Computer Standards & Interfaces 36 (2014) 928–940

Contents lists available at ScienceDirect

Computer Standards & Interfaces

journal homepage: www.elsevier.com/locate/csi

Software reference architecture for smart environments: Perception

A. Fernández-Montes [a,*], J.A. Ortega [a], J.I. Sánchez-Venzalá [a], L. González-Abril [b]

[a] ETS Ing. Informática, Universidad de Sevilla, Spain
[b] EU Estudios Empresariales, Universidad de Sevilla, Spain

ARTICLE INFO

Article history:
Received 27 May 2012
Received in revised form 14 January 2014
Accepted 3 February 2014
Available online 19 February 2014

Keywords:
Smart environment
Software architecture
Ambient intelligence
Perception

ABSTRACT

With the increase of intelligent devices, ubiquitous computing is spreading to all scopes of people life. Smart home (or industrial) environments include automation and control devices to save energy, perform tasks, assist and give comfort in order to satisfy specific preferences.

This paper focuses on the proposal for Software Reference Architecture for the development of smart applications and their deployment in smart environments. The motivation for this Reference Architecture and its benefits are also explained. The proposal considers three main processes in the software architecture of these applications: perception, reasoning and acting.

This paper centres attention on the definition of the *Perception* process and provides an example for its implementation and subsequent validation of the proposal.

The software presented implements the *Perception* process of a smart environment for a standard office, by retrieving data from the real world and storing it for further reasoning and acting processes. The objectives of this solution include the provision of comfort for the users and the saving of energy in lighting. Through this verification, it is also shown that developments under this proposal produce major benefits within the software life cycle.

1. Introduction

A smart environment (SE) can be defined as *one that is able to acquire and apply knowledge about the environment and its inhabitants in order to improve their experience in that environment* [1].

Smart home technologies are an important part of ubiquitous computing. Mark Weiser [2] outlined the principles of Ubiquitous Computing: the purpose of a computer is to help someone do something. Nowadays, due to the popularisation of computational devices and applications, ubiquitous computing is recognised as a revolution in the development of smart environments.

Nevertheless, software artefacts related to ubiquitous computing, together with the wide spectrum of computational devices (and the software needed to fulfil their missions) are too heterogeneous and hence difficult to compare or classify. Each piece of software evolves in an isolated way or only in relation to the hardware for which it has been developed. The problem addressed in this paper involves the orchestration of the architecture of a general software model for the development of SEs.

This Software Reference Architecture would favour the development of a smart environment solution by increasing the reuse of components, promoting interoperability, and defining the competences of each part of the software.

A good comparison for this could be the Open System Interconnection (OSI) model, which is a prescription for characterizing and standardizing the functions of a communications system in terms of abstraction layers.

The ambitious goal of this architecture forces it to remain very general and to leave specific aspects until the implementation stage.

The benefits of the approach include a better understanding of the issues that must be faced when developing each component of a smart environment solution. The Software Reference Architecture reduces the costs of the main cycles of software (design, development, deployment, and maintenance) and favours the interoperability between various solutions.

The main goal of this work is the proposal of Software Reference Architecture for the development of SEs (see Section 3), where all the components can interact flawlessly and reach automatism objectives.

To this end, the architecture proposed seeks to improve the modularity, reusability and extensibility of solutions, thereby allowing a more coordinated evolution of SEs, which currently remain under individual and isolated development. The architecture defines a middleware framework that connects the modules and establishes the responsibility of each module. The benefits for developers using a defined framework or standard architecture for the domain have been thoroughly studied by Fayad and Schmidt [28], and include: a reduction and focus of the effort involved, a soft learning curve, integrability, maintainability, easier validation, efficiency, and a higher level of standardization.

* Corresponding author.
E-mail addresses: afdez@us.es (A. Fernández-Montes), jortega@us.es (J.A. Ortega), jisanchez@us.es (J.I. Sánchez-Venzalá), luisgon@us.es (L. González-Abril).

http://dx.doi.org/10.1016/j.csi.2014.02.004
0920-5489/

As an example of the architecture usage, this paper presents the Perception process and provides an example of implementation by following the Software Reference Architecture proposed.

Typical components of a SE have been thoroughly studied in the literature, although the approach of Cook and Das [20] deserves special mention since it is currently the most widely accepted approach. Fig. 1 shows the general organization of these components. Components are divided into four layers: a) physical; b) communication; c) information; and d) decision. This approach joins hardware with software agents, and hence very heterogeneous elements, such as a decision maker and sensors or actuators, appear in the same component model.

All these components must collaborate in order to achieve the goals of automatism that a SE requires. Which tasks belong to each component and how they should collaborate constitute the main motivation of the Software Reference Architecture proposed.

The Software Reference Architecture proposed is divided into three main parts: Perception, Reasoning and Acting. This paper focuses on the definition of Perception, as the first step in the general process. Section 2 analyses related work in this area, and in Section 3, Reference Architecture is presented and the Perception process is explained. Finally, verification with a prototype of the Perception process is shown in Section 4, and conclusions are drawn in Section 5.

2. Related work

Ambient intelligence is a trending topic, and hence a wide variety of related research initiatives have appeared. One of the most common applications in ambient intelligence is that of SEs. Many researchers around the world are developing projects which involve SEs.

In this section, some of the most popular projects are reviewed and compared. The section has been divided depending on where each project is focused: general smart environments, technologies or architectures.

2.1. Smart environment projects

Da Costa [12] focuses on the challenges and issues that ubiquitous computing applications have to deal with and summarizes them: heterogeneity, scalability, dependability and security, privacy and trust, spontaneous interoperation, mobility, context awareness, context

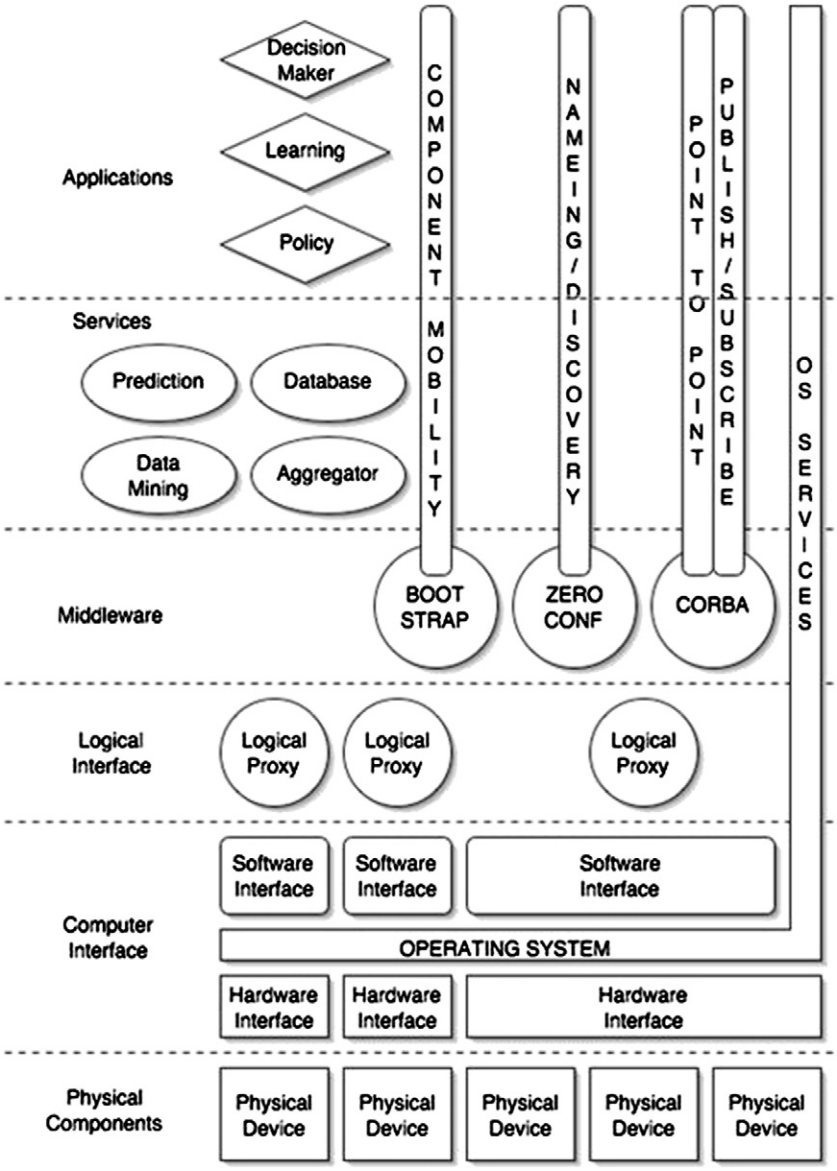

Fig. 1. The components of a SE by Cook and Das [20].

management, transparent user interaction and invisibility. Thus, SE projects, as subproducts of ubiquitous computing, have to address those challenges.

For example, the objective of the MavHome project [3] is to create a home that acts as an intelligent agent. Its architecture organizes the environment as an agent that can, in turn, be divided into several intelligent agents which interact. The technologies within each agent are separated into four cooperating layers: decision, information, communication, and physical. Perception is a bottom-up process, while acting is top-down.

DomoSEC project [5], offers a home automation solution that covers necessities in indoor domotics. DomoSEC is based on OSGi and composed of an embedded computer that centralizes the home "intelligence", and communicates with the controlled devices.

The intention of the Aware Home project [8] is to produce an environment capable of managing information about itself, its occupants, and their activities. It is composed of a set of subsystems responsible for the various technologies deployed: human–computer interaction, machine learning, computational perception, wearable computing, ethnography, software engineering, and sensors.

Energy Aware Smart Home [7] focuses on the integration of heterogeneous embedded devices and power metering plugs by middleware called Hydra. Its purpose is to retrieve energy-consumption data for the monitoring and analysis of energy consumed, in order to programme, control and use home appliances efficiently.

The Place Lab project [9] is centred on capturing inhabitants' behaviour through a group of cabinetry components, which contain a microcontroller, speakers, cameras, and a set of sensors. These sensors record a complete audio–visual log of activity. Place Lab data streams are employed to develop new context-detection algorithms and context-aware computing applications.

Patch Panel [18] is a mechanism for the incremental addition of modification of behaviour in existing ubiquitous computing environments (such as iRoom), for example, by adding new input modalities or choreographing the behaviour of existing independent applications. It provides a general facility for retargeting event flow, and enables interactions between networked hardware and software components to be created or modified in ubicomp environments.

EasyLiving [16] is a Microsoft project which pursues the easy aggregation of I/O devices into a single and coherent environment. The systems include middleware to facilitate distributed computing, world modelling to provide location-based context, perception to collect information about the world's state, and service description to support decomposition of device control, internal logic and user interface.

Other systems, such as the Sentient Computing System [19], can change their behaviour based on a model of the environments they construct by using sensor data. This system strives to remove obstacles by sharing and configuring devices for exploitation in the world model.

CASAS [6] is an adaptive smart-home system that utilizes machine-learning techniques to discover patterns in the daily activities of residents and to generate automation policies. Data from sensors is analysed in order to determine activity patterns of interest for automation. Patterns are modelled continuously in a multilevel structure to build the context. Finally, a selection of the activities to be automated is performed.

2.2. Projects with technological improvements

Other projects are more focused in technology, and provide several kinds of technological improvements. Some examples of this are described below.

The ATRACO project [4] aims to support everyday activities in a meaningful way. It uses a remote OSGi platform connected with a residential gateway, which manages the devices, sensors and actuators via UPnP protocol. These elements working together offer a context-aware service for the environment.

A high-level programming language, called Visual RDK, is proposed by Weis [10], for prototyping pervasive applications. This language generates a debugging application and a prototype application from the same source. The main advantage of this proposal is that context is tightly integrated into the language itself, and hence developers can attach functionality to locations, people, or situations instead of to the device.

However Bannach [11] focuses on the problem from another point of view: prototyping of Activity Recognition applications. It features mechanisms for distributed processing and supports for mobile and wearable devices. The CRN Toolbox is a tool set specifically optimized for the implementation of multimodal, distributed activity and context recognition systems running on Posix operating systems. It also contains a collection of ready-to-use algorithms (signal processing, pattern classification, and so on). Its implementation is specially optimized for mobile devices.

2.3. Projects with relevant architectures

The major contribution of some projects lie in the architecture proposed by them. The following projects are examples of this case.

The GAIA [13] metaoperating system extends the reach of operating systems to manage ubiquitous computing habitats and living spaces as integrated programmable environments. It presents a middleware infrastructure for active spaces. This middleware, unlike the architecture proposed in this paper, is localized at the operative system level.

JCAF (Java Context-Awareness Framework) [14] is a Java-based context-awareness and service-oriented infrastructure and an API for creating context-aware applications. It groups the elements of the architecture into four categories: Context Services, Entities and Context, Context Clients, and Context Events. These elements communicate between each other.

Interplay [15] is middleware software which integrates heterogeneous home devices to simplify their control to the user. It allows users to use a pseudo-English interface to achieve home tasks without difficulty. Its architecture is organized into five layers, which sends orders from the user interface to an underlying middleware for device management. Its objective is to provide advanced control of the devices.

I-Centric Services [32] from Fraunhofer FOKUS Berlin, establishes a taxonomy of roles that are assumed by the different nodes without distinguishing between node types. Each node, usually a software service, provides standard interfaces for a variety of tasks. This plain architecture is very ubiquitous-computing-oriented, by giving all the components the consideration of nodes with different roles, but without more structure.

Mundo [33], from Darmstadt University, establishes a structured node classification centred on the scope of the node. This project introduces major aspects, such as the communication and the association of nodes, but it remains a low-structured node classification more than an architecture, from a component-level point of view. MundoCore [27] communication middleware designed for the requirements of pervasive computing is also developed in Darmstadt University. It is low-level software oriented, as opposed to our proposal that describes a higher-level software architecture.

Gator Tech Smart House [31] from the University of Florida, proposes a layered Reference Architecture, divided into four layers: application, service, node, and physical. Each layer includes a set of sublayers and components that form the whole architecture. It also includes OSGi as a solution for the management of devices in the service layer. This is an architecture very similar to our approach, although we pursue an architecture of a simpler and more abstract nature. The main difference, between their proposal and ours, is that Gator architecture does not explicitly include an acting layer and does not define a cycle. Moreover Gator architecture does not define tasks for their Sensor layer, meanwhile our proposal defines five main tasks for it.

GAS-OS [17] is the closest approximation to the software Reference Architecture proposed in this paper. This software implements the Gadgetware Architectural Style (GAS) approach, in which people configure complex collections of interacting eGadgets, in a similar way to that of a system builder in designing a software system and components. The benefits of the ubicomp are the same as those demonstrated by software engineering: encapsulation and composition.

As can be concluded from a detailed analysis of these projects, their architectures remain heterogeneous, although several common characteristics can be identified such as: general Perception-Reasoning-Acting cycle (commonly mentioned and explained in the bibliography, for example by Russel and Norvig [25]), or a certain kind of integration and interaction between devices (the OSGi framework is frequent), or there maybe treatment of context-related information. The projects usually take a layered approach, but fail to follow a common structure, and their layers, components and/or devices differ greatly and are also very architecture-dependent.

The evolution of these systems by integrating new devices, algorithms or methodologies that improve or perform new tasks involves a complex task, due to the low level of encapsulation and modularization. Software design is a key feature in the organization, integration, and scalability of these numerous components.

This problem is one of the prime reasons why SEs have yet to evolve, and why they belong more to the world of research and academia than to industrial solutions. Solving this pushed us to propose the following Software Reference Architecture that organizes those common tasks which need to be accomplished in the design of a SE from a conceptual point of view, independent of the implementation. This necessity has already been indicated by Muhlhauser and Gurevych [26].

3. Reference Architecture

3.1. Introduction

This section explains the proposal of Software Reference Architecture to develop the software layer in SEs. The proposal is based on the goals of ubiquitous computing proposed by Weiser [2]. Taking this as a starting point, automation in SEs can be organized as a continuous interaction between three main processes: a) perception, b) reasoning and c) acting (see Fig. 2).

As indicated in the Introduction, this work is centred on the Perception process of the Reference Architecture, which is explained below.

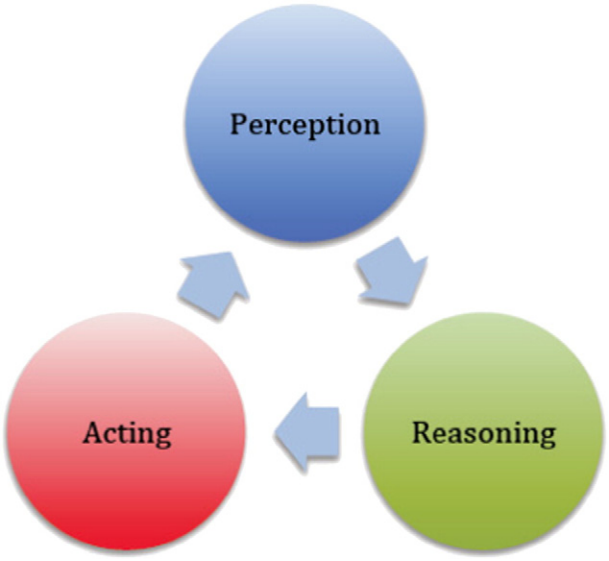

Fig. 2. The cycle of the automation process in a smart environment.

3.2. Perception

The *Perception* process should be divided in order to split complex perception activity into several attainable tasks. The result of these tasks must be an accurate perception of the real world (see Fig. 3).

Perception has to deal with low-level details to retrieve data from real world and to adapt it to a knowledge base, which must agree with the ontology of the SE. A simple information model for the SE used in the prototype scenario is proposed in Table 1. The process has to clear this retrieved data of any erroneous, insignificant, and redundant values in order to build an accurate representation of the real world, as required by the following process.

Some of these tasks have common features with ordinary pre-processing of data such as normalization, adding attributes or replacing missing values. Pre-processing of data for SE has been studied by several authors. Stankovski and Trnkoczy's [34] proposal defines a table from the data collected in order to generate inputs for following processing, but does not cover error detection, reparation or the devices which perform this pre-processing. On the other hand, Elnahrawy [35] proposes two general processes

- cleaning data (considering cleaning at sensor level or cleaning at database level) by applying probabilistic uncertainty models and
- querying data.

Moreover Wu and Clements-Croome [36] mention a *Data preparation* step where data miners create relevant subsets but do not list scopes, reparation or error detection proposals, like Zhang [37] which just mention that a pre-processing step is required for integration of low-level sensor data. The author's approach includes concepts presented by previous proposals and generalizes them.

3.2.1. Data collector

This is the lowest-level task, and its aim is to retrieve data from physical devices within the SE. The Data Collector usually has to deal with gateway devices of every type of sensor technique deployed in the SE.

Data can be generated by numerous kinds of sensors such as temperature, pressure, optical, acoustic, mechanical, motion, vibration, flow, position, electromagnetic, chemical, humidity, and radiation, and therefore a crucial question that must be addressed concerning the task of the Data Collector is that of the unification of data types.

In fact, only a small subset of the environment properties (Table 1) is necessary to perform a particular application or automation process (e.g. switching lights off when nobody is at home does not need conditioning information from the environment). Similarly, sensors must be deployed in an organized manner in order to prevent the processing of useless information.

Regarding the execution of the data processing, it is important to consider where the processing should be performed since a number of devices allow internal programming while others are pre-programmed.

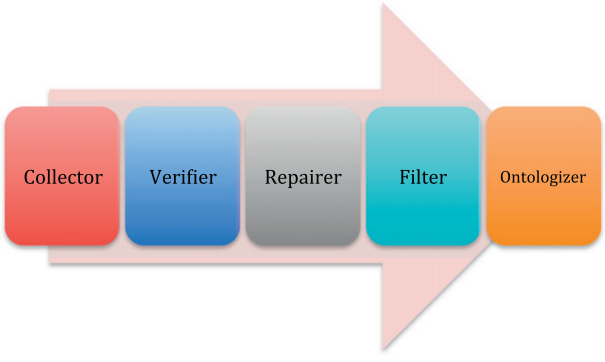

Fig. 3. Tasks of the perception process.

Table 1
Proposal for modelling smart environments.

Categories	Fields	Description	Example
Device-related	Status	Current state of the devices	Sensor temperature measures 25 °C.
	Location	Where the devices are	Cleaner robot is in the living room.
Inhabitant-related	Personal data	Name, age, sex	Diane is 45 y.o.
	Location	Where the inhabitants are	Mark is in the bedroom.
	Physical state	Illness, injuries, and condition of inhabitants	Roy has a cold.
	Mental state	Psychological state of the inhabitants	David is depressed.
Environment-related	Date, time	Temporary information	Current time is 13:36.
	Environmental conditions	The phenomena that are currently occurring in the atmosphere	It is rainy.
Home background	Inert entity location	Where these entities are	The sofa is in the living room.
	Home limits and properties	The properties of the home structure and its limits	Bedroom window opacity is 70%.

The processing can be distributed, centralized, or even a combination of the two if both kinds of devices are present in the environment.

Since the majority of devices are pre-programmed and cannot extend their basic functionality (e.g. X-10 motion sensors), the design of the SE has to be suitably adapted and developers have to adapt to them. On the other hand, a growing number of devices have now the ability to extend or modify their performance (e.g. Sentilla Tmotes).

Programmable devices are much more flexible, and therefore pose a greater challenge to the designer, since they can be adapted to current needs for a specific application or circumstance. In this case, the task of the Data Collector must involve the use of daemons developed to retrieve information, and also small applications running on devices, and hence both elements have to agree on what information is sent, the periodicity of these requests, and so on.

3.2.2. Verifier

The main purpose of the *Verifier task*, as its name implies, is to verify that the data is being received by the *Data Collector* correctly. However the challenge here is how the *Verifier* can determine whether the data is correct or not. There is no unique solution for all possible environments, so the verification has to be adapted to each environment by taking into account the ontology used.

In our proposal, *Verifier* maintains a rule engine where verification rules can be deployed, modified and checked in order to determine where data is right or wrong for the current environment. The rule engine has to offer programmers a flexible way to add, modify and delete rules, and therefore these should be human readable. The set of rules of the *Verifier* should evolve over time, as environment changes are produced.

This task has to work side by side with the *Repairer* when incorrect data is received in order to fix invalid data. The mechanism of communication between the *Verifier* and the *Repairer* must be determined. Typical implementations of this mechanism include the publication of a repair service by the *Repairer* that is invoked by the *Verifier*.

It can be concluded that the *Verifier* can be seen as a filter applied over all the data received, and it can be used to reject data for any reason (incorrect, redundant…) by using the aforementioned rule engine and based on the ontology model proposed in Section 3.2.5.

3.2.3. Repairer

The task of the *Repairer* is to fix incorrect data detected by the *Verifier*. The repair applied to the data must always consider the defined ontology and can be performed in a wide variety of ways, such as:

- Ignore data. The first option is to ignore the data, which means setting it with an unknown value. The *Ontologizer* saves this value as required by the storage system (e.g. the Weka-arff '?' character, or the SQL null value).
- Adjust data. If a value is incorrect, but its distance from a correct value is less than a previously specified threshold amount, then the value could be adjusted to the nearest correct value.
- Replace data. Another option involves replacing data with previously correct data (e.g. if temperature sensor returns 100 °C, and previous data is 25, then the current value can be replaced with 25).
- Reject data. If current values are not suitable for repair, then the *Repairer* will reject them.

Once a set of data has been received, verified and repaired, it has to be sent to the *Ontologizer* to be organized and stored.

3.2.4. Filter

Sometimes, not all the data received from the environment is necessary for the reasoning process, and hence the ontology dismisses this information. This is the objective of the filter: to prevent this superfluous information from being sent to the *Ontologizer*. The implementation of this filter, if present, should be based on the ontology defined for the representation of the environment.

3.2.5. Ontologizer

An ontology represents the knowledge about the world (or environment) as a set of classes, properties and relationships, within a domain. The reasoning is performed using the entities represented, and hence data retrieved from the environment should be organized before applying artificial intelligence techniques in order to have a solid knowledge base with which to work.

The main goal of the *Ontologizer* is to organize, homogenize, synchronize and aggregate data to form a model of the real world supported by the ontology defined for the SE.

It is necessarily an important effort for building a model of a SE which provides data interoperability and makes possible to realize inference, as suggested by Nucci [29] where an ontology framework is used to describe all relevant information of the environment: devices, services and context. It also tackled the further difficulty for device manufacturers because of the lack of standardization in semantic technologies within these scenarios. Energy is another key aspect to take into account for the ontology as proposed by Kofler [30], where ontology includes information not only related to the environment, but also about energy supply and provider.

Some other studies like Cook [3], Das [21] and Li [22] have helped in the composition of the abstract model proposed, which has been arranged into four main categories as explained in Table 1. This model can help developers ascertain the main entities that need to be monitored in the environment.

Device related — This category is the most obvious, and it is related with the main elements in a SE. Ambient intelligence algorithms should be aware of the following main fields:

- Status. Algorithms must know the current states of devices installed in the SE. Obviously this is essential for these algorithms, and one of the prime factors for building of future predictions. Energy aware algorithms may also need information about energy needed by these devices to operate in order to apply any energy saving policy.
- Location. Devices usually remain at a location for a long time, and hence this information can be used by ambient intelligence algorithms. The model must also be able to handle mobile devices, such as motorized cleaner robots.

Inhabitant-related — SE algorithms must be aware of the inhabitants' status to offer appropriate predictions for any user or for the whole group of inhabitants. Along this line, several of the necessary fields to infer inhabitant-aware predictions are discussed:

- Personal data. This field includes all the data concerning a particular person, such as name, age, and gender.
- Location. Inhabitants can move between different spaces, so SE systems should be able to identify and locate each inhabitant.
- Physical state. This field is related with the illnesses and injuries that an inhabitant can suffer. SE technologies must adapt to these situations and offer appropriate responses.
- Mental state. The state of mind of a person can be defined as the temporary psychological state. The behaviour of a depressed inhabitant usually differs from that of a euphoric inhabitant, and hence SEs must be consistent with these circumstances.

Environment-related — This category is probably the most diffuse since it covers heterogeneous and difficult-to-limit fields, as discussed in the following list:

- Date, time, season. Obviously SE behaviour differs under each temporal condition. For example, the air conditioning policy is altered between summer and winter.
- Environmental conditions. This field is comprised of current environmental conditions (sunny, cloudy, rainy, among others). A SE should also request a weather forecast, which could be significant in the assessment of future decisions.

Home background — This category must contain all the relevant items regarding inert entities and their properties and qualities. This category is the least relevant discussed, but could remain significant in certain specific applications. Two related fields are proposed in the following listing:

- Furniture location and position. Furniture occupies space at home and can be moved. Location (room where the furniture is located) and position (place within the room) should be registered by the smart home systems since it could be useful in specific applications, such as robot movement-related algorithms, and presence detection-related algorithms.
- Home limits and properties. The texture of a floor, the colour of a wall, and the opacity of the windows could be significant in specific cases, such as temperature-adjustment applications.

There are yet two more issues concerning the *Ontologizer*: Synchronization and Aggregation.

Synchronization — The *Ontologizer* has to synchronize data from a world full of asynchronous devices, and events. Response time constitutes a major factor when reasoning about events. Automation applications usually need sets of data composed of values from multiple devices, captured at various moments. Data from a variety of devices must be synchronized for its latter aggregation, and hence this task has to define the logic to synchronize values from multiple and very heterogeneous sources. Implementations of the synchronization process vary depending on the goals of each specific smart application and on its type of data. However all implementations share certain common elements such as:

- Data buffer, which stores received data that is waiting to be paired with other data.
- Garbage collector, which supervises the size of the buffer, and periodically cleans the buffer of data that cannot be paired.

Aggregation — Once data is synchronized, it is aggregated in a set of data for a specific smart application that conforms to the SE model. When data is aggregated, it is ready to be stored in the knowledge base that feeds the reasoning tasks and learning process. The knowledge base format can take the form of any of the de facto standards, such as *arff* (Weka software format file for input data), or that of a relational database.

3.3. Device Abstraction

There is a gap that needs to be covered between the modules defined by the Reference Architecture and the physical devices. One interesting initiative which solves this problem is *Device Abstraction*. This is the result of previous work by authors within the OSAmI project [23].

In Device Abstraction a device abstraction layer is provided which describes a set of standards and conventions for controlling, configuring and accessing the data generated from all kinds of devices related to Ambient Intelligence. The integration of sensors and actuators is provided by following the *Device Abstraction model* in their control software.

This hierarchical model, shown in Figs. 4 and 5, unifies criteria in order to facilitate the access to the devices, their functionality, and their generated data. In this way, the methods used are independent of the underlying protocols, and allow easier device switching. The use of Device Abstraction is a step towards standardization of SEs.

The primary role of Device Abstraction is the classification of devices in sensors and actuators. Sensors are categorized as either meters or detectors, while actuators depend on their functionality as pulse, switch, dimmer and movement. Each device category provides a set of specific methods for control of the device actions.

There is still another software artefact between Device Abstraction and the devices: the API which translates high-level methods invoked in protocol-dependent requests. These APIs are usually provided by device manufacturers.

3.4. Communications between processes

Reasoning processes in SEs can be separated into several tasks which interact to achieve three main goals: a) to learn, b) to reason, and c) to predict.

Finally, in order to close the circle, SEs must act automatically to achieve a specific smart application. This is the main purpose of the Acting process. The decisions and specific tasks ordered by the Reasoning process, have to pass through three main taskmasters: a) policy manager, b) task scheduler, and c) task runner.

In order to feed both Reasoning and Acting processes, an event-driven architecture paradigm (EDA) is proposed, in the form of a publishing-subscribe message system. Reasoning tasks are *subscribers* of data generated by *Perception* processes and Reasoning tasks are *publishers* of inferred knowledge. In this way, Acting tasks become *subscribers* of the knowledge generated by the Reasoning process.

4. Verification with prototype

An example of the *Perception* process implementation, which follows the Reference Architecture proposed in this paper, is presented in this section as proof of its usefulness.

4.1. Prototype overview

The main objective of the application is the perception of a smart office. It retrieves data on the localization of workers, and on luminosity, temperature, and humidity (see Fig. 6). The purpose of the retrieval of this information is to acquire knowledge about inhabitants/workers regarding their habits with respect to lighting conditions, temperature, etc. in order to make smart use of artificial lighting for energy saving purposes.

The hardware of the development environment is composed of:

- a computer, which acts as the data receiver gateway,
- three Sentilla Tmotes (two in the office as shown in Fig. 6 and a third in another office), as sensor devices for the collection of data on the quantity of light, temperature, and humidity,
- an X10 motion sensor in order to determine whether the office is occupied,

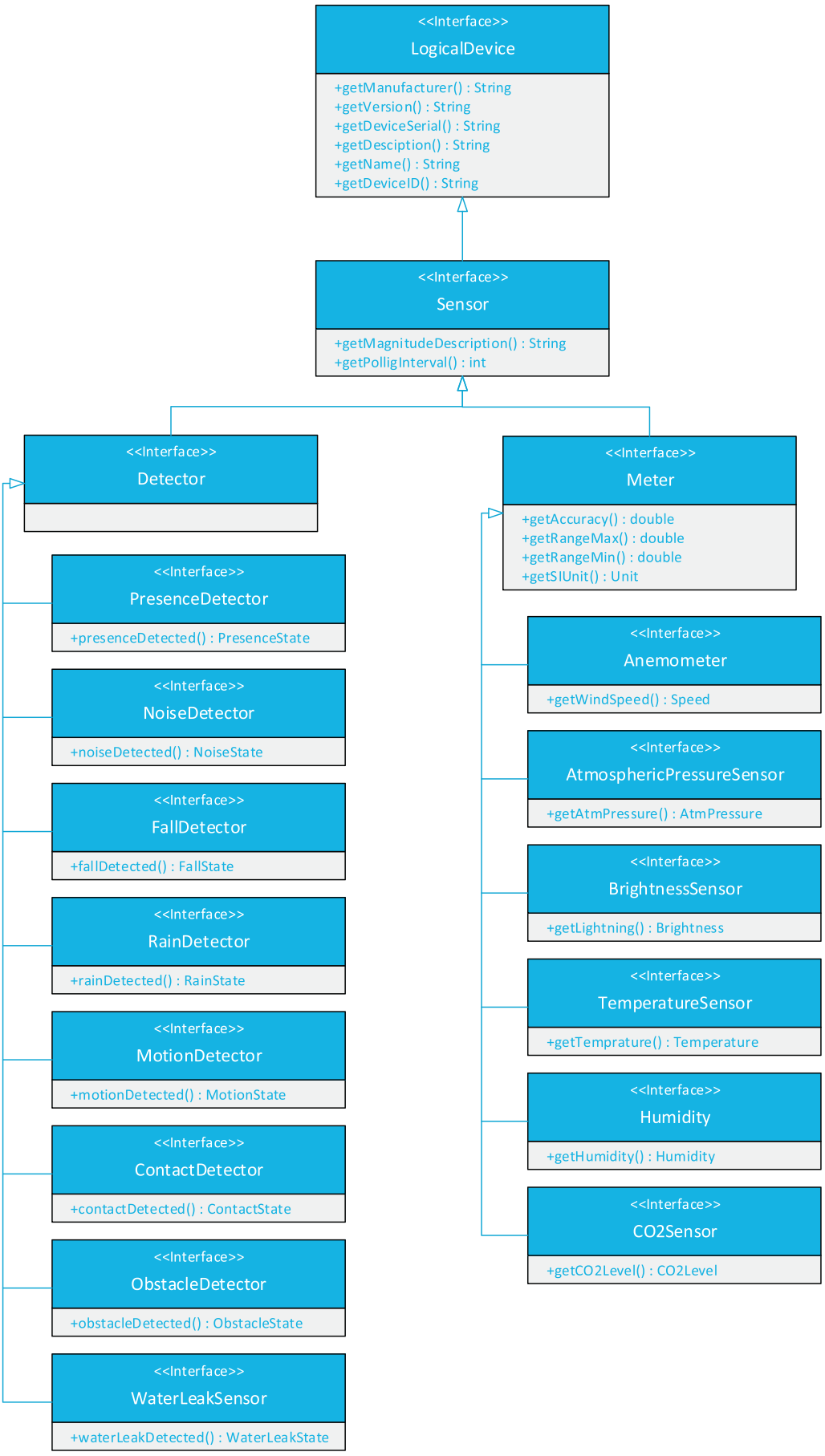

Fig. 4. Device Abstraction class model for sensing.

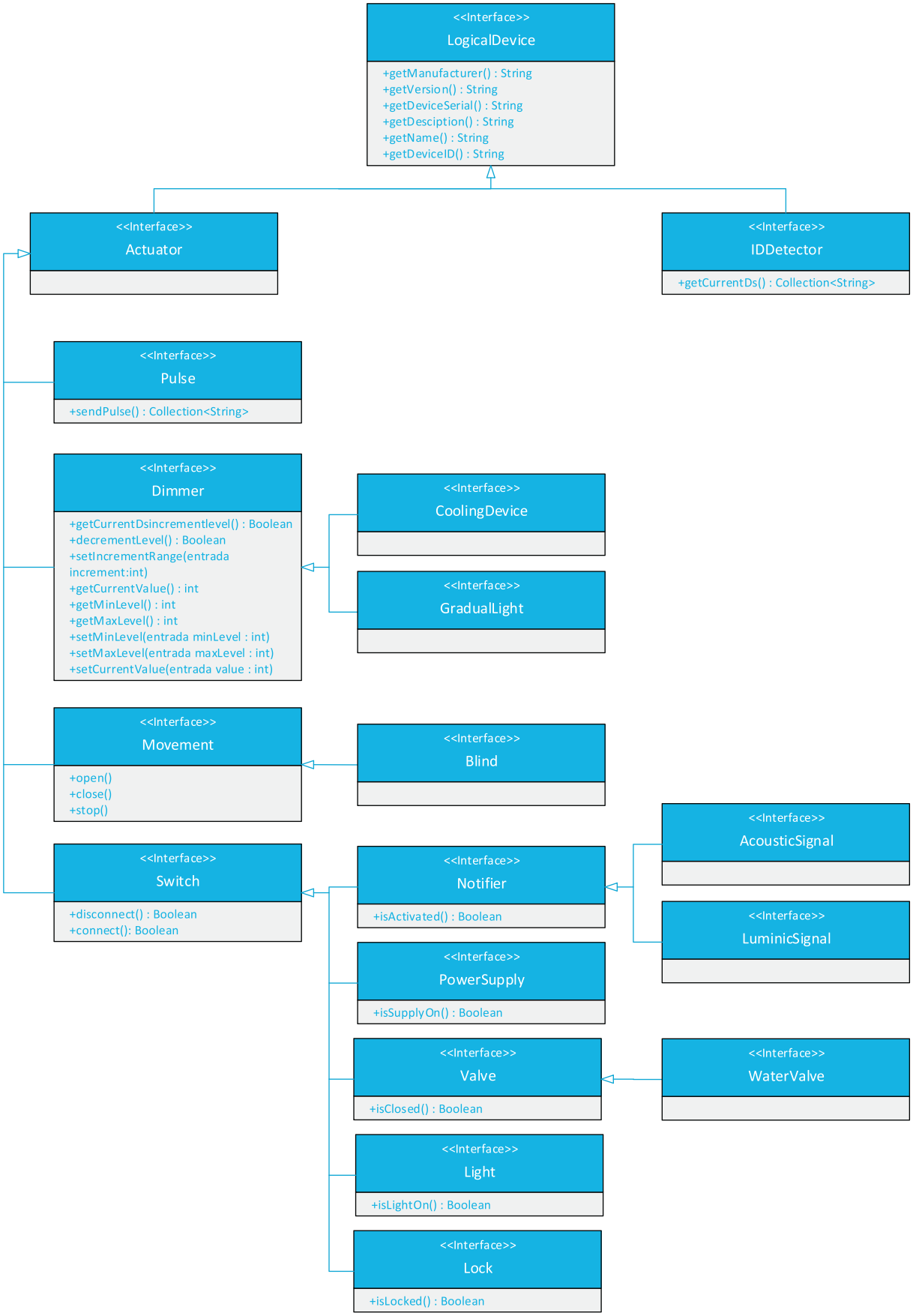

Fig. 5. Actuation class model for Device Abstraction.

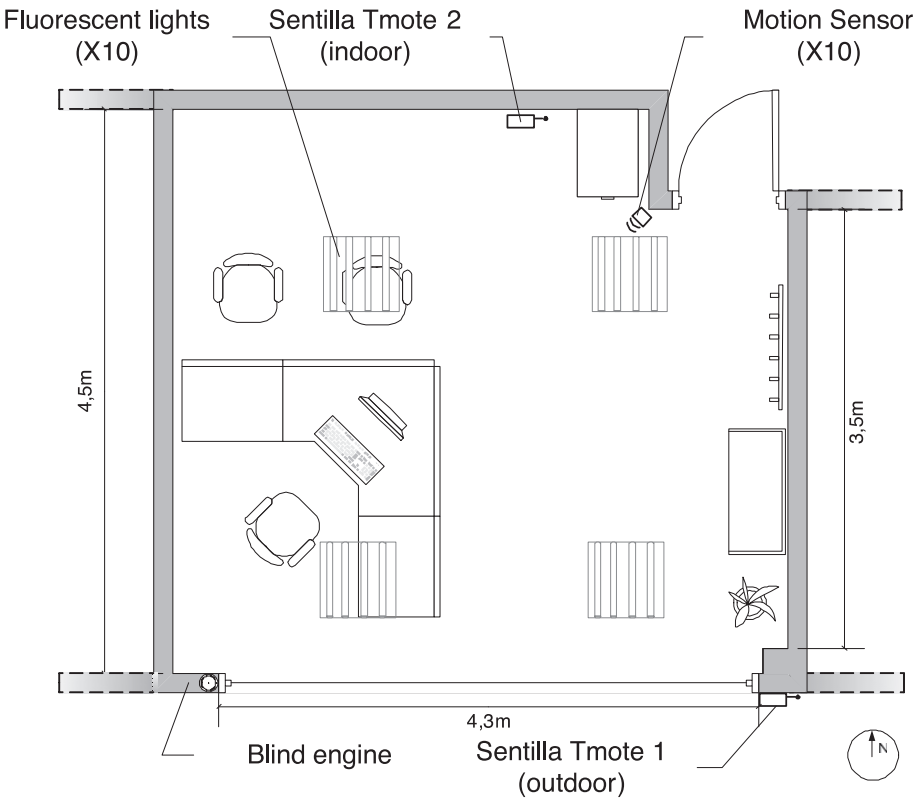

Fig. 6. Room setup.

- IEEE 802.15.4 (ZigBee) protocol bridge connected to the computer in order to communicate with the Sentilla Tmotes,
- X10 transceiver to communicate with actuators and the motion sensor.

The software of the development environment is composed of: Sentilla Work, which is an Eclipse-based IDE for the creation, deployment, and debugging of ubiquitous applications; Weka, a tool suite which facilitates the use of machine learning techniques; and the developed software based on the proposed Reference Architecture and on the Device Abstraction Model as the software paradigm for the device management software.

4.2. Prototype implementation

The software developed by the authors follows the Reference Architecture outlined in Section 3 and is focused on the Perception process detailed in Section 3.2. This software therefore follows the tasks proposed in the Perception process and is responsible for the low-level interaction with devices (Data Collector task), in the form of verification, repair and filtering of the data, and of the storage of the data by aggregating several data sources and by following the Device Abstraction Model. The software itself is distributed between various devices, and hence a number of these tasks are performed by the central computer while others are performed by the Sentilla Tmotes.

4.2.1. Sentilla Tmotes software

Sentilla Tmotes implement a Java Virtual Machine (JVM) called Sentilla Point so that it can run Java applications. In order to access the hardware capabilities of the device, Sentilla offers a Java library which provides access to the data gathered by the sensors and other elements such as extension ports and leads. This low-level software constitutes the device API which is used by the software to access devices. The Java application run by Tmotes accesses sensor data and transmit it via the Zigbee interface. The sensors form a mesh network where motes act as repeaters. This type of network makes it possible to cover wide areas even though Zigbee protocol has a radio scope of a mere 10 m.

When developing this application, it was observed that the quantity of luminosity (measured in luxes) captured by its photosynthetically active radiation (PAR) sensor fluctuated if the fluorescent light of the office was left switched on. The reason for this behaviour is that fluorescent light is constantly switching off and on but it is not perceptible by the human eye due to its high frequency. However, this behaviour posed a problem with the software, so it had to be tackled and included as a part of the *Repairer* task (see Section 3.2.3) at this point. The solution is quite simple: instead of retrieving just one value, n values are retrieved and their average is computed and then sent to the central gateway.

4.2.2. Mote Dashboard

In order to receive all the information from the Sentilla *motes*, an application has been developed, which implements the appropriate interfaces in each case: MotionDetector, TemperatureSensor, BrightnessSensor, or HumiditySensor.

Moreover, the authors have developed software for the central station called Mote Dashboard. This software shows the information that is being received from the motes in real time. This application carries out all the main tasks of the *Perception* process in Section 3.2. A summary of the classes and interfaces of the implementation is shown in Fig. 7:

1. Data collector. This process is responsible for managing the reception of data from the motes, and for using the mote gateway supplied with the development kit, and an X10 controller developed by the authors.
2. Verifier. Three simple verifiers have been developed and perform simple tests, similar to preconditions, over data received from TSR-PAR luminosity, humidity, and temperature sensors (e.g. luminosity ≥ 0).

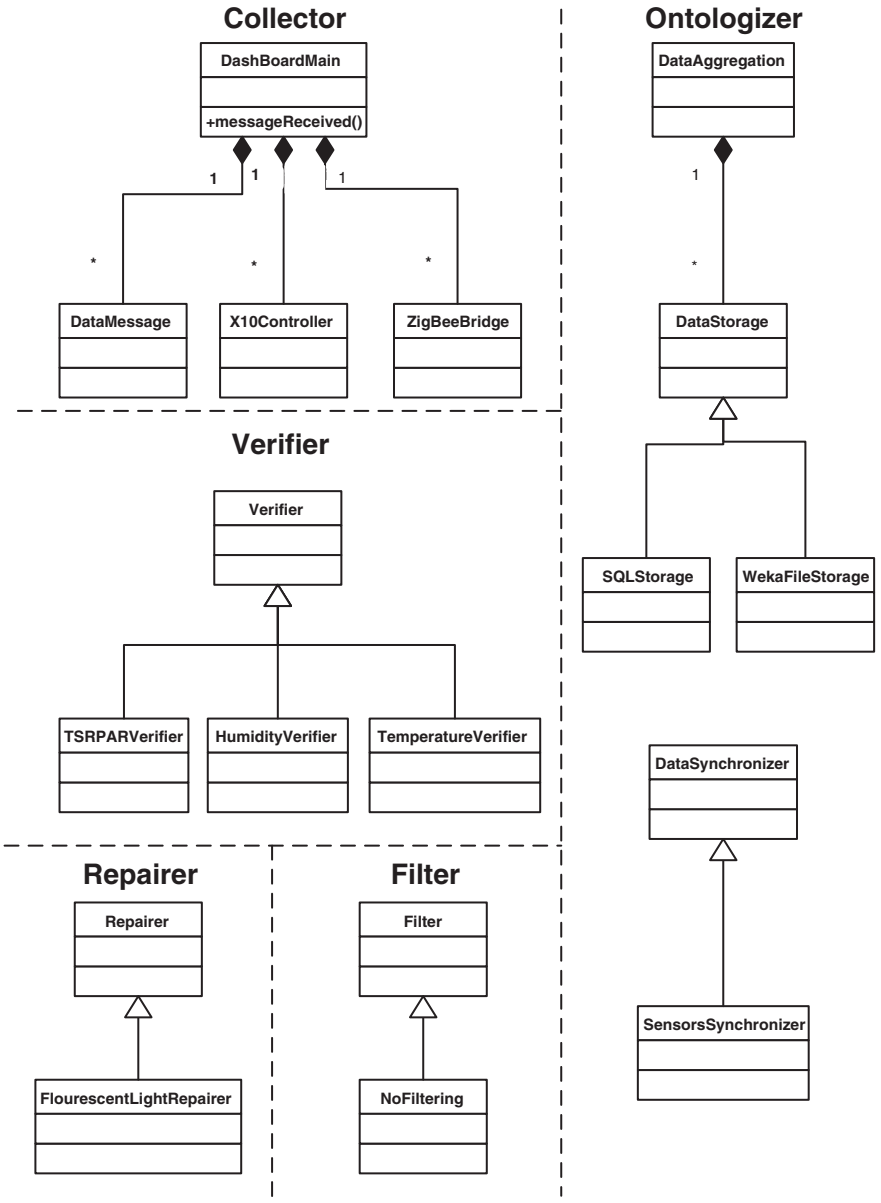

Fig. 7. Summary of implementation of Mote DashBoard software.

3. Repairer. When the verifier task detects erroneous data, as mentioned above for luminosity, the Mote Dashboard software chooses between two options:
   - Replace. If previously correct data was received a short time before, then specific erroneous values are replaced with previous values. The time difference is customizable. Note that the software developed by the authors and run in the Tmotes also repairs wrong values retrieved by the PAR lighting sensor in the class *FluorescentLightRepairer*, due to the fluorescence issue explained earlier.
   - Reject. On the other hand, if previously correct data was received only a long time before, then current erroneous data is rejected.
4. Filter. In this particular case, no superfluous information is received, and hence the *Filter* of this applications does not remove any data.
5. Ontologizer. Finally, correct data is processed and stored by the *Ontologizer* task. This performs three operations:
   - Synchronization. A buffer of data received from every *mote* in SensorsSynchronizer is retained. When information from outdoor and indoor *motes* is taken at approximately the same time, then it is synchronized by the aggregation subtask.
   - Aggregation. Once data is synchronized, it is added to the same instance of the input defined in Section 4.2.4 Input and output.
   - Store. Finally, the Dashboard software stores all instances in *arff* format and also in SQL in the relational database, since these are useful for machine learning tools, such as Weka [24].

Mote Dashboard has other minor functionalities:

- Show information. The information received from the motes is displayed in a Graphical User Interface (GUI) (Fig. 8), and a list of active motes is shown in the mesh network.
- Reset. Resets the Dashboard GUI. This does not affect stored data, but cleans buffers and GUI of all data received.

4.2.3. Database

A relational database has been chosen to store the data retrieved. It currently uses MySQL due to its simplicity and free cost, and since the

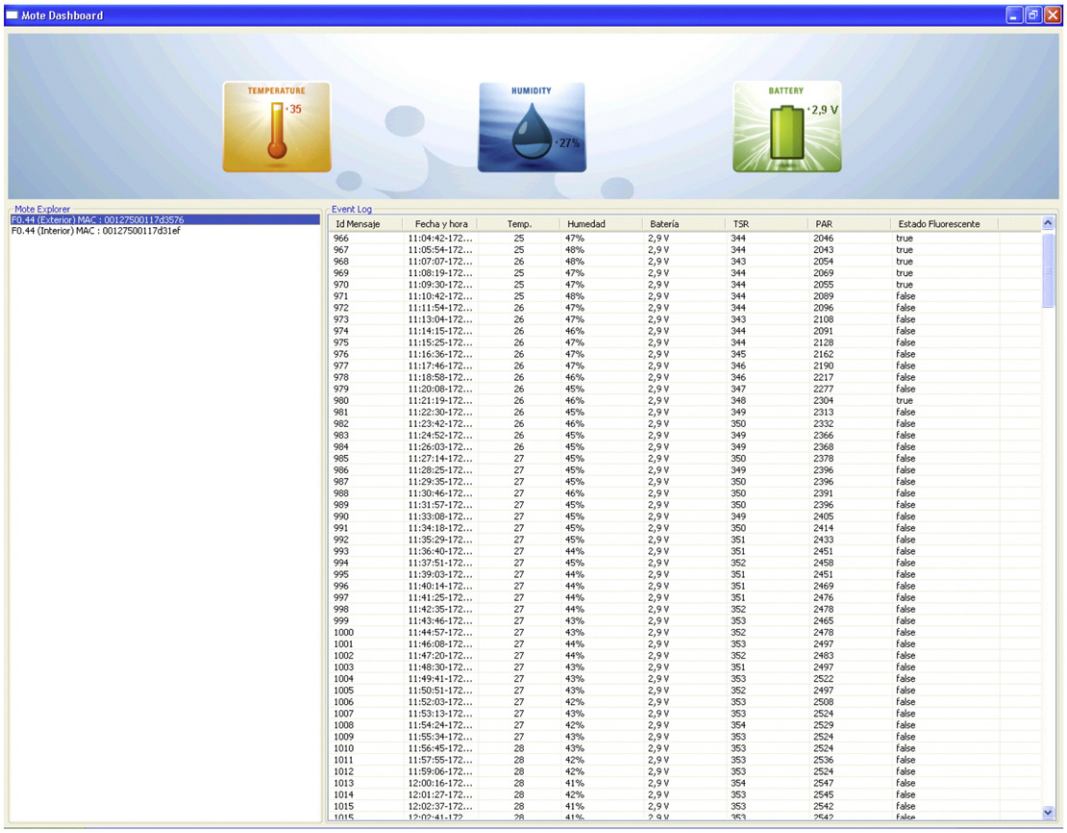

Fig. 8. Mote Dashboard GUI.

specific application does not require a powerful database manager. MySQL organization also offers a Java Database Connectivity (JDBC) connector driver for free, which is a requirement of both the application and of Weka.

4.2.4. Input and output

Table 2 shows the input variable X proposed and two sets of sample input data:

- Outdoor lighting. This variable represents the quantity of light received from the outdoor Sentilla Tmote sensor. Continuous variables from 0 to 1.
- Indoor lighting. This variable represents the quantity of light received from the indoor Sentilla Tmote sensor. Continuous variables from 0 to 1.
- Indoor light state. This variable represents the state of the artificial lights of the room as received from the X10 appliance module. Discrete variables; 0 for lights off, and 1 for lights on.
- Blind state. This variable represents the state of the blinds or curtains. Continuous variables from 0 to 1; 0 for totally closed and 1 for totally open.
- Motion. This variable represents the detection of motion sent by the MS13A X10 device. Discrete variables; 0 for no motion, 1 for motion detected.
- Action over light. This variable represents the action carried out by the user regarding the indoor artificial lights. Discrete variables; −1 lights switched off, 0 no action, +1 lights switched on.
- Action over blind. This variable represents the action carried out by the user regarding the blinds/curtains. Continuous variables; −1 means the user closed it totally, 0 no action, +1 means the user opened it totally.
- Threshold. Represents the current user's lighting preference. Discrete variables;0 represents minimum room lighting, 1 represents maximum room lighting.

4.3. Prototype and development analysis

Once the implementation of the prototype is presented, we analyse the benefits of the approach and the Software Architecture presented. In summary, the results are very satisfying due to the improvements achieved following the paradigms described above. These improvements and benefits were analysed in comparison to the development of this software following general software development architectures, such as OSGi Reference Architecture. The development team is composed of a Software Architect, a senior programmer and a junior programmer. Once the development was finished, another junior programmer added certain functionality to the software. During the development, the improvements that can be indentified include:

1. Design time savings. Once the Software Architect studied the paradigms of the Software Architecture, then the essential issues that had to be tackled were understood, and future development problems were identified from the outset. For example, the synchronization and aggregation of *perceived* data were solved within the design itself.

Table 2
Input sample.

Outdoor lighting [0,1]	Outdoor lighting [0,1]	Outdoor light state {0,1}	Blind state [0,1]	Motion {0,1}	Action over light {−1,0,1}	Action over blind [−1,1]	Threshold [0,1]
0.9	0	0	0	1	0	0	0.5
0.9	0.7	1	0.5	1	1	0.5	0.5

2. Development time savings. Since developers understood the architecture and could work independently from each other, both time and cost were minimized. In this case, one developer was in charge of programming solutions for *Data Collector*, *Verifier* and *Repairer*, since the other developer focused on *Filter* and *Ontologizer*. From the experience of the Software Architect, the developers worked more swiftly.
3. Maintenance time decreased. Since the software was of a modular nature, changing, modifying or improving the software artefacts was child's play. Understanding code was also easier, since every artefact fulfils specific responsibilities.
4. Modularity of development improved. The separation of responsibilities was delivered on separation of software artefacts. This constitutes one of the most obvious benefits.
5. Reusability of software artefacts. Since development was separated by responsibilities, artefacts developed can be easily reusable for further developments. Some of them can also be distributed as components in order to be used by other developers such as data collectors, verifiers of lighting, humidity or temperature measurements, or the fluorescent light repairer.
6. Ease of extension. When another developer, not previously involved in the original version of the software, had to make modifications and improvements, the ease of understanding of the software architecture proved very useful in order to ascertain why certain solutions were adopted, where they were applied, and which points of the *Perception* process were involved.
7. Bug detection and identification improved. Related to reduced time of maintenance, searching for bugs in the source code was straightforward, since any malfunction in the software could be swiftly matched with the corresponding tasks of the *Perception* process.

In short, benefits in the general cycle of software are realized since not only developers accomplish their tasks, but goals of the development are also achieved, which implies that this Software Architecture is highly recommendable. Let the reader notice these improvements.

5. Conclusions

In this work, a general Reference Software Architecture for a SE is presented, which identifies *Perception*, *Reasoning* and *Acting* as the most important areas involved in SE applications.

The alternatives for the techniques used in the implementation of the perception process are presented, explained and referenced.

The objective of this paper is to define the Software Architecture for the *Perception* process as a framework for SEs. This Architecture should be followed in order to accomplish successful solutions for the implementation of this kind of software. This Software Architecture is intended to produce benefits, as shown through verification with a prototype.

Future work on this matter should advance in three ways. The first is to describe Reasoning and Acting processes in the same way as Perception has been described in this paper. Second, a framework based on this Software Reference Architecture could be implemented, which facilitates SE implementation solutions by providing common services and supporting several modules, as information, communication and control systems. And third, innovative techniques could be developed for each of these modules related to previously described processes.

Other important areas which should be taken into account within the framework include user interface and security. Both areas are vital for a complete smart environment experience. Further challenges involve not only enabling the SE to fit user preferences, but also applying it to change behaviour in the individual which could, for example, arm the community with sustainability policies for future SEs, through teaching inhabitants to be more environmentally aware.

Acknowledgements

This research is partially supported by the projects of the Spanish Ministry of Economy and Competitiveness ARTEMISA (TIN2009-14378-C02-01) and Simon (TIC-8052) of the Andalusian Regional Ministry of Economy, Innovation and Science.

References

[1] G.M. Youngblood, E.O. Heierman, L.B. Holder, D.J. Cook, Automation intelligence for the smart environment, Proc. Int. Joint Conf. Artif. Intell. 19 (2005) 1513–1514 Lawrence Erlbaum Associates Ltd.
[2] M. Weiser, The computer for the 21st century, SIGMOBILE Mob. Comput. Commun. Rev. 3 (3) (1999) 3–11.
[3] D. Cook, M. Youngblood, S. Das, A multi-agent approach to controlling a smart environment, Lect. Notes Comput. Sci 4008 (2006) 165.
[4] A. Meliones, D. Economou, I. Grammatikakis, A. Kameas, C. Goumopoulos, A context aware connected home platform for pervasive applications, Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops, 2008, pp. 120–125.
[5] M.A. Zamora-Izquierdo, J. Santa, A.F. Gómez-Skarmeta, An integral and networked home automation solution for indoor ambient intelligence, Pervasive Computing, IEEE 9 (4) (2010) 66–77.
[6] P. Rashidi, D. Cook, Keeping the resident in the loop: adapting the smart home to the user, IEEE Trans. Syst. Man Cybern. Syst. Hum. 39 (5) (2009) 949–959.
[7] M. Jahn, M. Jentsch, C.R. Prause, F. Pramudianto, A. Al-Akkad, R. Reiners, The energy aware smart home, 5th International Conference on Future Information Technology, 2010, pp. 1–8.
[8] C.D. Kidd, The aware home: a living laboratory for ubiquitous computing research, Proc. The Second International Workshop on Cooperative Buildings — CoBuild, 99, 1999, pp. 93–95.
[9] S.S. Intille, K. Larson, J.S. Beaudin, M. Tapia, P. Kaushik, J. Nawyn, T.J. Mcleish, The PLACELAB, a Live-In Laboratory for Pervasive Computing Research (Video), 2005.
[10] T. Weis, M. Knoll, A. Ulbrich, G. Muhl, A. Brandle, Rapid prototyping for pervasive applications, IEEE Pervasive Comput. 6 (2) (2007) 76–84.
[11] D. Bannach, P. Lukowicz, O. Amft, Rapid prototyping of activity recognition applications, IEEE Pervasive Comput. 7 (2) (2008) 22–31.
[12] C. Da Costa, A. Yamin, C. Geyer, Toward a general software infrastructure for ubiquitous computing, IEEE Pervasive Comput. 7 (1) (2008) 64–73.
[13] M. Román, C. Hess, R. Cerqueira, R.H. Campbell, A middleware infrastructure for active spaces, IEEE Pervasive Comput. 1 (4) (2002) 74–83 (IEEE).
[14] J.E. Bardram, Design, implementation, and evaluation of the Java Context Awareness Framework (JCAF), Context (April), Technical Report CfPC, 2005, pp. 1–14.
[15] A. Messer, A. Kunjithapatham, M. Sheshagiri, H. Song, P. Kumar, P. Nguyen, K.H. Yi, InterPlay: a middleware for seamless device integration and task orchestration in a networked home, Fourth Annual IEEE International Conference on Pervasive Computing and Communications PERCOM06, 2(1), 2006, pp. 296–307.
[16] B. Brumitt, B. Meyers, J. Krumm, A. Kern, S. Shafer, EasyLiving: technologies for intelligent environments, Handheld and Ubiquitous Computing, 1927/2000, Springer, 2000, pp. 12–29.
[17] A. Kameas, I. Mavrommati, Computing in tangible: using artifacts as components of ambient intelligence environments, Intelligence (2005) 121–142.
[18] R. Ballagas, A. Szybalski, A. Fox, Patch Panel: Enabling Control-flow Interoperability in Ubicomp Environments, Event (London), 2004.
[19] M. Addlesee, R. Curwen, S. Hodges, J. Newman, P. Steggles, A. Ward, A. Hopper, Implementing a sentient computing system, Computer 34 (8) (2001) 50–56.
[20] D. Cook, S. Das, How smart are our environments? An updated look at the state of the art, Pervasive Mob. Comput. 3 (2) (2007) 53–73.
[21] S.K. Das, D.J. Cook, Designing smart environments: a paradigm based on learning and prediction pattern recognition and machine intelligence, Lect. Notes Comput. Sci. 3776 (2005) 80–90.
[22] J. Li, Y. Bu, S. Chen, X. Tao, J. Lu, FollowMe: on research of pluggable infrastructure for context-awareness, 20th International Conference on Advanced Information Networking and Applications, vol. 1, 2006, pp. 199–204, (AINA'06).
[23] http://thewiki4opentech.com/index.php/OSAmI-ES-Device-Abstraction-HowTo .
[24] I. Witten, E. Frank, Data Mining: Practical Machine Learning Tools and Techniques, Morgan Kaufmann, 2005.
[25] S. Russel, P. Norvig, Artificial Intelligence, A Modern Approach, Pearson, 1995.
[26] M. Muhlhauser, I. Gurevych, Handbook of Research: Ubiquitous Computing Technology for Real Time Enterprises, IGI Global, 2007.
[27] E. Aitenbichler, J. Kangasharju, M. Mühlhäuser, MundoCore: a light-weight infrastructure for pervasive computing, Pervasive Mob. Comput. 3 (4) (2007) 332–361.
[28] M. Fayad, D. Schmidt, Object oriented, application frameworks, Commun. ACM 40 (10) (October 1997).
[29] M. Nucci, M. Grassi, F. Piazza, Ontology-based device configuration and management for smart homes, Neural Nets and Surroundings, vol. 192013. 301–310.
[30] M.J. Kofler, C. Reinisch, W. Kastner, A semantic representation of energy-related information in future smart homes, Energy Build. 47 (2012) 169–179.
[31] S. Helal, W. Mann, H. El-Zabadani, J. King, Y. Kaddoura, E. Jansen, The Gator Tech Smart House: a programmable pervasive space, Computer (2005) 50–60.

[32] S. Steglich, R.N. Vaidya, O. Gimpeliovskaja, S. Arbanowski, F.S. Fokus, S. Sameshima, et al., I-Centric Services Based on Super Distributed Objects Symposium on Autonomous Decentralized Systems, 2003.
[33] A. Hartl, E. Aitenbichler, G. Austaller, A. Heinemann, T. Limberger, E. Braun, M. Max, Engineering Multimedia-aware Personalized Ubiquitous Services, 2002.
[34] V. Stankovski, J. Trnkoczy, Application of Decision Trees to Smart Homes, Springer, 2006.
[35] E. Elnahrawy, B. Nath, Cleaning and querying noisy sensors, Proceedings of the 2nd ACM International Conference on Wireless sensor Networks and Applications, 2003.
[36] S. Wu, D. Clements-Croome, Understanding the indoor environment through mining sensory data, a case study, J. Energy Build. 39 (2007).
[37] S. Zhang, S. McClean, Using Duration to Learn Activities of Daily Living in a Smart Home Environment, Pervasive Computing Technologies for Healthcare (PervasiveHealth)2010. 1–8.