DAAF: a Device Abstraction and Aggregation Framework for Smart Environments

A. Paz-Lopez, G. Varela, V. Sonora, J. A. Becerra
Integrated Group for Engineering Research
University of A Coruña
Ferrol, Spain
e-mail: alpaz@udc.es

***Abstract*—Largely motivated by the ubiquitous computing vision, computers have been steadily becoming integrated in people's life, expanding the ability of software applications to interact with the environment and the users. This has led to the creation of smart environments that support people in their daily life by exploiting information acquired using sensor networks. These systems are usually populated by multiple non-interoperable technologies that make the development, deployment and reutilization of applications a complex and repetitive task. This paper presents DAAF, a Device Abstraction and Aggregation Framework that provides a common conceptual model of a device network, and a set of tools, which allow a flexible integration of heterogeneous sensor networks, enabling fast development and reconfiguration of smart environment applications.**

Keywords-smart environment; ubiquitous computing; sensing; model; device abstraction; RFID

I. INTRODUCTION

During the last few years a new class of sensor networks has appeared motivated by advances in computers and the ubiquitous computing vision. Thus, computers are now integrated in every day objects expanding our ability to sense the environment anytime and anywhere, just like Weiser envisioned in the early nineties [1]. These smart environments assist people by anticipating their needs, exploiting information acquired from the users and from the environment by means of sensor networks.

Regrettably, the integration of device networks in smart environments introduces several new challenges. In this sense, the heterogeneity of available technologies in the device network domain makes development, deployment and reutilization of applications a complex and repetitive task. Specifically, smart environments are usually populated by multiple non-interoperable network and device technologies associated with different application domains like multimedia entertainment or home automation. Furthermore, many device networks require a complex configuration during the deployment phase, and make dynamic reconfiguration of the system an even more complex task. Another challenge particularly related with smart environments is the dynamic discovery of device services, because of the innate dynamism of user-centric applications. Pervasive sensing and monitoring of the physical world also involves significant data integrity and privacy concerns. All of these technical issues are holding back the development and deployment of rich services that could result from the integration of multiple devices and information sources.

In this paper we are mainly focused on the development of a Device Abstraction and Aggregation Framework (DAAF) together with a set of software tools which supports the flexible integration of different device networks, enables fast development of new independent services and provide for the dynamic reconfiguration of the system.

In the device network and integration context, largely motivated by the ubiquitous computing vision, numerous technologies have been proposed. From standardization proposals like UPnP [2] to solutions coming from Ambient Intelligence projects or general purpose solutions in the area of wireless sensor networks (WSN).

Ambient Intelligence and ubiquitous computing are research fields that have to address the device abstraction and integration problem. Thus, projects like AMIGO [3], SAIL [4] or AmI-Space [5] propose solutions that abstract the functionalities of the sensor networks as interoperable OSGi services. Furthermore, these approaches usually provide a middleware layer that exports sensing services with interfaces that are compatible with external protocols, like UPnP.

Taking a different approach, some research projects that are building Intelligent Domotic Environments propose the development of control gateways in charge of providing uniform access to different domotic technologies. A remarkable example of these approaches is the Domotic OSGi Gateway (DOG). This solution provides a device gateway that contains software plugins to support existing domotic technologies and a homogeneous API to control the available domotic devices. DOG also makes use of the DogOnt [6] ontology as a common semantic model for a home automation environment, that provides the solution with inference capabilities, introspection of device operations and states at runtime, etc.

Another interesting approach for the uniform integration and discovery of heterogeneous sensor networks is the one adopted by the GSN (Global Sensor Networks) project [7]. GSN's central concept is the virtual sensor abstraction, which enables a declarative description of sensor capabilities according to XML specification files. Using these virtual sensor components, the GSN middleware allows the flexible integration of local and remote sensor data through plain SQL-based queries.

Other research initiatives in areas related to the Internet of Things (IoT) vision are focused on semantic sensor information description and processing. Specifically, the W3C's Incubator Group on Semantic Sensor Networks (SSN) has developed an ontology to model sensor devices and their capabilities [8]. The SSN ontology supports the description of the physical and processing structure of sensors together with the description of observation and measurement data aspects. However this ontology does not include other modeling aspects of interest like units of measurement or contextual knowledge related to sensor data.

All of these projects and solutions represent isolated efforts towards the development of general purpose middleware systems that minimize the costs of development, deployment and maintenance of large and complex sensing environments. Additionally, having a common middleware for device networks not only reduces costs, but also enables the possibility of integrating data among heterogeneous devices. However, none of the existing technologies provides an all-in-one solution for the instrumentation of smart environments.

The Device Abstraction and Aggregation Framework (DAAF) presented here, in addition to decoupling applications from hardware technology, provides capabilities to dynamically create higher-level device abstractions by defining new virtual devices that aggregate other devices in order to achieve new functionalities.

This paper is organized as follows. Section II is dedicated to a detailed description of the design and implementation of the proposed Device Abstraction and Aggregation Framework. Section III presents a use case example of the solution developed. Finally, in IV we present some conclusions and future work.

II. Device Abstraction and Aggregation Framework

In this section we are going to describe the design and implementation details of the proposed device abstraction and aggregation framework.

The system described here is a high level framework for hardware abstraction, distributed hardware access and sensor aggregation. It can be viewed from two different points of view. On the one hand there is a conceptual model that describes and abstracts the usual concepts managed by hardware devices. On the other hand, there is an implementation of that conceptual model and a set of tools to operate it. Together, they provide a complete framework for the design, implementation and deployment of software that requires distributed hardware access using multiple heterogeneous technologies.

The solution proposed in this paper is based on previous developments we have produced in the area of hardware abstraction [9]. That previous system was designed only for hardware abstraction and with many constraints to make it deployable on embedded hardware. However, the solution proposed here increases the level of abstraction by focusing on device and sensor aggregation, supporting device abstraction from a higher-level point of view.

We are going to divide the description of the solution according to the two different aspects of the system, the conceptual model, and its implementation.

The conceptual model provides system developers with a homogeneous vision of a heterogeneous network of devices. It takes the similarities between the different heterogeneous technologies and assembles a new set of concepts with the essential characteristics of every hardware technology considered. This set of concepts abstract the peculiarities of each technology and device, providing developers with a common language to discover devices and interact with them.

The proposed conceptual model has been designed as a pair of two decoupled models. On the one hand, an abstract model to represent the high-level concepts (and their relationships) that support the operation and management of distributed hardware devices. On the other hand, a classification of specific instances of those abstract concepts.

This division of the conceptual model allows an easy addition of new specific types of devices. It only requires describing them in the classification of devices using the concepts of the abstract model.

Fig. 1 displays the different concepts and relationships supported by the proposed abstract conceptual model. It is made up of eight main concepts:

- **Devices.** These are the devices that a user expects in his network. They provide end users with services and functionalities.
- **Physical devices. C**onceptual representation of the real hardware devices.
- **Groups of devices.** Multiple devices can be grouped together, and a group has the same entity as one device. An application can send commands, queries, etc. to a group of devices in the same way as to a single device.
- **Aggregation of devices.** Similar to groups, but instead of blindly redirecting actions to every device

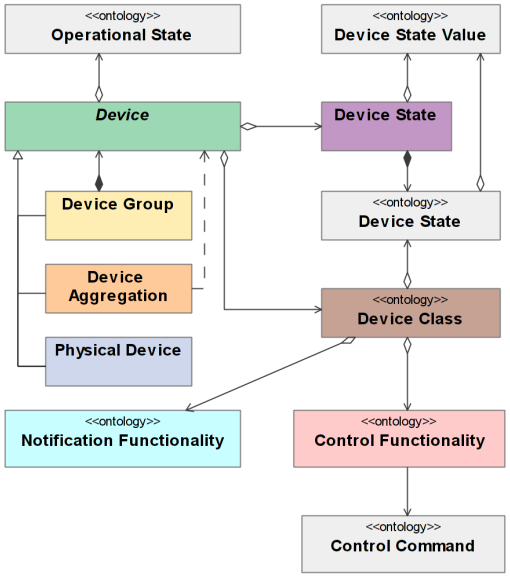

Figure 1. Abstract conceptual model

of the group, an aggregation contains logic to manage the received orders and translate them to actions in its aggregated devices.

- **Classes of devices.** The class of a device represents its type and metadata. It supports inheritance.
- **Device states.** Represent the type of states that a device has (for example, real world values sensed by the device). A device can have multiple different states. They are specified by its device class.
- **Device commands and device notifications.** Together they represent the functionality of a device. A device can receive commands and it usually reacts by changing its state and acting over the environment modifying it. Notifications represent changes in the state of a device. They are specified by the device class.

These concepts are highly inspired by the DogOnt ontology [10]. This is because, as will be explained later, we decided to use DogOnt for the classification of devices.

The main difference with respect to the DogOnt model is the extension of the device concept with three sub-concepts, and, in particular the Device Group and Device Aggregation concepts. As show in Fig. 2, the former provides a way to create sets of devices that share the same functionality (same color), and operate them as if they were a single device. The later allows the creation of new device types, with new functionalities, that are implemented by aggregating other heterogeneous devices.

The abstract model is complemented with a classification of the different types of devices supported by the system. This classification provides a taxonomy of the different classes of devices supported, including metadata about each class of devices, like inheritance, supported device states, or the commands that each type of devices accepts.

The conceptual model and the device type classification, build an abstract conceptual framework for device abstraction that can be implemented using many different technologies.

We have decided to use ontologies for the implementation of the classification of device types.

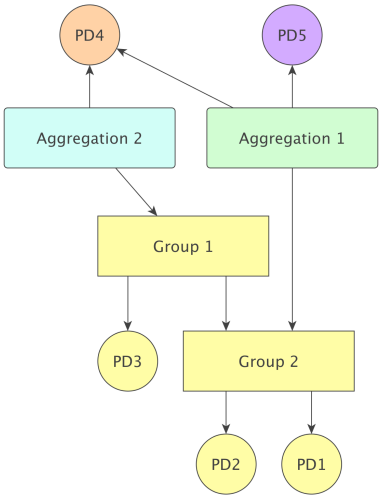

Figure 2. Device group and device aggregation concepts

Ontologies provide powerful tools to describe extensive semantics about concepts, content and relationships between them [11]. There exist some ontologies for device description. One prominent example is the DogOnt ontology. It was created as part of the DomoticOSGi Gateway (DOG) project [12], and it is a very comprehensive taxonomy of devices typically used in home environments.

We decided to use the DogOnt ontology because it is quite complete, and matched the conceptual model we were designing. In fact, we have modified the conceptual model to use some concepts directly extracted from DogOnt.

Regarding the conceptual model, it was implemented using multi-agent technology. This kind of technology allows for the distribution and decoupling of behavior and responsibilities [13], which is an interesting characteristic in a framework for distributed hardware access. Furthermore, we have been developing an Ambient Intelligence development platform using multi-agent technologies [14], and we want to use the solution presented here as its hardware access technology.

The core element of the implementation is the Virtual Device Agent (VDA). It is the software representation of the device concept from the conceptual model. It can thus act as an abstraction of real hardware devices, or as a group or aggregation of other virtual devices, providing higher-level abstraction capabilities. Fig. 3 shows the different elements that build a VDA.

As an implementation of the device concept, a VDA has a device class, a Type, States for representing sensed values, and Commands for representing actions it can perform in the environment using its related hardware devices. Furthermore, as an agent, it has a Behavior that defines the logic that drives the operation of the device.

There exist four different kinds of VDAs, mapping the different sub-types of the device concept, plus one more extra:

- **Abstraction VDA**. This kind of VDA represents real hardware devices available in the system. They include behaviors to interact directly with a hardware device and translate, to our conceptual model, the concepts used by the underlying device technology. These VDAs are used to provide homogeneous access to hardware devices using the proposed conceptual framework.
- **UniDA VDA**. The framework presented here can be used in combination with our low-level hardware access technology UniDA [9]. This kind of VDA includes behavior that allows the direct combination of multiple UniDA devices in order to build a higher-level device.
- **Group VDA**. Connects multiple VDAs of the same type (device class), providing a single control point for all of them.
- **Aggregation VDA**. They operate in a similar way as the UniDA VDAs, but aggregating other VDAs. It allows the creation of new higher-level VDAs that finally rely on abstraction VDAs to interact with the hardware.

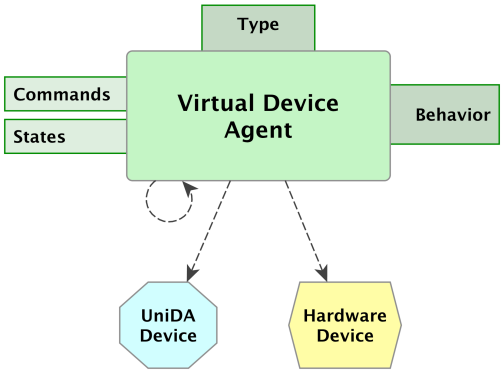

Figure 3. Virtual Device Agent structure

The structure of each kind of VDA is the same, being the only difference between them their Type and Behavior.

The Type defines metadata about the VDA. It specifies the device class from the DogOnt ontology, thus specifying the commands and states supported by it. In the case of Aggregation and UniDA VDAs, their type also includes a way to specify what kind of devices can be aggregated.

The behavior of a VDA specifies how it operates during runtime. It defines, for example, sensing data aggregation strategies, what to do when a command is received, how a state must be updated, etc. The behavior can be programmed specifically by a developer, for example for abstraction VDAs or complex aggregation VDAs. Or it can be selected from a repository of behaviors that is provided with the system. This repository includes, by default, a group behavior, a UniDA VDA behavior and a simple aggregation behavior. Developers and system administrators can add new types and behaviors to the repository.

The group behavior redirects received commands to all its associated VDAs. The aggregation behavior uses the metadata available in the VDA type to associate states of VDAs to the execution of commands in other VDAs. This behavior allows the easy building of higher level VDAs that take sensing data from one device and cause a change in other devices.

As the structure of a VDA remains always the same, it is possible to dynamically build new VDAs by selecting the different elements that build them. A developer, or even a user of the system, can define new particular VDAs by selecting a type and a behavior, and providing some metadata required by the type.

This way, programmers can develop their VDAs decoupled from particular devices. And the users or installers can define the specific VDAs required by theirs systems, for example, creating some abstraction VDAs associated to the particular devices they have to control, and then creating some group VDAs and aggregation VDAs to define the behavior of the system.

For this last purpose, the system incorporates some graphical tools to define new VDAs and store the definition information in a database. This database is used by the system at start time to setup and launch all the required VDAs. More information about those tools will be given in the next section.

From a system developer perspective, developing new types of VDAs is fairly simple. Their only responsibility is to define a new type, and to program a set of behaviors, as complex as required, to control the operation of the VDA. For the implementation of the behavior, they have to implement two strategies that include callback methods that are called whenever the VDA receives a request, a command or a state query from another component of the system (another VDA, an application, etc.). Therefore, the implementation of those strategies must include logic to process commands, device state value requests, etc. This logic can interact with other behaviors that the developer may include in the VDA, for example, for processing sensing data coming from other VDAs to obtain a combined state.

Accessing the system from external software is also fairly straightforward. Fig. 4 shows the architecture of the system. Applications can use the VDA repository to discover the devices available (VDAs) by searching for them by functionality. They can then interact with those devices using the known communication interface of the VDAs, which allows the execution of commands, the query of states and the subscription to state changes notifications. The information of the supported commands and states can be obtained from the device type in the DogOnt ontology. This way, applications are decoupled from hardware devices, as they only interact with devices through the VDAs' common interface, and with information coming from the DogOnt ontology.

There can be many distributed VDA repositories in different hosts, so that an application can use distributed VDAs that operate hardware devices located in different physical places. Furthermore, it is even possible to create high-level VDAs that aggregate VDAs from multiple hosts.

The device abstraction and aggregation framework is complemented by a set of tools that allows the management and discovery of the virtual devices available in a system. This way applications and users can access the resources managed by the system.

The VDA repository shown in Figure 4 is a core element of the system. It not only provides a place for applications to discover the devices available in the system, but it is also a place for system administrators and users to setup and manage their network of devices. Currently, VDAs are added manually to the system.

As shown in Fig. 5, the repository is made of three elements:

- A database to store information about the defined VDAs. We used an embedded RDMS, HSQLDB, because it provides the benefits of a relational database without the overload of a full-fledged database management system with an external server.
- The VDA Manager Agent is in charge of managing the lifecycle of the VDAs, and providing interfaces for applications to query the available VDAs, define new VDAs, etc.

- A web application user interface. The manager agent can be accessed through a web interface which users and system administrators can use to define new VDAs, view the available VDAs, etc.

The manager agent has access to the different types of VDA supported by the system, as well as to the behaviors associated to each type. Using that information the user interface allows users to define new VDAs of any of the supported types, including also support for specifying the metadata required by each type. For example, when creating an abstraction VDA it will require the specification of a URL for connecting to the real device. To create a group or aggregation VDA it will require the association of other VDAs, and in the case of aggregation, it will allow the establishment of relationships between input and output VDAs.

Finally, the system has integrated support for the UniDA system [9], so it is possible to employ the user interface to explore a UniDA network and create VDAs that aggregate its devices. Fig. 6 displays a screenshot of the web user interface of the management system.

III. CASE STUDY

This section will explore an example case study to demonstrate the benefits of using DAAF for the development of smart environment applications.

The example has been developed in an experimental setup deployed on our laboratory, populated by multiple heterogeneous instrumentation technologies. It is a simple example for illustration purposes focused on comfort and power management functionalities. Fig. 7 shows a block diagram of the software components involved in the example.

The experimental setup consists of two rooms where presence detection, identification and home automation devices are deployed. For presence detection we have installed classical domotic presence sensors and some Bluetooth antennas that detect user personal devices within coverage. Regarding identification technologies,

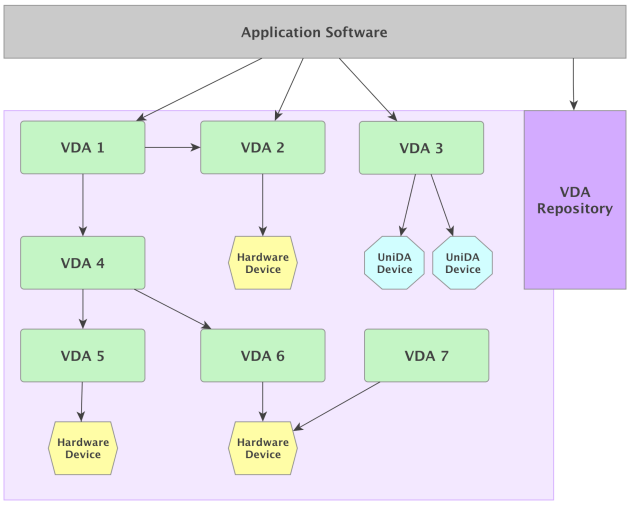

Figure 4. Virtual Device Agent system architecture

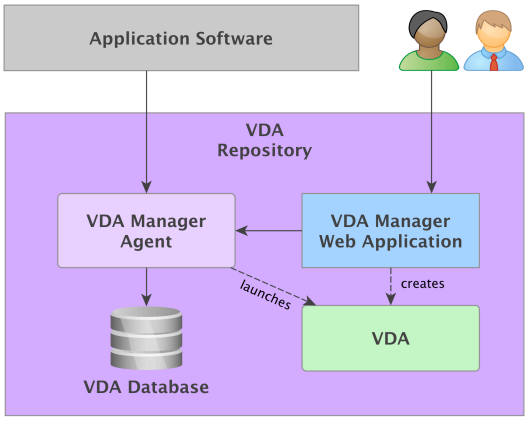

Figure 5. VDA repository and associated management tools

we have deployed RFID readers in the doors of the rooms, a voice recognition system using the MARF library [15], and we also use the Bluetooth antennas for indentifying users. Concerning home automation, it is equipped with a KNX domotic network populated with devices for climate control and lighting management.

On top of these hardware devices, a set of VDAs was developed in order to illustrate how DAAF can isolate applications from the complexities of the underlying hardware, and facilitate the use of the multiple presence and identification technologies available without developers even noticing the presence of different incompatible technologies.

First, various VDAs were developed to adapt the underlying technologies to the conceptual model used by the DAAF. For example, the Voice Identification VDA uses the PC sound subsystem and the MARF library, previously trained with sample data from the users, to create a list of identified persons in a room. Additionally, to improve the detection process, it uses a presence sensor VDA to activate the recognition engine only when someone is in the room. This virtual device agent uses the *IdentificationSensor* device class from our extension of the DogOnt ontology, exporting the *IdListState* through the common interface of the VDA. These abstraction VDAs can be used by applications to interact with hardware devices of the same kind, for example presence sensing devices, using a homogeneous API, independent of the underlying technology.

Based on the abstraction VDAs previously mentioned, a Multimodal Presence VDA and a Multimodal Identification VDA were defined as group VDAs. Each one of them is an association of multiple implementations of devices of the same type, thus creating two high-level virtual devices that provide a single point of access to all the presence detection capabilities of the system, and another one for the identification capabilities. These VDAs can be used by applications to directly access all the presence detection or identification capabilities available in the system. Independently of the number and particularities of the particular hardware technologies deployed in the system.

Finally, it is even possible to include high level control logic in some VDAs, so that they can be more autonomous, leaving external applications as simple monitors of the

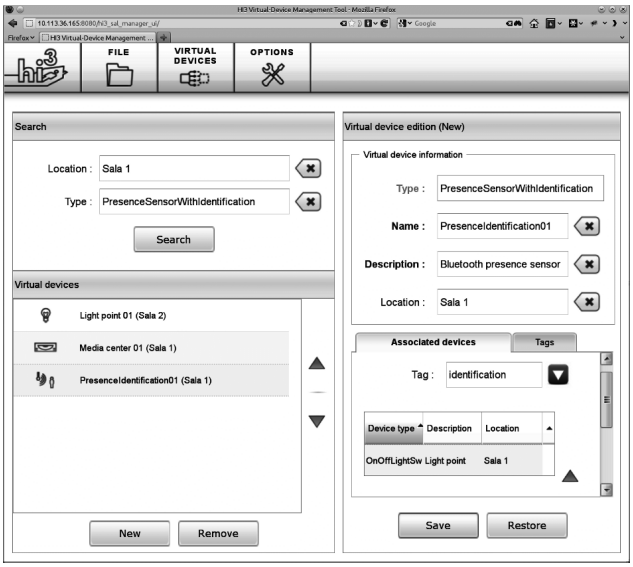

Figure 6. Screenshot of the VDA Manager web application

behavior of the virtual devices. In this example, two aggregation VDAs were developed, the Power Management VDA and the Comfort VDA. They directly introduce some automation capabilities in the system. For example, the former aggregates the Multimodal Presence, the Climate Control and the Lighting Control VDAs to manage the activation and deactivation of physical devices depending on user presence in the rooms. It uses an ad-hoc behavior that receives state notifications from the Multimodal Presence VDA, and executes activation and deactivation commands in the Climate Control and Lighting Control VDAs.

All the VDAs expose a common interface supported by the DAAF conceptual model, particularized to each of them by particular instances selected from the classification ontology. Therefore, client applications, and even higher-level VDAs, can be implemented completely decoupled from the available devices and their underlying hardware.

IV. Conclusions

Smart environments are starting to land in people's homes mainly driven by devices accessible through smartphones and tablets. Unfortunately, the capabilities of such systems are usually quite limited because of the isolation of the devices, which live on artificial islands created by the different technologies used.

As a fit-them-all standardization technology seems almost a utopia, device and network abstraction technologies seem the way to new advanced smart environment applications, capable of exhibiting more useful, intelligent and user-adapted functionalities.

This paper has presented an integrated solution for device abstraction whose main benefits are that, one the one hand it allows the development of interoperable and reusable applications thanks to a common device network model that decouples applications from the underlying technologies. And, on the other hand, it allows for the dynamic definition of new higher-level abstract devices, facilitating the aggregation of existing devices to add new functionalities

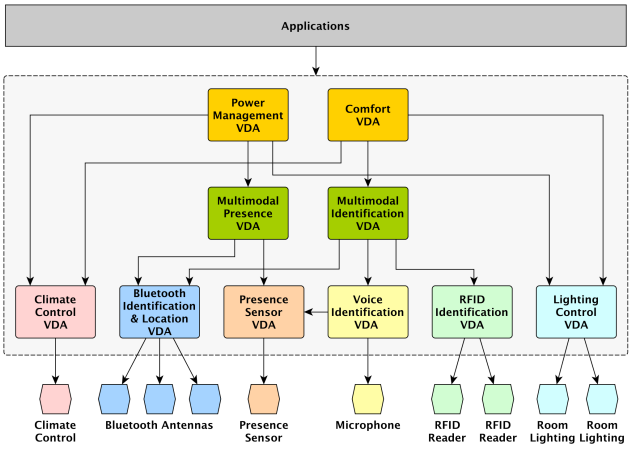

Figure 7. Use case example block diagram

to the system.

Acknowledgment

This work was partially funded by the Xunta de Galicia and European Regional Development Funds through projects 09DPI012166PR and 10SEC019E.

References

[1] M. Weiser, "The Computer in the 21st Century," *Scientific American*, vol. 265, no. 3, pp. 94-104, 1991.

[2] A. Donoho, "UPnP™ Device Architecture 1.1," *Architecture*, no. October, pp. 1-136, 2008.

[3] G. Thomson, D. Sacchetti, Y. D. Bromberg, J. Parra, N. Georgantas, and V. Issarny, "Amigo Interoperability Framework: Dynamically Integrating Heterogeneous Devices and Services," *Constructing Ambient Intelligence*, vol. 11, pp. 421–425, 2008.

[4] M. Girolami, S. Lenzi, F. Furfari, and S. Chessa, "SAIL: A Sensor Abstraction and Integration Layer for Context Awareness," *2008 34th Euromicro Conference Software Engineering and Advanced Applications*, pp. 374-381, Sep. 2008.

[5] C. Rui, H. Yi-bin, H. Zhang-qin, Z. Yong, and L. Hui, "Framework for Local Ambient Intelligence Space: The AmI-Space Project," *Computer Software and Applications Conference Annual International*, vol. 2, no. Compsac, pp. 95-100, 2007.

[6] D. Bonino and F. Corno, "DogOnt - Ontology Modeling for Intelligent Domotic Environments," *The Semantic Web-ISWC 2008*, pp. 790–803, 2008.

[7] K. Aberer, M. Hauswirth, and A. Salehi, "A middleware for fast and flexible sensor network deployment," in *Proceedings of the 32nd international conference on Very large data bases*, 2006, pp. 1199–1202.

[8] L. Lefort et al., "Semantic Sensor Network XG Final Report," *W3C Incubator Group*, 2011. [Online]. Available: http://www.w3.org/2005/Incubator/ssn/XGR-ssn-20110628/.

[9] G. Varela, A. Paz-Lopez, J. A. Becerra, S. Vazquez-Rodriguez, and R. J. Duro, "UniDA: Uniform Device Access Framework for Human Interaction Environments," *Sensors*, vol. 11, no. 10, pp. 9361–9392, 2011.

[10] D. Bonino and F. Corno, "DogOnt - Ontology Modeling for Intelligent Domotic Environments," *The Semantic Web-ISWC 2008*, pp. 790–803, 2008.

[11] G. Antoniou and F. Van Harmelen, "OWL Web Ontology Language," *Ubiquity*, vol. 2007, no. September, pp. 1-1, 2004.

[12] D. Bonino and E. Castellina, "The DOG gateway: enabling ontology-based intelligent domotic environments," in *Consumer Electronics, IEEE*, 2008.

[13] M. Wooldridge, *An Introduction to MultiAgent Systems*. Wiley, 2009, p. 484.

[14] A. Paz-Lopez, G. Varela, S. Vazquez-Rodriguez, J. A. Becerra, and R. J. Duro, "Some Issues and Extensions of JADE to Cope with Multi-agent Operation in the Context of Ambient Intelligence," *Trends in Practical*, 2010.

[15] "MARF: The Modular Audio Recognition Framework and its Applications," *MARF Research and Development Group*, 2008. [Online]. Available: http://marf.sourceforge.net/.