We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

6,900
Open access books available

185,000
International authors and editors

200M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities



WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com



1

Integrating Ambient Intelligence Technologies Using an Architectural Approach

A. Paz-Lopez, G. Varela, S. Vazquez-Rodriguez, J. A. Becerra and R. J. Duro
Grupo Integrado de Ingeniería, Universidad de A Coruña
Spain

1. Introduction

It seems difficult to agree on a concise and accurate definition of a concept as broad as Ambient Intelligence. In short, it could be said that in such a context people interact with each other and with the environment and are assisted in their tasks in a certain smart way. The term Intelligence concerns the response that users expect from the system in terms of what Artificial Intelligence stands for, that is, proactivity, predictability and adaptability in its behaviours. This behaviour collection constitutes the functionalities that characterize an AmI system. On the other hand, the term Ambient is related to human factors and conducts and the ubiquitousness of a non invasive system. Indeed, the main goal should be user expectation fulfilment and the dynamic adaptation of the system to his/her needs, preferences and habits. In this context, the environment must be understood as an extended environment where a person works, has its family life, its recreational time and its social life. Therefore this environment involves the workplace, the home, the car or public places such as malls or sports centers, among others.

At this point, some questions arise: How should so vast and dynamic environment be managed? Where should the different software/hardware components be located? How do these components interact each other? How are new hardware/software components incorporated to an AmI system and how is it rearranged in order to integrate those elements? What are the channels through which information flows and who should supervise them? How could we take advantage of the numerous information sources/targets, like sensors, actuators or microprocessor appliances, and bus technologies available in a certain context? The idea that underlies all of these questions is the concept of an Architecture that should provide support and consistency to hardware/software AmI components and should allow AmI functionalities to be developed and integrated. This chapter deals with what an architecture conceptually involves and the description of possible approaches to the different issues related to it.

An architecture implies aspects concerning low-level hardware access, middleware and architectural software considerations. Therefore, it doesn't provide any AmI functionalities by itself but, on one hand, provides tools to system designers for the rapid development, deployment, reusability and interoperability of AmI solutions; On the other hand, it serves

as a technological wrapping that grants coherence and compatibility between AmI components contained in it by defining a specific model for each element.
At a first glance the approach of using the general architectural model would seem to be a good beginning in order to try to solve the problems indicated before. There is, however, no real consensus on this point in the literature. Many researchers just emphasize the development of AmI functionalities while hardware/software structural issues are pushed into the background. Hardware access matters and the complexity of physical network structures are often underestimated. Also information concerning AmI services cannot usually be shared with others and, currently, most AmI components are not aware of the existence of other components within the same system.
Despite this conundrum of approaches and solutions, it seems clear that the design of a global AmI architecture is necessary if good software/hardware engineering practices are to be achieved and the systems show some of the properties desired such as scalability, fault tolerance or ease of deployment and integration. It is in this line that some researchers have opened up the field to AMI architecture design. It is the case of the SOPRANO (Wolf et al., 2008; SOPRANO, 2007) and PERSONA (PERSONA, 2008) projects where platforms are based on OSGi and SOA (Service Oriented Architectures) were employed. From another point of view, a multi-agent based platform is proposed by AIT (Athens Information Technology) through the CHIL (Soldatos et al., 2007) project. The AMIGO project proposes a combination of both technologies, that is, SOA and agent based in (Vallee et al., 2005).
We will start this chapter by identifying the multiple topics that an architecture involves and the needs that come up when developing real solutions for them. As a result of this first analysis, we will define the conceptual model of an AmI architecture that is dealt with in next section. It is structured as a collection of subsections concerning the different model challenges. For each one of them an analysis of interest and needs, a review of the state of the art and a description of our approach as compared to others are presented. In short, structural software matters concerning the software platform, resource cooperation, knowledge representation and ubiquity are dealt with as well as user-system interaction, hardware virtualization and artificial intelligence functionalities. All the proposals that we describe here were developed in the scope of a global architectural project that we know as HI3 Architecture. After this, we will consider an example of using an HI3 system. The usefulness and the interoperability of the modules described will be shown with the aid of simple cases. Some conclusions and further avenues of research are outlined at the end of the chapter.

2. Needs and challenges

There are many requirements in the design and development of an AmI architecture. Some of them are common in distributed systems (with or without Artificial Intelligence), whereas others are very specific of AmI environments.
First of all, it is obvious that the whole system will be made up of many different elements: sensors, actuators, applications providing functionalities to users, intermediate services that provide common functionalities to applications, system level utilities for system control and logging, etc. All of these elements need to communicate to each other, consequently, communications are our first clear and important need. Due to the fact that some components, mainly sensors and actuators, will be physically distributed, special attention

will have to be paid to communications in order to guarantee reliability. Communications should also be optimized in two aspects; minimize resource consumption, as some components will have to run in devices with much reduced computational power, and minimize processing time, because some applications will need real time, or near real time, capabilities. Some sensors are susceptible of requiring high throughput, video cameras for example, so the system should also support such high level throughput. In addition, every AmI system should have the capability of growing, it is quite frustrating to try to expand an existing installation and find a limit in the number of elements or throughput (as is usually the case in current commercial systems). Thus, scalability is almost a must, and this brings together the ability to process a large volume of data in communications. The final characteristic related with communications is security. Information managed in an AmI system is usually sensitive, as it could be used to describe private aspects of its users. Therefore, communications should be secured through encryption and authentication. Security should also be included at a component level, in order to protect malicious replacement / injection of elements in the system, and at the user level, for obvious reasons.

Reliability is not only applicable to communications, it is a general requirement of any AmI system, due to their nature: AmI systems have to interact with human beings in a non-intrusive way, and it would be unacceptable for the user to have to restart an AmI system when something fails. That is, reliability is necessary to achieve high availability. In this context, redundancy is fundamental, although not sufficient, to obtain reliability. It means that if any software or hardware component fails, its functionality is immediately replaced by other components. Another characteristic that is necessary for reliability is fault tolerance. Eventually, everything can (and will) fail, consequently hardware and software must be designed to be able to recover when that happens. Related with this, some kind of hardware hot-swap is, obviously, very interesting.

From a software engineering point of view, all the elements should be designed bearing reusability in mind. The main reason is that an AmI system can, potentially, grow throughout its operational life, and new needs should be solved as soon as possible reusing components that are already available to the largest extent possible. In order to maximize the possibility of reusing components, system elements should be as modular as possible. Related to this and to high availability, it would be interesting to have also software hot-swap capabilities, so that a given element in the system can be replaced without interfering with the other elements. So as not to be tied to a specific hardware / O.S., something that is not desirable as it would reduce application scope, the software employed and programmed should be multi-platform, making hardware / software dependencies as low as possible in terms of software layers. Other related thing to take into account is that, if we really want hardware independence at sensor level, the AmI architecture should support different protocols of different hardware makers, trying to abstract these different protocols as soon as possible in the layer hierarchy. Finally, it is necessary to have some tools to control, manage and inspect the behaviour of an AmI system.

Human activity is complex, highly unpredictable and situation dependent. Unobtrusive applications within these intelligent spaces need information about the current state of the environment to dynamically adapt to different situations. Adaptation and unobtrusiveness are essential to social acceptability of AmI systems. These characteristics of human environments make it impossible to pre-program the appropriate behaviours for a context-aware service or application. Hence, an AmI architecture should include an extensible,

adaptable and efficient model for handling and sharing context information, providing a structured and integrated view of the environment in which the system runs.
These requirements and needs really condition the way an AmI architecture should be designed. They are obviously general principles that have to be taken into account from the beginning when considering the design of the main components and structure of the architecture. Many more needs and requirements will arise as we go deeper into de details of each component, and we will comment on them in the corresponding sections.

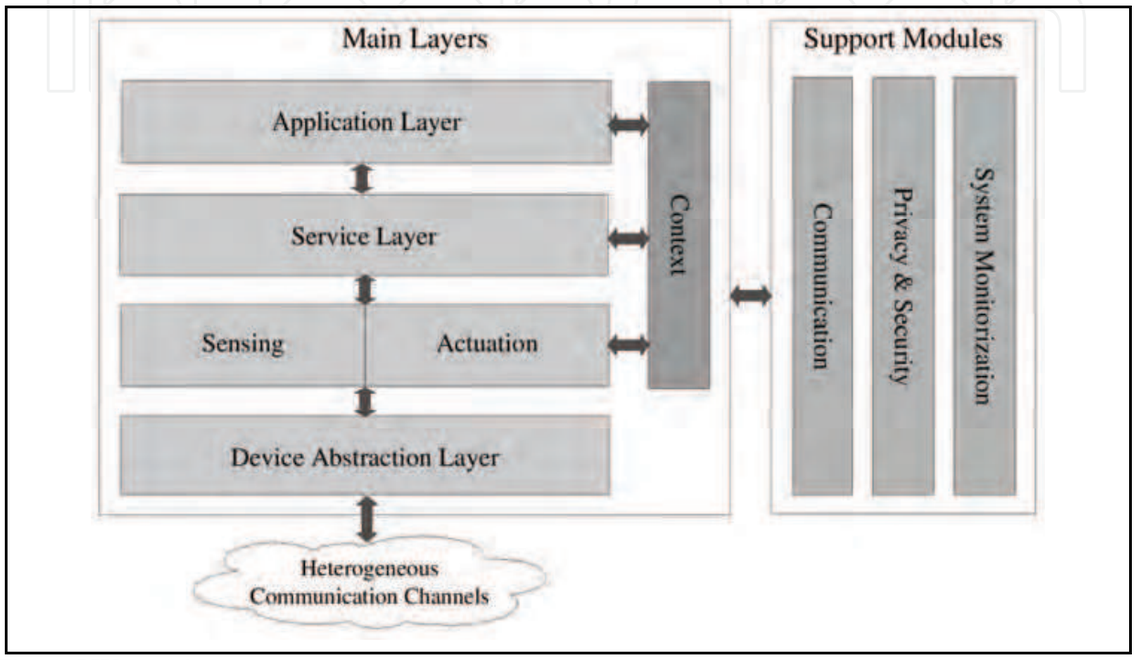


Fig. 1. HI3 architecture conceptual model

3. Architecting ambient intelligence

In the previous section we have identified a set of properties or key areas that we think an integral Ambient Intelligence software development platform should tackle in order to alleviate the problems associated to AmI software development. In this section we are going to address thewse diverse topics, providing a brief review of the most interesting approaches found in the literature to provide solutions to each of them.
Along with the detailed description of each key area and its state of the art review, we will present our version of an integral general purpose AmI system development architecture,. This allows us to provide a complete vision of the AmI software development field, going from the problems and requirements of the developers to the construction of a complete solution for them.
Before starting the discussion about the different key topics, and in order to facilitate the description of the different areas of the proposed solution, we will give here a brief overview of our approach. In the last few years we have been working on the development and application to different environments of a multi-layer intelligent agent based software architecture called HI3 (Varela et al., 2007; Paz et al., 2008, a; Paz et al., 2008, b). HI3 stands

for Humanized, Intelligent, Interactive and Integrated environments. The objective of this architecture is to provide the methodology, structure and base services to easily produce modular scalable systems in ambient intelligence settings that can operate without downgrading their performance as the system grows. The main idea behind our proposal is a layer based design that enables the division of system elements into levels, reducing the coupling between modules, facilitating abstraction as well as the distribution of responsibilities. A high level view of the design proposal is shown in Figure 1. A further description of this design will be presented in following sections and more information can be found in (Varela et al., 2007; Paz et al. 2008, b).
In order to implement this conceptual model, we have chosen a multiagent technology based approach supported by the JADE agent framework. Many of the characteristics of this paradigm fit perfectly with the main objectives for our architecture, such as autonomous operation, distribution and collaboration on task resolution, proactivity, intrinsic communications capabilities, mobility and adaptability.

3.1 Conceptual modeling

AmI systems are much more complicated than traditional computing systems. Hence, characteristics such as adaptability, flexibility, interoperability and modularity are more important. Furthermore, these systems must provide common improvements such as service discovery, self-organization, rich knowledge representations and context-awareness. To address these challenges, especially when seeking real AmI system interoperability, a general reference conceptual model and architecture is necessary. In recent years, a lot of research institutions and universities have been working on the definition and development of architectures and middleware that deal with the complex characteristics of AmI environments. However, only a small part of these projects try to formally define conceptual models that are not restricted to particular AmI domains or environments.
Some recent research efforts in the direction described above can be found, for instance, in (Rui et al., 2009) where the authors defend the need of general reference models in AmI and present a conceptual hierarchical model to describe a typical AmI application system. They propose a theory to describe AmI systems using a five-layer model: sensors and actuators layer, AmI network and middleware layer, devices layer, service layer and AmI applications layer. They do not include, as part of this work, high-level development tools that could closely guide developers in the process of building systems that fulfill the proposed conceptual model. Another example is the PERSONA project (PERSONA, 2009) which focuses on the specification of a reference architecture for ambient assisted living (AAL) spaces. The authors formally define a logical architecture that abstracts all functionality not leading to context, input, or output events as services.
Inspired by some of the ideas underlying these projects we developed the HI3 architecture, focusing on interoperability, scalability and ease of deployment.

HI3 Architecture conceptual model

As previously described, the HI3 architecture proposed here has a conceptual structure based on a multi-layer design schematized in Figure 1. This conceptual model establishes criteria and defines recommendations that facilitate developers the division of AmI systems into decoupled modules as well as the distribution responsibilities using homogeneous design principles and patterns. Communications between layers is vertical. Within layer,

horizontal communications are allowed thus permitting the cooperation between elements. The main layers defined in this model are:

- Device Access Layer: It includes elements that encapsulate concrete logic to access physical devices. It is divided into two sub-layers: device drivers and device access API.
- Sensing and Actuation layer: There are two types of elements in this layer: sensing elements and actuation elements. Sensing elements provide access to sensor type devices (using the device access API). Similarly, actuation elements must provide the required functionality to command actuator type devices. The elements of this layer are proxies of the physical devices.
- Service layer: A service is an element designed to solve a high level task that does not provide a complete solution for a system use case. Services are managed through a repository that enables the discovery of services required by others to perform their task.
- Application layer: This layer hosts the elements representing and implementing particular functionalities that a user expects from the system. These components make use of services registered in the system to carry out their tasks.

Furthermore, there is a component that is shared by the three higher level layers, the context. Its main goal is to define and manage the elements that must be observed in order to model the current state of the environment.

The previous high-level conceptual model provides a common vocabulary and guidelines to make the construction of AmI systems easier through technology abstraction and component composition and reusability. However, we believe that this model must be supported by middleware that abstracts developers from repetitive tasks and complex interactions with particular technologies. Specifically, we chose a multi-agent based approach to develop a middleware solution (fulfilling the hi3 architecture conceptual model requirements) that builds up a container for AmI applications providing advanced features such as high level inter-agent communications facilities, multi-agent models with support for the declarative definition of components, distributed management tools or development utilities (see section 3.2 for technical details).

This multi-agent based middleware promotes the division of large and complex AmI systems into highly decoupled components and, at the same time, provides what we call a multi-agent declarative model. The multi-agent declarative model is, on one hand, a model to describe a multi-agent system and its components and, on the other, it is a set of tools to declaratively define these components. As a model it brings together all the different elements that our platform leaves in the hands of the developers to build new applications. The top level concept of our platform is the multi-agent system (MAS). The platform is prepared to support several types of MAS with different high level characteristics but, in its basic form, they are a collection of instances of different types of agents that work together to perform a task. Directly below the MAS concept is the agent-type concept. An agent-type specifies the components that will define an agent in our platform, providing a way to declaratively build new agent-types by reusing pre-existing components and mixing them with new ones. Agent-types are defined according to a set of concepts that also have independent and declarative definitions: behaviours, message processors, publications, arguments for agents parameterization and descriptions of public functionalities.

Together, the high-level conceptual model and the multi-agent declarative model provide common guidelines and tools that control the entire process of developing interoperable AmI systems.

3.2 Middleware fundamentals

As previously stated, sharing a common vocabulary, concepts and abstractions among AmI developers is the first step towards more interoperable, reusable and easier to design systems. In the end it brings to the AmI field what UML or the object oriented programming paradigms brought to the programming field. Like paradigms such as OO, a conceptual model for AmI systems by itself does not produce a huge improvement on AmI development efforts. Developers still need to implement those designs that will be heavily based on complex distributed operations and on the integration of a huge number of techniques and technologies. In order to speed up the development of AmI systems, developers will need software, or more precisely, middleware, that supports the previously described models and provides them with tools and utilities that ease the development by allowing them to focus on the logic of the desired functionality, rather than on the interaction and integration of the different underlying technologies and devices.

As AmI is a relatively new field (it started to capture attention on the last ten years) until recently, the majority of projects had their focus set on achieving functionalities for very concrete niches (some examples may be found in (Tapia et al., 2004; Brdiczka et al., 2009; Krumm et al., 2000; Bernardin et al. 2007), taking ad-hoc design and implementation decisions focused on the particular environment dramatically reducing the possibility of adapting those projects to other environments, objectives or technologies. In order to operate in a real life scenario, every one of those projects needed solutions for very similar problems, like distributed component deployment and communications, sensor/actuator hardware interaction or fault tolerance. To put an end to that process of continuously reinventing the wheel, much in the same way as had previously happened before in other areas like multi-agent systems, numerous projects have been started with the aim of producing general purpose middleware that make the development of AmI systems easier.

From a very superficial point of view, it is possible to classify them according to their objectives. On one hand there are projects that are focused on delivering solutions to specific environments, they make decisions and adopt technologies that may make them less general purpose but easier to use in those environments. Prominent examples are projects like SOPRANO (Wolf et al., 2008), focused on elderly care scenarios, (and which relies on abstractions from that environment for component definition and on an ontology based semantic system for component interaction). The already mentioned PERSONA project devoted to increasing the independence of old people or the iRoom (MIT Oxygen, 2009), centred on the work environment. On the other hand we may find projects with a general purpose in mind. These projects usually devote great efforts to technology interoperability aspects and to providing high level abstractions and communications capabilities suitable for the typical requirements of AmI systems. Examples are the AMIGO project (AMIGO, b, 2009), although mostly focused on the home entertainment environment it provides a very versatile architecture, the EMBASSI project (EMBASSI, 2009), that tries to provide the needed technological infrastructure for the development of AmI systems in environments like the home, automotive vehicles and kiosks. The works carried out by Chen Rui et. al (Rui

et al., 2009), towards the definition and implementation of a general purpose AmI framework and our own approach, the HI3 project (Varela et. Al, 2007; Paz et al., 2008, b).
Before starting the discussion on the different technical approaches to the development of ambient intelligence middleware, it is important to take a look at the objectives of AmI in order to establish a series of prerequisites or key areas that a middleware for AmI development should deal with. One key differentiation factor of AmI with respect to general software development is that AmI systems are intended for operation in the real world, interacting with the environment and its inhabitants, this fact alone brings in some strong requirements such as:

- Multi-platform compatibility and interoperability.
- Modularity.
- Distributed operation.
- Dynamic management of components.
- System management.

Multi-platform compatibility and interoperability
AmI software needs to support its deployment on very different software and hardware platforms, from PCs to mobile phones, going through home appliances, automotive vehicles or consumer electronic devices. Because of this almost all the projects having to do with the development of AmI middleware are implementing them with technologies that inherently support multiple platforms, like interpreted programming languages such as Microsoft .Net or SUN Java and XML based mechanisms for communications.

Modularity
Most authors agree that AmI systems need to be highly modular in order to model and interact with a highly dynamic and complex environment like the real world. They need to contemplate and interoperate with a large number of devices and technologies to sense, reason and act on the environment. A design based on loosely coupled components is essential to accommodate that disparity in a sustainable and scalable system. It also forms the basis for component reusability, a key capacity to make more extensible and adaptable systems.
Modularity is a key aspect of AmI middleware, this can be seen in the fact that the projects we cited before can be divided by the techniques they apply to define components. Two main approaches can be distinguished. Projects that rely on Service Oriented Architecture models (SOA), and projects that use a multi-agent approach.
In the last decade, SOA has been widely adopted in the enterprise development field as a good approach for obtaining more sustainable and scalable systems. The SOA approach promotes a system design based on services (components) that define an interface that other components can use to access the functionality of the services without knowing anything about the service´s internal workings. The key design principle of SOA is that the services must be highly decoupled and reusable so that systems can be built by the aggregation or orchestration of multiple services. Numerous AmI projects have adopted SOA as their basis for system design and modularization, mainly due to the fact that it provides a good solution for modularization. Some examples of those projects are SOPRANO (Wolf et al., 2008), which is based on the development of services that provide semantic contracts related to different levels of an elderly care scenario to specify its functionality and requirements to

other services. AMIGO (AMIGO, c, 2009), that make use of services to dynamically accommodate the different devices and technologies needed to autonomously operate in a home environment. The PERSONA project (PERSONA, 2009, a), on the other hand, distinguishes different types of components by using multiple event-based buses with subscriber and publisher components for direct input/output and context management, but relies on services for the abstraction of the high level functionalities.

The other approach for system modularization is the multi-agent paradigm. It is based on the division of the system into multiple autonomous components, called agents, that collaborate to resolve a functionality. The key characteristic is agent autonomy, as it implies that the agents must present other characteristics that make them very attractive to AmI [aitami][chen-rui], like loose coupling, reactivity, proactivity and robustness. To operate in an autonomous way, agents needs to react to changes in the environment, a good property for systems that will operate in such a highly dynamic environments as the real world. They also need to be proactive, because every one of them should pursue its own goals, taking initiative and interacting with others and the environment as needed, again a good characteristic for AmI systems that must adapt themselves to the environments and their inhabitants, learning from them in order to predict their needs. And finally agents must be robust to operate in an autonomous manner with very little human intervention. These characteristics have led to different AmI projects adopting the multi-agent paradigm as its technological base. Prominent examples are the work by Soldatos et. Al from the AIT (AIT, 2009) within the CHIL (CHILL, 2009) project (Soldatos et al., 2007), which presents a three tier architecture with a tier of agents for high level functionality that rely on the other two layers for sensing and signal processing, or the Intelligent Room (MIT Oxygen, 2009) project at MIT which uses agents through its Metaglue (Hanssens et al., 2002) and Hyperglue (Peters et al., 2003) systems to represent resources and their interactions. Chen Rui (Rui et al., 2009) from the Embedded Software and Systems Institute of the Beijin University of Technology present a general purpose AmI architecture with a multi-layer design and use multi-agent systems to implement it. In our own approach, the HI3 project, we have also proposed a multi-layer design (Paz et al., 2008, b), and we use multi-agent technology to implement the different elements that populate each layer.

Distributed operation

Highly related to the modularization problem and the nature of the environment in which AmI systems must operate, the components of an AmI system should operate in a distributed fashion. They need to rely on multiple heterogeneous devices to sense and act on the environment, and more importantly, they need to provide its functionality in an ubiquitous way, and thus need to operate multiple devices at multiple locations. This increases the importance of the communications capabilities that a middleware for AmI provides to its components.

With respect to the projects that use SOA, like SOPRANO or AMIGO, almost every one of them employs OSGi (OSGi, 2007), an specification for a dynamic module system for Java that is a de facto standard for SOA based systems. There exist numerous implementations of the OSGi platform specification that provide tools for the development, deployment and management of services. Due to the nature of SOA, systems using this approach employ RPC (Remote Procedure Call) like communications almost exclusively. That is, for instance, the case of SOPRANO and AMIGO. With respect to the technical solutions for RPC

communications within OSGi, the initial versions of the OSGi specifications did not contemplate distributed operation, so numerous projects started to provide that functionality, the most widely distributed one is R-OSGi. The last revision of OSGi, release 4, version 4.1 (OSGi, 2007), contemplates support for the uPnP protocol and for the publication of services through the HTTP protocol. With respect to multi-agent based systems, they normally rely on ad-hoc message interchange for component communication and on FIPA (FIPA, 2009) compatibility for interoperability with other systems.

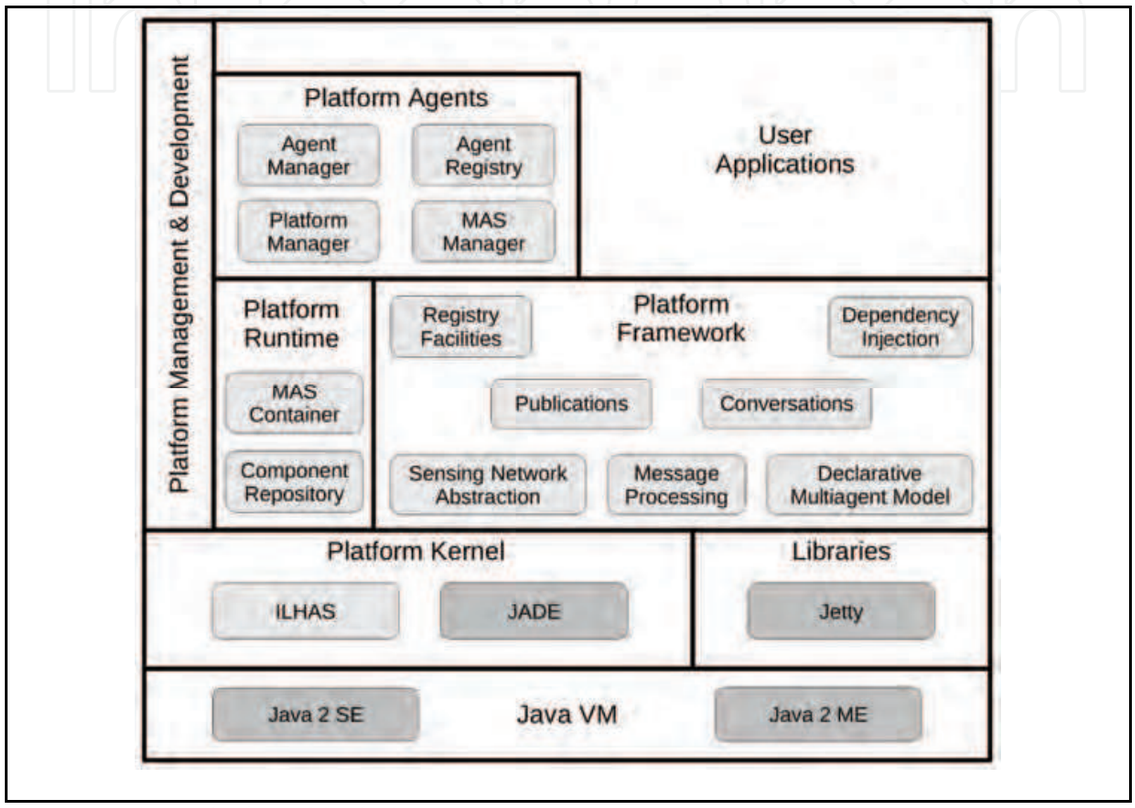


Fig. 2. HI3 AmI platform block diagram

We think that RPC like and simple message passing communications are not enough to support the complicated interactions between components that will be necessary to operate on an environment as complex and dynamic as the real world and to handle its relations with its inhabitants. Developers need more elaborate and high level communications tools to model and implement the interactions among its components. Event-based communications are a common solution for loosely coupled one-to-many communications, so they perfectly fit the requirements of AmI components. They have been used by several projects, like AMIGO, which considers an event based mechanism for context change management, PERSONA that introduces event buses as its primary communications, input/output and context management system. In HI3 we have also implemented an event based communications system for agents, providing them with the capability to publish information on publications that other agents can be subscribed to. Complex one-to-many, and many-to-many communications are also desirable in order to coordinate societies of

components to pursue a common goal. This is also a typical problem of multi-agent systems, and as a consequence FIPA provides some facilities for conversations, especially for the case of one-to-many. In HI3 we have developed an extensible system to manage complex many-to-many agent conversations, every HI3 agent can initiate and participate in conversations whose behaviour is controlled by different conversational models that contain the logic to drive a particular type of conversation. This include how to invite new members into a conversation, how to leave a conversation or how to notify members of message reception.

System management

System management is often overlooked in AmI systems or at least, there is not much information about the techniques used by the different projects. Some of the systems are intended for home use thus, they try to avoid the need for management. However, when intended to be used in large installations like hotels or hospitals, where there will be professional staff supporting the systems, system management capabilities are a must. The general approach is to rely on the management capabilities of the underlying component technology, like remote service deployment on OSGi platforms. One noteworthy exception is the case of AMIGO and its OSGi Deployment Framework (AMIGO, a, 2009) that is able to dynamically deploy services taking into account their semantic descriptions. With respect to the HI3 project, we have developed a declarative model and its associated tools that make the declarative definition of multi-agent systems (and its components, agents, behaviours, etc.) possible. It uses a self designed URL system for component resolution, thus providing loosely coupled component aggregation. Currently it only resolves components that are available within its own component container, but we are working to support remote dynamic deployment and instantiation of components. Along with this system, HI3 provides an AmI system execution container that integrates a lightweight embedded web server to provide a remotely accessible web interface for component life cycle management and deployment. It also provides support for the components to use that web server to provide web interfaces for management or even for web service compatibility.

To illustrate a complete proposal for base middleware in an AmI system, intended to alleviate the problems of AmI system development, Figure 2 displays a block diagram of the AmI system development and execution platform developed for HI3. The platform uses Java as its programming language and, as mentioned before, is based on multi-agent technologies. It uses the JADE (JADE, 2009) agent platform as its base, and on top of that provides all the functionalities that have been describe throughout this section, such as component definition and description, high level communications capabilities like conversations and event-based communications, component management and searching capabilities and hardware abstraction facilities.

3.3 Context-awareness

In this section, we review various existing context-aware systems focusing on context middleware and architectures, which facilitate the design and development of context-aware services and applications. We discuss important aspects in context-aware systems and present some ideas related to context management in the HI3 architecture.

Many researchers have debated definitions of context for many years without reaching a clear consensus (Dourish, P., 2001). In the literature the term context-aware was first introduced by Schilit and Theimer (Schilit and Theimer, 1994) in 1994. They define context

as "the location and identities of nearby people and objects and changes to those objects". Dey, in (Dey, 2001), reviews definitions of context and provides an operational definition of context as "any information that can be used to characterize a situation". This is the sense in which we understand the term context. The context defines the elements that must be observed in order to model the current state of the environment (situation). Additionally, context-aware applications must be able to dynamically adapt their behaviour to the current context without direct human intervention, in order to increase their usability and improve the end-user experience.

Dey and Abowd (Dey and Abowd, 2001) propose the division of context entities into three categories: places, people (individuals or groups) and things (physical objects). These entities can be described by attributes which can be classified into four categories: location, identity, status and time. Another popular way to classify context instances is the differentiation between the external and the internal dimension. The external dimension refers to context that can be sensed by hardware sensors, whereas the internal dimension is extracted from user interactions.

Related with the method used for context information acquisition, we can enumerate three different approaches (Chen et al., 2003):

- Direct sensor access. The applications gather the information directly from the sensors. This hinders system scalability, adaptability and distribution.
- Middleware infrastructure. This approach introduces a layered architecture that hides low-level sensing details. Hence, facilitates reusability and encapsulates the hardware dependent software.
- Context server. This approach extends the basic middleware based architecture by introducing distributed access to remote context data sources. We need to consider that probably the majority of devices used in ubiquitous environments are mobile devices operating over different network protocols.

Additionally, in order to manipulate and store context data in a computational form, we need to define a context model. There are different context modelling approaches that use different data structures for representing and exchanging contextual information: simple key-value models, object oriented models, logic based models or ontology based models. Ontologies are the most expressive models and are a very promising instrument for modelling contextual information (Strang & Linnhoff-Popien, 2004).

Focusing attention on AmI environments, there is a high level of consensus among researchers on the idea that it is essential to provide a framework for supporting context-awareness. One of the goals of a smart environment is that it supports and enhances the abilities of its occupants in executing tasks. In order to support the inhabitants, the smart environment must be able to detect the current state of the environment and determine what actions to take based on this context information. Therefore, applications in AmI environments need to be context-aware so that they can dynamically adapt themselves to different situations (Hung et al., 2003).

Related projects

Many ad-hoc context-aware systems designed to deal with predefined and restricted scenarios can be found in the literature. These systems may be optimized for this scenario but they aren't flexible or expandable. As a contrast, and following the global architectural

approach we promote in this chapter, we will focus on middleware or architecture based systems. Middleware should provide uniform abstractions and independence from hardware. It must simplify the development and maintenance of context-aware systems. It should also simplify the interaction between different context-aware applications by means of the use of common abstractions and tools. In what follows we will introduce various frameworks that meet some requirements such as architectural approach, historical data management, context model definition, sensing abstraction model or security and privacy.

An example of agent based architecture for supporting context-aware computing in AmI environments is the Context Broker Architecture (CoBrA) (Chen, 2004). The core of this middleware platform is the intelligent context broker that manages a shared contextual model making it accessible to a society of software agents. The context broker also includes components that provide a context knowledge base, components that encapsulate the context data acquisition process and components that grant private access to the information managed by the system.

The Service-Oriented Context-Aware Middleware (SOCAM) project (Gu et al., 2004) defines an architecture for the rapid prototyping of context-aware mobile services. It includes a server that collects and manages context data using distributed context providers. Services can make use of different levels of context data in order to perform their context-aware tasks.

Another middleware platform based on a layered architecture is developed as part of the Hydrogen project (Hofer, 2002). The main particularity of this framework is that it manages a local and a remote context. The local context only contains information provided by local devices, the remote context provides the information that other remote devices know. The devices share their context information when they are able to establish a communications. With this strategy, a device's context consists of its own local context and a set of remote contexts obtained from other devices. The architecture also includes a software layer that abstracts the access to contextual information from the client applications.

The Gaia project (Roman, 2002; Gaia, 2009) defines another framework that extends typical operating system concepts to include context, location awareness, mobile computing devices and actuators. The project is focused on building applications in a generic way without assumptions about the current resources of an active space. The platform must be able to transparently locate appropriate devices, detect the addition of new devices to the system, and adapt content when data formats are not compatible.

The PRIMA project (PRIMA, 2009) is about machine perception of people and their activities. This project includes the development of a lightweight middleware intended to help developers to create ubiquitous applications. The authors argue that context is a key concept in ubiquitous computing and propose a layered architectural model for services based on human activity (Emonet, 2006; Crowley, 2006). This model defines a conceptual and operational division of the system in four layers: the sensor-actuation layer that encapsulates the diversity of sensors and actuators through which the system interacts with the world; the perception and action layer that operate at a higher level of abstraction than sensors and actuators; the situation model layer that allows the system to focus perceptual attention and computing resources according to the current state of the environment; and, at the highest level of abstraction, the service layer.

Other important projects in the area of AmI, such as AMIGO (AMIGO, c, 2009) or PERSONA (PERSONA, b, 2008), develop context-awareness frameworks as an important

part of architectures that facilitates the development and integration of AmI applications. Both projects together with the PRIMA project share important design criteria, such as a layered approach with different levels of abstraction, use of ontologies to represent contextual information or the inclusion of components for user profiling and subscription/notification mechanisms.

Context model
We believe that an extensible, adaptable and efficient model for handling, sharing and storing context information is fundamental to providing effective and usable applications to a variety of users in scenarios with large variations in resources and activities. Context is too complex to be defined and programmed as a fixed set of variables. This demands the use of higher-level knowledge approaches together with reasoning and learning techniques.
The context model and context processing logic are the major components for providing intelligent, adaptable and proactive context-aware services and applications. As indicated before, ontologies provide an interesting formalism for specifying contextual information. Based on ontological models we can develop more sophisticated tools that provide a generic middleware platform to support context modelling in heterogeneous smart environments.
In the conceptual model of the HI3 architecture, we propose the division of AmI systems into four main layers (Figure 1). These layers represent different levels of abstraction and, as a consequence, the system can manage multiple views of the same contextual information in different layers and with different levels of abstraction. Hence, middleware must provide context aggregation capabilities that permit the composition of fine-grained context data in order to obtain higher-level context representations of the information. Middleware must also provide context interpretation capabilities to perform transformations of context data.

Awareness and notification
The awareness and notification capability must provide the functionality required to develop applications that can easily stay aware of any significant change in context. To achieve this, we propose a subscription/notification mechanism that allows services to register rules that specify what changes in context they should be notified of.

Historical information
Managing historical context data provides the possibility of implementing highly adaptable context-aware services. Furthermore, using learning algorithms, contextual information can be predicted to proactively adapt system behaviour to user habits. Related with historical data we can introduce in our architecture the concepts of granularity and period of validity. These properties are important to determine the relevance of contextual data to solve a certain task.

Information storage, privacy and security
We believe that it is necessary to use persistent storage systems to save coherent context data. This storage system must manage privacy policies and different levels of abstraction for data representation. Furthermore, is important that this component has facilities to query historical context data. Context usually includes sensitive information about people (location, activity, medical data, etc.). Hence, it is necessary to be able to protect privacy specifying policies and ownership of context information. We propose a transversal view of privacy and security components that operates in all the layers of the architecture. Thus, at

every level of abstraction, one must incorporate mechanisms to support privacy, trust and security.

User modelling and profiling
A major requirement of AmI systems is that they must be capable of learning from the use and habits of the users in order to predict and anticipate them. Moreover, these systems must exhibit proactive, adaptive, predictive and autonomous behaviour. These goals are achieved by constructing, maintaining and exploiting user models and profiles, which are explicit representations of individual user preferences. Hence, an ambitious AmI architecture and context-aware framework must provide components and structures that facilitate the development of applications that make use of user models and profiles.

3.4 Mobility
Ambient Intelligence systems should provide an ubiquitous experience to its users, in other words, as far as possible, they should offer their services and functionalities independently of user location. For example, if the user is in his office accessing a travel planning application to prepare his next work trip and suddenly the user needs to go out to a client's office, the application should transfers its execution to his mobile phone, maybe by changing to simpler algorithms for data processing due to the lack of resources of mobile devices, and obviously changing the user interface. As soon as the user is in his car, the system transfers the functionality to the car computer in order to increase resources to use the more demanding algorithms and again changing the user interface, for instance to voice commands. Examples of this desired behaviour can be found almost for every functionality that an AmI system could provide to its users, except those that are highly related to a specific hardware device or installation.
Achieving these types of mobility capabilities is not an easy goal and is yet in an early research phase. To provide such capabilities to its developers, AmI architectures and their supporting middleware must support classical techniques from the fault tolerance field, such as component duplication and component state synchronization. They should also provide new techniques that make use of the semantic descriptions and contextual information in order to correctly move and duplicate functionalities between devices and spaces, taking into account component requirements, such as memory footprint or specific devices needed for its operation, as well as context information, such us preferences or even abilities of the users in order to select the correct user interface.
Mobility has been a hot topic in multi-agent systems research because with mobility agents gain numerous interesting capabilities (Oshima & Lange, 1999). For this reason, projects that use multi-agent paradigms have a huge advantage on this particular aspect. In fact multi-agent platforms like JADE provide basic support for agent mobility. Agent based approaches can focus on providing high level features that use their semantic and context systems to achieve intelligent component mobility.
With respect to SOA based approaches (usually based on OSGi platforms), OSGi does not support integrated mobility or fault tolerance features so, currently, a lot of efforts are being devoted to the development of OSGi extensions to support fault tolerance (Ahn et al., 2006; Filipe & Torr, 2009) that could be used as the basis for mobility features by reusing its component replication capabilities. Some work on SOA services mobility is starting to be reported. An interesting paper in this line is (Preuveneers & Berbers, 2008). It provides a

good overview of mobility in pervasive systems and, in particular, it proposes solutions for mobility in OSGi based systems, also presenting an interesting implementation example.

3.5 Hardware abstraction

Ambient Intelligence systems are intended for operation in real environments, more specifically; real life social environments like homes, hospitals or hotels, inhabited by people, with the objective of making people's life easier and more comfortable. To achieve this, AmI systems must infer quality contextual information about the environment and its inhabitants. Complex algorithms and techniques are being proposed for this purpose but, in the end, all of them must rely on real data coming from the environment, and there is no other way to obtain those data than using sensing/actuation devices deployed in the environment or even on the people (wearable sensors, etc.).

There are many technologies competing for the domestic and social market, each one offering adequate solutions for its portion of the market. AmI software must be able to take advantage of these technologies but, the huge disparity of the different technologies and interfaces results in a series of problems:

- Difficulty of interoperating groups of devices of different natures.
- Portability of software components to different hardware platforms.
- Risk of translating the complexity of the hardware set-up to the software applications.
- Risks derived from the dependence on a given technology vendor.
- In real environments there may already be hardware elements present, such as classical domotics buses, which require their integration.

Solving these problems is not an easy goal, and the majority of AmI projects have mostly ignored them, adopting concrete technologies. This approach anchors the software solution to the selected hardware technologies, introducing unnecessary complexities on the software side and distracting developers from the core logic of its functionality. Furthermore, the software needs to be adapted to the hardware characteristics, loosing its generality and making its reuse on other project or environments difficult.

Almost all the projects working on integral architectural solutions for AmI have identified the hardware access as a key aspect. The good news is that all of them agree on the paradigm that should be applied to deal with this problem and are developing hardware abstraction layers that hide the complexities and differences of heterogeneous technologies. The main differences found between the approaches have to do with the design of those layers and the type of devices they are dealing with. On one hand there are projects that have focused on particular niches. For example, projects focused on home entertainment are mainly working on the interoperation of consumer electronic devices. This is the case of the INTERPLAY system (Messer et al., 2006), that tries to provide integration among home entertainment devices by using a three-layer architecture to support multiple device technologies and hide their heterogeneity.

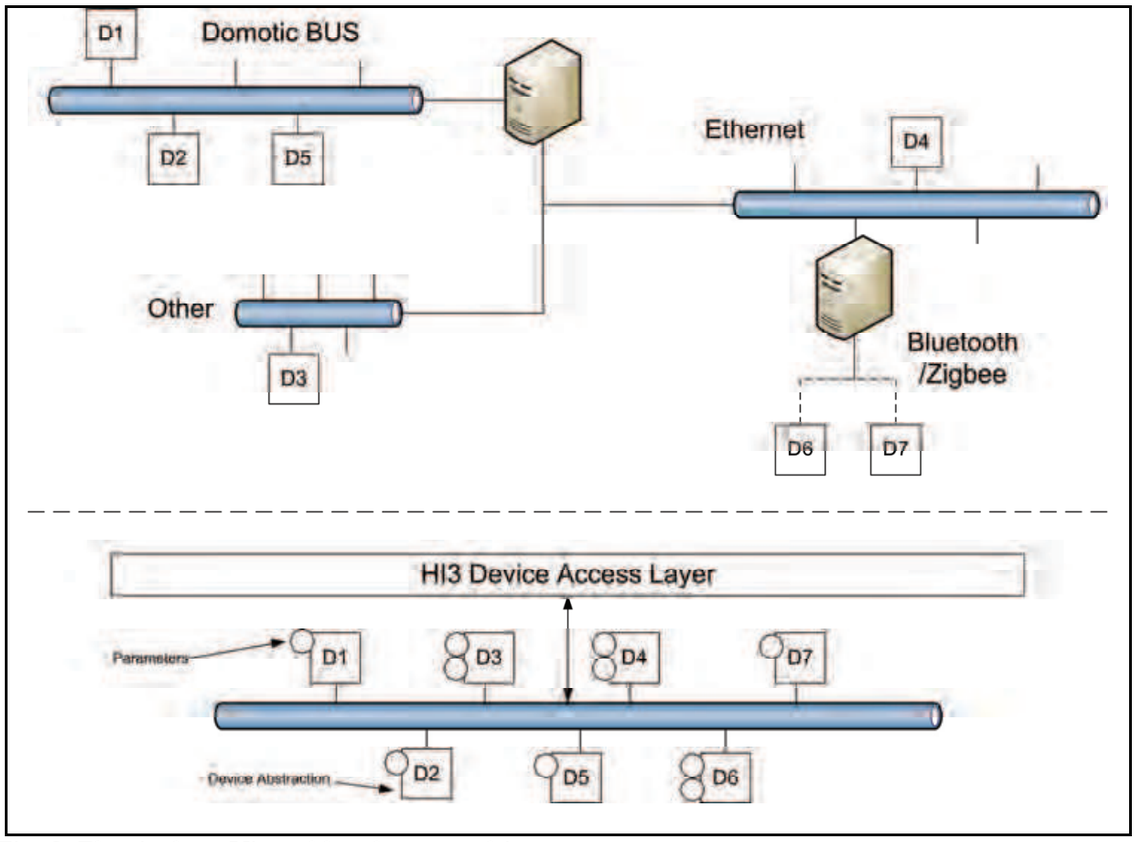


Fig. 3. Physical vs. Virtual hardware architecture

On the other hand there are projects that try to provide a general solution to the hardware integration problem. A prominent example is the AMIGO project (AMIGO, c, 2005) that has worked on the integration of heterogeneous technologies coming from domotics and consumer electronics. AMIGO relies on standardized solutions as much as possible, developing abstractions to hide the heterogeneity of the devices and translate their functionality to AMIGO compatible services. AMIGO proposes uPnP as a standard for hardware access so, when a non uPnP compatible device needs to bee used, AMIGO develops proxy services that use ad-hoc drivers to interact with the device and translate its interface. Proxies have been developed for two widely used domotic technologies, BDF and EIB. Chen Rui et al. (Rui et al., 2009) propose a similar solution to AMIGO, integrating technologies through the use of encapsulating proxies. The main difference is that they also provide a hardware solution to simplify the physical integration of technologies as well as the development of new devices.

From the beginning of HI3 we though that a solution for the integration of hardware technologies was the first step towards more reusable and maintainable AmI systems, so we started the development of the HI3 architecture from its hardware abstraction layers (Varela et al., 2007). Our approach is similar to the ones proposed by AMIGO and Chen Rui et al. It is based on the definition of a standardized paradigm for hardware access and the development of proxy-like devices to adapt current existing technologies.

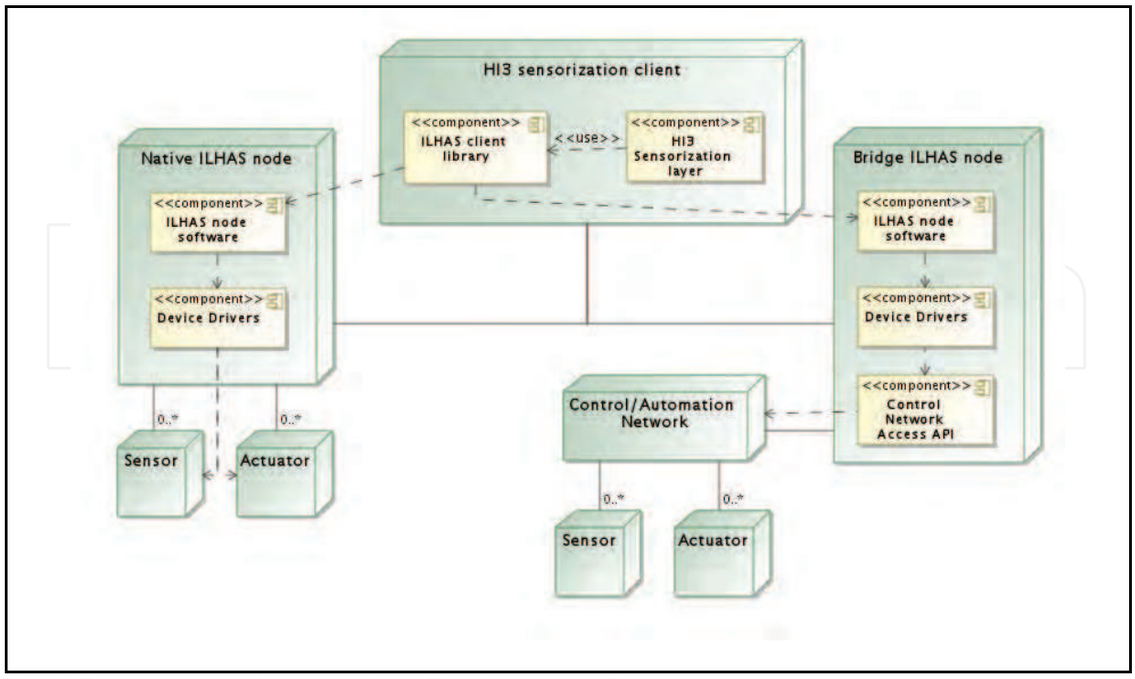


Fig. 4. Deployment diagram of the Device Access Layer components.

The HI3 project proposes a conceptual model based on a multilayer design with two layers in charge of abstracting hardware technologies. Going from the top down, the Sensing and Actuation layer is made up of agents that provide higher level views of hardware devices, for example aggregating them, applying some basic signal processing algorithms, etc. The Device Access Layer takes inspiration from the IEEE-1451 (NIST, 2009) standard (Varela et al., 2007). It is based on a distributed model inspired by the network and application layers of IEEE1451.1 standard and is supported by three basic pillars:

- Unified and generic view of the different hardware devices.
- Definition of a paradigm for the operating model of the sensing/actuation device network.
- Abstraction of the device network topology.

The first two abstractions provide functionality-based access to devices. They are accessed through read/write operations over parameters that represent the objects or states of the physical environment that a device can operate with. The third one is implemented as a network layer and a node abstraction that enables the integration of multiple and heterogeneous device networks.

This model is summarized in Figure 3 which displays the two visions of a complex scenario. On the top there is the physical vision of the installation, in which the software is exposed to the complexities of each different device. The bottom represents the homogeneous view the hardware abstraction subsystem provides.

The Device Access Layer is implemented through two components Figure 4, a software library to provide homogeneous and ubiquitous access to a network of devices (called ILHAS), and node software that can be used to integrate devices into that network. We have also developed a hardware prototype device (Varela et al., 2007) that integrates this node

software with COTS hardware, creating a good low cost solution for the development of new devices as well as hardware proxies for existing devices or technologies.

4. Applying HI3 technology

In the previous section of this chapter we have presented a review of different interesting approximations to provide Ambient Intelligence with models and supporting software to speed up the development of AmI systems. The HI3 architecture, a general purpose base architecture for AmI, has been presented along with these descriptions. In order to clearly illustrate how AmI support middleware, and more specifically the HI3 architecture, can help in the development of AmI software, we are going to show how it can be used to build a simple but interesting example that includes many of the typical requirements of AmI systems.

The example is a fairly typical one, a simple meeting management application. Similar examples have been used in numerous AmI articles to date as this type of application has functionalities that are easy to explain but presents complex underlying requirements like heterogeneous hardware integration, dynamic component collaboration, distributed operation or IA capabilities. It is an AmI application in charge of the management of one or more meeting rooms. It has to control all the devices involved in a meeting process, like the lights of the room, the projector display or the computer that shows the presentation. It also needs to autonomously know when a meeting can start by looking at who is in the room and checking if all the expected members have already arrived. In order to introduce some high level data processing techniques, we also want the application to autonomously extract user preferences about meeting room configuration.

In order to implement this example, we have at our laboratory a meeting room with all the required hardware devices:

- An EIB bus with light actuation devices, presence sensors and a collapsible projector display.
- A video projector with ethernet connectivity that is compatible with the PJLink protocol for remote control purposes.
- One computer with its video output connected to the video projector in order to display presentations.
- One computer with various bluetooth dongles that will provide a list of MAC addresses that are detected inside the room. Therefore, the users must simply carry a mobile device with bluetooth support in order to be located and identified.

Following the HI3 architecture model guidelines, those devices will be abstracted as ILHAS parameters that represents their device functionalities, like the state of the lights and operations to read/write it, the state of the presence sensor or a list of detected MAC addresses in the case of the bluetooth device that will be used to know who is in the room. The type and number of ILHAS parameters is standardized across devices that provide the same functionality, in other words, two light switching devices from different vendors, for example EIB and X10, will present the same parameters to the ILHAS subsystem. As the ILHAS library provides direct access to those parameters, homogeneous access to heterogeneous device technologies is effectively achieved.

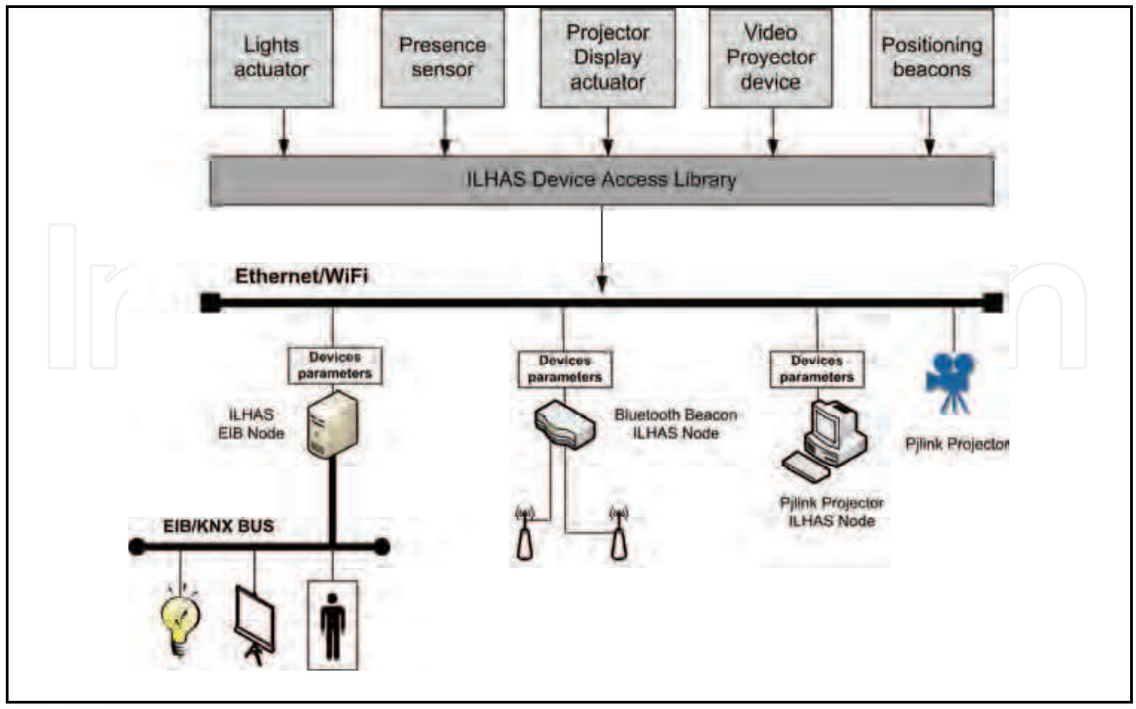


Fig. 5. Hardware deployment diagram

ILHAS provides also a homogeneous network topology on top of complex topologies that integrate multiple network technologies. As can be seen on Figure 5, there is at least one device adaptor for each technology that translates its concrete interaction protocols to the ILHAS parameter read/write paradigm. In order to use EIB devices, a bridge node has been implemented by using the ILHAS node software we talked about in the previous sections. It is deployed on a PC and connected to the EIB bus through a RS-232 interface using the Calimero EIBnet/IP tunneling library and the Tweety EIBnet/IP tunneling server from the TU Vienna (TU, 2009). With respect to the video projector, even though it has TCP/IP connectivity by using the PJLink protocol, a bridge node was also developed. This bridge node uses the PJLink protocol to interact with the projector and provides its functionality as parameters through ILHAS. This PJLink bridge node can be directly reused with other PJLink projectors. Finally, the bluetooth devices can be seen as a native ILHAS node because in this case there are no translations involved. The bluetooth dongles are deployed on a PC that runs an ILHAS node with device drivers that access those bluetooth devices.

As proposed by the HI3 architecture, all the devices are accessed through elements in the Sensing/Actuation layer that will translate them to a higher level view, by, for example, incorporating new information like their location, and providing automatic fault tolerance by transparently changing from one device to another in case of failure or by aggregating them. In this example we are going to keep this layer as simple as possible so it can be seen as redirecting proxies. It is important to notice that these elements are independent of the hardware technologies of the devices they represent because they access the hardware by using the ILHAS library, thus the video projector component can use any type of video projector, from a PJLink compatible one to an RS-232 device, as long as they are adapted to ILHAS.

As shown on Figure 6, the proxy components are elements of the Sensing/Actuation layer of the HI3 architecture. As such, they are implemented as multi-agent systems (in this particular case, they have only one agent) and they have an associated formal description of their functionality that is registered by the HI3 container at deployment time, and that can be used by other system elements to dynamically find them. The functionality descriptions of every element (in any of the layers) can be parameterized at deployment time or at runtime with specific values like, for example, a location in which the element operates.

Looking again at Figure 6, a high level view of the system's design is shown. As proposed by the HI3 architecture model we have designed the system as a set of loosely coupled components that collaborate to resolve the proposed functionality. Those components have been developed within each one of the proposed HI3 layers by looking at their purpose. Hardware access components are on the Sensing/Actuation and Device Access layers. General purpose components that can be reused by other components or projects are inside the Service layer. This is the case of components in charge of identifying the detected users of the system or controlling the set-up of a meeting room. Those components are examples of loosely coupled functionalities that, by having them modularized, can be directly reused in other applications and environments. Finally there are the components that provide concrete, not easily reusable functionalities; we call them applications, as they are the components that implement the use cases that a user expects from the system. In this example we have only one application, the meeting room management application. It relies on the available services to provide its required functionality and incorporates custom logic for its specific functions, such as, for example in this case, the profiling of the meeting room configuration.

We are now going to look more carefully at how the different elements resolve the proposed functionalities.

To achieve user location and identification functionalities a simple approach was applied. Users must carry a device with bluetooth support (for example a mobile phone) and we will use bluetooth dongles integrated in the system to detect those mobile devices. On top of these detection devices there will be a Presence service. It is associated to a location and knows who is in that location. It infers that information in a multi-modal way by using two types of information. Bluetooth devices to know who are in the room, and discrete presence sensors for fault tolerance and to alleviate the problems of the bluetooth technology like detecting people that is the contiguous room. Together with the Presence service there is the Identification service that uses the presence information and user data to generate user location and identification information. The communication between the Identification and Presence services, is achieved using the publish/subscribe capabilities provided by the HI3 middleware. The Identification service is subscribed to a Presence information publication. The same strategy is applied for the Presence service and the devices it uses. It is important to highlight that multiple services of each type can be present in the system at any time and the connection between them is performed dynamically. Their functionality is registered in the HI3 container and, as we have already indicated, can be parameterized, for example by the location they control. Thus, for example, when an identification service is deployed it will search for an adequate Presence service by using its standardized functionality description parameterized with its location.

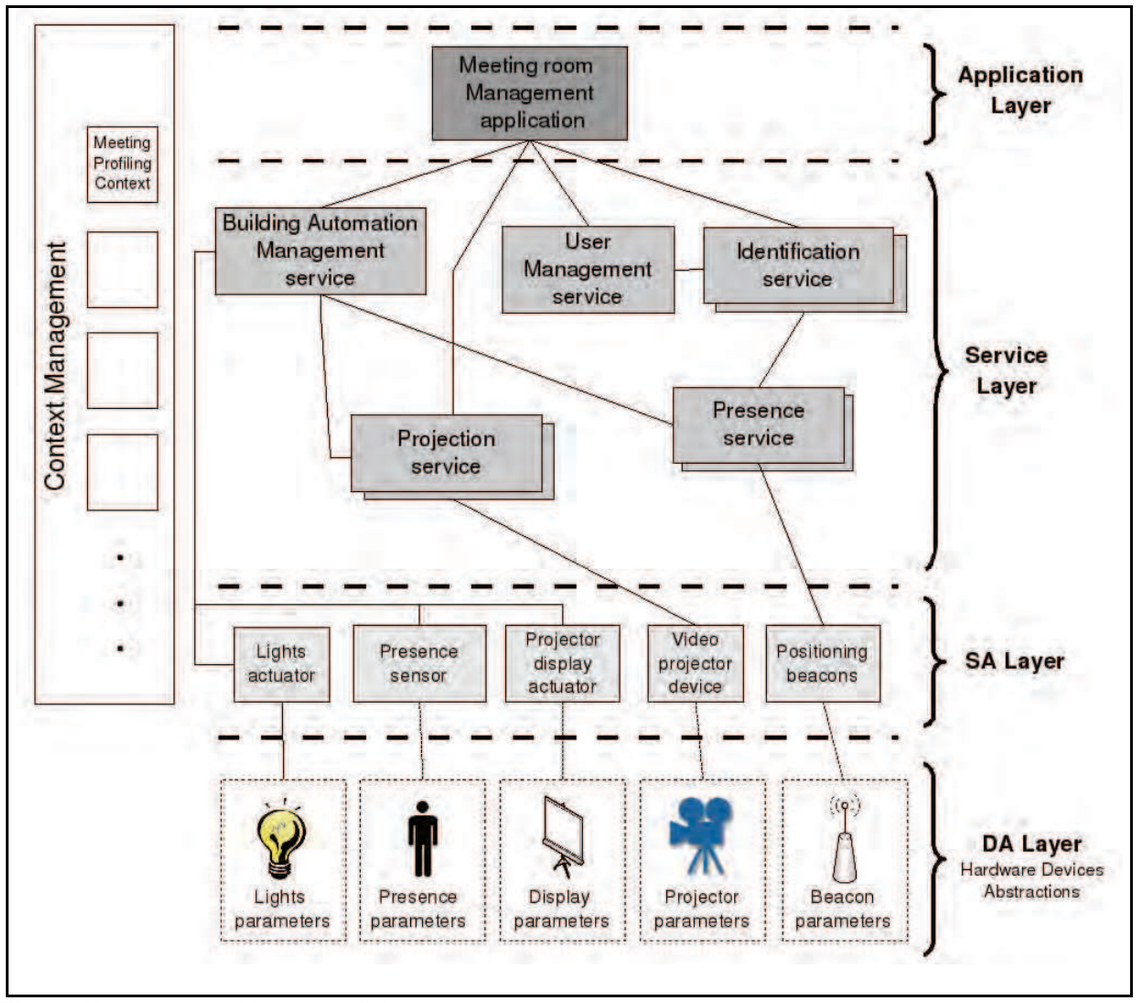


Fig. 6. Component diagram of the proposed example

The Projection service is similar in concept to the Presence service. It is in charge of aggregating and providing control over the typical devices found in a meeting room. There can be multiple meeting rooms in a building, so there is one Projection service for each room and at runtime it will search for partners (video projector, projector display, lighting, etc.) by using their descriptions parameterized with the location value. As with the case of the identification and presence services, this effectively achieves dynamic component composition and collaboration.

The meeting room management application uses all these services to control the devices involved in a meeting and to access information about the environment and its users, like the people present in the meeting, the state of the lights, etc. It applies profiling algorithms to this information in order to extract preferences about the lighting configuration that a particular user (or group of users) prefers for a meeting. More specifically, it monitors the state of the meeting room lights during meetings detecting patterns in their behaviour. For example, when the system does not know the preferences of a user it may, by default, turn off the lights when a presentation starts, if the user does not like that configuration it will turn on the lights (or a given subset of them) using any of the available actuators. This is the

information used by the system to learn the preferences of the user. This information is associated with the context and objective in order to increase the probability of the system automatically turning on some lights during a presentation when the context is similar.
In this simple example we have introduced typical AmI application requirements, like environment sensing and actuation through multiple paths, dynamic component composition and collaboration and even IA algorithm integration. It also has presented how an AmI general purpose architecture can speed up the development of AmI systems by providing solutions that are ready to use in multiple problems and a structured way of integrating them.

5. Conclusion

This chapter provides a review of some of the most important operational aspects of Ambient Intelligence as well as their bearing on the development of the software/hardware platforms that must support it. In this line, a series of issues that an architectonic approach to AmI software development should tackle have been identified and we have shown how the research community in general, and our laboratory, in particular, is dealing with them. Through the study and presentation of the HI3 architecture proposal and by relating it to other approaches found in the literature, the different solutions to the issues that arise and how they must be considered in the design and tool development phase have been visited. Some of these solutions have been taken to a high level of development through their use in other fields. This is the case of distributed operation technology, modularity and low level communication. Others are currently in an intermediate state, some good advances have been made, but much work is needed to fulfil requirements like ubiquity and full autonomous operation; this is case of technologies for issues such as security, mobility, fault tolerance or hardware access and configuration. Finally other areas are still in an early stage of development, some promising work has been carried out, but it is still isolated (not integrated) or assumes too many simplifications in order to deal with the problems. In this situation we find areas such as context-awareness, component self-organization and collaboration or natural human-computer interaction. As a consequence, it is easy to see that there is still much to do in this area and formal and structural approaches are still needed in order to achieve the operational degree that would be required of such a system operating in real life situations with humans. Obviously, it is a very interdisciplinary subject, that will require bringing together knowledge and technologies from very different fields, both technological and social, within a framework that really allows a systematic approach to the construction and adaptation of these structures without leading to a quagmire of ad hoc solutions or a complete deadlock as the systems scale. And it is also evident that to achieve this end powerful design and development tools and architectures will be a key issue.

6. References

Ahn, H., Oh, H., & Sung, C. O. (2006). Towards Reliable OSGi Framework and Applications, *Proceedings of the 2006 sympossiumon Applied computing*, Vol. 6, 1456-1461, Dijon, France, ACM, New York.

AIT. (2009). Athens Information Technology, Center of Execelence for Research and Graduate Education, http://www.ait.gr.

AMIGO (2009), a. AMIGO OSGi Deployment Framework, http://amigo.gforge.inria.fr/home/components/wp3/OSGi_Framework/index/index.html, 2009.

AMIGO (2009), b. AMIGO Project, http://www.hitech-projects.com/euprojects/amigo/.

AMIGO (2005), c. Specification of the Amigo Abstract Middleware Architecture.

Bernardin, K., Ekenel, H. K., & Stiefelhagen, R. (2007). Multimodal identity tracking in a smart room. *Personal and Ubiquitous Computing*, Vol. 13(1), Issue 5, 25-31.

Brdiczka, O., Crowley, J. L., & Reignier, P. (2009). Learning situation models in a smart home. *IEEE transactions on systems, man, and cybernetics. Part B*, Vol. 39(1), 56-63.

Chen, H. (2004). An Intelligent Broker Architecture for Pervasive Context-Aware Systems, PhD Thesis, University of Maryland, Baltimore County.

Chen, H., Finin, T. and Joshi, A. (2003). An ontology for context-aware pervasive computing environments, *The Knowledge Engineering Review*, Cambridge University Press, Vol. 18, pp.197–207. 2003.

CHIL (2009). CHIL - Computers In the Human Interaction Loop, http://chil.server.de.

Crowley, J., Brdiczka, O., & Reignier, P. (2006). Learning situation models for understanding activity, *International Conference on Development and Learning*.

Dey, A. (2001). Understanding and using context, *Personal and ubiquitous computing*, 5(1), 4–7. Springer.

Dey, A. K. and Abowd, G.D. (2001). A conceptual framework and a toolkit for supporting rapid prototyping of context-aware applications, *Human-Computer Interactions (HCI) Journal*, Vol. 16, Nos. 2–4, pp.7–166. 2001

Dourish, P. (2001). Where the Action Is: The Foundation of Embodied Interaction, MIT Press, Cambridge, 2001.z

EMBASSI (2009). EMBASSI - Multimodal Assistance for Infotainment and Service Infrastructures, http://www.embassi.de.

Emonet, R., Vaufreydaz, D., Reignier, P., & Letessier, J. (2003). O3MiSCID, a Middleware for Pervasive Environments, *1st IEEE International Workshop on Services Integration in Pervasive Environments*.

Filipe, C., & Torr, L. (2009). Fault Tolerance in the OSGi Service Platform.

FIPA (2009). FIPA - Foundation for Intelligent Physical Agents, http://www.fipa.org.

Gaia (2009). Gaia Project, http://gaia.cs.uiuc.edu/

Gu, T., Pung, H.K. and Zhang, D.Q. (2004). A middleware for building context-aware mobile services, *Proceedings of IEEE Vehicular Technology Conference (VTC)*, Milan, Italy.

Hanssens, N., Kulkarni, A., Tuchinda, R., & Horton, T. (2002). Building agent-based intelligent workspaces, *ABA Conference Proceedings* (p. 675–681).

Hofer, T., Schwinger, W., Pichler, M., Leonhartsberger, G. and Altmann, J. (2002). Context-awareness on mobile devices - the hydrogen approach, *Proceedings of the 36th Annual Hawaii International Conference on System Sciences*, pp.292–302.

Hung, N., Ngoc, N., Hung, L., Lei, S., & Lee, S. (2003). A Survey on Middleware for Context-Awareness in Ubiquitous Computing Environments, Korean Information Processing Society Review, 1-20.

JADE (2009). JADE - Java Agent DEvelopment Framework, http://jade.tilab.com/.

Klein, M., Onig-ries, B. K., Ussig, M. M. (2005). What is needed for semantic service descriptions - a proposal for suitable language constructs, *International Journal of Web and Grid Services*, 1(3/4), 328-364.

Krumm, J., Harris, S., Meyers, B., Brumitt, B., Hale, M., Shafer, S., et al. (2000). Multi-camera multi-person tracking for easyliving, *IEEE Workshop on Visual Surveillance* (Vol. 6, pp. 1-8).

Messer, A., Song, H., Kumar, P., Kunjithapatham, A., & Sheshagiri, M. (2006). Interplay: a middleware for integration of devices, services and contents in the home networking environment, *3rd IEEE Consumer Communications and Networking Conference*, pp. 1083-1087.

MIT Oxygen (2009). MIT Oxygen Project – E21: Intelligent Spaces, http://oxygen.lcs.mit.edu/E21.html.

NIST (2009). NIST IEEE-P1451 Draft Standard, IEEE1451, http://ieee1451.nist.gov/.

OSGi Alliance (2007). OSGi Service Platform Release 4 Version 4.1, http://www.osgi.org/Download/Release4V41.

Oshima, M., & Lange, D. B. (1999). Seven Good Reasons For Mobile Agents, *Communications of the ACM*, 42(3), 88-89.

Paz, A., Varela, G., Monroy, J., Vazquez, S., & Duro, R. J. (2008), a. HI3 Project: Software Architecture System for Elderly Care in a Retirement Home, *3rd Symposioum of Ubiquitous Computing and Ambient Intelligence*, 11-20, Salamanca, Spain.

Paz, A., Varela, G., Monroy, J., Vazquez, S., & Duro, R. J. (2008), b. HI3 Project: General Purpose Ambient Intelligence Architecture, *Proceedings of the 3rd Workshop on Artificial Intelligence Techniques for ambient Intelligence*, AITAmI08, 77-81, Patras, Greece.

PERSONA (2009), a. PERSONA Project, http://www.aal-persona.org.

PERSONA (2007), b. Full version of ambient assisted architecture specification, http://www.aal-persona.org/deliverables/D3.1.1-Full_version_of_ambient_assisted_architecture_specification.pdf.

Peters, S., Look, G., Quigley, K., Shrobe, H., & Gajos, K. (2003). Hyperglue: Designing high-level agent communication for distributed applications.

Preuveneers, D., & Berbers, Y. (2008). Pervasive services on the move: Smart service diffusion on the OSGi framework, *Lecture Notes in Computer Science*, 5061, 46–60. Springer.

PRIMA (2009). PRIMA Project. http://www-prima.imag.fr

Roman, M., Hess, C., Cerqueira, R., Ranganathan, A., Campbell, R.H. and Nahrstedt, K. (2002). A middleware infrastructure for active spaces, *IEEE Pervasive Computing*,Vol. 1, No. 4, pp.74–83.

Rui, C., Yi-bin, H., Zhang-qin, H., & Jian, H. (2009). Modeling the Ambient Intelligence Application System: Concept, Software, Data, and Network, *IEEE transactions on systems, man, and cybernetics. Part C.*, 39(3), 299-314.

Schilit, B. and Theimer, M. (1994). Disseminating active map information to mobile hosts, IEEE Network, Vol 8 pp 22-32, 1994.

Soldatos, J., Dimakis, N., Stamatis, K., & Polymenakos, L. (2007). A breadboard architecture for pervasive context-aware services in smart spaces: middleware components and prototype applications, *Personal and Ubiquitous Computing* , Vol 11, Issue 3, 193-212.

SOPRANO (2007). Review state-of-the-art and market analysis, http://www.soprano-ip.org/ecportal.asp?id=350&nt=18&lang=1.

Strang, T. and Linnhoff-Popien, C. (2004). A Context Modeling Survey, *First International Workshop on Advanced Context Modeling, Reasoning and Management, UbiComp*.

Tapia, E., Intille, S., & Larson, K. (2004). Activity recognition in the home using simple and ubiquitous sensors, *Lecture Notes in Computer Science*, 158–175. Springer

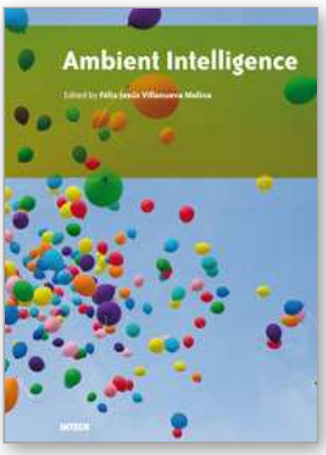
TU Vienna (2009). Institute of Computer Aided Automation, Automation Systems Group, https://www.auto.tuwien.ac.at/a-lab/knx-eib.html.

Vallee, M., Ramparany, F., & Vercouter, L. (2005). Dynamic service composition in ambient intelligence environments: a multi-agent approach, *Proceeding of the First European Young Researcher Workshop on Service-Oriented Computing* (pp. 1-6).

Varela, G., Paz-Lopez, a., Vazquez-Rodriguez, S., & Duro, R. J. (2007). HI3 Project: Design and Implementation of the Lower Level Layers, *2007 IEEE Symposium on Virtual Environments, Human-Computer Interfaces and Measurement Systems,* 36-41, Ostuni, Italy.

Wolf, P., Schmidt, A., & Klein, M. (2008). SOPRANO – An extensible, open AAL platform for elderly people based on semantical contracts, *Proceedings of the 3rd Workshop on Artificial Intelligence Techniques for ambient Intelligence*, AITAmI08, 11-15, Patras, Greece.

IntechOpen



Ambient Intelligence

Edited by Felix Jesus Villanueva Molina

ISBN 978-953-307-078-0
Hard cover, 144 pages
Publisher InTech
Published online 01, March, 2010
Published in print edition March, 2010

It can no longer be ignored that Ambient Intelligence concepts are moving away from research labs demonstrators into our daily lives in a slow but continuous manner. However, we are still far from concluding that our living spaces are intelligent and are enhancing our living style. Ambient Intelligence has attracted much attention from multidisciplinary research areas and there are still open issues in most of them. In this book a selection of unsolved problems which are considered key for ambient intelligence to become a reality, is analyzed and studied in depth. Hopefully this book will provide the reader with a good idea about the current research lines in ambient intelligence, a good overview of existing works and identify potential solutions for each one of these problems.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

A. Paz-Lopez, G. Varela, S. Vazquez-Rodriguez, J. A. Becerra and R. J. Duro (2010). Integrating Ambient Intelligence Technologies Using an Architectural Approach, Ambient Intelligence, Felix Jesus Villanueva Molina (Ed.), ISBN: 978-953-307-078-0, InTech, Available from: http://www.intechopen.com/books/ambient-intelligence/integrating-ambient-intelligence-technologies-using-an-architectural-approach

INTECH
open science | open minds

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

© 2010 The Author(s). Licensee IntechOpen. This chapter is distributed under the terms of the Creative Commons Attribution-NonCommercial-ShareAlike-3.0 License, which permits use, distribution and reproduction for non-commercial purposes, provided the original is properly cited and derivative works building on this content are distributed under the same license.

IntechOpen

IntechOpen