12

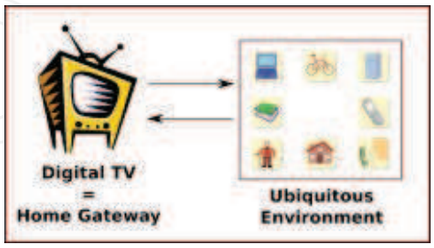
Using the iDTV as the Center of an Ubiquitous Environment

Orlewilson B. Maia, Nairon S. Viana and Vicente F. de Lucena Jr
Universidade Federal do Amazonas (UFAM)
Brazil

1. Introduction

The characteristics of the new interactive Digital Television (iDTV) systems are assuming a very important role in modern life. In fact, these features are being increasingly expanded, from simple signal decoders to sophisticated devices that allow the execution of interactive applications related to the content displayed, providing services of all kinds like Internet access, TV-Banking, TV-Mail, TV-Commerce, TV-Health, Games and so on. Moreover, recently, the main iDTV features have been used to integrate themselves with other consumer electronics (CE) devices in intelligent and ubiquitous environments. This new trend improves the user's experience with Digital Television and introduces several challenges for integrating heterogeneous devices, centralizing their services through a common iDTV set.
Nowadays one possible way of expanding the functionalities of a Set-Top Box (STB) is to connect it with other devices equipped with computational power such as some popular electronic equipments of a Home Network system. The firsts ideas were related to establishing a communication link between Digital TV and CE devices through one central processing unit of the Home Network, also known as Home Gateway (see Figure 1(a)). Another possibility is to integrate these two devices on a single platform, using the STB as a Home Gateway (see Figure 1(b)).

(a) Digital TV interacts with an ubiquitous environment through a Home Gateway



(b) Digital TV interacts with an ubiquitous environment directly

Fig. 1. Two ways for integrating iDTV in an Ubiquitous Computing Environment

Several scenarios can be provided by the use of Digital TV (DTV) receivers as Home Gateways. One possible example is the administration of a smart home through the iDTV, performing common tasks in the ubiquitous environment, such as the control of doors,

cameras, temperature sensors and security applications. In a more specific scenario the iDTV multimedia content can be synchronized with the television program. For example, the user is able to print a document linked to a TV show or to control the environment synchronized with audio/video transmission while a he/she is watching a movie.

Indeed, based on the tecnology available nowadays, other useful real life scenarios where the iDTV set is the information center in an ubiquitous environment can be created. For example, when Maria arrives at home, the TV recognizes her through her mobile phone and recommends a list of programs that she could watch based on her known profile. After that, she chooses a movie program. A few minutes later, Maria decides to eat something and moves to the kitchen carrying her mobile phone. At this moment, she is recognized by a TV set located in the kitchen that starts automatically to show the movie she was watching in the living room.

These scenarios bring to the market new possibilities of developing convergent applications, for instance in Brazil, where most people (over 97.1 %, according to a recent research from the Brazilian National Agency of Electrical Energy, ELETROBRAS, performed from 2005 to 2007 (ELETROBRAS, 2011)) use the TV quite frequently and are large consumers of services. Additionally, the iDTV can be used not only for providing conventional services (like TV-Mail or TV-Commerce), but also for accessing services of devices in a Home Network. This allows content sharing among iDTV and mobile phones, which is another potential field in the Brazilian market (ANATEL, 2010).

The goal of this study is to present some research results about the relationship between Digital TV and electronic devices in ubiquitous environments and to describe a collaboration model between Home Networks and iDTV Systems based on the Brazilian standard. We present a platform built over a common software environment for Digital TV and Home Network applications. With this platform it is possible to create several use cases scenarios, specially those related to composed services, i.e., those who integrate services from various devices to improve the user's experience with the connected ubiquitous home. The platform considers some issues of software environments for iDTV and Home Network, and tries to offer to the programmer an appropriate environment for rapid development of applications, in a level that isolates the main software components of the two environments. This study, from the overall technologies to the specification of the platform, will be described in the next sections.

2. Overview of digital TV and home networks

2.1 Digital TV

Ever since the first TV channel (BBC London in 1936), the TV went through several changes. The first most significant occurred in 1950 with the emergence of the color TV Sets and the growing of program options to the viewers (Schwalb, 2004). In the 1980s came the first digital production islands. A consequence of this was the transmission of digital content which were decoded in devices called Set-Top Boxes (STBs) in order to enable an analog TV to show the digital content.

According to Morris & Smith-Chaigneau (2005), the Digital TV (DTV) offers many advantages, such as the delivering of more programs in the same transmission band; the improvement of audio and video quality; the absence of interference at reception; and the emergence of interactive services and applications.

2.1.1 Interactive digital TV

Originally the TV does not provide interactivity to the viewers, i.e., the viewers watch the content of the programs and there is no interaction between them; the viewers only turn on/off the TV or switch the channel. Over the years, some television programs started using telephone lines to ask the viewers their opinion about the content being watched. With the advent of the DTV, services and interactive applications are created to allow the viewers to interact with the programs through an application broadcasted with the progams. This kind of interactivity allowed the emergence of the new interactive Digital TV (iDTV).

According to Schwalb (2004) and Morris & Smith-Chaigneau (2005), the iDTV can be classified as *Enhanced TV* with texts, graphic elements and improvement of audio and video quality; *Internet on TV* where the Internet can be accessed through the TV; *personalized TV* where the TV is adapted according to the viewer's profile; *Personal Video Recorder (PVR)* where the TV content is recorded from a genre, title or schedule; *Walled Garden*, a portal from which the user can browse for some desired iDTV application; *Games*, where the viewers use the TV to play a game or to connect with other players located connected in network; *Electronic Program Guide (EPG)*, similar to the Walled Garden, but it has more details about the channels; and *Teletext* which shows information (economics, wheater, last news and so on) provided by the broadcasters in a text format.

Another issue that enables the use of interactivity for viewers is the usage of the return channel. The return channel is used to send a request from a viewer through an iDTV application (such as ordering pizza or answering a question from a quiz) to the broadcaster over the Internet or a phone line, for example. Thus, the viewers can interact with the broacasters returning a feedbak about their content programs.

2.1.2 Components of an interactive digital TV system

An iDTV System is generaly composed by the following components: the broadcaster, the receiver, the broadcast network, and the communication link (see Figure 2). These components or subsystems deal with the digital information as services, audio/video/data elementary packets that compose a program channel. The services are phisically organized as a *Transport Stream* (TS) in which some operations are performed, such as coding and multiplexing defined by ISO-IEC 13818-x (ISO, 2000).

The broadcaster is represented by a TV channel or another entity able to perform the editing, formatting and distribution of content throught the broadcast network. This process involves steps of audio/video encoding (ISO-IEC 13818-1,2), data/application insertion (Digital Storage Media Command Control standard, ISO-IEC 13818-6) (ISO, 2002) and multiplexing (ISO-IEC 13818-2). In this stage, the data that will be broadcasted is in TS format (defined by MPEG-2 Systems). The last stage is the modulation to the broadcast network which will depend on the iDTV standard assumed. The broadcast could be done by three ways: cable, terrestrial or satellite.

When the digital signal is received, the STB performs the reverse process on the TS. The signal received is demodulated, demultiplexed and delivered to the audio/video/data decodificator which processes and displays the content in a TV screen. An important feature of the STB is the support for executing interactive applications which may or may not be related to the audio/video content and allow sending information back to the broadcasters. This feature is enabled in a iDTV System by the return channel.

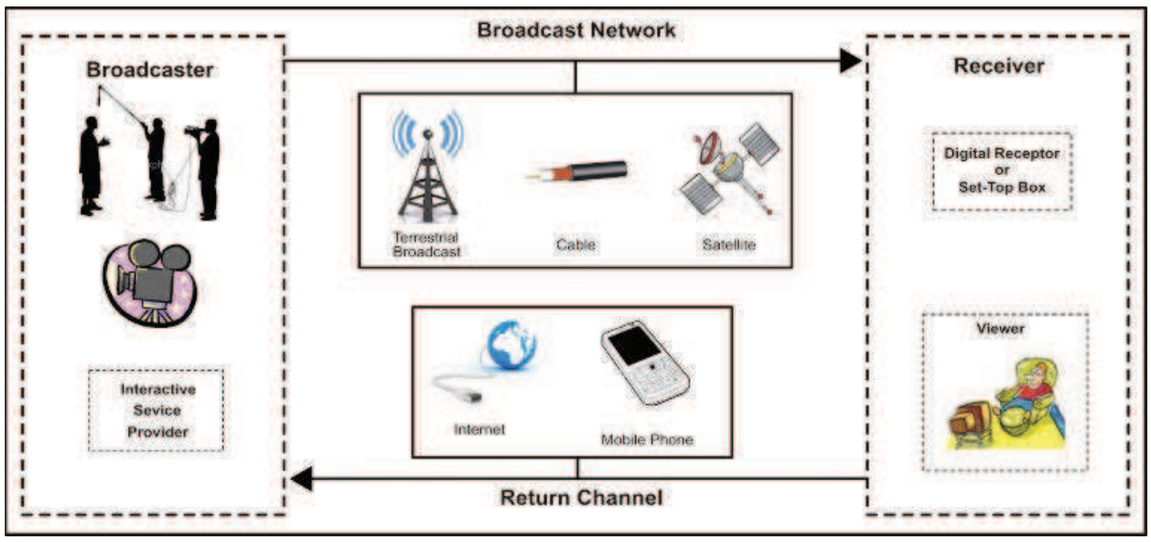


Fig. 2. General iDTV Components

2.1.3 Digital TV systems and middlewares

The main entities involved on iDTV business joined efforts to create specifications or reference models for iDTV in order to guarantee the interoperability among hardware manufactures, businesses and iDTV applications developers. Each specification has some particular caracteristic related to the regions they were developed. The main specification systems for iDTV are the American Advanced Television Systems Committee (ATSC)(ATSC, 2009), the European Digital Video Broadcast (DVB)(DVB, 2010), the Japanese Integrated Services Digital Broadcasting (ISDB)(ISDB, 2011), and more recently the Brazilian International Standard for Digital TV (ISDTV)(ISDTV, 2011).

In a generic architecture for the receiver, the main interface between the iDTV applications and the devices themselves is a component named *middleware* which consists of a set of *Application Programming Interfaces* (APIs) that provides a platform-independent execution environemt for iDTV applications. Thus, the middleware ensures portability between STBs and provides less effort for developing iDTV applications because the developers do not need to worry about how to access the low level receiver functionalities. The main existing standards for middlewares in the world are the European Multimedia Home Platform (MHP) (MHP, 2010); the American Advanced Common Application Platform (ACAP) (ACAP, 2009); the Japanese Association of Radio Industries and Businesses (ARIB) (ARIB, 2009); and the Brazilian Ginga (ITU-T, 2009).

The execution environment used in MHP is based on the Java Virtual Machine. A DVB application developed in this environment is called DVB–J. Moreover, the MHP 1.1 allows the usage of a declarative language similar to HTML called DVB–HTML. These types of applications have the capabilities of downloading and storing iDTV applications (in storage devices like USB Flash Drive or Hard Disc), access smart cards readers, and control iDTV applications over the Internet.

Similar to MHP, ACAP enables the usage of declarative languages as XHTML-based or ECMAScript. However, applications developed to this middleware are not complatible to MHP. To overcome this problem, a stable subset of API was extracted from MHP and defined as a standard common API, assuming interoperability among applications built in

any middleware. This interoperable platform is called *Globally Executable MHP (GEM)*. All the middleware specifications are supposed to have a GEM-compliance subset.
ARIB is composed by ARIB STD-B24 standard (Data Coding and Transmission Specification for Digital Broadcasting) which allows the development of declarative iDTV applications through the Broadcast Markup Language (BML).
Ginga is the middleware of the International Standard for Digital Television (ISDTV), the Brazilian Digital TV System. The specification is compliant with ITU J.200, J.201 and J.202 recommendations by providing an environment that supports declarative and procedural content, using an execution machine for Java-based applications (xlets) and a presentation machine for the treatment of declarative content based mainly on NCL (Nested Context Language) (Soares et al., 2007). In the Ginga software stack, a core structure (Ginga Common Core) is responsible for providing services that integrate the two environments. This part of the stack contains common content decoders that serve both the Ginga-NCL and the Ginga-J as well as implements a bridge mechanism to enable interaction between Java applications and NCL documents.
The first specifications released for Ginga-J defined the middleware as a set of 3 API's (named Green, Yellow and Blue), each one of them grouped some common functionalities for iDTV APIs (media management, life cicle control through JavaTV, return channel, and so on). This specification had a special feature of a device support API integrated to the middleware, which allowed sharing content among the iDTV and other devices. Currently, the Ginga-J specification has changed, and some of these APIs were replaced by the JavaDTV standard (ABNT NBR 15606-4, 2010), especially those related to GEM compliance. In spite of these modifications in Ginga-J, the basic functionalities of a Java-based iDTV programming environment were maintained, such as the xlet lifecycle model, media management, and the inter-xlet communication mechanism.
The Ginga-NCL provides a presentation engine for multimedia content based on scripts. The logical subsystem of Ginga-NCL handles documents in the pattern of the NCL language. NCL was designed for presentation and synchronization of multimedia content, based on NCM (Nested Context Model). Other types of content supported by Ginga-NCL are the ECMA Script, CSS and XHTML. Ginga-NCL also offers support for treatment of procedural content, through the implementation of the Lua API (Soares et al., 2007), allowing developers to construct audiovisual programs and to insert some sort of dynamic programming. The support for content (such as scripts, Lua, JPEG, PNG, and MPEG) is guaranteed by the Adapter, a component located on Ginga Common Core. Ginga-NCL also provides a device support mechanism, mainly related to sharing multimedia content among iDTV and mobile devices (Soares et al., 2009).
A summary of the procedural/declarative technologies available for the middleware specifications presented before is depicted in Table 1.

2.2 Home networks

The emergence and growth of the communication technologies enabled the interconnection of multiple devices located in a residence creating a smart home environment (Dixit & Prasad, 2008). This kind of home environment is characterized by allowing users to control, access and share information through these devices. Some applications can be cited, as for example, scheduling an air conditioner to turn on automatically when the environment temperature is above 25 celsius degrees.

Middleware	iDTV System	Declarative Environment	Procedural Environment
ACAP	ATSC (American)	ACAP-X [ATSC A-101 2005] Languages: XHTML like and ECMAScript	ACAP-J [ATSC A-101 2005] Language: Java
MHP	DVB (European)	DVB-HTML [ETSI TS 102 812 v 1.2.2 2006] Languages: XHTML like and ECMAScript	MHP [ETSI TS 102 812 v 1.2.2 2006] Language: Java
ARIB	ISDB (Japonese)	ARIB-BML [ARIB B-24 2004] Languages: BML(XHTML like) and ECMAScript	Optional (GEM [ETSI TS 102 819 v 1.3.1 2005]) Language: not implemented
Ginga	ISDTV (Brazilian)	Ginga-NCL [ABNT NBR 15606-2 2009] Languages: NCL and Lua	Ginga-J [ABNT NBR 15606-4 2010] Language: Java

Table 1. Declarative and procedural environments of iDTV middlewares

Furthermore, advanced types of systems may be constructed, in a near future such as those who have the ability to learn from user's everyday activities at home and, using artificial intelligence techniques, create an adaptable environment based on the user profiles. For example, when someone arrives at home, the environment recognizes his/her profile and sends a message to the stereo system which selects his/her favorite music track. Another point that have contributed to the creation of these kind of environment was the popularization of wireless communication which enabled the interconnection of several devices located in a home.

2.2.1 Open standards used in a home network

The Universal Plug-and-Play (UPnP) technology provides communication between electronic devices as well as the sharing of services, without the need of human intervention and configuration (Miller et al., 2001). The architecture has zero configuration support and the services discovery is automatic. A Device Control Protocol (DCP) was created for interconnection of electronic devices. Another important feature is that each manufacturer can create its own API and describe their services by an Extensible Markup Language (XML) file.

Jini, for example, is a middleware that offers APIs for the development of services, as well as network protocols allowing the data sharing among electronic devices with different types of communication technologies (Gupta et al., 2002). Each device supplies an interface to access available methods. Thus it ensures standardization and the sharing of services among them. This interface is represented by a Java object that implements its methods. The Remote Method Invocation (RMI) is used to access its methods through *invoke* and *lookup* commands.

The Home Audio Video Interoperability (HAVi) specification is used for sharing multimedia content between audio and video devices through FireWire (IEEE 1394) and their managment is centered on TV. To do so, a set of APIs and a middleware were created to implement functions, such as detect the devices in a Home Network, install the interfaces, and guarantee the interoperability between the devices (HAVi, 2001). The HAVi main features are that any

device can control other device, each device has a Java interface which is available to other devices, the updates are done via download, and it gives support for legacy systems and Plug-and-Play (Marshall, 2001).
One of the most known specifications for Home Networking is the Open Services Gateway initiative (OSGi). The framework supports applications from modular units known as bundles. In fact it controls the bundles life-cycle (adding, removing and replacing bundles at run-time) and verifies the dependencies between them (Tavares & Valente, 2008). Each bundle is represented as a service in the OSGi architecture. The services are registered in the OSGi Service Registry, which allows the discovery and access of services in an OSGi environment (Marples & Kriens, 2001).

2.2.2 Communications technologies used in a home network

In a home, there are different types of CE (TV, mobile phone, digital camera, and so on) that have a communication technology which allows to exchange information among them. This trend is followed by the growth in the usage of some communication technologies (Bluetooth, Universal Serial Bus (USB), Firewire etc). Among the most promising communication technologies commonly used in a Home Network, the two types above deserve a special attention: Wireless Networks and No New Wires Networks.
Wireless networks, such as IEEE 802.15 (Bluetooth), IEEE 802.15.4 (ZigBee) and IEEE 802.11x (WiFi), have grown significantly in recent years because of their convenience in easily connecting with new placed devices, the integration with several types of devices, and the gradually low cost of these solutions. They provide a transmission rate from 115Kbps (ZigBee) to 54Mbps (802.11g), measurable Quality of Service (Bluetooth and Zigbee), and a transmission range from 1m (Bluetooth) to 1.6 km (ZigBee). This kind of network is very useful in a home environment because of the typical distance between the CE devices, which is, in general, between 1m and 50m, its ability to communicate with more than one device at the same time, and, most importantly, because of the lack of cables that facilitates the replacement of devices.
The No New Wires Networks reuses the existent wired infrastructure of the house, such as the electrical wiring (HomePlug) and phone line (HomePNA) to establish a communication mechanism. As a matter of fact, the electrical wiring used is not considered an apropriate network media for data transmission because of its noise, the presence of interference, and fading. However, the emergence of new techniques in digital signal processing and code error control has allowed the increase of data rate transmission through the power lines and the reduction of noise (Lin et al., 2002). The main features of these networks are data rate transmission between 10 Mbps (HomePNA) and 14 Mbps (HomePlug), transmission of multimedia contents and connection of up to 50 devices in the same network. Nevertheless, there are some technical difficulties in building these networks, such as scalability of the number of connected devices in the network and security issues in data traffic (Zahariadis et al., 2002).

3. State of the art

The design of solutions incorporating electronic devices with some communication technology that promotes the exchange of information among them has been the topic of research in several recent works. These works diverge on their respective adopted approaches. Three different aspects of these approaches were analyzed: the way a device communicates

with others, the specification used to manage electronic devices, and the collaborative ways between iDTV and Home Gateway.

When talking about the diversity communication mechanisms, we may cite Kanma et al. (2003), which describes a scenario where a cellular phone communicates with electronic devices, such as an air conditioner and a washing machine, over Bluetooth. To do so, adapters with integrated Bluetooth technology through device's serial ports (RS 232) were developed. Some services were deployed, including the monitoring and remote controlling, updating of the adapter software, diagnosing of the failures, and getting technical information on the electronic devices. The need to create new adapters to each new device that joins to the network is a significant disadvantage.

The work described in Al Mehairi et al. (2007) used Short Message Service (SMS) to request different kinds of services from a Home Server that controls/monitors electronic devices through Bluetooth. To do so, in the SMS, the device and command names (turn on/off lamp, for example) were informed. In this approach, the Home Server receives the command and sends the information to the appropriate device. As a disadvantage of this work we may cite the fact that SMS is a payed service.

Considering the specification used for managing electronic devices, the work in Kim et al. (2007) used OSGi in a Residential Gateway to create a common ground for electronic devices from a wide range of communication technologies. In this work the electronic devices are managed remotely through a Web page available by the Residential Gateway. This proposal was focused on the use of limited resources, such as low data rate transmission and low energy consumption.

Another possibility may be illustrated by a scenario where UPnP and Jini specifications communicate with OSGi by offering a large scope to manage electronic devices (Dobrev et al., 2002). In this context, two components were created, a Jini Driver and an UPnP Base Driver. These components offer an access interface to registered services in OSGi environment. Thus, it is possible to create several applications, such as accessing a printer from Jini or a digital camera from UPnP. This way, when a device (a PC or cellular phone, for example) requests some service, such as printing a text or getting photos from a digital camera, the OSGi communicates with Jini requesting the printing of a text, or with UPnP requesting the content from a digital camera.

Some works related to the possible colaborative mechanims suggest the use of the TV-Sets as Residential Gateways mainly due to the emergence of the DTV and its potential for integrating other electronic devices in the Home Network. One of the first ideas was to create a protocol layer to control multimedia devices (audio and video) through HAVi specification (Marshall, 2001).

In Tkachenko et al. (2005) a framework named DTV-HNF (Digital TV - Home Network Framework) is described. This framework enables managing access requests from iDTV applications to available services from electronic devices on the Home Network. However, only iDTV applications access electronic devices services and there is a possibility of communication failure between iDTV and the Residential Gateway because of the technical nature of the communication used (TCP/IP).

To overcome these difficulties, the approaches in Cabrer et al. (2006), Bae et al. (2006), Yang et al. (2007), Redondo et al. (2007) and Lin et al. (2008; 2009) describe a collaboration mechanism between iDTV and Residential Gateway in the same environment, i.e., in the same device are included features from the iDTV and from the Residential Gateway. Other

characteristic of these works is the bidirectional communication, i.e. an iDTV appl ication can access a service from an electronic device and vice-versa.
In Cabrer et al. (2006) and Redondo et al. (2007) a new kind of application named XBundLET was created, which has features of MHP applications (xlet) and OSGi services (bundle). XBundLET is in compliance with xlets and bundles specifications: it is managed by an Application Manager and communicates with other xlets through Inter-Xlet Communication (IXC). At the same time provides services to other bundles and/or invokes services from other bundles through OSGi Service Registry.
A convergent architecture between data broadcasting and home networking services is proposed in Bae et al. (2006), based on the ACAP middleware and the UMB (Universal Middleware Bridge) protocol stack. This model has two sets of software components; a Service Proxy under the ACAP architecture and a Service Broker on the UMB stack. These components establish connections through the Simple Service Discovery Protocol (SSDP) enabling the discovery and use of network services for ACAP-J applications.
The proposal in Yang et al. (2007) implements a collaborative model that modifies as little as possible the characteristics of the MHP and OSGi native components. The work defines bridge structures between the two platforms that allow the passing of context parameters from MHP xlets to OSGi bundles and vice-versa. These structures exploit the Java Class Loader features, as well as implement a security mechanism to ensure that unauthorized access will not occur. Although, this model is simpler than the previous ones, its main disadvantages are that the developers of both xlets and bundles need to know the new components, besides the need to export bridge-components as Java Virtual Machine (JVM) static libraries outside the MHP and OSGi environments.
The approach of Lin et al. (2008) used components under the Wrapper and Broker design patterns, to isolate iDTV and Home Network software components. MHP xlets and OSGi bundles communicate with specific proxies for each platform, implemented within a Broker bridge component. This model has the MHP Application Manager as an OSGi service, through *MHPAPPmanager* bundle. Inside this bundle are the Broker component and the proxies. A significant disadvantage is the need to modify the MHP and OSGi components to support this new model, which just increases its complexity. The work of Lin et al. (2009) is an extension of the previous one. It presents a general classification of MHP-OSGi architectures, designing some of them in order to make a qualitative/quantitative comparison, considering issues of resources consumption, time for loading components, startup time, etc.

4. Integrating digital TV in an ubiquitous environment

The iDTV usage as a part of the ubiquitous environment is a topic of recent research. The collaboration strategies presented in Section 3 offer software platforms to build new application scenarios to the home user. One possible trend, analysed by Maia et al. (2009), is to extend a software platform for iDTV in order to increase the scope of these new applications. This way, it is possible to create from monitoring/controlling to more ubiquitous related scenarios, such as personalizing the home configuration to the user preferences.
To deliver ubiquitous services to the Brazilian iDTV, it is desired to create an integration mechanism between the Ginga middleware and some Residential Gateway specification (OSGi, Jini or UPnP) that can live together in only one CE device. In the literature, there is the work of Viana et al. (2009) which shows an introductory idea of such integration. Moreover, some features from the DTV middleware can be used, such as spatial-temporal synchronization between media (video, audio, text) and electronic devices.

In this section will be described one constructed component-based model for the Brazilian DTV middleware Ginga and the OSGi framework. The proposed model explores the features of the two Ginga environments (Ginga-J and Ginga-NCL) offering software components that interact with applications (xlets) or declarative content (NCL), exporting its features to the OSGi domain. The components and the two Ginga-OSGi approaches (GingaJ-OSGi and GingaNCL-OSGi) will be shown in details. Similarly, it should be possible for Ginga-J and Ginga-NCL applications accessing OSGi services to have interactive control over devices in Home Networks.

The general architecture of the model is shown in Figure 3. In this figure are Ginga-J, Ginga-NCL and OSGi components and the software bridge between them. Some features for the new Ginga-OSGi platform have been identified to meet the needs of integration between Home Networks and the Brazilian Digital TV. They are:

- ***Allowing the registry and the discovery of OSGi services in a Home Network*** – Services are the basis of the OSGi model. A service is the interface to operations provided by the networked devices. The service discovery in a Home Network is a key step for integrating it with iDTV applications.
- ***Interoperation between OSGi services and Ginga functionalities*** – The environment must allow Ginga to access OSGi services. That is a difficult task because it requires an effort to integrate these different software platforms. Similarly, this model should enable the access of Ginga functions by OSGi.
- ***Providing a useful description of the services to the user*** – Service description is the key to success in attracting a user. In addition to the information provided by OSGi services, such as name, version, and provider, other information should be released, such as device name and methods description.

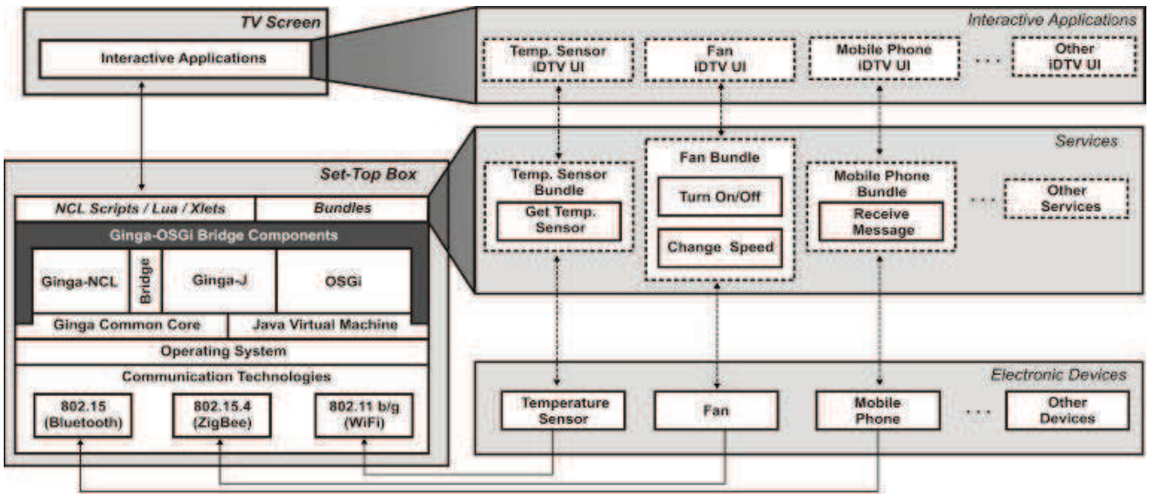


Fig. 3. General Ginga-OSGi Architecture.

The developed strategy considers mechanisms for collaboration between Ginga and OSGi. The two Ginga-OSGi approaches (GingaJ-OSGi and GingaNCL-OSGi) will be described in the next sections.

4.1 GingaJ-OSGi model

The procedural approach is in line with the desired requirements previously mentioned, and follows a model supported by components of the basic modules of Ginga-J and OSGi APIs. Therefore the model can be extended to other middleware like MHP. Indeed, it is based on the approaches of Cabrer et al. (2006), Yang et al. (2007), Lin et al. (2008) and Redondo et al. (2007), providing components that act as interfaces between the two frameworks. However, for this model, it was tried to change as little as possible both environments so that the xlets/services programmers do not need to know complex mechanisms of the platform.

The software layer of GingaJ-OSGi presents seven components that enable the flow of service objects between the two areas following one common pattern. The whole interaction process is focused in the registering of entities for Ginga-J features (IXCRegistry) and OSGi services (Service Registry). Once some Ginga-J functionality has been registered in the IXCRegistry, a component notifies the OSGi side through the registration of a corresponding service. Similarly, every time a bundle registers its service in OSGi, another entity will pass the service reference for the Ginga-J side. These entities act as listeners in both environments and the objects are registered through a generic interface following the Proxy and Decorator design patterns as described in Gamma et al. (1995). They protect the direct access to methods of the object, providing new methods for accessing the object with security, using features of the Java Reflection API. The seven developed components are:

- *GingaJ-OSGiRegister* – This component exports Ginga-J features as services in the OSGi domain. It implements a listener to registry events in the *IXCRegistry* (bind, unbind, rebind). *GingaJ-OSGiRegister* gets the Ginga-J and OSGi contexts through *ContextBridge*. Thus, it registers the Ginga-J functionality as an OSGi service using a *WrappedProxyObject*. This is done by accessing a service provided by the GingaJ-OSGiBundle component.
- *GingaJ-OSGiBundle* – This component provides a service in the Service Registry through which it is possible the registering of *WrappedProxyObject* objects relating to interfaces of the Ginga-J. This is necessary because since the *GingaJ-OSGiRegister* component is not a bundle, it is not authorized to directly register OSGi services.
- *OSGi-GingaJRegister* – Supports the reverse process, it implements a listener to registry events in the OSGi Service Registry and accesses the *ContextBridge* to export the OSGi service as a *WrappedProxyObject* related to the functionality to be registered on the Ginga-J IXCRegistry.
- *XletContextExporter* – This is an xlet that exports a singleton *XletContext* object (in textitjavax.tv.xlet package) to the *ContextBridge*.
- *BundleContextExporter* – It is a bundle that exports a singleton *BundleContext* object (in textitorg.osgi.framework package) to the *ContextBridge*.
- *ContextBridge* – This is a bridge component that allows accessing xlet/bundle contexts. This component receives context objects from *XletContextExporter* and *BundleContextExporter*, making these context objects available in the environment.
- *WrappedProxyObject* – That is a generic object that implements an interface (*WrappedProxyInterface*) which has methods for protecting the services that are being registered (into Service Registry or IXCRegistry).

These components, which are the basis of GingaJ-OSGi platform, operate with the aim of ensuring two interaction mechanisms between the platforms: OSGi services should be accessed by Ginga-J applications, and OSGi applications should access functionalities exported by Ginga-J. The two mechanisms are described in the following subsections.

4.1.1 Accessing OSGi services through Ginga-J applications

As ilustrated in Figure 4, this mechanism is supported by four components: *XletContextExporter*, *BundleContextExporter*, *ContextBridge* and *OSGi-GingaJRegister*.

Once the system is launched, the Application Manager defines a mechanism based on Java class loaders for activating *XletContextExporter* and *BundleContextExporter*. When these two objects are initialized, their contexts are passed to the Singleton object *ContextBridge* which remains active until the shutdown of the framework. At this point, the *OSGi-GingaJRegister* is waiting for notification of any OSGi service registration, through a ServiceListener object (in org.osgi.framework package). When a bundle registers its service in the OSGi (step 1), a reference is retrieved by the *OSGi-GingaJRegister* (step 2). The reference object (ServiceReference) provides methods for accessing some properties of the service through a query language expressed in Lightweight Directory Access Protocol (LDAP) (Howes, 1997). The *OSGi-GingaJRegister* has a key method that passes the service object to a *WrappedProxyObject* instance (step 3), which in turn is passed to IXCRegistry (step 4). This way, all access to the service object from Ginga-J applications is protected by the *WrappedProxyObject*, as outlined before. Thus the xlet, after retrieving the service object from IXCRegistry (step 5), needs only to know the name, the parameters and the types of the desired method in the service.

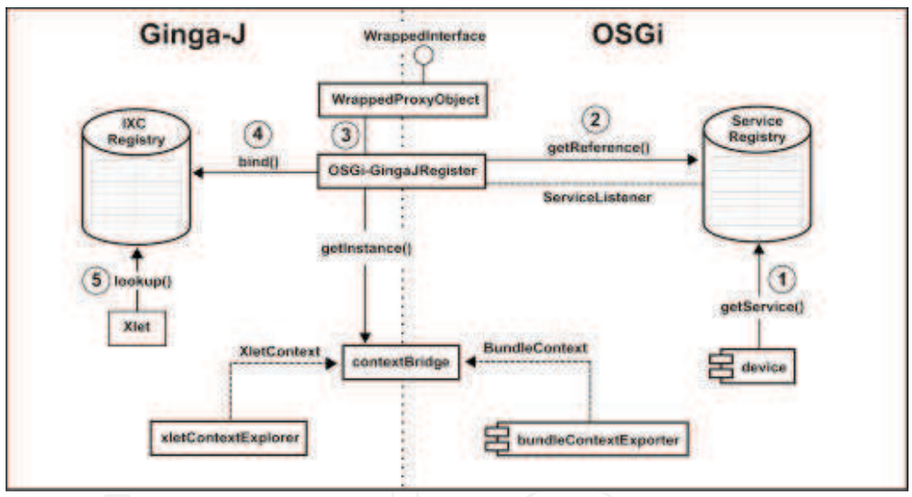


Fig. 4. Exporting OSGi services to Ginga-J domain

4.1.2 Accessing Ginga-J functionalities through OSGi bundles

To register xlet functionalities on the OSGi Service Registry, there are five components: *XletContextExporter*, *BundleContextExporter* and *ContextBridge*, *GingaJ-OSGiRegister*, and *GingaJ-OSGiBundle*(see Figure 5).

Once *XletContextExporter*, *BundleContextExporter* and *ContextBridge* are initialized, the xlet must register an interface in IXCRegistry (step 1). At this point a Singleton object, *GingaJ-OSGiRegister* is listening to events on the IXCRegistry (BIND, REBIND and UNBIND, which means that an interface has been registered, updated or removed, respectively) (step 2). From this point the process is similar to the previous one: the *GingaJ-OSGiRegister* activates a *WrappedProxyObject* to protect the methods of the object registered in IXCRegistry (step 3). After that, the *GingaJOSGiRegister* accesses a service from *GingaJ-OSGiBundle* to register the *WrappedProxyObject* as an OSGi service (step 4). A client bundle for this service needs to know

how to get a ServiceReference for the *WrappedProxyObject* (step 5). A bundle that needs to access a method of the interface registered in IXCRegistry must pass the name, types and parameters to *WrappedProxyObject* which will automatically invoke the method. In this way, the bundle is enabled for using a Ginga-J functionality without directly interacting with the middleware.

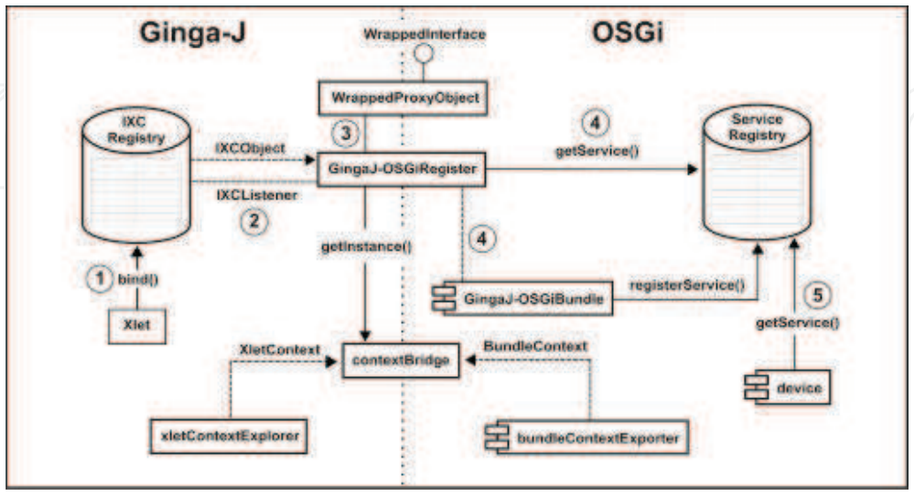


Fig. 5. Exporting Ginga-J funcionalities as OSGi services

4.2 GingaNCL-OSGi Model

The software layer created on the GingaNCL-OSGi platform has components that enable the passing of objects related to services between the OSGi and Ginga-NCL environments. Two Ginga-NCL Adapters are working as bridge components. An *Adapter* is a Java-based component that acts as a container for presenting the media in the interactive audiovisual NCL document. For each *Adapter* a specific region on the iDTV screen can be declared, having its properties defined by the *Descriptor*, that is associated with a *Player* component for playing the media. The six types of components presented in the GingaNCL-OSGi software layer are:

- *DeviceBundle* – This component represents a device located on the Home Network. The information contained in its Manifest file is used to describe it, such as the vendor, the interface name, and the service description. Moreover, a Property file is created to offer a useful description of its services, such as device name, and a description of the available methods.
- *ListenerServerBundle* – That is a component that listens to the registration of *DeviceBundle* services in the Service Registry and stores the information contained in the Manifest and Property files in the XML format.
- *Communication Bridge* – It is composed of two Java static classes (*BundleContextBridge* and *GingaNCLContextBridge*). These Java classes store a bundle context and a Ginga-NCL object, respectively, and enable the communication between Ginga-NCL applications and OSGi bundles.
- *ExporterBundle* – For a Ginga-NCL application to access some *DeviceBundle* services, it is necessary to get its reference on the Service Registry. To do so, this component exports its bundle context to *BundleContextBridge*. In this way, the Ginga-NCL application can use the *OSGiAdapter* for accessing the OSGi services.

- *GingaNCLBundle* – This bundle component registers a Ginga-NCL application as a service after getting the Ginga-NCL object stored in the *GingaNCLContextBridge*. After that, any *DeviceBundle* can access the Ginga-NCL services by retrieving its references on the Service Registry.
- *Extended Adapters and Players* – Two new Adapters were created, the *OSGiAdapter* and the *GingaNCLAdapter*. The first one is responsible for getting a bundle context stored in the *BundleContextBridge* and accessing the registered services by requesting to the *ListenerServerBundle*. After that, the *OSGiAdapter* accesses a *DeviceBundle* and uses its services. The last one exports a Ginga-NCL object to the *GingaNCLContextBridge* enabling the *GingaNCLBundle* to register it as a service on the Service Registry. Thus, a *DeviceBundle* can access this service to communicate with a Ginga-NCL application.

By using these components, the Ginga-NCL and OSGi characteristics were not modified, i.e. Ginga-NCL applications are accessed by OSGi bundles through a service that represents it in the OSGi environment and OSGi services are accessed by Ginga-NCL applications through their discovery in the Service Registry. The following subsections describe the proposed communication mechanism between Ginga-NCL applications and OSGi.

4.2.1 Accessing OSGi services by Ginga-NCL applications

In Figure 6 is shown how Ginga-NCL applications access OSGi services. The following components are presented: *ListenerServerBundle*, *DeviceBundle*, *ExporterBundle*, *BundleContextBridge*, and *OSGiAdapter*.

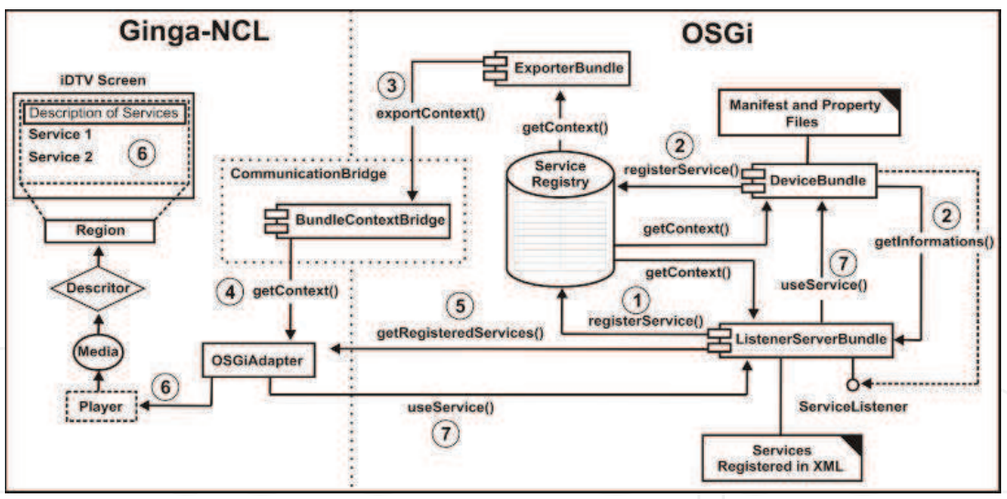


Fig. 6. Mechanism of exporting OSGi services to the Ginga-NCL domain.

When the system is launched, *ListenerServerBundle* registers its services in Service Registry (step 1). One of its functions is to provide a list of available services in the Service Registry in an XML format. To do this, it awaits the registration of new services on the Service Registry through a Listener. When a *DeviceBundle* registers its service, *ListenerServerBundle* gets some information contained on its Manifest and Properties files (step 2). Thus, this more detailed information helps the choosing of services by the users. To ensure that the services described in XML format by *ListenerServerBundle* are viewed by Ginga-NCL applications, another bundle named *ExporterBundle* is used, which exports its context to a bridge-structure named *BundleContextBridge* (step 3). With this, the *OSGiAdapter* gets a bundle context stored

on the *BundleContextBridge* (step 4). After that, *OSGiAdapter* uses this context to get the desired service from *ListenerServerBundle* by searching In the Service Registry (step 5) and displaying on the TV screen the available services (step 6). Finally, the *OSGiAdapter* informs to the ListenerServerBundle to invoke the service chosen by the viewer (step 7).

4.2.2 Accessing Ginga-NCL applications by OSGi bundles

Figure 7 shows how the OSGi bundles access Ginga-NCL applications. The following components are involved: *GingaNCLAdapter*, *GingaNCLBundle*, *DeviceBundle* and *GingaNCLContextBridge*.

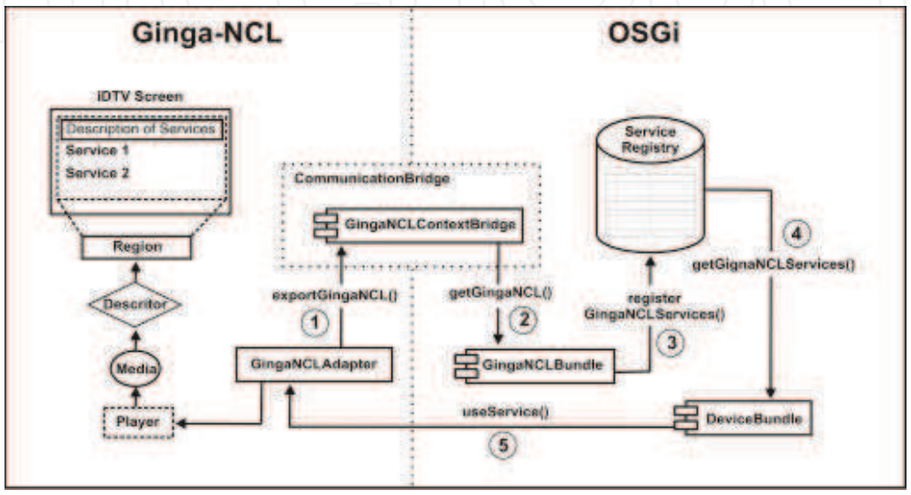


Fig. 7. Mechanism of exporting OSGi services to the Ginga-NCL domain.

When a *GingaNCLAdapter Player* executes a media object, *GingaNCLAdapter* exports a GingaNCL object to *GingaNCLContextBridge* (step 1). In this bridge-structure another Java static class named *GingaNCLContextBridge* is created, which stores the GingaNCL object. After that, *GingaNCLBundle* gets this object (step 2) and registers it in the Service Registry as a service (step 3). Finally, when a *DeviceBundle* needs to use this service, it is invoked after being searched in the Service Registry (step 4). In this way, the *DeviceBundle* sends an information to a specific region described on the NCL document through a correspondent GingaNCL object (step 5).

5. Implementation and experiments

The prototype was implemented in a PC platform with Intel Pentium 1.3GHz, 512MB RAM, 80GB HD, with WiFi, Serial RS-232 and USB interfaces. The operating systems was Windows XP SP2, with the Java 2 Platform Standard Edition 1.4.2_17 and 1.6.0_07. The PC simulates a single STB-HG (Set-Top Box Home Gateway) platform. The procedural part of the model used components of the XleTView 0.3.6 platform (XleTView, 2011), in which two main features were created: support for Inter-Xlet Communication through Java CDC Personal Profile RI (Personal Profile, 2006), and a model for IXC event listeners. The Ginga-NCL Emulator 1.1.1 (Ginga-NCL, 2011), a Java-based environment for creating interactive audiovisual documents based on NCL scripts, was chosen as a declarative middleware. The Knopflerfish 1.3.4 OSGi framework (Knopflerfish, 2011) was adopted for managing OSGi services. In order to test

the proposed platform, some case studies were built under the new Ginga-OSGi platform for procedural and declarative environments.

5.1 Building the procedural environment

Once registered in the OSGi, the services must be exported to the Ginga-J as xlet remote objects, (using the platform components in the *br.ufam.tvdihn.* package*). In the Ginga-J environment, the Application Manager uses system API's (*havi, javatv, cdc personal profile, nanoxml*) to control the xlet execution (using *xjavax.microedition.xlet.** components), which can register java.rmi.Remote objects in the IXCRegistry (using *com.sun.xlet.ixc.IXCRegistry* package). A listening mechanism ensures the exporting of ICXRegistry objects to the OSGi. It is important to say that the API's for graphical interface such as HAVi and DAVIC (*Digital Audio Video Council*) are not present in the Ginga-J middleware but they were used on this project because of the MHP nature of the Java iDTV emulated environment (XleTView). In spite of that, as they are graphical components, they do not interfere in the GingaJ-OSGi components.

Inside the *br.ufam.tvdihn.** package there is the bridge entity *ContextBridge*. This is a Singleton object, which means there is only one instance of *ContextBridge* in the entire system. Once the system is started, *ContextBridge*, *XletContextExporter* and *BundleContextExporter* are activated. *XletContextExporter* is an xlet which exists in the Ginga-J only while performing its function of exporting an xlet context and *BundleContextExporter* is a bundle installed in the OSGi. The *ContextBridge* initialization is done in the startXlet of *XletContextExporter* and start of *BundleContextExporter*.

The central components of the model, *WrappedProxyObject* and *WrappedInterface* are in the *br.edu.ufam.tvdihn.wrapped* package. The former stores information about a service object (an object to be exported between the domains), and the later is an interface that contais the operations that can be done on the service object. *WrappedProxyObject* protects direct access to the service objects it represents (an OSGi service or a Ginga-J remote object). Thus, *WrappedProxyObject* is an implementation of the Wrapped and Proxy design patterns, described in Gamma et al. (1995).

The *WrappedProxyObject* uses three additional components to store service object information: *java.lang.Object object* (related to the service object), *java.reflect.Method method* (an object that represents a method to be invoked in the service object) and *java.lang.object.Object methodObject* (the result of the invocation of *method*). These components work together in order to guarantee the protection of the service object.

The last group of common components is in the *br.ufam.tvdihn.listeners* package, which presents classes for creating a listening mechanism for events in the IXC. This is an important mechanism because it allows registering Ginga-J remote objects from the IXCRegistry to the OSGi Service Registry. The main classes of this group are: *IXCEvent*, which represents three types of events: BIND (when an xlet registers a remote object), UNBIND (when an xlet unregisters a remote object) and REBIND (when the remote object is updated); *IXCListener*, which in fact is a listener to IXC events; and *IXCListeners*, responsible for storing all existing *IXCListener* objects.

In the proccess of exporting OSGi services to the Ginga-J domain, there is the *OSGiGingaJ-Register* component which is represented by the entity *OSGiGingaJRegister* in the *br.edu.ufam.tvdihn* package.

To implement its function, the *OSGiGingaJRegister* has a listening mechanism for service events on Service Registry, using the *org.osgi.framework.ServiceListener* component. Once is

detected a service registration, the related object is passed to Ginga-J through *ContextBridge* and *WrappedProxyObject*, following the *WrappedProxyObject* mechanism, described earlier.
The *OSGiGingaJRegister* has three main functions: catch OSGi service objects, register these objects in the Ginga-J IXC and remove the objects from the IXC. The first function is carried out using a standard OSGi element, the *ServiceEvent*, which represent three types of service events in the Service Registry: *ServiceEvent*.REGISTERED (a service has been registered), *ServiceEvent*.UNREGISTERING (a service has been removed) and *ServiceEvent*.MODIFIED (the service properties were changed).
When the service object is retrieved, the OSGiGingaJRegister checks the following service properties through a LDAP query: SERVICE_ALIAS = "HOME_NETWORK_ALIAS" and SERVICE_TYPE = "HOME_NETWORK_SERVICE". If it do not match, the service is invalid and it will not be exported to the IXC Registry; otherwise, the *OSGiGingaJRegister* will execute the second function: to register the object in the IXC. To do so, the service object is initialized (using a bundle context from *ContextBridge*), passed to *WrappedProxyObject*, and then the IXCRegistry is invocated (using an xlet context from *ContextBridge*) through the *IXCRegistry.bind* operation. An alias for identifying the object is retrieved from the OSGi service, using the SERVICE_IXC_ALIAS property. This process is illustrated in Figure 8.

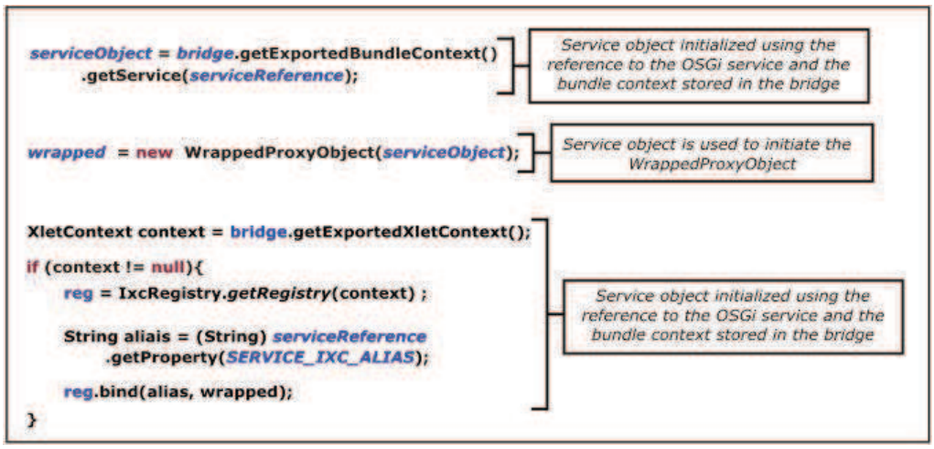


Fig. 8. *OSGiGingaJRegister*: code for registering an OSGi service as a Ginga-J remote object.

The last function of *OSGiGingaJRegister* is the reverse proccess: removing the IXC remote object from IXC Registry, when a service is uninstalled from the Service Registry. This is done by verifying the *ServiceEvent.UNREGISTERING* event, using the service properties to search for the corresponding object in IXC and then performing an *IXCRegistry.unbind* operation.
Regarding the process of registering remote objects from the IXCRegistry (in the *WrappedProxyObject* format) to the OSGi Service Registry, there are two components; *GingaJ-OSGiRegister* and *GingaJ-OSGiBundle*, both configured in a single bundle, *BundleGingaJOSGiRegister*, in the *br.edu.ufam.tvdihn* package.
The component *GingaJOSGiRegisterActivator* activates the *BundleGingaJOSGiRegister*. It implements a listener for events on the IXC Registry (through the *IXCListener* object in *br.ufam.tvdihn.listeners*). The main functions of *GingaJOSGiRegisterActivator* are: capturing the IXC remote object, registering and removing this object from the Service Registry. The registering/unregistering of service objects is preformed by the *GingaJOSGiRegisterActivator* in a different way of the *OSGiGingaJRegister*: in the OSGi domain there is only one service, *GingaJOSGiService*, which maintains an object repository, related to all IXC remote objects from

Ginga-J. The *GingaJOSGiService* can be accessed by bundles that need to use some functions of an IXC remote object. For each IXC remote object there is a corresponding *WrappedProxyObject* stored in the *GingaJOSGiService* repository.

5.2 Building the declarative environment

Similar to the procedural environment, the instances of Ginga-NCL and OSGi reference implementations were configured in a single JVM. The Ginga-NCL emulator source code was modified to define a specific ClassLoader for loading Ginga-NCL and Knopflerfish classes. In this way, the Ginga NCL Adapters can use OSGi code to access OSGi services. The same way, OSGi bundles can use functionalities of Ginga-NCL Adapters.

The two Ginga-OSGi adapters are Java classes built on Ginga-NCL middleware. As mentioned before, an Adapter is associated with media type and a Player object for this media. For each *Adapter* defined at *br.ginga.core.adapters.bridge* package, a corresponding Player which extends a *DefaultPlayerImplementation* object that represents *IPlayer* interface was created. The new *Adapter* must also extend a standard abstract class (*DefaultFormatterPlayerAdapter*) that has methods for managing the media presentation. The relationship between *Adapter* and *Player* objects is managed by *PlayerAdapterManager* object that works with two property-based standard files in the emulator: *controldefs.ini* and *mimedefs.ini*, both in *gingaNclConfigs.players* package. In the first file an alias for each *Adapter* class and its full name is defined. The second file defines the type of media for the *Adapter*, which can be played in Ginga-NCL Emulator. To show the media in the emulator, the node of the region and the corresponding media are configured in NCL file.

Figure 9 shows the description of the *GingaNCLAdapter* and the *OSGiAdapter* in the files mentioned above. In order to run an application the developer needs to create one region for each Adapter and define their presentation moment in the NCL file.

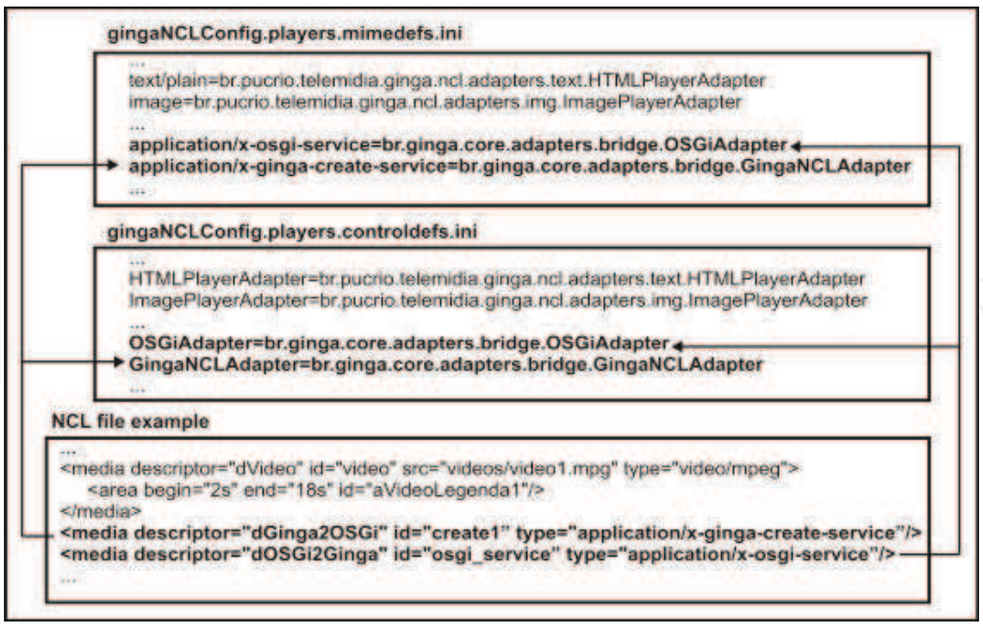


Fig. 9. Defining new Adapters in Ginga configuration files

5.3 Experiments

In order to clarify how the system works, two categories of examples are going to be explainded. The first contains one experiment based on procedural model (GingaJ-OSGi) and the second contains other experiment based on declarative model (GingaNCL-OSGi).

The first experiment allows bundles to use xlets functionalities, as described in Section 4. In this scenario we used an embedded microcontroller device (with WiFi interface) and a Webcam. When a user interacts with the device's keypad, it sends a request to a bundle server through the WiFi connection. The server bundle uses a Webcam service and takes a snapshot of the user attending to the request. The snapshot is passed to an iDTV area controlled by the xlet that exported its functionalities for the OSGi side as a service. At this moment, the iDTV user can use the remote control to send a reply message to the device. This situation simulates a home access control managed by the Digital TV. Figure 10 illustrates the final interface of the procedural application for the access control scenario.

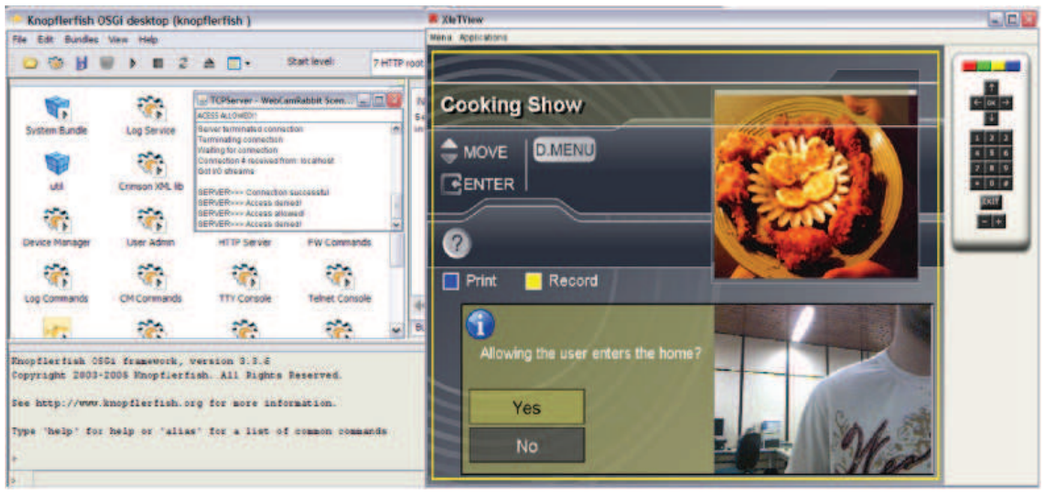


Fig. 10. The Procedural Access Control Scenario

In the declarative experiment, a *BluetoothServerBundle* application was developed which provides a Bluetooth service in the OSGi network. It provides a Bluetooth connection server for mobile phones by means of Java Bluetooth API. It registers an interface in OSGi Service Registry that provides methods to send a simple text message from a mobile phone to an iDTV with Bluetooth interface. When a region on the TV screen is created to allow the *GingaNCLAdapter* to receive messages, the *GingaNCLAdapter* exports a GingaNCL object to the *GingaNCLContextBridge*. After that, the *GingaNCLBundle* gets a GingaNCL object and registers it as a service in the Service Registry. From now on, when a cellular phone sends a message to the iDTV screen, the *BluetoothServerBundle* will get the Ginga-NCL service and communicate with the *GingaNCLAdapter* sending the message information. After that, the message is shown on the TV screen. This scenario is illustrated in Figure 11.

For the last one the *TempSensorBundle* was created. It manages an interface with a temperature device (MSP430-based platform) through a RS-232 interface by means of a Java-based API, through which the master device can monitor some others installed in a WiFi sensors network. In this bundle, there are the Manifest and Property files that have some information about its services, such as description of methods, device name, version and so on. The *TempSensorBundle* is started and registers its services on the Service Registry.

Fig. 11. The Declarative Mobile Scenario

After its registration, the *ListenerServerBundle* gets the information contained in Manifest and Property files of this bundle and saves into an XML format. After that, the *ExporterBundle* is activated. It exports its bundle context into the *BundleContextBridge*. In the user TV screen, the possible devices located in the house are shown. One of them is the Temperature Sensor. When the user selects this device, the *OSGiAdapter* is started and gets the bundle context located in *BundleContextBridge* to access the *ListenerServerBundle*. After that, the *OSGiAdapter* requests the service to *ListenerServerBundle* by searching the service on an XML document. After the desired service is found, *ListenerServerBundle* sends the method name to get the temperature sensor from *TempSensorBundle*. After a few seconds, is shown on the user TV screen the description and the methods available by *TempSensorBundle*. Finally, the user selects the desired service (Get Temp. Sensor Value, for example) and a few seconds later it is shown on the TV screen (see Figure 12).

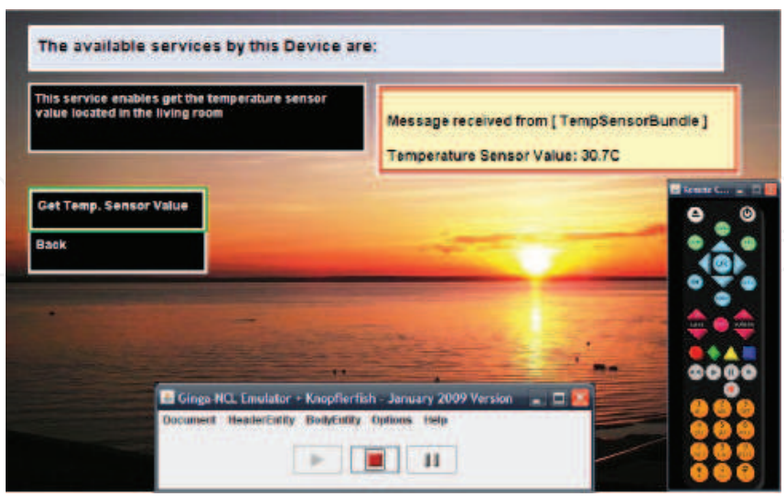


Fig. 12. The Declarative Temperature Monitoring Scenario

6. Conclusion and future scenarios

The expansion of the iDTV features and its recent use as a Home Gateway, makes the iDTV a key element for providing several kinds of services in the home networked environment buildind useful ubiquitous systems. Thus, this chapter aims to contribute to consolidate an iDTV-HN collaboration model, allowing the emergence of useful Home Automation applications and improving the user's experience and quality of life. The existing iDTV-HN convergence models were analyzed and a model that explores new features of the Brazilian iDTV middleware was proposed. This model can be easily exported to other iDTV standards. The main features obtained from this research are summarized in the key points described in the following:

- The fact of using two software platforms in a single environment allows the direct communication between the iDTV Ginga middleware and OSGi framework components. This is a useful advantage over some related proposals that use TCP/IP network.
- The xlet/bundle programmer does not need to know the new components: he/she can work with standard Ginga/OSGi components. In the case of the NCL programmer, he/she only needs to know the name of Adapters created for each service.
- The bridge-mechanism does not modify the Ginga/OSGi core components.
- In both models there is a difficulty in exporting iDTV functionalities (both for Ginga-J xlets and Ginga-NCL Adapters). It was necessary to use additional bundles to make this work (*GingaJ-OSGiBundle*, for example).
- In the case of the Ginga-NCL based model, the fact of managing the presentation of OSGi services just configuring some properties in a XML-based file, without any procedural code, is a useful advantage.
- In the case of Ginga-J-based model the components are less complex than the XBundLET of Cabrer et al. (2006); Redondo et al. (2007) and it is not necessary to allocate bridge components outside the GingaJ-OSGi environment, as Yang et al. (2007). In addition the Ginga-J Application Manager remains in its domain: it is not exported to OSGi, as in Lin et al. (2008).

In addition to the model based on the procedural specification (using Ginga-J middleware), the novelty of the work is in using declarative middleware features (through Ginga-NCL) to provide integration between script-based content and OSGi services. It considerably extends the scope of home automation applications that can be constructed using this new model. In the future the platform will be extended to support more kinds of ubiquitous scenarios, inserting components that have some intelligence to provide a dynamic adaptation of the application to the user's context. This model will allow to build user centered scenarios, such as those related to context aware systems and ambient intelligence.

7. Acknowledgements

The authors would like to thank the following funding institutions: CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior) and CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico) which provided support for the implementation of the work described in this chapter.

8. References

ABNT NBR 15606-4 (2010). Digital terrestrial television – Data coding and transmission specification for digital broadcasting Part 4: Ginga-J – The environment for the execution of procedural applications, Document ABNT NBR 15606-4, 2010.

ACAP (2009). ATSC Standard: Advanced Common Application Platform (ACAP), Document A/101A, 12 February 2009.

Al Mehairi, S., Barada, H. & Al Qutayri, M. (2007). Integration of Technologies for Smart Home Application, *Computer Systems and Applications, 2007. AICCSA '07. IEEE/ACS International Conference on*, pp. 241 –246.

ANATEL (2010). Brazilian National Agency of Telecommunications. Mobile users exceed 185 million subscribers in June 2010. Report on Personal Mobile Services, *Available at http://www.anatel.gov.br/Portal/exibirPortalNoticias.do?acao = carregaNoticia&codigo=20824. Accessed on January 10, 2011*.

ARIB B-24 (2009). ARIB Standard B-24 Data Coding and Transmission Specification for Digital Broadcasting, version 5.2-E1, 2009.

ATSC (2009). Digital Television Standard: Part 1 - Digital Television System, Document A/53 Part 1:2009, 2009.

Bae, Y.-S., Oh, B.-J., Moon, K.-D. & Kim, S.-W. (2006). Architecture for Interoperability of Services between an ACAP Receiver and Home Networked Devices, *Consumer Electronics, IEEE Transactions on* 52(1): 123–128.

Cabrer, M., Diaz Redondo, R., Vilas, A. & Pazos Arias, J. (2006). Controlling the smart home from TV, *Consumer Electronics, 2006. ICCE '06. 2006 Digest of Technical Papers. International Conference on*, pp. 255–256.

Dixit, S. & Prasad, R. (2008). *Technologies For Home Networking*, Wiley-Interscience, New Jersey.

Dobrev, P., Famolari, D., Kurzke, C. & Miller, B. (2002). Device and service discovery in home networks with OSGi, *Communications Magazine, IEEE* 40(8): 86–92.

DVB (2010). Multimedia Home Platform (MHP) Specification 1.2.2, ETSI Doc. No. TS 102 727 V1.1.1, 2010.

ELETROBRAS (2011). PROCEL. Brazilian Center for Information on Energy Efficiency. Research on Appliances Own and Energy Consuption Habbits. Summary from 2005-2007, *Available at http://www.eletrobras.com/pci/main.asp?View=05070313-120A-45FD-964D-5641D6083F80. Accessed on January 10, 2011*.

Gamma, E., Helm, R., Johnson, R. & Vlissides, J. (1995). *Design patterns: elements of reusable object-oriented software*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

ITU-T (2009). Consented Recommendation H.761. Nested Context Language (NCL) and Ginga-NCL for IPTV Services, 2009.

Ginga-NCL (2011). Ginga-NCL tools. Ginga-NCL Emulator, *in* Ginga-NCL (ed.), *Available at http://www.gingancl.org.br/ferramentas.html. Accessed on January 10, 2011*.

Gupta, R., Talwar, S. & Agrawal, D. (2002). Jini home networking: a step toward pervasive computing, *Computer* 35(8): 34 – 40.

HAVi (2001). The HAVi Specification: Specification of the Home Audio/Video Interoperability (HAVi) Architecture, version 1.1, 2001.

Howes, T. (1997). The String Representation of LDAP Search Filters, Document RFC 2254, December 1997.

ISDB (2011). Integrated Services Digital Broadcasting, *Available at http://www.dibeg.org. Accessed on January 10, 2011*.

ISDTV (2011). International Standard for Digital Television. Brazilian Digital TV System Forum (SBTVD), *Available at http://www.forumsbtvd.org.br. Accessed on January 10, 2011*.

ISO (2000). ISO/IEC 13818-1:2000. Information Technology - Generic Coding of Moving Pictures and Associated Audio Information.

ISO (2002). ISO/IEC 13818-6:1998/Cor2:2002. Information Technology - Generic Coding of Moving Pictures and Associated Audio Information: Part VI: Extensions for DSMCC.

Kanma, H., Wakabayashi, N., Kanazawa, R. & Ito, H. (2003). Home appliance control system over Bluetooth with a cellular phone, *Consumer Electronics, IEEE Transactions on* 49(4): 1049 – 1053.

Kim, K.-S., Park, C., Seo, K.-S., Chung, I.-Y. & Lee, J. (2007). ZigBee and The UPnP Expansion for Home Network Electrical Appliance Control on the Internet, *Advanced Communication Technology, The 9th International Conference on*, Vol. 3, pp. 1857 –1860.

Knopflerfish (2011). Open Source OSGi Service Platform, *in* Knopflerfish (ed.), *Available at http://www.knopflerfish.org. Accessed on January 10, 2011*.

Lin, C.-L., Wang, P.-C. & Hou, T.-W. (2008). A wrapper and broker model for collaboration between a set-top box and home service gateway, *Consumer Electronics, IEEE Transactions on* 54(3): 1123–1129.

Lin, C.-L., Wang, P.-C. & Hou, T.-W. (2009). Classification and Evaluation of Middleware Collaboration Architectures for Converging MHP and OSGi in a Smart Home, *Information Science and Engineering, Journal on* 25(1): 1337–1356.

Lin, Y.-J., Latchman, H., Lee, M. & Katar, S. (2002). A power line communication network infrastructure for the smart home, *Wireless Communications, IEEE* 9(6): 104 – 111.

Maia, O., Viana, N. & de Lucena, V. (2009). Using the iDTV for Managing Services in the Ubiquitous Computing Environment, *Ubiquitous, Autonomic and Trusted Computing, 2009. UIC-ATC '09. Symposia and Workshops on*, pp. 143 –148.

Marples, D. & Kriens, P. (2001). The Open Services Gateway Initiative: an introductory overview, *Communications Magazine, IEEE* 39(12): 110 –114.

Marshall, P. (2001). Home networking: a TV perspective, *Electronics Communication Engineering Journal* 13(5): 209 –212.

MHP (2010). Digital Video Broadcasting (DVB); Multimedia Home Platform (MHP) Specification 1.2.2, ETSI Doc. No. TS 102 727 V1.1.1, 2010.

Miller, B., Nixon, T., Tai, C. & Wood, M. (2001). Home networking with Universal Plug and Play, *Communications Magazine, IEEE* 39(12): 104 –109.

Morris, S. & Smith-Chaigneau, A. (2005). *Interactive TV Standards: A Guide to MHP, OCAP and JavaTV*, Focal Press, Burlington.

Personal Profile, R. I. (2006). JSR-000216 Personal Profile 1.1 (Final Release). Java Community Process, 2006.

Redondo, R. P. D., Vilas, A. F., Cabrer, M. R. & Arias, J. J. P. (2007). Exploiting OSGi capabilities from MHP applications, *Journal of Virtual Reality and Broadcasting* 4(16).

Schwalb, E. M. (2004). *iTV Handbook: Technologies and Standards*, Pearson Education, New Jersey.

Soares, L. F. G., Costa, R. M., Moreno, M. F. & Moreno, M. F. (2009). Multiple exhibition devices in DTV systems, *MM '09: Proceedings of the seventeen ACM international conference on Multimedia*, ACM, New York, NY, USA, pp. 281–290.

Soares, L. F. G., Rodrigues, E. F. & Moreno, M. F. (2007). Ginga-NCL: The Declarative Environment of the Brazilian Digital TV System, *in* B. C. Society (ed.), *Journal of the Brazilian Computer Society*, Vol. 13, Brazilian Computer Society, pp. 37–46.

Tavares, A. L. & Valente, M. T. (2008). A gentle introduction to OSGi, *SIGSOFT Softw. Eng. Notes* 33(5): 1–5.

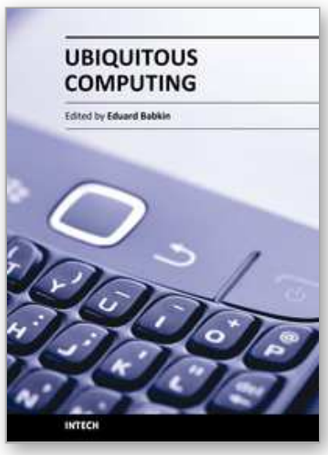
Tkachenko, D., Kornet, N., Dodson, A., Li, L. & Khandelwal, R. (2005). A framework supporting interaction of iDTV applications and CE devices in home network, *Consumer Communications and Networking Conference, 2005. CCNC. 2005 Second IEEE*, pp. 605 – 607.

Viana, N., Maia, O., de Lucena, V. & Pinto, L. (2009). A Convergence Proposal between the Brazilian Middleware for iDTV and Home Network Platforms, *Consumer Communications and Networking Conference, 2009. CCNC 2009. 6th IEEE*, pp. 1 –5.

XleTView (2011). MHP Emulator for viewing xlets on PC, *in* M. Svenden (ed.), *Available at http://www.xletview.org/. Accessed on January 10, 2011*.

Yang, M.-C., Sheng, N., Huang, B. & Tu, J. (2007). Collaboration of Set-Top Box and Residential Gateway Platforms, *Consumer Electronics, IEEE Transactions on* 53(3): 905–910.

Zahariadis, T., Pramataris, K. & Zervos, N. (2002). A comparison of competing broadband in-home technologies, *Electronics Communication Engineering Journal* 14(4): 133 – 142.



Ubiquitous Computing

Edited by Prof. Eduard Babkin

ISBN 978-953-307-409-2
Hard cover, 248 pages
Publisher InTech
Published online 10, February, 2011
Published in print edition February, 2011

The aim of this book is to give a treatment of the actively developed domain of Ubiquitous computing. Originally proposed by Mark D. Weiser, the concept of Ubiquitous computing enables a real-time global sensing, context-aware informational retrieval, multi-modal interaction with the user and enhanced visualization capabilities. In effect, Ubiquitous computing environments give extremely new and futuristic abilities to look at and interact with our habitat at any time and from anywhere. In that domain, researchers are confronted with many foundational, technological and engineering issues which were not known before. Detailed cross-disciplinary coverage of these issues is really needed today for further progress and widening of application range. This book collects twelve original works of researchers from eleven countries, which are clustered into four sections: Foundations, Security and Privacy, Integration and Middleware, Practical Applications.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen