuration Protocol [DHCP]. In an unmanaged network UPnP uses AutoIP [Troll00], which defines how to choose an IP address from a set of reserved addresses.
- Discovery - given an IP address, the device is added to the network and advertises its services to control points. A Discovery message contains little information about the devices such as type, identifier and a URL pointing to a location containing more information.
- Description - after a control point has discovered a device, it retrieves more info from the URL given in the Discovery message. The device can host many different services.
- Control - the control point now has to retrieve a more detailed description for each service in the device. This description is expressed in XML and includes a list of commands, called actions, and parameters, called arguments, for each action. The description also includes a list of variables, which model the state of the service. To control the service, a control URL is used to send XML messages using SOAP.
- Eventing - when a variable that describes the state of the service changes, a control point can subscribe to be notified and receive this information. The event notification and subscription is handled using GENA.
- Presentation - if a device has a URL for presentation, the control point can retrieve a page from this URL, load the page into a browser, and depending on the capabilities of the page, allow a user to control the device and view the device status.

As with HP Chai technology, UPnP may easily pass through a firewall since it uses XML over HTTP, and HTTP is not normally blocked. This point is not true for Jini for which ways have to be found for passing firewalls. Nonetheless, if SOAP is not blocked by firewalls now, that could change in the future.

3.1.3. Jini

The Jini (Java Intelligent Network Infrastructure) system [JINI] extends the Java application environment from a single virtual machine into a distributed network of virtual machines. Hence the basic scenario is that a Jini enabled device contains a JVM (Java Virtual Machine).

The purpose of the Jini architecture is to federate groups of devices and software components into a single, dynamic distributed system. The ability in Jini to be a

spontaneously created, self-healing community of services is based on a couple of key concepts. There are three different classes of component:

- Service: piece of functionality made available, e.g., devices or software components
- Client: device or software component that would like to make use of a service
- Lookup service: facility to help clients find and connect to services

Thanks to a feature central to Java, in Jini, services are always accessed via an object provided by the service itself, a piece of Java bytecode downloaded to the client called a proxy. The lowest-level services that Jini provides are listed below:

- Discovery: used to find proxies for the lookup service
- Join: used by services to reference themselves in a specific lookup service
- Lookup: used to find and access services
- Leasing: allows distributed systems to be self-healing (essentially Jini never grants access to resources indefinitely but requires that resource holders continually “renew” their leases on resources)
- Remote Events: allows distributed applications to communicate with one another by using events (the event model used is the Java Distributed Event Model)
- Transactions: allows members of a Jini community to share Java objects

To simplify the implementation of Jini entities, there are a few managers available. Some are more advanced, at a higher level of abstraction, than others. The managers, described in [ER01], can be used by a service or a client but cannot be used as stand-alone services or clients. There are :

- The join manager
- The service discovery manager
- The lookup discovery manager
- The lease renewal manager
- The RMI [JRMI] security manager

There are several Jini services available in the Jini 1.1 release. These services run on their own JVM and perhaps on a different host in the Jini community. Usually they are doing things on behalf of other services and clients. For example, a service that does not do anything for a long time could let someone else handle the renewing of the leases and go to sleep. In fact, Java RMI brings the Activation functionality, which enables clients to sleep if they have nothing to do.

Below is the list of those high-level services:

- Lookup Discovery Service – called Fiddler
- Lease Renewal Service – called Norm
- Event Mailbox Service – called Mercury
- Transaction Service – called Mahalo
- Lookup Browser

The JavaSpaces [JSPAC] technology is basically a way to store Java objects somewhere and is implemented by Jini as a service. Objects stored in a JavaSpaces service are leased. A remote event listener can be registered to be notified when new matching objects are inserted into the space. JavaSpaces is the only service included in the Jini 1.1 release that uses transactions. The JavaSpaces service implements the “net.jini.space.JavaSpaces” interface, which supports only four basic operations:

- read – object can be read from the space.
- write – object can be written to the space.
- take – object can be taken (i.e. read and removed) from the space.
- notify – register a listener to be informed about objects inserted in the space.

The Jini technology benefits from the security of the standard Java API. Furthermore, an on-going project of the Jini.org community, called the Davis project [DAVIS], is improving the security architecture. One of those improvements is a configuration programming model and API for deployment-time customisation of applications without modifying code. In fact, the environment is really open and offers many development shortcuts if we look at all open source projects. The community working for the evolution of Java components is huge.

Nonetheless, Jini requires working with Sun’s RMI implementation for reaching its full potential as has been shown in [HW00]: “there are alternative implementations of Sun’s RMI classes” but “none of these implementations does seem to fulfil the purpose of replacing Sun’s RMI to be able to run Jini” (p.31). Furthermore, this implementation remains big for devices with small capacities such as a mobile phone. It is more a method-driven approach since remote method invocation is used. It is worth noting that the JDK1.3 RMI [JRMI] supports IIOP (Internet Inter-ORB Protocol) [IIOP] and contains improvements for penetrating firewalls.

Sun's proposal on how to get Jini into non-Jini devices is called the Surrogate Architecture [SURA]. The idea of the Surrogate solution is to let a collection of Jini-enabled Java classes (the surrogate) run on a surrogate host running a JVM on behalf of a device, that cannot run one directly. Hence this could mean that Jini is going to obtain a very high-level of interoperability. Another improvement will come from the ServiceUI [SERUI] API, which will provide a “standard” approach for attaching user interfaces to Jini services.

3.2. ***The missing link***

Something is missing! Sun Microsystems [SUN00] has been one of the first to notice that the diversity of home networking technologies will require another level of abstraction. As we have seen in chapter 2 and the beginning of chapter 3, the difference between those technologies is significant. Moreover differences reside at different levels. Today it is clear there will be more than one standard in existence in the medium term at least. The broad range of requirements at an end-user level have demonstrated a need for a high level of abstraction. There is a mismatch with the low-level technologies and the high-level requirements. Even the advanced home network middleware such as Jini or UPnP are not adequate to fulfil all requirements. There are powerful tools at a programmer level, but they cannot be used by end-users. A piece of work has to wrap those technologies to give a global view of the home network, including all that is provided at the level below.

This is the reason Sun Microsystems has created OSGi, explained in section 3.2.1.1. As presented in section 2.1.3, it will be more likely to have a gateway in which high-level actions can be performed. As presented in OSGi, this gateway should not be dependent on a specific home network technology to achieve a global view. It has been thought that home network middleware, such as Jini, could rely on bridges or proxies to other technology to give a global view of the home network. However, due to the fact that bridges are difficult to build, an unpopular technology could not be bridged. Consequently some parts of the home network could be excluded from the view. Another flaw of some home network middleware such as Jini is that they are designed for local networks. In fact, it is not straightforward to manage the home network remotely over the Web using Jini. For example, the protocols used locally cannot pass easily through firewalls.

By reading the requirements in section 2.3, what is needed seems to be something like the old-fashioned house-keeper. A house-keeper had to take care of the house globally, without excluding a part of it due to a certain incompetence. This is related to the fact that the residential gateway should understand all the complexity, all underlying technologies without exception. The house-keeper was the person to contact to make a change in the home, a central point of communication. Once the family member instructed him, the house-keeper was in charge of contacting all other entities involved. The idea is that a kind of abstraction has to be provided to the end-user, hiding the complexity of the subtasks that have to be done for an oral task. The multi-user requirement points out that the house-keeper had to change his behaviour depending on whom he was dealing with. He was in charge of security in the home. He had to follow the orders of different persons who had different levels of power described in a hierarchy. In the past, a house-keeper took invisible actions when needed, without disturbing the home members. The end-users want to avoid being spammed by tons of emails: they expect the behaviour of a house-keeper.

The following sections review two candidates that could be a base for this next generation of electronic house-keeper in the tradition of their human ancestors.

3.2.1. Candidates

3.2.1.1. OSGi

An overview of the OSGi specification is presented first. Then, the Sun implementation of OSGi, called JES, is detailed.

3.2.1.1.1. Overview

Sun Microsystems [Sun00] found there was a “missing link” (p.16) between the Internet and the in-home network and created OSGi (Open Services Gateway initiative) [OSGI] to fill the gap.

[Meadows00] thinks that the service gateway, as described in the OSGi's specification [OSGi00], should:

- Be open and independent of platform
- Have a standard
- Have advanced privacy and security features
- Support all types of home networking
- Be reliable [Meadows00]

In comparison to several other consortia, which are primarily industry-specific (e.g., construction, consumer electronics, telecommunication or mobile phone), OSGi is more sector-crossing and platform-independent. In domain analysts opinion [LF+01], OSGi is well accepted even though there is also widespread support in the industry for HomePNA and Phoneline Alliance. It is worth pointing out that these technologies are not at the same level in home networking architecture.

OSGi intends to integrate services of all common standards:

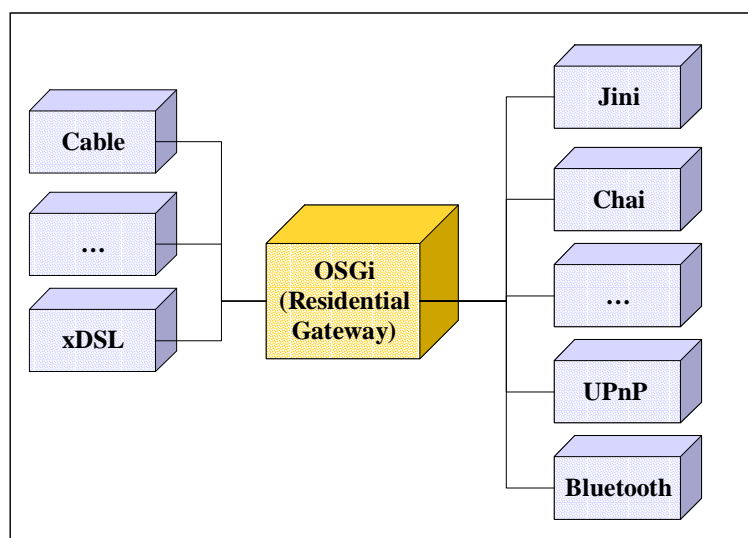


Figure IV: Simplified architecture of an OSGi solution

OSGi defines a service gateway as an administrator-free embedded server in a network providing an interface between the internal and the external network. OSGi specifies API for the life-cycle management of services, data management, device management, client access, resource management and security. As OSGi's specifications are independent of the

manufacturer and the operating system, it is possible to integrate various technologies over API although OSGi's core technology is Java based, running a JVM. For example, wireless protocols such as Bluetooth, HomeRF and ShareWave can also be integrated as can networking standards competing with Jini such as UPnP; as well as HomePNA, X10 or HAVi. OSGi concentrates on specifications for the application and context layer in the (advanced) OSI (Open System Interface) reference model and is open to standards affecting other layers.

A service gateway solution based on the OSGi standard can in essence be built on the components shown in the illustration in rectangles with a plain background in the figure below. Other services alongside the OSGi-API could be added. They are represented in rectangles with a mosaic background.

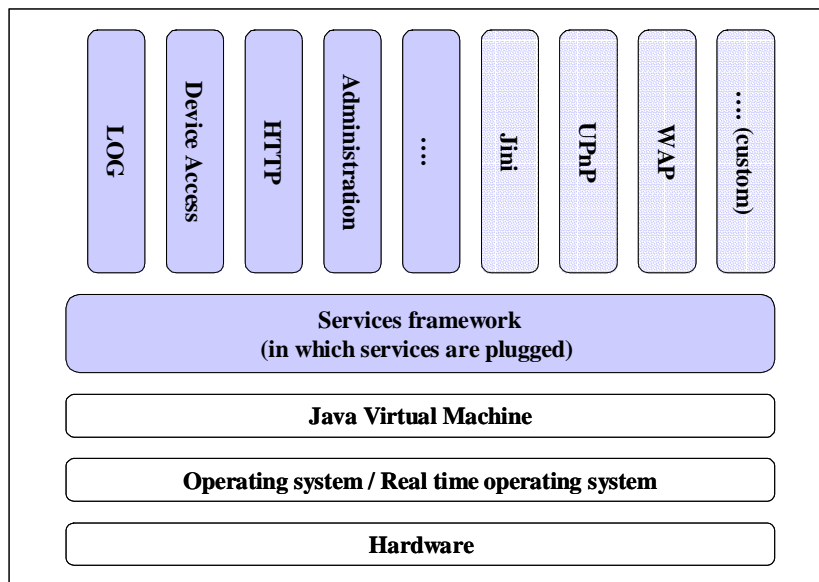


Figure V: Components of an OSGi-based residential gateway

Fundamentally, OSGi only describes the framework for the integration of services and devices in the network. Only Device Access Control, Log-Service and HTTP Service are specified as its own services. Other services are currently being developed. The framework details how the download of services is to be executed and how the interdependencies between the services are regulated. From OSGi's point of view, Jini, UPnP and WAP only symbolise the integration of another service. Although the architecture is predominantly Java-based, it is nonetheless open to other standards. OSGi describes itself as a collection of APIs, which defines standards that are required to define Services gateways. There are two

sets of APIs. The core API includes service delivery, dependency and life-cycle management, resource management, remote service administration and device management. In addition, the optional API describes the mechanisms for the integration of the client and the gateway and data management. Moreover, some Java API is also delivered, e.g., Jini. OSGi members are not obliged to offer the optional API, however, if they offer solutions that relate to the optional API, they do have to offer OSGi's optional API.

Remote process control is not possible using OSGi yet. However, remote diagnostics is possible (how many hours until the next service, consumption ratios, etc.), as well as remote updates and service and remote control (on / off). OSGi's fundamental advantage is that proprietary standards will not be used. This, however, is not the case with the numerous solutions for home networking technology.

3.2.1.1.2. JES

JES (Java Embedded Server) [SUN00] is the Sun implementation of the OSGi specification 1.0. Freely downloadable, this small footprint application server can be embedded into any networked device. It has two primary components:

- Framework: composed of APIs for life-cycle management of services and applications, this modular model manages installation, versioning, content management, monitoring and service discovery.
- Services: comprised of servlet-like objects that represent the modular services and can be deployed over the Internet, including HTTP, logging, thread management, remote administration and servlet support.

The JES product comes bundled with a set of core services:

- Log service – log error and events
- HTTP service – access resources and servlets using HTTP
- Device access – dynamically detect and refine devices
- Servlet and JSP framework support
- SSL support – secure communication over HTTP
- Management panel – remote administration over the Web
- Command console – perform administrative tasks

3.2.1.2. *The Chai Place Manager*

We have already seen that the HP Chai technology is related to the HP Labs Cooltown project. The Cooltown vision is that places, people and things have a Web representation. [DC00] describes a horizontal and uniform software architecture for building a Web presence for places, people and things. [CD00] goes further in the description of a general place manager infrastructure useful for creating Web representation for physical places. Several of the requirements listed in section 2.3 come from this latter report. They highlight the fact that the environment of a place involves multiple users and that secure relationships with different entities are needed. One of their assumptions is that there will be a place master, a kind of Webmaster who will manage the different Web pages describing the smart place. In fact, they stressed that there should be an easy-to-use authoring environment for such a place Webmaster. The service is dedicated to end-users: it has to be user-friendly. They use the well-known metaphor of a Web site with Web pages and hyperlinks. Another point is that the architecture should allow for the extension of the content of an entity description. The architecture should also be open to different discovery protocols.

Nevertheless, the first implementation of their place manager is closely related to the HP Chai technology. As we have seen, the architecture should be independent of the underlying home networking middleware as in OSGi. In the beginning of July 2001, HP has released the WPM (Web Presence Manager), which is downloadable from the Web. There are two versions of this WPM, one that runs on top of a ChaiServer and another that runs on top of a Java Servlet Container.

Concerning the software infrastructure, each Web presence hosted by the WPM has a data repository called the relationship directory that stores links to other related Web presences. This directory is queried or updated programmatically or manually using its HTTP interface. Another data repository, called the description, stores all other information concerning what is represented by the Web presence, such as its name or its location in XML format. Following is the diagram representing the architecture.

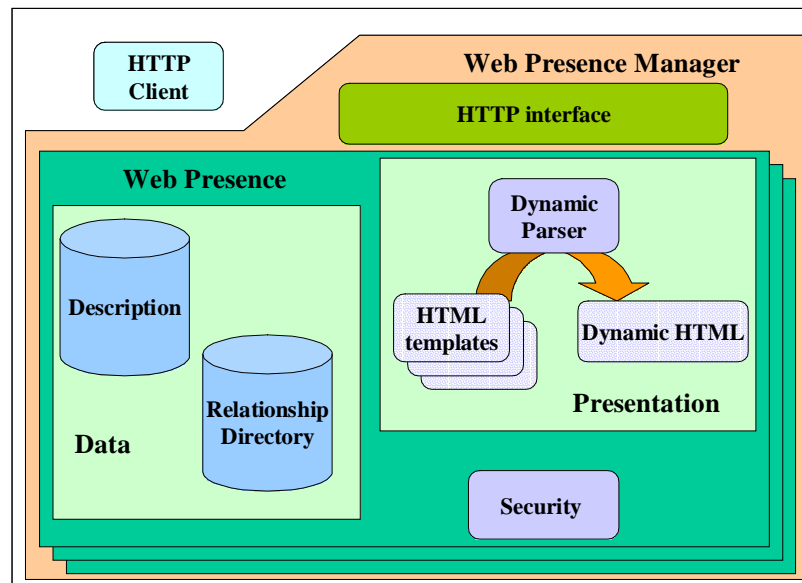


Figure VI: HP Cooltown Web Presence Manager architecture diagram

A presentation module can retrieve both the relationship directory and the description to generate a Web page dynamically. Finally, a security module allows for permissions to be set to see restricted information. An administrator of the Web place can change the rights of users. How could a user give rights to his assets? For example, the administrator can specify that a user is a “familyMember”. Say that this user has an MP3 player, which is going to be added in the list of Web presences for things. It is almost certain that this user would like to give access to his MP3 player to specific users. It would be easier if the user did not have to become an administrator or to contact the administrator every time he needed to change access control of his assets. The HOUSE-KEEPER service provides a way to improve that. In the HP WPM, the XML description must be available over the Web and thus have a corresponding URL. Could it be possible to avoid having a constraint on how to retrieve the XML description? What is needed is to retrieve the XML descriptions not to know how it has been retrieved. Thus, the WPM is another solution that cannot fulfil all the requirements listed in section 2.3. The next section summarizes what has been written so far concerning this point.

3.2.2. Is it possible to fulfil the requirements with the actual technologies?

As it has been written, in OSGi, a complete residential gateway operates independently of a personal computer and contains a modem and networking software. This small box solution could limit the power of the services provided. A solution inside a PC could take advantage of its greater computational power. Artificial intelligence components modules could be added and complex mathematical algorithms could be run faster to provide a convenient user experience. Located inside the PC, it could simplify exchange between the standard software applications, such as an organizer, and the services of the residential gateway. In the OSGi approach, much more has been made for remote application service providers than for residential end-users, due to the fact that it should be a zero-administration box. To the contrary, the Chai Place Manager is dedicated to end-users. The service is built to be user-friendly. Nevertheless, the idea of an architecture independent of the home networking middleware is less true than for OSGi.

Moreover, other requirements are not effectively addressed by the actual technologies. Concerning the security requirement, UPnP is working to provide secure access to device description. Security of version 1.0 of UPnP is focused on physical security, but security services including authentication and authorization are subject to discussion for support in future versions [UCSQ01]. HP Chai at least gives a framework. The core ChaiService BasicGateKeeper implements the basic authentication scheme described in the rfc1945 [BFF96]. It uses also the rfc1421 [Linn93] for encoded passwords. SSL [SSL], Secure Sockets Layer, has not been implemented yet, but it is on the HP plan. Jini offers the standard security of the Java API. Nevertheless, no environment offers the abstraction for managing security easily. The Jini.org Davis project [DAVIS] is developing a framework to add declarative security features to Jini services. In the next release of Jini, for example, it could be possible to grant access to a Jini service dynamically without recompiling some code, but this is not possible in the current release. We have seen that the domain is multi-user and requires security as well as mechanisms to ease its management. Again UPnP does not give an answer that meets the requirements. JES, the OSGi implementation of Sun, has SSL support in its core services, but role-based functionalities have not been implemented yet. Thus, no candidate seems to fulfil the security requirement mixed with the requirement of simplicity and catering for a multi-user domain. It is worth noting that for OSGi the home

service gateway is a zero-administration box, and the security should be provided by a security application provider. How will the residential user give access to his MP3 player? Will he have to contact his security application provider first?

The notion of persistence required is not really addressed by UPnP. Basically, the Chai technology provides hooks for adding a database. In fact, a real component has been added, which is the Oracle [ORAC] Lite database product. Persistence is offered at different level in Jini. The Javaspace service can store objects in a defined space. Thanks to its services, OSGi provides a way for persistent data. Nonetheless, when transactions have to be used, the programmer still has to be careful about what is done when the database or the Javaspace is accessed concurrently. The Jini event model offers an EventMailbox service that stores and holds remote events until the client is ready to receive them. On the contrary, without creating a personal solution from scratch both HP Chai and UPnP lose events if the client is not ready. OSGi has no real event service and behaves differently from other solutions. OSGi has been created with the notion of a gateway of services rather than with the objective of controlling one specific device remotely like in HP Chai. Accessing Jini services over the Internet is a big issue. A solution using RMI such as Jini cannot easily pass firewalls. The Jini.org SOAPUDDI project is developing a way to access a Jini service thanks to a SOAP counterpart. Since UPnP and Chai use HTTP, they should pass firewalls as long as HTTP is not blocked by firewalls. OSGi is designed to be accessed from the Internet.

Interoperability is still in its infancy for all candidates. Jini with its Surrogate Architecture has opened an interesting door for creating bridges between the different technologies. The release version of HP Cooltown provides a way to integrate UPnP devices. UPnP is more likely to be well spread due to the number of vendors who have joined the consortium. OSGi is also going to be widely spread and could integrate any other technology. OSGi has been built to be open but services for each home networking technology have to be built. It is worth noting that Gatespace [GSPAC] has just released a non-free version of its service gateway solution implementing a UPnP bridge service.

What about the physical location of the device in the home? All solutions provide a service discovery. Nevertheless, those services give the user a way of finding things by what they are rather than where they are. [CD00] explains that “inter-place relationships is a topic for further research” (p.2) in the Cooltown representation for places. The name of the

physical location where the user thinks the device is can obviously be a simple metaphor for finding the device that he is looking for. The user could have the different rooms of the house listed as hyperlink texts. This list could be split in smaller lists to be more user-friendly. The “first floor” link could point to a list of rooms on the first floor. The user could choose to click on the “first floor” link and then on the “dining room” link to find the smart TV in the dining room located on the first floor. It means the system should know where the devices are and if there is a relation between different places. A way to do that could be to add the location of the smart device to its attributes and to store this information to display it when needed. Additional information could be added to build the relation between different places. This solution has been chosen in the HOUSE-KEEPER service. In this case, a presentation layer is added above what is provided by the underlying technologies. Thus HOUSE-KEEPER is based on a specific architecture that helps to meet the requirements previously listed. Without this architecture the actual solutions cannot fulfil all the requirements. The next chapter describes the proposed architecture and how this system helps to meet the requirements.

4. **HOUSE-KEEPER : the new step**

The chapter describes HOUSE-KEEPER, which aims to fulfil most of the requirements described in section 2.3. The choice has been made to describe the architecture and to explain the main design ideas and the benefits of the chosen technologies along with the description. Firstly, the assumptions on which the design relies are explained. Afterwards, the high-level view of the architecture is given. Then, some special issues addressed by the architecture are detailed. A discussion about other issues that could have been dealt with differently follows. Finally, there is a description of the prototype and what the different implementations have brought to light.

4.1. **The big picture**

4.1.1. **Assumptions**

We assume there will be a central home gateway between the outside and the in-home network. The trend is more likely to have stand-alone devices in the home as stated in [BMKKS00]. Nevertheless the connection to the Internet should be unique to protect the home network by passing through a firewall to limit the paths of accessing it maliciously as in the current corporate LAN model. This does not mean that the home network system has to be centralised. It means that we have to view the home network from the outside and the view has to be built somewhere. In HOUSE-KEEPER, it has been decided to do that in the residential gateway, which will be the PC itself. In the home, the PC still has a high level of computational power. OSGi has taken the fact that there should be a central point for bridging the Internet and the home network. Nevertheless, [SUN00] claims that just putting a wired PC in the home is not the answer since reliability is poor and maintenance is complex. They argue that all the services of the home gateway can be managed by external service providers. In fact, it makes sense, but it does not take into account that the user would also like to customize and control some of the features. The user research [FDS00] concludes there is “more comments about access and control than about usability and reliability” (p.20). The PC provides a convenient metaphor to control the smart home system.

The HOUSE-KEEPER is not only a home gateway in the sense of the OSGi. It needs more resources than just a TCP/IP stack and a persistent storage of at least 1 MB, as for OSGi. In addition to being a hub to connect and manage the smart devices in the home and the communication gateway, HOUSE-KEEPER filters the information sent to users and presents a comprehensive metaphor of what the aggregation of those smart devices has created. A tree of places in the home is dynamically generated. For each place, the list of subplaces and devices in this place are displayed. The latter point should help to reach the design goal, explained in [Hedberg00], of allowing the user to focus on the task itself rather than on the devices needed to operate the system. The residential user can go to the bedroom and check the temperature sensor to know the temperature of the bedroom without specifically knowing the identification number of the specific temperature sensor.

As written at the beginning of this document, the domain of the HOUSE-KEEPER architecture and services is the home, which is different to the corporate domain or the public domain. We will not speak about employee or customer end-users. The users will be residential users. There will be a hierarchy amongst the residential users: users with the most privileges will get the role of landlord; members of the household will get the role of family member; and occasional users will be given the role of guest.

Security is detailed in section 4.2.4. The assumption concerning this issue is that the in-home network will be considered as a different zone from the external network. High-level attacks, requiring expert knowledge, could be attempted by people outside of the community of residents. The in-home network, more likely to be locally accessed by the residential users, will be protected from common threats.

Home networking middleware used directly in the HOUSE-KEEPER service should be high-level home networking technologies such as Jini, Chai and UPnP. The reason is that the architecture will use some of the services provided by these technologies. Such middleware will need to provide a means for discovery and lookup. An event service and the functionality to download a new piece of code dynamically could really improve the potential of what is feasible.

It is assumed that the goal of the project was neither scalability nor mobility. Millions of Web devices were not considered to be present in a home. A simple update of what is in the

home has been done. We do not speak about frequently moved devices such as mini MP3 players, but rather appliances such large TV screens or fridges.

4.1.2. Multi-tier architecture

The architecture is a multi-tier one with a Web-tier for the presentation level. Basically, the protocol chosen to control and monitor the smart home remotely has been HTTP. As in OSGi, the architecture is designed to be independent of the underlying home networking technology. The service is dedicated to residential users. It must be user-friendly as stated in the HP WPM. The back end of the architecture is described first. The front-end is presented afterwards.

The back-end of the architecture consists of the in-home network up to the residential gateway, which is the home PC in the case of the HOUSE-KEEPER service. Below is the diagram representing this first part:

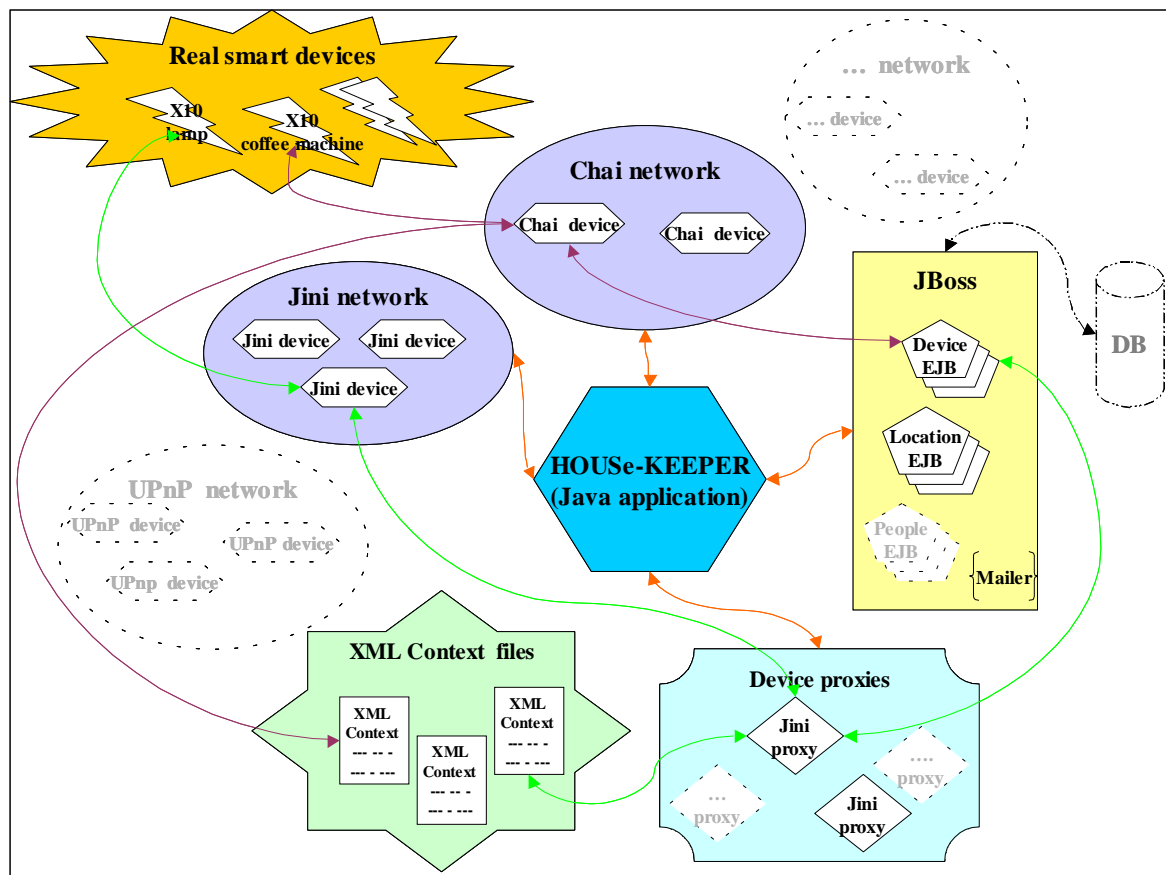


Figure VII: Back-end of the HOUSE-KEEPER architecture

Physically, we have different smart devices present in the smart home. For HOUSE-KEEPER, these devices use X10 technology to be connected to the home network. This technology uses the powerline to communicate. Other types of devices could be used, such as devices communicating using the phoneline or Sharewave [SWAV] wireless devices. Since low-level home networking technologies do not provide discovery and lookup of devices, high-level home networking middleware has to be used. In HOUSE-KEEPER, X10 devices are wrapped in Chai or Jini pieces of software. Thus, they can be dynamically discovered by the HOUSE-KEEPER service. A simple Chai to X10 bridge has been built by using the X10 Java API [JX10]. Concerning Jini, a Jini.org project called JADE [JADE], which provides a bridge to X10, has been reused.

At this level, HOUSE-KEEPER acts as a client of the high-level home networking technologies. HOUSE-KEEPER is a set of Java classes. Their instances run in different threads and some of them are notified by Jini or Chai when a device is found in the home network. Other high-level middleware such as UPnP could be used. HOUSE-KEEPER has already implemented Jini and Chai. This part of the design, as will be explained further, contributes to the achievement of a service independent of the underlying technology.

When the HOUSE-KEEPER Java application is notified that a device has entered the home network, it starts to interact with an EJB (Enterprise JavaBeans) server [EJB], which is JBoss [JBoss]. The principle is that one device has one entity EJB counterpart. Each device has a specific and unique EJB counterpart. Such an EJB is the counterpart of a specific and unique smart device. Hence, any information related to this specific device can easily be stored persistently. An EJB server is in charge of managing the connection with a database in which entity EJBs are stored on behalf of the programmer. This “device EJB” has information concerning the description and the context of the device. It can also send a specific command to this device or retrieve a specific state from this device. It knows how to communicate with the real device. Another entity EJB type populates JBoss. This one is called “location EJB”. This entity represents a place in the home. It knows what other places are a part of this place, called subplaces. For example, the ground floor place could have two subplaces the kitchen and the dining room. A location EJB also stores the list of devices present in the place. It could store more. The location of a device is a part of its context. This introduces another principle of the HOUSE-KEEPER service.

Each device has one XML file, which describes its context, called “XML context”. The content of the context will be described later, but roughly it contains the description of the device, its location and its access control list. The HOUSE-KEEPER service uses JDOM [JDOM] to parse this XML file to know the description and the context of the device in order to populate the JBoss server correctly. The XML file facilitates the management of the device, as we will see later in section 4.2.3.

Where should the XML context be located? It is true that small smart appliances with their embedded systems will have less computational power and restricted functionalities. One restriction could be that the device cannot store any files. The solution is explained later, but the main idea is that a proxy will store the file on behalf of the device.

For devices that cannot be used alone in HOUSE-KEEPER, the chosen solution is to provide a proxy. In fact, as we will see, each Jini device has a specific piece of software, which helps the Jini device to be used. As in the previous paragraph, another type of proxy could be a proxy which stores files on behalf of the device. The proxy should allow the use of a device even though the device knows nothing about the HOUSE-KEEPER service as detailed in section 4.3.2.

The front-end of the architecture is diagrammed below:

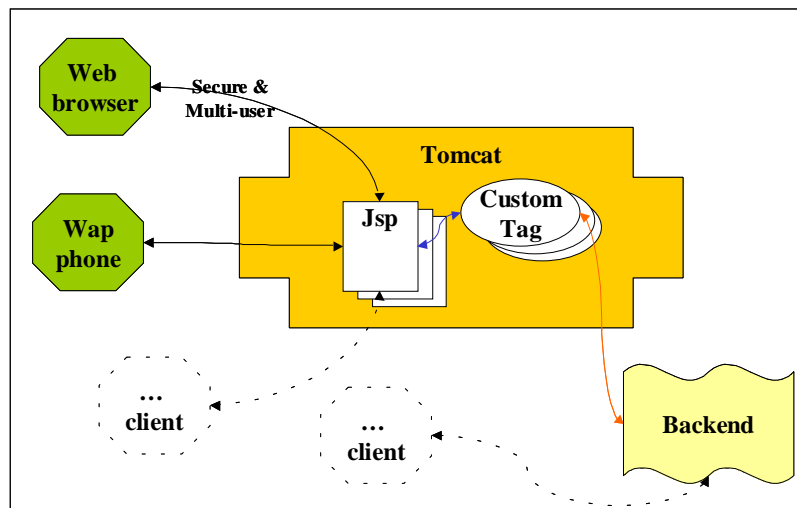


Figure VIII: Front-end of the HOUSE-KEEPER architecture

It is worth noting that the front-end requires that the back-end of the HOUSE-KEEPER is available. It can be considered as a Web tier, which provides the presentation of the service. The main component of the front-end is Tomcat [TOMC], an open source servlet [SERV] and JSP (JavaServer Pages) [JSP] engine. In fact, some custom tags — technology explained in [P+01] (pages 263-405) — have been written to query the JBoss server. These tags reuse some of the classes of the HOUSE-KEEPER Java application. For example, one of these reused classes provides different methods to extract information from the set of “location EJB” entities. One method is used to get a reference to a specific “location EJB” entity.

The tags are used in JSPs, which give a view of the smart home to the residential user. The representation is discussed later in section 4.3.1. By using tags, a small amount of Java code is inserted in the JSP pages. It also allows the retrieval and presentation of interesting information contained in the JBoss server. One of the custom tags extracts the list of subplaces of a place given as an attribute. Another extracts the list of devices. Given a specific device, another tag lists the different methods of the device. No Java code has to be written to build a Web page presenting this kind of information. This is at the level of a standard Webmaster. The feature is quite user friendly. However, a JSP container such as Tomcat has been used to provide a dynamic view of the home network without requiring the Webmaster to build static pages.

There is a set of reused JSP pages. When the residential user connects to the Web site of the home, a page that points to the root of the home is displayed. This page displays the subplaces and the devices available from this current place. If the residential user selects a subplace, the same JSP page is called but it displays the information related to the new place. If the smart home user clicks on the link to a device page, another JSP page is called. The idea is that there are few JSPs and that what is displayed depends on arguments given in the page request.

The following two points will be debated afterwards. Firstly, the presentation of the home can be adapted to the device capability. Using the same idea as for a browser client, a set of JSPs and custom tags has been written to provide a view of the service for a WAP phone. Secondly, the security for a Web client has been improved. The introduction of roles and access control lists has enabled a better multi-user environment.

To summarize before going into more detail, following is a simplified technology stack used in the architecture:

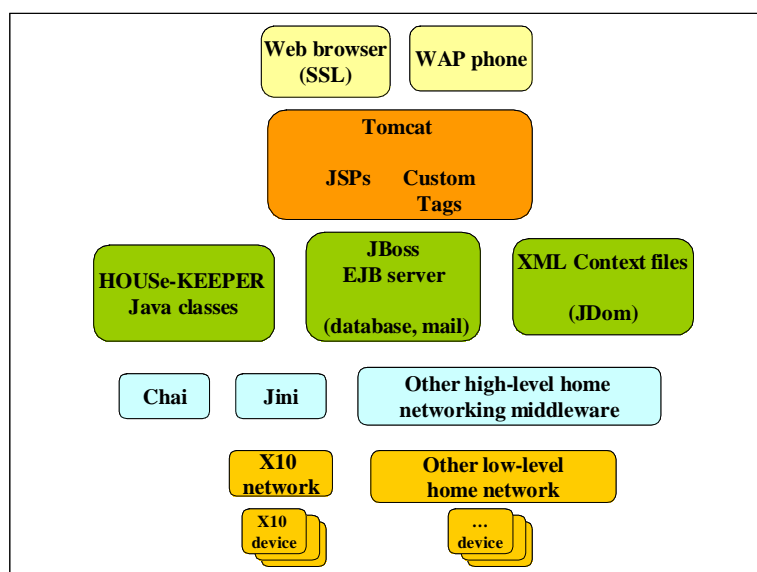


Figure IX: Simplified technology stack of HOUSE-KEEPER

4.2. Further detail

4.2.1. The choice of the server

Why do not we use JES, the Sun Microsystems implementation of OSGi, to build the HOUSE-KEEPER service, even though it is located in the PC? JES is like a small footprint Java application server. By being located in the home PC, HOUSE-KEEPER can share the important quantity of information stored in the PC. For example, if artificial intelligence components were added in HOUSE-KEEPER, these components could get interesting information about the behaviour of the user during his computer session. Such more advanced modules in HOUSE-KEEPER would need higher computational power. A real application server would be needed to run these components effectively. JES brought the idea that an architecture based on components should increase the level of openness. To integrate any kind of components that will require high computational power, a solution in Java is to use an EJB server. The main requirements include persistence of context and easy use of security. EJB servers reduce the complexity of developing middleware by providing automatic support for middleware services such as transactions, security and database

connectivity. Since the HOUSE-KEEPER service runs in a PC, using an EJB server instead of JES allows the service to take advantage of all the computational power of the PC. JES is optimised for a box with less computational power than a PC. The optimised component model for a PC is an EJB server. The fact that an EJB server is in charge of storing any entity EJB persistently in a database on behalf of the programmer is a key feature. The requirements show that to keep a history of the state of the system is important. By using an entity “device EJB” counterpart for each device, we gain the possibility of easily storing anything related to a device in a database.

Different EJB servers are available on the market. JBoss has been chosen for two main reasons. Firstly it is an open source J2EE-based implementation [J2EE]. Secondly, thanks to its implementation with JMX (Java Management eXtensions) [JMX], it provides a common spine into which additional modules can be plugged. The implementation is modular. The goal of JMX is to provide tools for building distributed, Web-based, modular and dynamic solutions for managing devices, applications and service-driven networks. Even though this part of JBoss has not been used in the HOUSE-KEEPER service, it is an interesting functionality, knowing that HOUSE-KEEPER strives for easy management of a home network with smart devices. To have tools to reach this goal is an advantage.

4.2.2. A service independent of the home networking technology

It was helpful to look at alternatives for designing an HOUSE-KEEPER independent of the home networking technology. The look at OSGi helped in choosing how to design the overall architecture of HOUSE-KEEPER. From OSGi, it has become clear than there are two ways for interoperability. The first way is when an application relies on a single home network technology and uses bridges provided by this technology to speak to other home network technologies. For example, it would mean that HOUSE-KEEPER would be fully dependent on the HP Chai technology, as it shown in the following diagram.

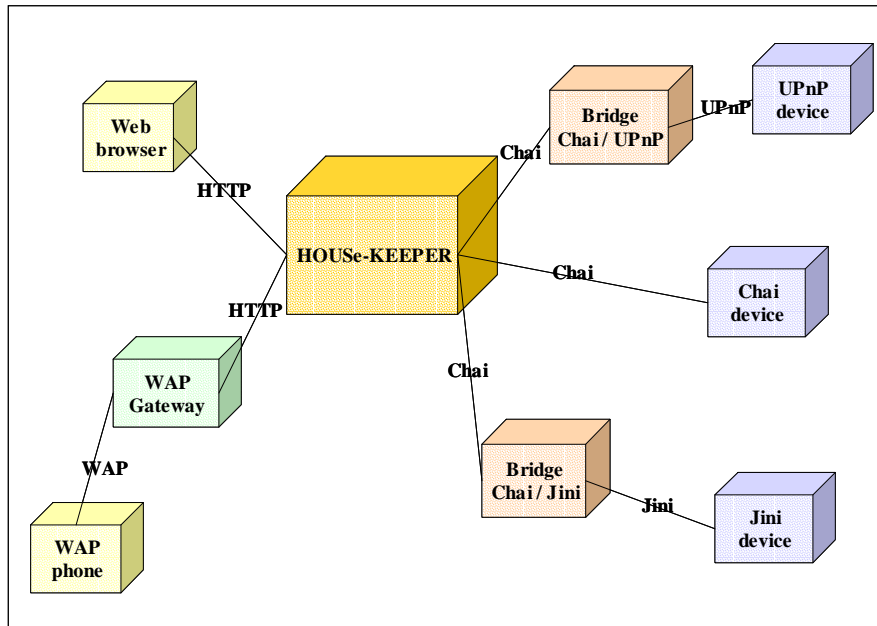


Figure X - HOUSE-KEEPER tightly coupled with Chai

Since Chai has no bridge available yet, it would imply that HOUSE-KEEPER could not use another home network technology until there is a bridge available on Chai for this technology. The second way is that of OSGi. Rather than having an application dedicated to one technology, the application offers a means for plugging in modules, one module speaking directly to one home network technology. The application becomes independent of the home networking technology but needs to have a high level of reusability of its components.

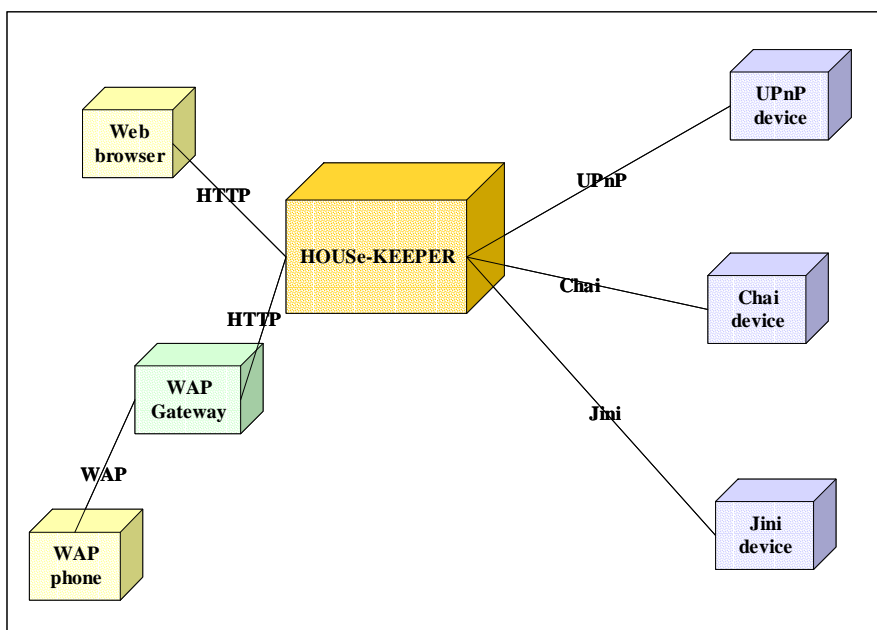


Figure XI - HOUSE-KEEPER independent of the home networking technology

As has been said in [CD00], to obtain a service independent of the home networking technology, there should be a decoupling of the discovery of a new device and its registration. We have seen that there are different types of discovery protocols. HOUSE-KEEPER has no discovery service able to find all the devices in the home network. HOUSE-KEEPER is a client of the different discovery services, listening or subscribing to them in order to find out what devices compose the smart home. From an implementation point of view, HOUSE-KEEPER will have to be considered by each middleware as one of its members to be able to understand the protocol. HOUSE-KEEPER may run a ChaiServer to understand Chai, a Jini client to speak with the Jini network and so on.

In HOUSE-KEEPER, a specific class has the responsibility to become a client of the service discovery of the home network. Furthermore, this is an abstract class. A subclass of this class has to be written for each home networking technology due to the fact that the listening process differs between the different technologies. When HOUSE-KEEPER is notified that a device has entered the smart home, another class has the responsibility to check whether the device has to be registered. Since the communication process also differs between the different home networking technologies, this class is an abstract class, and a subclass of this class exists for each technology.

Once a new device has been registered, its “device EJB” counterpart is created. Every “device EJB” has a “process” method, which is used to send a command to the device. Since the communication process is different from one home networking technology to another, the body of this method varies depending on the type of smart device. Hence, there are different types of “device EJB”, “Chai device EJBs” and “Jini device EJBs”. The following class diagram describes the idea.

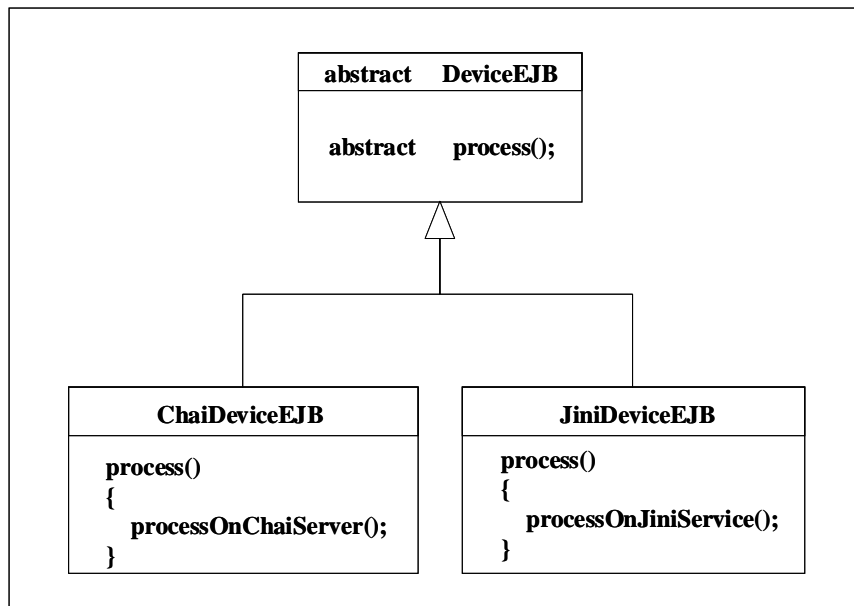


Figure XII: Subclasses of an abstract “device EJB” class

To summarize, there is a set of core abstract classes. Each of these classes has few responsibilities, which must be provided to plug a new home networking technology into the HOUSE-KEEPER service. In fact, when a new home networking technology has to be plugged into the service, subclasses of these abstract classes must be created. After that, the main flow stays the same: a client of the discovery service of the new technology runs and passes devices entering the smart home for registration or update if needed. When a command has to be carried out on a device, the command is generated by a “device EJB” that knows how to communicate this command by using the protocol of the home network type of the device.

4.2.3. An easy and vendor-independent declarative process for device context

As explained in the big picture of the architecture, one device has one XML file that describes the context of the device. It is a principle to apply to each smart device independently of its home networking technology. This principle facilitates the management of the devices since it is the same process independent of the type of home network.

What is required is a method to get the XML content of the device context. It is not limited to getting the content using a specific protocol such as in the HP WPM, where the XML description has to be accessed at a specific URL. The implementation of how to get the content is left to the programmer.

The look at UPnP helped to choose XML for device representation and also to define what kind of information is required. UPnP relies on standardized XML descriptions of devices. This good idea has been applied to HOUSE-KEEPER. Alternatively, we could have used the description of the device provided with the discovery service of the high-level home networking middleware such as Jini. To do that, all home network technologies should provide this description with its discovery service. The issue is that it is not the case. The Chai discovery service does not provide information about the device easily. The UPnP discovery service stores simple attributes describing the device, but they are not sufficient to know everything about the device. UPnP can fetch the UPnP XML description to know more. In Jini, a device may register its service along with a set of objects, which would best describe the device characteristics. The information provided by the home networking middleware could be used to fill the different fields of the XML context. However, an XML context file is still needed to get a unique model to describe the context of a device, even when the technology does not provide specific information.

Below is the final state of the XML context model:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<device>
  <nickname>jmDesktopLamp</nickname>
  <owner>jm</owner>
  <emailOwner>seigneuj@tcd.ie</emailOwner>
  <description>HKX10ChaiLamp</description>
  <networkType>chai</networkType>
  <entityType>x10chailamp</entityType>
  <htmlRep>http://www.cs.tcd.ie/JeanMarc.Seigneur/WebHeke/coffeeMachine.htm</htmlRep>
  <location>
    <locationName>G30</locationName>
    <locationDescription>The NDS room</locationDescription>
    <locationPartOf>home</locationPartOf>
    <locationPartOfDescription>The Computer Science buildings of Trinity College
  </locationPartOfDescription>
  </location>
  <deviceACLList>
    <deviceUserRole>jm</deviceUserRole>
  </deviceACLList>
  <methodList>
    <method>
      <methodName>ProcessThis</methodName>
      <methodDescription>Process the command strings received on the X10 lamp
    </methodDescription>
      <methodACLList>
        <methodUserRole>jm</methodUserRole>
      </methodACLList>
      <methodParamList>
        <methodParam>
          <paramName>Command</paramName>
          <paramDescription>Value: "on/off" to switch on/off; "dim" to dim; "bright" to
            brighten</paramDescription>
          <paramType />
          <paramValue />
        </methodParam>
        <methodParam>
          <paramName>CommandValue</paramName>
          <paramDescription>Value: None for on/off; "1"... "10" for the intensity of
            dimming/brightening</paramDescription>
          <paramType />
          <paramValue />
        </methodParam>
        <methodParam>
          <paramName>HouseCode</paramName>
          <paramDescription>Should be "A" for the lamp</paramDescription>
          <paramType>hidden</paramType>
          <paramValue>A</paramValue>
        </methodParam>
        <methodParam>
          <paramName>DeviceCode</paramName>
          <paramDescription>Should be "3" for the lamp</paramDescription>
          <paramType>hidden</paramType>
          <paramValue>3</paramValue>
        </methodParam>
      </methodParamList>
    </method></methodList></device>
```

There are three main parts in the model:

- the description part of the device, which can be seen as its internal context: its user-friendly name; its textual description; what it can do and what kind of parameters are required; a URL pointing to a HTML representation of the device (such as some HTML inserting an image of the device); its type of home networking technology; and its general type (such as “lamp”)
- its external context: the owner’s name and email; its location with a textual description; and the container of this location (it could be “first floor” for a location “kitchen”) with a textual description
- a specific part of its external context, its access control lists: the access control list at the device level; and the access control list for each command available (this point is more detailed in section 4.2.4)

Nonetheless, this XML content has a high probability of change. This is the reason that, during the registration, the parsing of the XML context file is done in one specific class for any type of smart device. This requires only one point to change to make a global change. Such a change could imply backward compatibility issues amongst others. It is easy to change the model of the XML context, but such a change can be difficult to handle.

4.2.4. The answer to the multi-user security requirement

Both Jini and UPnP have weaknesses in solving security requirements. The Davis project [DAVIS] tries to improve the security in Jini. UPnP has recently approved in [UCSQ01] that it is not designed for a scenario in which the home is continuously connected to the global Internet. In both technologies, one of the main security requirements that they have defined, is that the residential user should be able easily to grant access to sensitive information and device functionalities.

In HOUSE-KEEPER, the security requirements have been taken into account from the start of the project. This is known as a good principle to make a system secure compared to adding “patches” to fix security holes in a system not designed with security requirements in mind. [MV00] states that “retrofitting security is always the wrong solution. Instead, security must be designed into software from the beginning”.

The whole system is separated into two parts: the in-home network and the Internet. It is the same kind of separation between the LAN of a corporate building and the Internet when the LAN is connected to the Internet. The residential users are not considered to have high

expertise and the means to break common security protections. On the contrary, people with high knowledge in security are present externally and could break these common security protections. This is the reason that the same paradigm as for e-commerce shopping has been chosen concerning Web access to the smart home. A secure HTTP connection, using SSL [SSL], is used when the dynamic Web pages of the smart home are accessed. This encryption of the communication increases the difficulty of “eavesdropping” attacks. On the other hand, in the in-home part at the level of home networking technologies such as Jini or Chai, the communication is not encrypted, at least at the HOUSE-KEEPER level. Residential users are not expected to take the time to build a system to “eavesdrop”.

HOUSE-KEEPER gets authentication and authorization functionalities from the e-commerce shopping paradigm and the use of J2EE [J2EE] security mechanisms. The use of logins and passwords decrease the risk of “masquerading” attacks. Thanks to Tomcat and the full J2EE stack, it is easy to add new residential users. It is a declarative process. The administrator of Tomcat has only to type the name, password and roles of the new user. By using sessions, the user has a single sign-on. It is also possible to use the same process to grant access to specific EJBs, up to specific methods of a specific EJB, in JBoss. Depending on the roles given to a residential user, this user can do more or less, and HOUSE-KEEPER behaves differently. There are three roles:

- landlord: A landlord is considered an administrator of the smart home. She or he has the highest privileges, including the privileges of other roles. She or he could change the roles of the residential users, add new residential users, etc.
- familyMember: This role allows the user to browse the different Web pages representing the places in the smart home as well as to access Web pages representing the different smart devices. If permitted by the owner of a smart device, she or he could use this smart device.
- guest: A guest is allowed to browse the different Web pages representing the places or the smart appliances available, but cannot use appliances by default.

To summarize, both declarative and programmatic security control are provided in the HOUSE-KEEPER architecture. Declarative security allows landlords to specify residential users, their roles and their passwords without changing the application’s internal structure or code. When Tomcat is run, its container enforces these specified details. Programmatic security is used internally at the level of the custom tags to enforce this model at the code level where methods can check specific permission in the code itself depending on the user logged in during the session.

The previous paragraph describes how a landlord can grant access to HOUSE-KEEPER. As we know, this is a multi-user environment in which assets are distributed and owned by different people. It is where the XML context file helps. The owner of a device can specify an access control list at the device level. Each element “<deviceUserRole>” represents the name of an entity allowed to access the device description. Moreover, it is possible to specify an access control list for each method available. Each element “<methodUserRole>” represents the name of an entity allowed to use the specific method. For example, say a residential user has an MP3 player. She or he can allow some users to access the device Web page by typing the names of the users in the XML context file of this MP3 player. Say that this MP3 player has two methods: to listen to a song and to download a new song. This user can allow some users to listen a song and other users to listen and to download by filling in the XML context file.

In fact, there is a separation of the security management. From the service point of view, landlords manage access and privileges globally. Concerning the set of distributed smart appliances owned by different people, device owners are in charge of granting access to the different functionalities of the device. Device owners keep the right to share their device, whereas landlords keep the right to grant access to the HOUSE-KEEPER service.

Thanks to the above means, the overall integrity has been improved. The leverage of the actual security features of the Web paradigm and the J2EE stack has helped the system to reach this level of integrity quickly. This is a good compromise knowing that “the design of secure systems is an exercise in balancing costs against the threats” [CDK00] (p. 259). To minimize confidentiality issues, when a residential user browses the different Web pages representing the smart home, it has been specified that the Web pages should not be cached in the clients since they can contain information about what is available in the smart home. One can imagine scenario in which the client could store the Web pages anyway. It shows that there are holes in the security, but again the compromise is good enough. If we had to follow strictly the principles of robust programming “paranoia, stupidity, dangerous implements, can’t happen” found in [Bishop98], it would mean that there is no balance between costs and threats. To finish with security, the look at availability is related to the risk of denial of service attack inherent to any Web site.

4.2.5. The need-to-know issue

One of the questions that have arisen during the implementation is how to be able to present and use the different methods available in a smart device. Normally a programmer should know what method to call to do a specific task. In the smart home case, the type of smart device the residential users will buy and what it can do is not known in advance. It is the need-to-know issue because the programmer must write some code without knowing what specific code to write at the time of writing.

To work adequately, when a new smart device enters the home network, it should be possible to create the Web page representing what is possible to do with this device on-the-fly. This is the reason that the XML context file of a device contains different methods and their parameters description. This information is used to create a form in the Web page of the device, which will allow a user to send the right command to the device by clicking on a button. For example, for a printer, pseudo-code to print a document could be “print(myDocument)”. The XML context should contain:

```
... <methodName>print</methodName>
<methodDescription>print the document given as a parameter</methodDescription>
...<methodParamList><methodParam><paramName>documentName</paramName>
<paramDescription>The name of the document to print</paramDescription>...
```

The Web page representing the device would contain a form with a “print” button after an input text area in which the user would have to type the name of the document to print. After pushing this button, the “device EJB” counterpart of the smart printer device would receive a call on its “process” method.

Any “device EJB” has a method “processMethod(String methodName, String paramString)”. In the case of our smart printer, the call to print the user’s document would be “processMethod(print, myDocument)”. Depending on the type of the underlying high-level home networking technology, the body of the method “processMethod” is more or less straightforward.

Concerning the HP Chai technology, the ChaiServer on the smart printer would wait for an HTTP request which would contain the name of the method to process and its different parameters as strings of characters. Pseudo-code of the Chai implementation could be: “sendHTTPRequestToChaiServer(ChaiServerAddress,RequestString)”.

In our smart printer scenario, it would give:

```
sendHTTPRequestToChaiServer(SmartPrinterChaiServerAddress,  
MethodName=print&ParameterDocumentName=myDocument)
```

The need-to-know issue arises in relation to the Jini implementation of the process method. In Jini, the assumption is made that the programmer of a client should know what interface to search for a specific service and to hardcode the known methods in the interface. This could be possible for common interfaces such as for a printer or a scanner, but all cannot be known in advance. Say that the method to print a document in a Jini interface would be “print(myDocument)”.

The pseudo-code of the Jini type “device EJB” should be: “print(myDocument)”.

The problem is that the programmer cannot hardcode “print(myDocument)”. The content of the “processMethod” should work for a printer as well as for a scanner or anything else. For a scanner, the pseudo-code content could be: “scan(myDocument)”.

The programmer has a need to know what to hardcode. The issue is that it is not known in advance. The solution is to write a generic body, which will work for any method. In HOUSE-KEEPER, it has been done by using the reflection and introspection features of Java. Following is a simplified piece of Java code, which gives an idea of how to do that.

```

/**
 * This method will call the desired method on the Jini HK service object
 * @param x Object the Jini HK service object
 * @param paramString String the string of arguments of the method in the same
 * order as in the real method
 * @return String the response of the method done, null if an exception is caught
 */

private String invoke(Object x, String methodName, String paramString)
{
    String response = null;

    try
    {
        System.out.println("Trying to invoke this method : " + methodName);
        Method m = null;
        Method [] methods = x.getClass().getMethods();
        System.out.println(methods.length + " methods found");

        for (int i=0;i<methods.length;i++)
        {
            String name = methods[i].getName();
            System.out.println("The name of the method "+ i + " is: " + name);

            if (name.equals(methodName))
            {
                m = methods[i];
                System.out.println("The method matching in the set of methods is: "+
                    m.toString());
            }
        }

        //Change the String of parameters as an array of Objects
        // one object is one parameter
        Object[] paramArray = getParamArray(paramString);
        response = (m.invoke(x, paramArray)).toString();
    }
    catch ( Exception e )
    {
        System.out.println("Exception when invocation of a special method on a "+
            "Jini HK service : " + e + e.getMessage());
        e.printStackTrace();
    }

    return response;
}

```

4.2.6. More about the Jini proxy for Jini devices

It has been mentioned in the big picture of the HOUSE-KEEPER architecture that a proxy design pattern could be used to include a smart device that lack the capabilities required to be integrated in HOUSE-KEEPER. For example, if the device could not persistently store the XML context because it has no hard drive, a proxy would store the XML context on behalf of the device. In HOUSE-KEEPER, all Jini devices have such a proxy. In addition to giving an example implementation of this design solution, it solves another problem.

An idea behind the Jini technology is the fact that we will not be interested in using a specific device but a kind of service. For example, in a smart building, a person who would like to print a document would not search for a specific printer, but would ask to find an available printer and to print the document on this printer. People would think more about services, what something they want to do, rather than about specific devices. Due to this idea, in Jini, the standard way is to identify the different services available in the home network rather than the devices themselves.

In fact, a device is more likely to be an aggregation of services. In HOUSE-KEEPER, a smart home is composed of smart devices. Each device has an owner, even though this device can be shared. There are not so many appliances providing the same services. The paradigm of services was not convenient. The use of a proxy helps thinking at the device level in Jini. A device should provide different services and implement different interfaces. In Jini, each of these services should get a unique identifier, even though they are the part of the same device. It could be better to have a unique identifier for a Jini service, which would aggregate all the different services of the device. Hence, the proxy of a Jini device in HOUSE-KEEPER is a stand-alone Jini service, which points to the different Jini services of the smart appliance.

In the XML context of the Jini service HOUSE-KEEPER proxy, which represents the device, each method is one of the methods available in the different services related to the device. For example, a Jini lamp would have a Jini service HOUSE-KEEPER proxy running somewhere in the home network. This proxy would store the XML context of the Jini lamp.

Since the lamp would provide a Jini service to switch off the lamp, the XML context would have this kind of content to specify this method with the unique identifier of the specific Jini service:

```
<method>
<methodName>turnOff</methodName>
<methodDescription>Switch off the lamp</methodDescription>
<methodACLList>
<methodUserRole>jm</methodUserRole>
</methodACLList>
<methodParamList>
<methodParam>
<paramName>jiniServiceTargetId</paramName>
<paramDescription>The unique id of the Jini service target</paramDescription>
<paramType>hidden</paramType>
<paramValue>2fea2965-e6bb-411b-86a3-ff572e1a12f1</paramValue>
</methodParam>
</methodParamList>
</method>
```

When a Jini service HOUSE-KEEPER proxy receives a command to process a specific method on the specific device that it represents, it uses the parameter value of the “jiniServiceTargetId” to process the right method on the Jini service targeted by this value, which is its unique identifier.

4.3. Critique of the architecture

4.3.1. Place metaphor

The HOUSE-KEEPER way to interact with devices remotely is to use the Web page paradigm. Alternatively, the Cyberspace project [CYBERS] has written specification for a Jini Place API . In [Venners99], the main idea is to use Jini to interact with smart devices represented as objects, which are organized in places. A real thing should be represented as it is represented in the real world. A toaster should look like a toaster and the user could be able to interact with this toaster as in the real world. The standard document-driven Web representation is not suitable for that. The choice of the Web page paradigm for monitoring and controlling smart devices introduces limitations on how it is possible to interact with the devices. Nevertheless, the Cyberspace option cannot work over the standard Web architecture. For example, a special Web browser has to be used to browse Cyberspace places.

In fact, this is an interesting point for the improvement of HOUSE-KEEPER. So far, a Web front-end has been created. More powerful clients could be created to provide more user-friendly representations. A GUI (Graphical User Interface) with 3D representations of the smart home could be imagined.

4.3.2. What has to be shipped?

We have already seen that the XML context model is difficult to define. The risk that something could change is high. To change the model is easy but to handle the change globally is hard due to backward compatibilities.

Another issue concerning the XML context is the fact that the methods available on the device have to be defined in the XML context. It means that someone has to write the correct information in the XML context file in order to display the methods available in the Web page describing the device. Many solutions are available to solve this problem. The obvious one is that the programmer of the device would fill the XML context part concerning the methods available. The smart device would be shipped to the residential user with the XML content file and the pieces of software required to be used in HOUSE-KEEPER. The remainder of the XML context information would be filled by the user or an automatic system. For example, the device location could be sent automatically to the device and an automatic write of the location and its description could be done. However, since HOUSE-KEEPER is not a standard, the probability to have the XML context file and all necessary software shipped from the source to the residential user is low. There are still at least two solutions.

Some of the high-level home networking technologies provide the possibility to upgrade the capabilities of a device on-the-fly. Jini and Chai have the possibility to move code from one entity to another and to add new services to smart devices. The solution should consist of uploading new services in a device to offer it the capability to be registered in HOUSE-KEEPER. Of course, a solution will have to be built for each type of home networking technology. Following is an implementation of what could be done to solve the problem. For Chai devices, when a smart device enters the home network, HOUSE-KEEPER is notified even if the device does not know anything about HOUSE-KEEPER. After this notification, HOUSE-KEEPER could try to upload what is needed for the device to be registered such as a

blank XML context and other ChaiServices. The owner of the device could update the XML context either using a GUI on the smart device, which could automatically fill the content, or typing directly into the file. Then the registration would be done as normally. A difficult point for the owner of the device will be to know what to write to describe the different methods available. For popular smart devices, it may be possible to have well-known Web sites with XML context examples, which could be downloaded, to help the owner.

The other solution is to take advantage of the proxy idea. This solution can be used with the actual HOUSE-KEEPER implementation for Jini devices. As explained in section 4.2.6, the Jini service HOUSE-KEEPER proxy aggregates the different services available on a device. This means that the unique identifier of each service has to be known because it has to be written in the XML context. To help the owner of the device to find these identifiers, a special Web page in HOUSE-KEEPER allows her or him to find any Jini services in the home network. This page can also present the unique identifier of each service, as well as all methods that it provides thanks to Java reflection and introspection features. The owner of the device just has to copy the information provided. In addition, the problem of knowing how to describe textually the methods available in the XML context is solved. What has to be written for a method is almost what is displayed on the Web page. As an aside, this process allows the creation of virtual devices by running a Jini service HOUSE-KEEPER proxy, which aggregates any wanted Jini services present in the home network.

4.3.3. Limitations of the command line paradigm

There is one implementation of the front-end, which uses the Web page paradigm. This gives a command line paradigm. The parameters must be textual. This is fine for simple methods, especially since the parameters are typed by human users. It is possible to print a given document, whose path has been given as a textual parameter, or to switch on and off an appliance.

To batch several simple actions in one click is more difficult using this paradigm. It could be possible to have a “macro” method available directly in the smart device to do more than one action with one click. It would be even more complex to create a kind of batch process from a Web browser. On the other hand, to process one method requiring a complex parameter cannot be done. Simple Java parameters such as primitive types, String objects or

Boolean objects can be processed. It is not possible to process a Java method requiring a “Car” object or any kind of complex object.

Nevertheless, it is possible to call any type of method over the Web. SOAP (Simple Object Access Protocol) [SOAP] allows software components and applications to communicate using standard Internet HTTP. More complex clients could be created to call complex methods on smart devices using SOAP. Thanks to the use of JBoss, this would not require changing the back-end of HOUSE-KEEPER. The new release of JBoss should provide a way to access and invoke EJBs using SOAP. The current version has a module called JBossZOAP [JZOAP], an alternative invocation layer using ZOAP. The Zero-effort Object Access Package is an open-source SOAP implementation for the Java 2 platform. These modules could be used to access the different HOUSE-KEEPER EJBs from “thicker” clients. It would only require to write new clients. Only the front-end should have to be improved. This feature should also ease the implementation of non-Java based high-level home networking technologies such as UPnP.

4.3.4. Is HOUSE-KEEPER a distributed system?

HOUSE-KEEPER is a distributed system which could have been more or less distributed depending on the choices in the design.

The Java application part of HOUSE-KEEPER could have been implemented as modules in many different boxes or devices with enough processing power to run it. The advantage of this solution is that there is no single point of failure. On the other hand, the difficulty in managing the whole system increases. The consistency between the different devices and their view of the smart home is difficult to maintain in case of network partition, for example. Another point is that many boxes with computational power do not exist in a home.

The second and chosen approach is to locate the whole HOUSE-KEEPER Java application in the home PC. At this stage, it looks like a centralized application. However it is not as clear as that. HOUSE-KEEPER is a part of the smart home system.

HOUSE-KEEPER deals with other distributed entities to reach its goals. Several functionalities of HOUSE-KEEPER rely on distributed components, amongst them:

- HOUSE-KEEPER does not implement the discovery service. It becomes a piece of the distributed system for gaining the functionality to find the other entities thanks to the discovery services of high-level home networking technologies.
- [DAS99] demonstrated that it is important to decouple the context information sensed in the environment and how it is sensed. HOUSE-KEEPER uses the context information stored in different XML context files. These XML files are distributed in the home network and can be stored on behalf of proxies, which are also distributed. The principle of decoupling the context information sensed and how it is sensed is applied because no assumption is made on how information is written in the XML context file. It can be updated manually by a user with a text editor or filled programmatically by a software. The location information could be extracted from a GPS system and updated in the file by software.
- In HOUSE-KEEPER, the storage of information is left to the JBoss server. JBoss manages the connection to the database. The new version of JBoss could allow the use of cluster technology to increase the level of distribution and availability.
- The right of giving access to a device is left to the owner of the device and distributed thanks to the presence of the XML context file, which can be located anywhere in the home network.
- The real smart devices are distributed in the home. The clients, which can access HOUSE-KEEPER, are also distributed. They could be PCs in and outside the home, WAP phones, other products with a standard Web browser, etc.

On the other hand, HOUSE-KEEPER has centralized parts. The important ones are:

- The gateway between the in-home network and the Internet is the PC in which Tomcat and JBoss, as well as the Java application, run. The Java application is also the client of the different discovery protocols and populates the EJB server.
- The right to give access to the smart home Web pages is left to landlords and centralized in Tomcat.

4.3.5. Adaptation to the client capability

The Web content of the Web pages representing the smart devices is dynamic. This means that HOUSE-KEEPER must be able to generate dynamic Web pages. Several technologies are available to do that. The Tomcat JSP container has been chosen because it is a well documented open source. Also, JSP is a part of the Java technology. Thus JSP can be used with the other HOUSE-KEEPER components easily and effectively.

Following this choice, there is still a requirement of adaptation to the client capabilities. The current implementation displays the smart home representation on a standard Web browser running on a PC or on a WAP phone. The adaptation to the client capability is done manually by choosing to display the JSPs dedicated to the right client. This way the programmer has to build each part more or less twice because they are quite similar. There are two sets of JSPs, two sets of custom tags: one set for Web browser clients and one set for WAP phones. It could be better to reuse some parts of the work.

Two solutions are available. A JSP could generate an XML document which contains the content to be displayed. After the generation of this XML file, a style sheet XSL [XSL] could be applied to this XML to generate the final document matching the type supported by the client device, for example an HTML document for a Web browser or a WML (Wireless Markup Language) [WML] document for a WAP phone. The Cocoon [COCO] project allows the generation of a final type document from an XML file automatically. Unfortunately, the Cocoon project does not work with JSP pages and is not a part of the J2EE. Due to that, this solution cannot be implemented without significant changes.

The good point of the design is that we have a database-like paradigm. The different EJBs in JBoss are in a certain way queried. Any new client has just to be able to communicate with this set of entities to extract the information it needs to generate the view of the smart home.

4.4. The prototype and its results

4.4.1. Test-bed

A test-bed has been set up to represent a real smart home environment. It allows the addition of “real” smart devices and real case scenarios to the software service provided by HOUSE-KEEPER.

There were no smart devices based on Chai or Jini available. Instead, X10 modules have been wrapped into software pieces of Chai or Jini code. Thus, a desktop lamp plugged into a X10 module can be discovered automatically on a Chai or Jini network. This real lamp can then be controlled from a Web page or a WAP phone simulator [WAPS]. This desktop lamp can be considered as a “real” smart lamp.

Thanks to the use of X10, which uses the powerline network and a laptop PC with a wireless network card, the test-bed approximates a smart home environment: a home network is set up without rewiring the house. The laptop is connected to the X10 network through its serial port in which a X10 module, called CM12U, allows the communication with other X10 modules through the powerline. In our case, the PC, which acts as the residential gateway, was already connected to a LAN and had access to the Internet. Of course, a home PC should be connected to the Internet, for example via an ADSL (Asymmetric Digital Subscriber Line) [DSL] connection, but concerning the wireless link between the laptop and this PC, a wireless technology should be chosen. The following two photos describe the final test-bed.



Figure XIII: Photo of the complete HOUSE-KEEPER test-bed

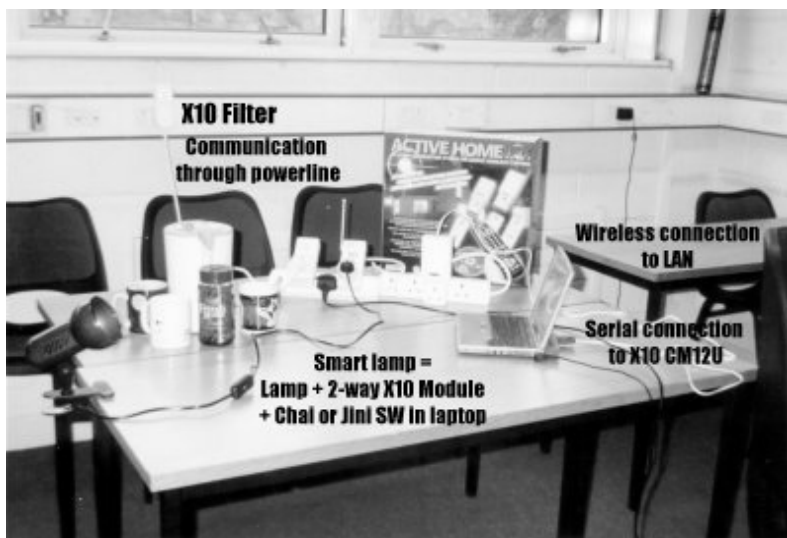


Figure XIV: Photo of the smart devices side of the HOUSE-KEEPER test-bed

Another solution for the low-level home network would have been to reuse a Lonworks network present in the Computer Science buildings of Trinity College Dublin. Most of the components, such as motion sensors and lights, of this network are fixed and located in different rooms. The X10 network allows users to quickly plug in and remove modules in the home network and can be set up in any room with electrical outlets. This facilitates experiments requiring discovery of appliances.

Quite popular in the US, X10 modules are now available for most of the European electrical outlets. HOUSE-KEEPER has shown that some of the X10 Java APIs [JX10] available to control American type X10 modules work for European type X10 modules. Authors of these APIs were not sure of that before the HOUSE-KEEPER prototype. This is another benefit to have used the X10 technology.

4.4.2. “Use-cases”

As we can see in the two following “snapshots”, the Web pages of a Web browser client are more user-friendly than the WML pages displayed on a WAP phone. This point demonstrates that the adaptation to the client capability is an important feature.



Figure XV: Web page representing the place “home”



Figure XVI: The place “home” as displayed on a WAP phone

By having separated the back-end from the front-end in HOUSE-KEEPER, we have been able to create views of the smart home adapted to the client capability.

In the following two other “snapshots”, we are in a scenario in which we have two different smart devices, a “jmCoffeeMachine” and a “jmLamp”.

The first one is a session of the user “jm”. “Jm” is the owner of the two devices and has not granted anyone else to access them. At the moment “jm” is browsing the Web pages, the “jmCoffeeMachine” has been removed from the home network. It could be because the “jmCoffeeMachine” is down. Since “jm” is its owner, the Web page displays a message warning him that the “jmCoffeeMachine” cannot be reached, whereas it should be.

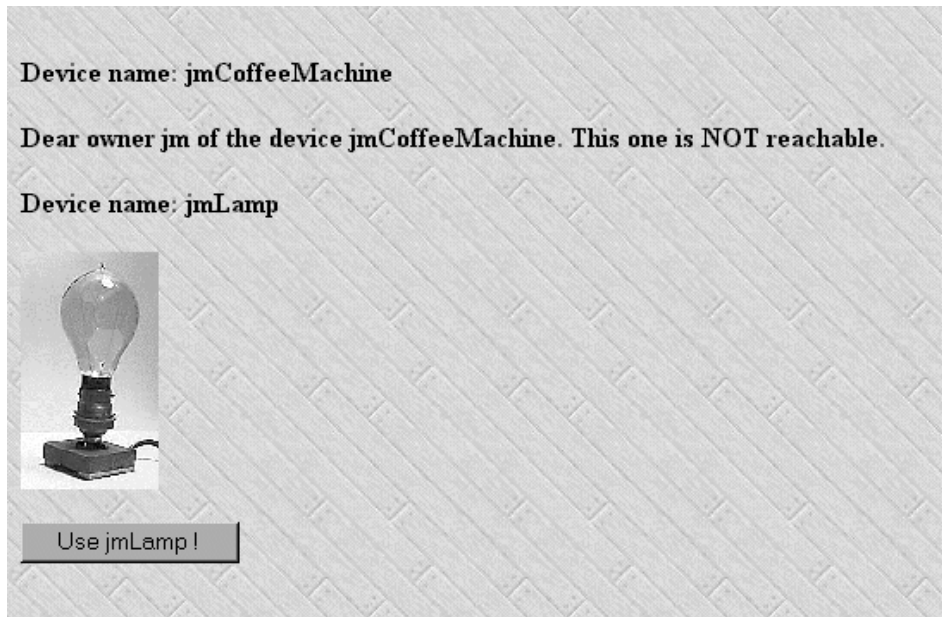


Figure XVII: Web content of the use case for “jm”

In the second “snapshot” below, the user browsing the Web pages is “alexis”. “Alexis” is a landlord, but is not the owner of the “jmCoffeeMachine”. HOUSE-KEEPER behaves then differently than in the first case and displays another kind of warning message while sending a warning email to the user “jm”.

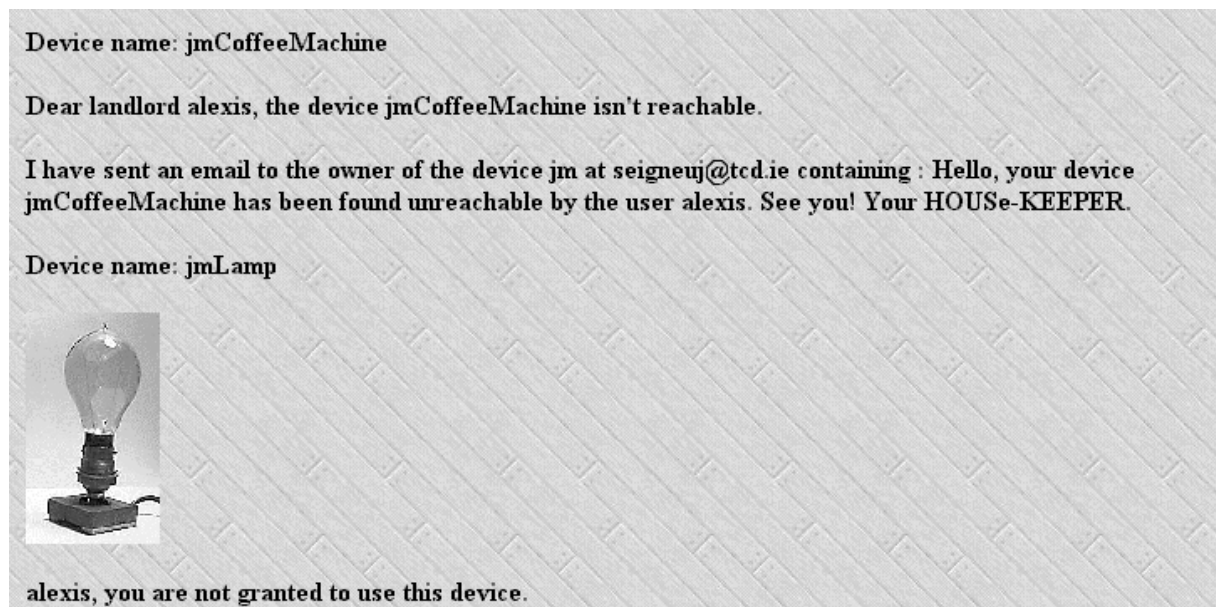


Figure XVIII: Web content of the use case for “alexis”

At the bottom of the previous “snapshot”, it is written that “alexis” cannot use the “jmLamp” because “jm” has not granted the use of his lamp. The above scenario demonstrates the type of multi-user environment and its security scheme that HOUSE-KEEPER provides thanks to authentication, session, authorization and declarative access control. In the same way, when a user is allowed to access a device but is restricted to using a subset of the methods available on this device, the Web page would only contain action buttons for the granted methods and display a message otherwise.

4.4.3. What did two implementations bring to light?

To implement two different home networking technologies has shown that HOUSE-KEEPER has a good level of independence against the underlying high-level home networking technologies thanks to a high degree of common structure. This section describes other points that have come up thanks to the presence of two different implementations.


It could be an issue to rely on a specific attribute in one technology to describe a part of a device because the same kind of information is not available in another technology. Jini offers many ways to describe a device, including some context information, and to retrieve this information. For example, in Jini, a device may register its service along with a set of objects, which would describe the device characteristics. Chai provides far fewer means to describe and retrieve device information than Jini.

The use of separate XML context files has shown that it helps to have the same kind of information for any devices. Nevertheless, doing that, the functionality to describe a device has moved from the level of high-level home networking technologies to the HOUSE-KEEPER level. We can still think about filling the XML context files programmatically thanks to the information provided by Jini or any other technology.

The two following “snapshots” represent how a Chai smart device and a Jini smart device are displayed.

HOUSE-KEEPER

Chai device: jmCoffeeMachine



Description: H

Actions:

Method name: ProcessThis

Method description: Process the textual command received on the X10 module

Param 1 name: Command

Param 1 description: Value: "on/off" to switch on/off, if possible on the X10 module "dim" to dim or "brigh

Type your value for this parameter here:

Param 2 name: CommandValue

Param 2 description: Value: None for on/off, "1"... "10" intensity to dim/brighten

Type your value for this parameter here:

Param 3 has been filled for you :)

Param 4 has been filled for you :)

Do ProcessThis !

Figure XIX: Web representation of a Chai device



Figure XX: Web representation of a Jini device

The point is that the representation process is the same for any device independent of the home networking middleware type. For residential users, we could avoid displaying type of home networking at the top of the Web page. Since it is transparent for a residential user to use a Jini or a Chai device in HOUSE-KEEPER, this information could be deleted.

Performance, as for the speed of processing, is not an objective of HOUSE-KEEPER as long as the processing time seems to be convenient for a human user. Both implementations either in Chai or in Jini have convenient delays in normal conditions.

Nevertheless, as an aside, the Jini implementation has shorter delays, e.g., to switch on a X10 lamp, than the Chai solution. In some scenarios, the speed of processing a command on a X10 module could be a key factor. To provide a Jini wrapping of the X10 module should be slightly faster. Anyway, we cannot speak about real time processing.

Another issue related to the time is how to define the right delays to tune the different services provided by home networking middleware, such as their discovery service. To tune Chai has been less difficult than to tune the different Jini leases.

In fact, it has not been straightforward to implement either Jini or Chai. Chai is simple. It has means to discover devices plugged into a Chai network and to send notifications to other devices among other services. Nevertheless, most of these services do not provide complex functionalities. Of course, these services could be improved by coding extra functionalities. That could increase the development time considerably. Jini has a longer learning curve because it provides so many services. Unlike Chai, thanks to these Jini services, it is possible to think about complex functionalities and to realize them by using the Jini services as they are. Furthermore, the huge community of Jini developers and their projects offer more services to be reused.

During the implementation of HOUSE-KEEPER, Chai was implemented faster than Jini concerning the discovery part. On the other hand, when it was the time to find a way to communicate with X10 modules, it was necessary to code a bridge from Chai to X10 “from scratch”, whereas HOUSE-KEEPER reuses a Jini to X10 bridge from a previous Jini.org community project.

In section 4.2.6, it is explained that Jini is more about Jini services than Jini devices. We could say that Jini is service-oriented. This is the reason that in section 4.2.6 also, a way has been described to create some Jini devices, because HOUSE-KEEPER is device-oriented rather than service-oriented.

The same issue did not arise with Chai because Chai is more device-oriented from an architecture point of view. It should be enough to have a ChaiServer, the Chai-specific Web server, per device. Since what is discovered is ChaiServers, it is possible to say that devices are discovered. In Jini, services are discovered. This is the reason that service-oriented high-level home networking technologies should be more difficult to implement in HOUSE-KEEPER. Nonetheless, it is possible since Jini has been implemented successfully.

5. **Conclusion**

The main result of the HOUSE-KEEPER project is the elaboration of a vendor-independent architecture for the easy management of smart homes. After a description of the domain of home networking, the requirements of such an architecture have been listed. Since HOUSE-KEEPER is dedicated for residential use, particular attention has been taken to define what they should expect from the services extracted from this architecture.

A home is a multi-user environment. The architecture allows the creation of different types of users with different privileges and assets. In a home, there is a “landlord”, who takes decisions that involve the whole home or one of its components. There are “familyMembers”, who are the other residents of the home. They have personal assets, such as smart appliances. Thanks to distributed XML context, the owners of smart devices keep the right to grant access to their devices. Moreover, they also have the possibility of choosing which specific service can be granted amongst the different functionalities provided by the device.

Any device has the same XML context model. That gives a global process to gather the important information extracted from the context. After having been extracted, it is also important to keep it for later use. This information will fuel the next generation of services in smart home systems. As more information is stored, the quality of these services will improve. Therefore, a technology that can store up to software objects persistently without much burden for the programmer was chosen to store the history of the state of the environment. This is JBoss, an EJB server.

The XML context model is open to extension. Elements could easily be added to describe further devices or locations overtime. The front-end representing the view of the smart home uses known metaphors, providing a high level of abstraction. Places appear in relation with other places. In fact, a place lists the “subplaces” reachable from this current place. HOUSE-KEEPER is aware of relations between locations and smart appliances. A place displays the different devices present there.

Concerning the security issues under work in all the high-level home networking technologies, HOUSE-KEEPER reuses efficiently some parts of the e-commerce and Web security mechanisms to meet the security requirement. There is an easy and global declarative process for granting access control to HOUSE-KEEPER services as well as for specific devices and their methods. Thanks to the J2EE-based components, the architecture is open for an easy and effective improvement of security protections.

The implementation of the architecture in two different home networking technologies has confirmed the point that HOUSE-KEEPER has a good level of independence from the underlying high-level home networking technologies. Thanks to the use of proxies, a device without the capabilities necessary to be registered can be integrated. The front-end can also be adapted to the client capability. The representation of the smart home is available in the form of standard Web pages for a PC Web browser and in the form of WML pages for a WAP phone. Other clients could be built knowing that the information can be retrieved easily as soon as the client can query JBoss.

HOUSE-KEEPER automatically generates a cohesive view of the smart home and its smart devices. It should also know how to react to special events in the smart home without contacting the users. To a certain extent, it customizes its behaviour depending of the member currently browsing the Web pages thanks to the session, authentication and authorization features in the Tomcat JSP and servlet engines. It is far from artificial intelligence functionalities but it is related. HOUSE-KEEPER decides whether to send an email to the owner of a suspected broken device depending on the context at the moment this device is found broken. It is a simple scenario which shows that interesting features can be added thanks to the multi-user environment and some context information.

This is to highlight the fact that HOUSE-KEEPER is a real service, made of more than 13,000 lines of code. Real X10 modules have been used. It is possible to boil real water from a Web browser or a WAP phone simulator, as well as to receive warning messages when the plug of a desktop lamp is removed from the powerline. Such X10 modules are wrapped in Jini or Chai pieces of software. To do that in Chai, a simple bridge from Chai to X10 has been built.

Much more could be added. Future work could be to improve mobility. The project did not aim to have mobile devices. Smart appliances were considered relatively fixed, which is the case of big appliances such as fridges or burglar alarms. There is a simple mechanism to update the different EJBs. HOUSE-KEEPER should become the client of the event services provided by high-level home networking technologies, as for their discovery services. The next update mechanism should use these event services. Special care should be taken concerning synchronization and caching issues of context information.

A further step could be to use the programmatic security at the level of the EJBs to take advantage of the easy security management functionalities provided by JBoss. By moving access restriction to methods at the EJB level rather than at the custom tags level, it would not be a security hole if a client could by-pass the custom tags and access the EJB methods directly. It could be worth using more declarative security features of JBoss. A third type of entity EJB could be inserted, the “people EJB” representing a residential user. To make the UPnP implementation of HOUSE-KEEPER would be significant since UPnP is going to be really popular. The SOAP alternative invocation layer to invoke EJBs, which should be implemented in the next version of JBoss, should ease significantly this implementation. This feature of JBoss will also open the door to more user-friendly clients doing more complex tasks.

6. References

- [ACKMM00] Dinesh Ajmera, Paul Castro, Ted Kremenek, Murali Mani, Richard Muntz, “My Building Knows Where I am! Using Jini™ Technology as a Framework for Supporting Smart Spaces”, *JavaOne 2000*, 2000, <http://msl.cs.ucla.edu/muse/slides/TS-1452.pdf>
- [Barron99] Peter Barron, “A Distributed Event System for Use in Mobile Environments”, Masters Thesis, Computer Science Department, Trinity College Dublin, September 1999, <http://citeseer.nj.nec.com/278773.html>
- [BFF96] T. Berner-Lee, R. Fielding, H. Frystyk, “Hypertext Transfer Protocol -- HTTP/1.0”, World Wide Web Consortium (W3C), May 1996, <http://www.w3.org/Protocols/rfc1945/rfc1945>
- [Bishop98] Matt Bishop, “Robust Programming”, 1998, <http://nob.cs.ucdavis.edu/~bishop/classes/ecs153-1998-winter/robust.html>
- [BMKKS00] Barry Brumitt, Brian Meyers, John Krumm, Amanda Kern, Steven Shafer, “EasyLiving: Technologies for Intelligent Environments”, In the *Proceedings of Second International Symposium on Handheld and Ubiquitous Computing, HUC 2000*, pages 12-29, Microsoft Research, September 2000, <http://citeseer.nj.nec.com/brumitt00easyliving.html>
- [BR00] Christian Bettstetter, Christoph Renner, “A Comparison Of Service Discovery Protocols And Implementation Of The Service Location Protocol”, Munich Institute of Communication Networks, 2000, <http://citeseer.nj.nec.com/bettstetter00comparison.html>
- [Buxton95] Bill Buxton, “Ubiquitous Media and the Active Office”, *Nikkei Electronics*, 3.27 (no. 632), pages 187-195, University of Toronto & Xerox PARC, 1995, <http://www.dgp.toronto.edu/OTP/papers/bill.buxton/ubicomp.html>
- [CCPS00] Vic Callaghan, Graham Clarke, Anthony Pounds-Cornish, Sue Sharples, “Buildings as Intelligent Autonomous Systems: A Model for Integrating Personal and Building Agents”, In the *6th International Conference on Intelligent Autonomous Systems, Venice July 25-27 2000*, University of Essex, 2000, <http://cswww.essex.ac.uk/intelligent-buildings/publications/essexias6.pdf>
- [CD00] Deborah Caswell, Philippe Debaty, “Creating Web Representations for Places”, HP Labs, 2000, <http://cooltown.hp.com/dev/wpapers/placeman/placesHUC2000.pdf>
- [CDK00] George Collouris, Jean Dollimore, Tim Kindberg, “Distributed Systems, Concepts and Design”, Addison-Wesley, Third Edition August 2001, ISBN:0201619180, <http://www.aw.com/info/collouris/>

[CEAGTH] “Guide to Home Networks”, CEA Home Networks and Information Technology Division, Consumer Electronics Association, <http://www.ce.org/networkguide/>

[CGKKNMR00] R. H. Campbell, M. Garland, R. Kravets, D. Kriegman, K. Nahrsted, M. D. Mickunas, D. Reed, “Active Information Spaces based on Ubiquitous Computing”, CS UIUC, Spring 2000, <http://choices.cs.uiuc.edu/2k/Internal/doc/gaia.pdf>

[CISG00] “Residential Gateways: Unleashing the Broadband Services Tsunami”, RC00-11HN, Cahners In-Stat Group, December 2000, http://www.instat.com/rh/ced/rc0011hn_story.htm

[Coen99] Michael H. Coen, “The Future of Human-Computer Interaction or How I learned to stop worrying and love my Intelligent Room“, *IEEE Intelligent Systems*, MIT Artificial Intelligence Lab, March/April 1999, <http://www.ai.mit.edu/people/mhcoen/ieee.pdf>

[DAS99] Anind K. Dey, Gregory D. Abowd, Daniel Salber, “A Context-Based Infrastructure for Smart Environments”, In the *Proceedings of the 1st International Workshop on Managing Interactions in Smart Environments (MANSE '99)*, pages 114-128, Georgia Institute of Technology, 1999, <http://citeseer.nj.nec.com/237330.html>

[DC00] Philippe Debaty, Debbie Caswell, “Uniform Web Presence Architecture for People, Places, and Things”, Technical Report HPL-2000-67, HP Labs, <http://cooltown.hp.com/dev/wpapers/uniform-Webpresence/uniform-Webpresence.asp>

[Dübendorfer01] Thomas Dübendorfer, “An Extensible Infrastructure and a Representation Scheme for Distributed Smart Proxies of Real World Objects”, TR 359, ETH Zurich, Institute of Information Systems, April 2001, http://www.inf.ethz.ch/vs/publ/papers/TR_359.pdf

[Elkin00] Tobi Elkin, “Simplifying Life @Home”, *Vision*, July/August 2000, http://www.ce.org/vision_magazine/editions/2000/julyaug/p24.asp

[Enikia99] “The IAN Information Appliance Network™ in Pervasive Home Computing”, Enikia, 1999, <http://www.enikia.com>

[ER01] W. Keith Edwards, Tom Rodden, “Jini Example by Example”, Prentice Hall, 2001, ISBN:0130338583, <http://www.kedwards.com/jini/jinibyexample/>

[FDS00] David Frohlich, Susan Dray, Amy Silverman, “Breaking up is hard to do: family perspectives on the future of the home PC”, Technical Report HPL-2000-117, HP Labs, 2000, <http://www.hpl.hp.com/techreports/2000/HPL-2000-117.html>

[FJHW00] Armando Fox, Brad Johanson, Pat Hanrahan, Terry Winograd, “Integrating Information Appliances into an Interactive Workspace”, *IEEE Computer Graphics and Applications*, Stanford University, May/June 2000, <http://graphics.stanford.edu/projects/iwork/papers/ieee-pda00/>

- [LF+01] Norbert Loeken, Wolfgang Fickus et al., “Analyst Report on Service Gateways”, WestLB Panamure, March 2001, <http://www.osgi.org/news/WestLBStudy.pdf>
- [Glick99] Erick Glick, “Home Networking The Time is Now”, *Vision*, September/October 1999, http://www.ce.org/vision_magazine/editions/1999/sepoct/pg20_24.asp
- [Hedberg00] Sara Reese Hedberg, “After desktop computing: a progress report on smart environments research”, *IEEE Intelligent Systems*, Volume 15, Issue 5, pages 7-9, 2000, <http://www.computer.org/dsonline/0101/>
- [HHSWW99] Andy Harter, Andy Hopper, Pete Steggles, Andy Ward, Paul Webster, “The Anatomy of a Context-Aware Application”, In the *Proceedings of the ACM/IEEE MobiCom, August 1999*, 1999, <http://citeseer.nj.nec.com/harter99anatomy.html>
- [HW00] Fredrik Hederstierna, Markus Wejrot, “Jini in a cellular telephone”, The Department of Computer Science, Lund Institute of Technology, Sweden, 2000
- [IECHN] “Home Networking”, International Engineering Consortium, http://www.iec.org/tutorials/home_net/
- [KBMBC00] Tim Kindberg, John Barton, Jeff Morgan, Gene Becker, Debbie Caswell, “People, Places, Things: Web Presence for the Real World”, Technical Report HPL-2000-16, HP Labs, February 2000, <http://cooltown.hp.com/dev/wpapers/WebPres/WebPresence.asp>
- [Linn93] J. Linn, “Privacy Enhancement for Internet Electronic Mail”, RFC1421, World Wide Web Consortium (W3C), February 1993, <http://www.w3.org/History/1993/WWW/AccessAuthorization/rfc1421.html>
- [MA97] Jen Mankoff, Gregory D. Abowd, “Domisilica: Providing Ubiquitous Access to the Home”, Technical Report GIT-GVU-97-17, Georgia Institute of Technology: Graphics, Visualization and Usability Center, March 1997, <http://citeseer.nj.nec.com/mankoff97domisilica.html>
- [MBBMM98] Elizabeth Mynatt, Douglas Blattner, Meera M. Blattner, Blair MacIntyre, Jennifer Mankoff, “Augmenting home and office environments”, In the *Proceedings of the third international ACM conference on Assistive technologies April 15 - 17, 1998*, pages 169-172, 1998, <http://www.parc.xerox.com/red/members/mynatt/pubs/assets.html>
- [Meadows00] Jennifer H. Meadows, “Residential Gateways & Home Networks”, Technology Futures, 2000, <http://www.tfi.com/pubs/19gateways.html>
- [Meier00] René Meier, “State of the Art Review of Distributed Event Models”, University of Dublin, Trinity College, March 2000, <http://citeseer.nj.nec.com/437791.html>

[Mills99] Kevin L. Mills, “AirJava: Networking for Smart Spaces”, Information Technology Laboratory, National Institute of Standards and Technology, 1999, <http://citeseer.nj.nec.com/330178.html>

[MM98] Michael C. Mozer, Debra Miller, “Parsing the Stream of Time: The Value of Event-Based Segmentation in a Complex Real-World Control Problem”, University of Colorado, 1998, <ftp://ftp.cs.colorado.edu/users/mozer/papers/eventbased.pdf>

[Mozer99] M. C. Mozer, “An intelligent environment must be adaptive”, *IEEE Intelligent Systems and their Applications*, No. 2, Vol. 14, 1999, pages 11-3, 1999, <http://www.cs.colorado.edu/~mozer/papers/ieee.html>

[MV00] Gary McGraw, John Viega, “Make your software behave: Assuring your software is secure”, *IBM developerWorks*, Reliable Software Technologies, February 2000, <ftp://www6.software.ibm.com/software/developer/library/assurance.pdf>

[OJM97] Frank Olken, H.-Arno Jacobsen, Chuck McParland, “Middleware Requirements for Remote Monitoring and Control”, Lawrence Berkeley National Laboratory, <http://www.objs.com/workshops/ws9801/papers/paper097.html>

[OSGi00] “OSGi Service Gateway Specification Release 1.0”, The Open Services Gateway Initiative, May 2000, <http://www.osgi.org/about/spec/osgi-spec.pdf>

[P+01] Grant Palmer et al, “Professional JSP 2nd Edition”, Wrox Editions, 2001, ISBN:1861004958

[Sanders00] Jane M. Sanders, “Sensing the Subtleties of Everyday Life”, *Research Horizons*, Georgia Tech Research Institute, Winter 2000, <http://www.gtri.gatech.edu/rh-win00/main.html>

[SUN00] “The Connected Home Powered by Java Embedded Server™ Software”, Sun Microsystems, December 2000, <http://www.sun.com/software/embeddedserver/whitepapers/dot.com.home.pdf>

[Troll00] Ryan Troll, “Automatically Choosing an IP Address in an Ad-Hoc IPv4 Network”, IETF (Internet Engineering Task Force) Internet draft, March 2000, <http://www.ietf.org/proceedings/00jul/I-D/dhc-ipv4-autoconfig-05.txt>

[UCSQ01] *UPnP connections*, volume IV, Second Quarter 2001, http://www.upnp.org/newsletter_06_2001/default.htm

[Venners99] Bill Venners, “A Walk Through Cyberspace”, *JavaWorld*, December 1999, http://www.javaworld.com/javaworld/jw-12-1999/jw-12-jiniology_p.html

7. World Wide Web resources

[APPLE] Apple, <http://www.apple.com>

[BATB] Batibus, <http://www.batibus.com/anglais/gen/>

[BTOO] Bluetooth, <http://www.bluetooth.com>

[CEBUS] CEBus, <http://www.cebuse.org>

[CES] Consumer Electronics Show, <http://www.cesWeb.org/>

[CHAI] HP Chai technology, <http://www.hp.com/products1/embedded/>

[COCO] Cocoon Apache project, <http://xml.apache.org/cocoon/index.html>

[COOLT] Cooltown, <http://www.cooltown.com>

[CORBA] Common Object Request Broker Architecture, <http://www.corba.org>

[CSHA] C#, <http://msdn.microsoft.com/vstudio/nextgen/technology/csharpintro.asp>

[CYBERS] Cyberspace Jini.org project,
<http://developer.jini.org/exchange/projects/cyberspace/>

[DAVIS] Davis Jini.org project, <http://developer.jini.org/exchange/projects/davis/>

[DCOM] Distributed Component Object Model,
<http://www.microsoft.com/com/tech/DCOM.asp>

[DHCP] Dynamic Host Configuration Protocol, <http://www.dhcp.org>

[DNET] .NET, <http://msdn.microsoft.com/net/>

[DSL] Digital Subscriber Line, <http://www.xdsl.com/content/backgroundinfo/overviews/>

[EIBK] EIB Konnex Association, <http://www.caba.org/standards/Konnex.html>

[EJB] Enterprise JavaBeans, <http://java.sun.com/products/ejb/>

[ETH] Ethernet, <http://www.ots.utexas.edu/ethernet/>

[FIRW] FireWire, http://developer.apple.com/hardware/FireWire/More_about_Firewire.html

[GENA] General Event Notification Architecture, <http://www.upnp.org/draft-cohen-gena-client-01.txt>

[GSPAC] Gatespace, <http://www.gatespace.com>

[HAVI] Home Audio Video Interoperability, <http://www.havi.org>

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

[HPLUG] HomePlug, <http://www.homeplug.com>

[HPNA] HomePNA, <http://www.homepna.org>

[HPOV] HP OpenView, <http://www.hp.com/ovw/>

[HRF] HomeRF, <http://www.homerf.org>

[HTML] HyperText Markup Language, <http://www.w3.org/MarkUp/>

[IBM] IBM, <http://www.ibm.com>

[IIOP] Internet Inter-ORB Protocol, <http://www.omg.org/news/whitepapers/iiop.htm>

[IRDA] Infrared Data Association, <http://www.irda.org>

[J2EE] Java 2 Platform Enterprise Edition, <http://java.sun.com/j2ee/>

[JADE] Jini Technology Application Demonstration Environment,
<http://developer.jini.org/exchange/projects/jade/>

[JAVA] Java, <http://www.sun.com/java/>

[JBoss] JBoss, <http://www.jboss.org>

[JC] Jini developers community forum, <http://www.jini.org>

[JDOM] JDOM, <http://www.jdom.org>

[JETS] HP JetSend, <http://www.jetSend.hp.com>

[JINI] Java Intelligent Network Infrastructure, <http://www.sun.com/jini/>

[JMX] Java Management eXtensions, <http://java.sun.com/products/JavaManagement/>

[JRMII] Java Remote Method Invocation, <http://java.sun.com/products/jdk/rmi/>

[JSP] JavaServer Pages, <http://java.sun.com/products/jsp/>

[JSPAC] JavaSpaces, <http://java.sun.com/products/javaspaces/>

[JX10] The Java X10 API 1.0.1, <http://www.jpeterson.com/rnd/x101.0.1/Readme.html>

[JZOAP] JBOSSZOAP project, <http://www.jboss.org/jboss-zoap.jsp>

[LWOR] Lonworks, <http://www.echelon.com/products/Core/default.htm>

[MPEG] Moving Picture Experts Group, <http://www.cselt.it/mpeg/>

[MS] Microsoft, <http://www.microsoft.com>

[OPPL] OpenPLANET, <http://www.openplanet.co.jp/Tour/html-e/e-beg1-1.htm>

[ORAC] Oracle, <http://www.oracle.com>

[OSGI] Open Services Gateway initiative, <http://www.osgi.org>

[SALU] Salutation, <http://www.salutation.org>

[SCP] Simple Control Protocol, <http://www.microsoft.com/HOMENET/scp/default.htm>

[SERUI] ServiceUI Jini.org project, <http://www.artima.com/jini/serviceui/index.html>

[SERV] Servlet, <http://java.sun.com/products/servlet/>

[SLP] Service Location Protocol, <ftp://ftp.isi.edu/in-notes/rfc2165.txt>

[SOAP] Simple Object Access Protocol, <http://www.w3.org/TR/SOAP/>

[SOAPUDDI] SOAPUDDI Jini.org project,
<http://developer.jini.org/exchange/projects/soapuddi/>

[SSDP] Simple Service Discovery Protocol,
http://www.upnp.org/download/draft_cai_ssdp_v1_03.txt

[SSL] Secure Sockets Layer,
<http://developer.netscape.com/tech/security/ssl/howitworks.html>

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

[SURA] Surrogate Architecture Jini.org project,
<http://developer.jini.org/exchange/projects/surrogate/>

[SWAV] Sharewave,
http://www.sharewave.com/HomePage/Home_Page_1/home_page_1.html

[TOMC] Tomcat, <http://jakarta.apache.org/tomcat/>

[TSPA] TSpaces, <http://www.almaden.ibm.com/cs/TSpaces/>

[UPNP] Universal Plug And Play forum, <http://www.upnp.org>

[V2] V2, <http://trace.wisc.edu/world/v2/>

[WAP] Wireless Application Protocol forum, <http://www.wapforum.org>

[WAPS] Openwave WAP phone simulator,
<http://developer.openwave.com/download/index.html>

[WETH] Wireless Ethernet, <http://www.wirelessethernet.org>

[WML] Wireless Markup Language, <http://www.wapforum.org/what/technical.htm>

[X10] X10, <http://www.x10.org>

[XML] eXtended Markup Language, <http://www.w3.org/XML/>

[XSL] eXtensible Stylesheet Language, <http://www.w3.org/Style/XSL/>