Artif Intell Rev (2014) 42:723–746
DOI 10.1007/s10462-012-9340-4

Combining the real world with simulations for a robust testing of Ambient Intelligence services

Teresa Garcia-Valverde · Emilio Serrano · Juan A. Botia

Published online: 29 April 2012

Abstract This paper proposes a general architecture for testing, validating and verifying *Ambient Intelligence* (*AmI*) environments: *AmISim*. The development of AmI is a very complex task because this technology must often adapt to contextual information as well as unpredictable behaviours and environmental features. The architecture presented deals with AmI applications in order to cover the different components of these kinds of systems: environment, users, context and adaptation. This architecture is the first one that is able to cover all these features, which are needed in a full AmI system. The paper shows that AmISim is able to cover a complete AmI system and to provide a framework which can test scenarios that would be impossible to test in real environments or even with previous simulation approaches. Simulated and real elements coexist in AmISim for a robust testing, validation and verification of the AmI systems, which provide an easier and less costly deployment.

Keywords Ambient Intelligence · Multi-agent based simulations · Testing · Ubiquitous computing

1 Introduction

Mark Weiser envisioned in his work The Computer for the twenty-first Century (Weiser 1995) that "the most profound technologies are those that disappear". He based on these arguments to introduce his vision of ubiquitous computing. The last years have seen substantial progress in Computing systems. Some of their previous constraints (like memory capacity, processor speed, communication bandwidth or even the cost) have been noticeably improved (Garlan

T. Garcia-Valverde (✉) · E. Serrano · J. A. Botia
University of Murcia, Murcia, Spain
e-mail: mtgarcia@um.es

E. Serrano
e-mail: emilioserra@um.es

J. A. Botia
e-mail: juanbot@um.es

et al. 2002). This has caused that computer systems are now widely spread. Furthermore, there have been great advances in some technologies such as wearable computers and devices, wireless networks, sensors devices and control appliances. The improvements regarding to their connectivity, battery life, weight and size make it feasible to integrate computer systems in our daily lives, making ubiquitous environments a reality.

The ubiquitous computing provides a smart environment, where a set of software and hardware elements supports an invisible and nonintrusive interaction between environment and users. This interaction makes it possible to acquire context knowledge from the environment, i.e. Context-Aware, and to provide services that support daily user tasks in a smooth and adaptive way. The convergence of these points is called Ambient Intelligence (AmI). AmI refers to a seamless and invisible computing environment, which is able to provide users with proactive and adaptative services.

AmI aims to create digital environments that are aware of our presence and context through the usage of embedded and nonintrusive devices. These devices generate information about the environment, users and the changes in both of them. This information allows services and applications in AmI systems to adapt to changes. Therefore, key features of AmI systems are *embedded*, *context-aware*, *personalized*, *adaptive* and *anticipatory* (Aarts 2004).

In AmI, access information, communication and services must be ubiquitous, wireless and transparent to users. Furthermore, the interaction between the system and the user must be unobtrusive and natural. Hence, embedded devices to access to the information in a natural and ubiquitous way are needed in an AmI environment. These devices are able to recognize users, the environment and the contextual information. This meaningful information about users and their environments can be used by the applications to adapt to them. In a nutshell, the system is context-aware. Contextual information allows applications to personalize, adapt and anticipate to the environment and to the users' necessities, desires and behaviours.

Considering all the described requirements (embedded, context-aware, personalized, adaptive and anticipatory), it is found that the following models must be considered in AmI:

– *Environment model*. A model that describes the physical world, i.e., buildings, complex objects, sensors, actuators, communications between devices, etc. This model must be flexible and configurable.
– *User model*. A complex multi-agent model that simulates physical and human features, interactions between them, and individual and group behaviours.
– *Context model*. This model gathers, merges, interprets, reasons and stores the contextual information, i.e., all relevant information that surrounds the environment and the users.
– *Adaptation model*. A model capable of supporting applications and services that use contextual information. This contextual information is used to adapt, personalize and anticipate to the environment and users in a reactive or proactive way.

There are a large number of developments and projects that apply the issues of AmI over different contexts. Specifically, most of them are focused on indoor environments. Popular examples of these projects are: Project Aura (Garlan et al. 2002), Oxygen Project (Rudolph 2001), House_n (Intille et al. 2000) or Aware Home (Kidd et al. 1999). These proposals include controlled imitation of real environments (called Living Labs (Gabel et al. 2005)) to test, validate and verify AmI systems.

Testing an AmI system includes detecting, locating and repairing errors on the system. Verification includes checking that the initial requirements have been achieved correctly. Validation checks that the functionality is achieved as expected. Testing, verifying and validating an AmI system by real environments is rather difficult. Moreover, the application of these labs is impractical in several situations, like an environment with thousands of individuals or an emergency scenario.

In order to address these problems, this paper proposes a general architecture for testing, validating and verifying AmI environments: AmISim. AmISim is a layered architecture which considers and covers the four aforementioned models of an AmI system. To the best of our knowledge, AmISim is the first architecture that is able to do this. Furthermore, AmISim enables developers to integrate real and simulated elements from these four models. AmISim provides a user model (essential in AmI system) without the need of interaction with the user. Therefore the number of individuals is not limited. This facilitates to test, validate and verify applications for hundreds of users or more. Finally, the contextual information in AmISim is formally represented using semantic technologies in an easy way.

The paper setup is presented as follows. Section 2 describes other works related with simulation in AmI or specific parts of AmI. Section 3 presents AmISim and the proposed architecture, which covers the requirements for the four models explained above. Section 4 illustrates the use of the AmISim architecture for an advanced AmI service into a scenario of an office building. Finally, Sect. 5 presents the conclusions and future work.

2 Related works

Most of the approaches for the testing, validation and verification of AmI systems are only focused on one or some parts of an AmI system. Nevertheless, to our knowledge no work covers all the requirements and necessities of an AmI system, i.e., the four models explained in the introduction. Therefore, the related works presented here about testing, validation and verification of AmI systems have been separated into four different topics according to their topics. (1) The first topic is Living Labs, which allow us to perform feasibility and usability studies by real life environments. The following three categories involve simulators, which deal with a specific part of an AmI system or emphasize such part. Thus, (2) the second topic is simulators that are focused on the use and analysis of the environment model of AmI systems. (3) The third category describes simulators that explicitly use a context-aware model. (4) Finally, the fourth topic presents the use of multi-agent approaches to simulate AmI systems. These last works usually emphasize the user model.

2.1 Living labs

The most intuitive approach for the testing, validation and verification of an AmI environment is the use of real users receiving feedback from them. A *Living Lab* is a laboratory that consists of real life environments where users and researchers look together for new AmI services. The first serious works to build laboratories that could be used to conduct feasibility and usability studies in AmI started in 2000 (Friedewald et al. 2005). As a result, the HomeLab of Philips was opened on April 24, 2002. Another well-known living lab was developed within the AwareHome Project (Kidd et al. 1999). This living lab has been used to research on context awareness and ubiquitous sensing, individual interaction with the home, person identification and location, and finding lost objects. Another example is the Essex intelligent apartment (iSpace), a test bed for ubiquitous-computing environments developed by Hagras et al. (2004). The iSpace is fitted with several sensors (temperature, occupancy, humidity, etc.) and actuators (such as door actuators, heaters and blinds). The target is to response to user's necessities. The *European Network of Living Labs* is a grown up initiative coming from the own European Living Lab and sponsored by the European Community. The Open Living Labs web[1] shows contact information for Living Labs in about 30 countries.

Developers in a Living Lab can make a compilation of relevant data about the execution of the application interacting with users. Then, these data can be analyzed to evaluate batteries of tests executed over the AmI application. This approach of real-world testing in laboratories has proliferated since the results are very reliable because the application is tested with the final users. However, the main disadvantage of the use of Living Labs is that it is not feasible in applications with hundreds or thousands of users. For example, the study of AmI services in office buildings, such as the location service used as case study in this paper, would be impractical with this approach. Furthermore, a huge economic investment would be needed. In such cases alternative approaches like simulations are useful even when the use of Living Labs is feasible. This is because simulations enable us to test services before their deployment, therefore, the cost of the faults is lower than if they appear in the development.

2.2 Simulators of environments

Building a Living Lab can be slow and costly. Furthermore, it can not be used with large-scale problems. Software developers should be able to develop AmI applications even if they do not have all hardware available yet. With the aim of solving this problem, AmI simulators have appeared in the last years. There are very few simulators designed specifically for this domain (Reynolds et al. 2006). They allow researchers to investigate AmI usability aspects and also functionality issues without physical limitations imposed from using real settings. This section deals with simulators modelling mainly the AmI environment.

Several 3D first-person-shooter (FPS) games, which have appeared for PCs since the late 1990's, have released software development kits (SDKs) allowing programmers to modify these games. Examples of these games are Half-Life, Quake III, and Unreal Tournament (O'Neill et al. 2005). Several proposals for simulating AmI system components try to exploit the 3D graphics engine of these games to model a realistic environment. For example, Quake III Arena is a FPS that models the physical environment in a 3D view. TATUS (O'Neill et al. 2005) is also an example of simulator with this philosophy. It also uses a 3D FPS network game, Half-Life (Coporation 1998), that allows up to 32 players competing in a single game. TATUS introduces an interesting feature; it supports research and development of adaptive software. The features of sensors and actuators are modelled by a SDK for Half-Life. This SDK includes map objects. A map contains information about objects, their names, types and coordinates on the environment. Defining physical objects, invisible and intangible entities like sound or lights, and triggers for modelling events is also possible with this SDK. An additional message XML-based definition tool is used to allow the information to pass between the simulator and the SUT, i.e. the system under test, during the experiment. Other popular cases of AmI environment simulations employing FPS games are UbiWise (Barton and Vijayaraghavan 2002) and QuakeSim (Bylund and Espinoza 2002). They are explained in next section under the topic of simulators that include a context-aware model since this model is their main focus.

The main advantage of these proposals is obtained by working with simulations. This is, quick and cheap prototyping without hardware or environment limitations. The major problem using these simulators is that they do not simulate the user. In these cases, the user is a player of the game (i.e. a person). Consequently, running a suite of experiments is slow and costly because the simulator requires interaction with the user (a player of the game)

[1] Open Living Labs website: http://www.openlivinglabs.eu/.

in every simulation. Therefore, these games limit the number of individuals in the environment, their relations and behaviours. Moreover, these proposals do not utilize a context-aware model. AmISim, presented in this paper, employs an environment modelling tool based on SweetHome3D,[2] a free indoor design application, in order to model realistic environments.

2.3 Simulators with a context-aware model

There are many definitions in the literature about *context*. Several authors had tried to characterize the term context-awareness since the first time that appeared, about 1994. Brown (1996) defines context as those elements of the user's environment that are known by the computer. Other authors like Abowd et al. (1999), define context as any relevant information for the applications about the entities in the environment. Nevertheless, all definitions concur in stating that the use of context-awareness is key for AmI systems in order to adapt to the demands of the users and their environments.

There are several works that explicitly use a context-aware model. An example of simulator with this philosophy is UbiWise (Ubiquitous Wireless Infrastructure Simulation Environment) (Barton and Vijayaraghavan 2002), which uses Quake III Arena. UbiWise is focused on the simulation of computing and communication devices. This simulator emerged from two existing simulators, UbiSim and WISE. UbiSim generates contextual information from raw simulated data in Quake III Arena and this information is processed in the Context Toolkit (Salber et al. 1999), which produces meaningful context to applications. The second simulator, WISE, offers a 2D view for simulating and setting devices, their connectivity, protocols, scenarios and the interaction between users and the environment.

Another example is QuakeSim (Bylund and Espinoza 2002), this simulator is a tool that interactively manages context information in real time. A 3D world and different kinds of context information are simulated in this tool. QuakeSim modifies Quake III Arena, like UbiSim does, to add simulation of sensors and actuators and the Context Toolkit manages all this information.

Morla and Davies (2004) present another approach with a context-aware model. The approach deals with testing and evaluation of network related issues on location-based applications. This proposal includes an environment model and uses the NS network simulator.[3] The authors illustrate their approach employing it to a mobile remote heart and to a lung health monitoring application.

Context-aware software is an emerging kind of application. These applications operate in a highly dynamic environment, where the testing is more complicated. These systems register parts of their context-aware logic in a middleware. However, most of the conventional testing techniques, such as unit tests, do not consider this logic (Lu et al. 2006). Consequently, some approaches to test context-aware applications have been proposed in recent years. Tse et al. (2004) use case generation based on metamorphic testing, a property-based testing strategy. Given multiple executions of the software under test, a metamorphic relation is an expected relation over a set of input data and their corresponding output values. The metamorphic testing is based on checking if a group of test cases satisfy these metamorphic relations. In the same group, Lu et al. (2006) have proposed a novel family of testing criteria that considers some contextual events and their associated actions. Starting from a context-aware data flow, the evolution of contexts is studied in order to obtain context-aware data flow associations

[2] Sweet Home 3D: http://www.sweethome3d.eu/es/index.jsp.

[3] NS network simulator website: http://www.isi.edu/nsnam/ns/.

and testing criteria. The authors illustrate their approach with the construction of adequate test sets and the evaluation of test results for an RFID-based location-sensing system.

These approaches, as the ones presented in the previous section, do not provide a user model, which is essential for testing AmI systems. AmISim, presented in this paper, utilizes semantic technologies to represent the context model. This ensures a common framework for: adding contextual information easily, reusing it, interpreting it, and reasoning over the context in order to integrate entities with a low level of information into other more abstract entities (Nieto et al. 2006). The result is that a more entire flexible context model is provided.

2.4 Simulators based on multi-agent technologies

One reason that justifies the necessity of new solutions for the testing, validation and verification of AmI systems is that one of the most complex parts of simulating these systems is the simulation of users. Despite this, there are few simulators that really manage complex behaviours of users and their relations, the contextual information and its use to adapt to the users. The use of social simulation in general and Multi-Agent Based Simulation (MABS) in particular provide the possibility of easily integrating sociological studies about user behaviours in the simulations. This is because many studies of this nature are carried out through MABS (Qiu and Hu 2010).

There are some works that have followed multi-agent based approaches for some components of AmI environments although they do not cover complete AmI systems. For example, Reynolds et al. (2006) simulate sensors, actuators, and the environment, but without simulation of context or adaptative services facilities. The work of Liu et. al (2006) proposes a scalable framework for prototyping and testing mobile context-aware applications. They use a distributed intelligent multi-agent system to model complex and dynamic user behaviours. For example, the user movement may be simulated based on policies specified by the software tester such as speed or fine grained trajectories. Another ubiquitous computing simulator in which agents are used to model users is presented in the work of Martin and Nurmi (2004). They simulate separately agents, environment and context. The context model is defined with variables of the context and maps. AmISim, presented in this paper, claims to be more complete than the approaches mentioned because it covers environment, context-aware, users and adaptation models.

The field of *Multi-Agent Systems* (MAS) is complementary in several aspects to MABS (Drogoul et al. 2002) and its borders are increasingly blurred. The MAS paradigm has been used widely for the development of AmI applications. This is demonstrated by numerous papers about what has become known as *Agent-based ubiquitous computing* (Mangina et al. 2009). The approach presented in this paper is not developing AmI using MAS, but using MABS to model some of the main parts in AmI systems, such as environment and users, where real tests are too costly or impractical.

3 The AmISim architecture

AmISim is a generic multi-layered architecture that enables to simulate complete AmI systems. On the one hand, the generic architecture enables developers to test, validate and deploy any kind of AmI systems. On the other hand, the different layers cover the four presented models of an AmI system: environment model, user model, context model and adaptation model. The layered architecture allows developers to change easily a model without interfering with the rest of the system.

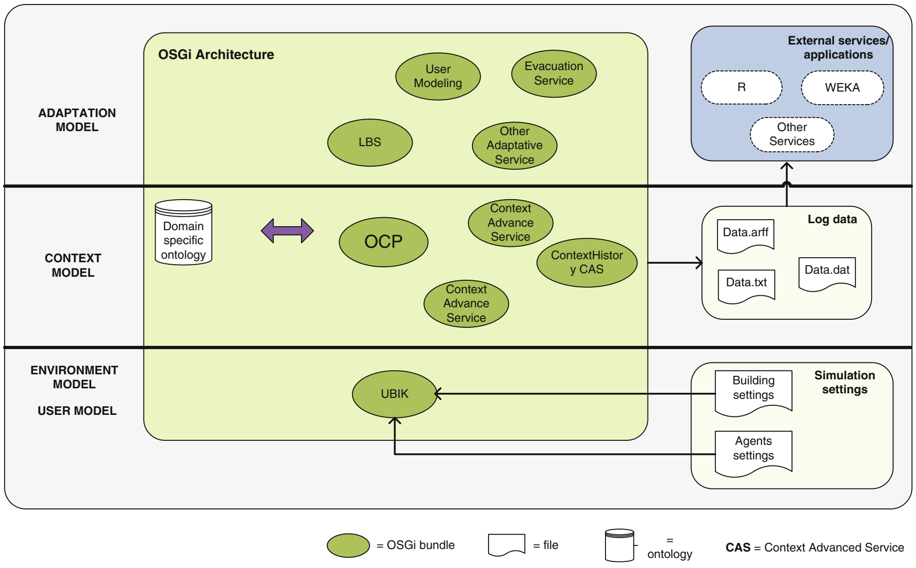


Fig. 1 AmISim architecture

The AmISim architecture is showed in Fig. 1. The multi-layered architecture is based on the Open Services Gateway initiative (OSGi) (Marples and Kriens 2001). OSGi is a framework for Java that supports the development and deployment of modular applications, called bundles. It manages the life-cycle of bundles and provides tools for the discovery, publishing and cooperation of devices and services. Thus, OSGi presents a large amount of advantages according to the features of the AmI systems, like easiness of deployment, portability, heterogeneity among devices and other specific features from the requirements of the ubiquitous computing, like the possibility of restarting a service without stopping the system, which supports the non intrusion requirement of a ubiquitous system.

In the proposed architecture, every component from the different layers is encapsulated in the form of a bundle within the OSGi environment. The lowest level of the architecture contains the bundles that configure the environment and user models. Both models are based on MABS. In the middle level, the bundles of the context model are found. In this level, a context manager bundle and a domain specific ontology are used to model the contextual information. Finally, the upper level contains located bundles which are able to use contextual information to adapt to the environment and the users.

Open Services Gateway initiative enables the inter-bundle communication among all the bundles. Nevertheless, AmISim architecture divides the bundles into levels according to the model they define. This division makes a natural separation between applications or services, their contexts and the technical details about the physical devices (i.e., each model that composes an AmI system). Therefore, the adaptation model knows the context model, but it does not need to know the features of the physical devices, their locations and the rest of low level details.

The use of independent bundles provides AmISim with an important feature: mixing simulated bundles with real bundles. Thus, it is possible, for example, using a real context middleware in the context model to test real services (i.e. real SUTs) whereas a simulated environment and user model are used. Another interesting example of this feature is the possibility of simulating some sensors over a simulated environment but using one or more real sensors in order to test their performance. Merging simulated and real components is one of the most important features in AmISim, since the final deployment over the real world will be easier.

The flow of inter-bundle information is performed by levels. In the environment model, there are simulated sensors and actuators that perceive the changes of the environment and users. These sensors and actuators are defined by bundles. When a change in the environment or user is produced, these bundles send these data to the context manager bundle, i.e., they produce context. The context manager bundle, using these data, generates contextual information and stores it in the historical context information database. Hence, the contextual information is available for the bundles in the adaptation model, i.e., the consumers of context. Using this meaningful information, these bundles can offer services adapted to the environment and the users.

This is the ascending flow of information in the system. In a similar way, there is a descending flow of information. The bundles in the adaptation model can generate new contextual information (for example, a service of machine learning that infers new information from the contextual information) or send to the actuators commands to order them some actions (for example, turn on the lights). In the first case, the context manager bundle stores properly the new contextual information. In the second case, it sends the orders to the actuators abstracting to the applications from the low level.

The next sections describe deeply each model that composes the system in depth. For each model, this work presents a proposal to cover it according its requirements and components. Therefore, the environment and user models are instantiated by Ubik (Serrano et al. 2009), the context model by the Open Context Platform (OCP) (Nieto et al. 2006) and the adaptation model by intelligent services and external tools. Nevertheless, as described above, the architecture of AmISim allows the developer to change one or more of these instantiations without changes in the others.

3.1 Environment model and user model

Multi-Agent Based Simulation can build powerful models to represent real world environments that have a degree of complexity and dynamic (Luck et al. 2005). In this work, the use of MABS for the environment and user models is proposed. Both models represent the real world, i.e., the scenario and the people regarding to this scenario. Therefore, they are suitable for representing with MASB. On the contrary, the other two models can not be represented using this technology because they do not simulate parts of the real world.

The environment model can be simulated in a large number of scenarios thanks to the flexibility of MABS. The simulated physical scenario can also be configurable in order to generate different case studies. Each physical space has particular properties derived from its own nature, for example, a person can walk in a room, but not in a window.

Another important factor in a simulator for AmI environments, as mentioned above, is the model of sensors and actuators since they enable sensing context and acting on it. In AmISim, sensors and actuators can be simulated. Thus, it is possible to simulate a sensor or an actuator and its behaviour, i.e., how the sensor receives the information from the environment, what information it receives, how the actuator sends the information to the environment and what information it sends.

With this aim, sensors and actuators in AmISim can have a wide range of features and properties. In this work, the approach for modelling these devices only describes their most fundamental features. A set of features is defined for setting several different behaviours of sensors and actuators. However, other features can be easily added thanks to the design philosophy of MABS.

Both sensors and actuators can be wearable or static, and they can generate data in periodical intervals or just only sporadically. All these features can be configured for the simulation setting. More examples of the features that can be configured for sensors and actuators are: their usual range of data, the probability or frequency of their occurrences, the probability that unusual data are produced (noise) or the range of data of these unusual occurrences.

Finally, other interesting properties can be defined for the simulation, such as the scope of the sensors and actuators. Out of this scope, sensors and actuators have no influence, i.e., sensors can not capture any data and actuators can not act in the environment. In AmI-Sim, actuators are considered in a similar way that sensors, but they have some particular properties, for example, the effect of an action in the environment.

Regarding the user model, as described in Sect. 2, simulators of AmI or similar environments usually forget or simplify the user model. Nevertheless, AmI is focused on users, their experiences and interactions. Thus, the user model is a fundamental goal in a simulator of this kind. The MABS technology, which is a branch of the social simulation, allows us to integrate sociological studies about the users' behaviours into the user model of AmISim. The result is a user model well defined, flexible and more reliable than previous approaches in simulation. Note that a simulated user model is not an overall solution because users' behaviours are not completely predictable. Thus, more tests would be conducted after the deployment. However, it offers a tool to minimize the error before the deployment. Therefore, the more realistic the user model is, the higher the probability of success will be in the deployment over the real world.

Thus, the behaviour of simulated people is modelled probabilistically in this work. Behaviours are defined as situations that the agent should play in each moment and transitions between behaviours are probabilistic. The underlying model is a hierarchical automaton (i.e. in the highest level there is a number of complex behaviours that the agent may play and, when it is in a specific state, there is subordinate automaton with simpler behaviours that defines that state). Hence, the modelling of each behaviour is treated separately and the modeller is abstracted of unnecessary details. In the lowest level (basic actions), each state is atomic. An agent never carries out two behaviours of the same level simultaneously.

In this approach, the behaviours of a user are divided into three types:

– Monotonous behaviours: the kind of behaviour that a person manifests always approximately in the same time slot, and on a daily basis (e.g. meeting or having coffee).
– Non monotonous behaviours: the kind of behaviour that a person usually manifests, not bounded to a concrete time slot, and repeated within a non constant period (e.g. going to the toilet).
– Any time behaviours: such behaviours are the usual behaviours and they are often interrupted by the others (e.g. working if the simulated person is a worker).

When the user is in a state, a change of state can happen, i.e., a new behaviour is arisen. As described above, these transitions between behaviours are modelled probabilistically. Then, probability distribution functions (pdf) are used according to the described type of behaviours. For example, the behaviour going to the toilet, is a kind of non monotonous behaviour and it can be fitted with an exponential distribution, which represents waiting time models (Garcia-Valverde et al. 2010).

As described above, one of the most important features of AmISim is the ability to merge simulated and real components. This ability is possible thanks to the flexibility of the architecture which enables the coexistence of real and simulated bundles. In the user and environment models this capability is a fundamental key. Recall that simulation is not an overall and final solution. Therefore, the possibility of having simulations composed of real and simulated elements increases the reliability of validating AmI services which are deployed by simulated environments. Thus, in AmISim is possible to achieve more realistic simulations using injection of real elements within the simulated world.

On the one hand, it is possible to simulate some sensors over a simulated environment but using one or more real sensors in order to test their performance as described at the beginning of this section. On the other hand, having real sensors also enables to gather information from real users. In that sense, merging real users with simulated users makes more realistic the behaviour of the user model. Thus, using AmISim is possible to have real users or some real users' features over real environments coexisting with simulated users. This ability is caused by the real context middleware.

The sensors installed in the real environment capture raw data regarding the real users, their movements, their habits and so on. This information is sent to the context middleware bundle which generates contextual information and stores it in the historical context information database. Therefore, as in the case of simulated users, the information of the users is available for the bundles in the adaptation model.

In the previous case, the user model is not simulated, but real like in the final deployment. Note that this case is not the final deployment since some sensors/actuators can be still simulated. Nevertheless, as the introduction described, having real users is unfeasible in several situations (remember the case of an emergency scenario or a scenario with thousands of users) or it is not possible due to some budget requirements, ethical problems, etc. or just because the development is in an initial phase which does not require real users yet.

Even in these cases, AmISim offers some possibilities to merging some real and simulated information, i.e., merging some real information from the users but keeping the simulated user model. As described, according to the type of behaviour a pdf is used. However, the parameters of theses pdfs can be slightly different for each person. For example, the real workers in an office use to have coffee at 9 a.m. while the simulated workers have it at 11 a.m. In both cases the pdf used is the same but the time slot is different.

This information is detected by the sensors and sent to the context middleware. Then, the adaptation model can use it to modify the parameters of the pdf in the user model. In a similar way, it is also possible to modify other parameters of the user model such as the common destinations, schedules, paths followed across the building (for example, a worker who avoids his boss' office), etc.

In both situations, the use of a full real user model or only some real features from it, merging simulated and real components, reduces the distance between the simulated models and the real world. This makes user model more realistic and facilitates the final deployment over the real world.

3.1.1 Ubik

This paper proposes a specific MABS called *Ubik* to simulate the environment and user models in AmISim. Ubik is developed in MASON.[4] There are numerous frameworks for the development of MABS and stands out from them MASON. MASON was chosen rather than other platforms because it is open source (very useful to understand deeply the implemented models), is fast (speed is necessary for models involving thousands of agents), replicable (quality needed to repeat exactly experiments of interest), and especially for being

[4] MASON website: http://cs.gmu.edu/eclab/projects/mason/.

self-contained (Luke et al. 2004). Being self-contained is important because allows MASON to be easily a sub element of another software. In this case, the simulations developed in MASON are a component of the AmISim architecture. Another good option to develop MABS is Repast,[5] which has a large community of users and, according to some opinions (Railsback et al. 2006), is the most complete platform for social simulations in Java. The web of the Open Agent Based Modelling Consortium[6] nowadays lists 21 of these frameworks and comparatives among them.

Using Ubik is possible to simulate different buildings. These buildings, as environment model of an AmI system, can be configured with different number of rooms, several kind of these, windows, doors, stairs, etc.

Each user of an AmI service is simulated in Ubik as an agent. Agents in Ubik have several features, both physical and related to the behaviours. For example, it is possible to define the age, role in the organization, speed of movement, its duties, its leadership skills, its happiness, etc. Furthermore, Ubik enables to configure the different agent's states, for example, working or running, and the actions that can be performed on the environment. All these features compose the behaviour of an agent in Ubik.

Like in the model of sensors and actuators, other behavioural features can be added to provide a wide set of configurations. In this case, this is possible thanks to the design philosophy of MABS in general and MASON in particular. This simulation platform was designed to allow a JAVA programmer to add new features easily (Luke et al. 2004).

Ubik uses a graphical 3D tool, Ubik3D, a Java3D editor to model the physical elements and the people that composes the environment and the user models in an easy way. As mentioned above, different structures of building with rooms, stairs, corridors, several floors, etc. can be simulated. The editor also contains a wide range of objects available to be incorporated to the simulated model, like different domotic devices, diverse furniture or different people. Any indoor space can be modelled using Ubik in an intuitive way without previous experience.

When the environment and user models are modelled, Ubik reads their configuration from Ubik3D and registers the elements into the context model bundle. Furthermore, Ubik defines the behaviour of each element and sets the step for each behaviour in the simulation using MASON.

Ubik is included in the AmISim architecture as a bundle of OSGi. When the elements from the environment and user models are registered in the context model, a bundle for each element is created for updating the changes of this element. This bundle will be a context producer or consumer bundle registered in OSGi (the following section gives further details).

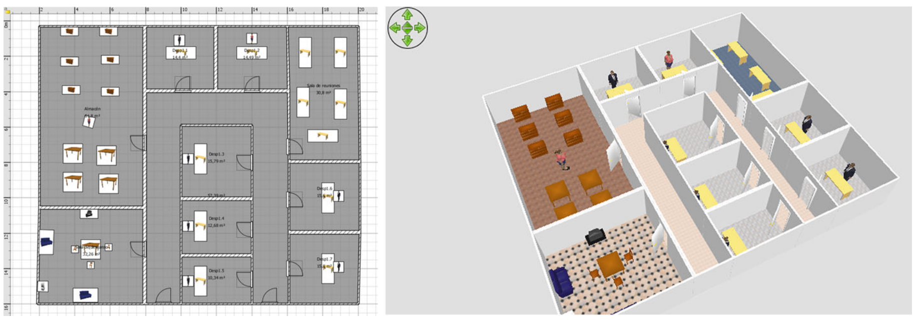
Figure 2 shows two figures of a floor in an office building modelled by Ubik. Both figures show the same floor with different views. There are several rooms connected by corridors. The rooms have one or more desks, chairs and some special rooms have a specific furniture. Finally, different users are simulated in this scenario.

3.2 Context model

Context-awareness is key for AmI systems in order to discover and take advantage of meaningful information from the users and the environment, e.g. user location, weather, devices, user activity, etc. Contextual information must be represented in a specific model to be used. In this work, the context model is built by a context manager (real or simulated). This context

[5] Repast website: http://repast.sourceforge.net/.

[6] OpenABM Consortium website: http://www.openabm.org/.



(a) (b)

Fig. 2 **a** Office floor in 2D. **b** Office floor in 3D

manager must model the contextual information for classifying situations of interest and triggering off the relevant actions at each moment.

The context manager represents the contextual information in an ontological model. Ontologies are essential for building AmI systems because they provide a common framework to share knowledge, interoperating between users and devices and reasoning about contextual information. The usefulness of ontologies in context-aware is widely known (Gu et al. 2004; Wang et al. 2004).

Thus, the context manager is a middleware in which the applications can consume or produce information. It captures the raw information from the devices in the environment model and provides contextual information thanks to the ontology. Furthermore, it obtains new information by inference and merging context. Finally, the applications in the adaptation model can also generate new contextual information (for example, using rule-based inference or machine learning) and send it to the context manager.

Another important component in the context model is the set of context advanced services (CAS). These services are specialized services, which offer elaborated information from the contextual information in the ontology or from the historic of the ontology using the context manager. A more efficient and scalable access to applications and external services is enabled by these services. An example of these services could be offering the number of individuals in a specific place.

There is a special CAS, the *ContextHistory Context Advance Service* (see Fig. 1). This CAS gives a visual interface between the users and the contextual information. Thus, the users can navigate and search visually the historic contextual information.

As well as the other context advanced services, this CAS has the semantic advantages of the semantic web technology given by the context manager. Then, SWRL rules or query languages for semantic data (like SPARQL (Pérez et al. 2009)) can be used in order to search specific contextual information. Thus, the developers are able to do complex queries over the contextual information (e.g. the produced information by the sensor with ID = 7 in the last 2 days at intervals of 30 min). Finally, they can extract the required information to files in the desired format, e.g. plain text or Weka (Hall et al. 2009) format (arff). The processed contextual information is ready to be used for external services like Weka, R (Team 2008) or external services defined by developers.

The context manager and the context advanced services are integrated into AmISim by OSGi bundles. Therefore, the context manager is a bundle that offers context management. Other bundles (in any model) use the context information as producer or consumer bundles of this context manager bundle. The ability of activate/deactivate bundles, adding or removing the context advance services according to the necessities of the system, is enabled by OSGi.

The OSGi framework maintains a central Service Registry. The Service Registry makes use of the WhiteBoard pattern. With this pattern multiple data sources as well as multiple viewers are allowed. It also has the effect of completely decoupling producers and consumers of context. Therefore, the services can be dynamically added and removed from the registry in an easy and unobtrusive way.

3.2.1 OCP

This work integrates the use of *OCP* (*Open Context Platform*) as an instantiation of the context model. OCP is a middleware that provides support for management of contextual information. This middleware represents the information in an ontological model and interprets it using SWRL rules (Horrocks et al. 2004).

Open Context Platform is integrated into the system as a bundle. Therefore, the integration and use of OCP is immediate. A context producer needs to implement: the interface *ContextProducer* to send contextual information to OCP, the interface *BundleActivator* to initialize the bundle, and the methods activate/deactivate to indicate to the producer that the service ContextService is active. Finally, the producer has to register the service in OSGi following the WhiteBoard pattern:

```
# Access to other methods: the bundle can interact with OSGi
private BundleContext bc;
...

#Register the service context producer
bc.registerService(ContextProducer.class.getName(), this,
idService);
```

When the context producer is registered in OSGi, it is able to send contextual information to OCP as follows:

```
private ContextService cs;
...

#Send context information: the new location
cs.setContextItemRelation(''Person'', idPerson, ''located'',
''Room'', idRoom);
```

As described in the example above, the context producer sends to OCP the new location (*idRoom*) of the person *idPerson*. In the same way, when a context consumer is registered in OSGi (in this case, it needs to implement the interface *ContextConsumer*), it can receive contextual information. The context consumer registers the entities in which it has interest. The interfaces to access as a context consumer are:

```
private ContextService cs;
...
#Register for listening the changes of Person
cs.register(''Person'', idPerson, this);
...
```

Then, the consumer is registered to receive the changes of context of the entity *Person*. When there is a change in the context of this entity, OCP notifies it to the consumer calling to the method *notifyContextChange*, which is implemented in the consumer:

```
public void notifyContextChange(ContextEntityItems cs){

#Send context information: the new location idRoom =
cs.getContextItemString( ``Person'' , ``412'', ``Room'');

}
```

Therefore, the context consumer obtains the ID of the room where the person 412 is located.

In OCP there is a generic predefined OWL-based ontology. This ontology should be refined with the domain specific ontology. The ontology contains a historic of contextual information (e.g. it may contain the values of a temperature sensor for the last 3 days). Some applications or services require this history of the context, for example: temporal reasoning, predicting future actions or locations, learning behaviour of the users, etc. Nevertheless, there are applications that can not store this information (e.g., a mobile device with memory constraints). In such cases, the middleware provides them with this service. The generic OWL-based ontology can be accessed from: http://darwin.inf.um.es/ocp.

The use of OCP as a context manager in the context model has an important advantage for the final deployment in the real world: it can be used directly as a real context manager of the AmI system. This means that no change is needed in the context model for the deployment over the real world.

3.3 Adaptation model

The adaptation model is the most flexible model in AmISim. It consists of all applications and services that are offered to the environment and the users. Both services and applications use contextual information from the context model to adapt to the environment and offer personalized and proactive services to the environment and users.

Services and applications in the adaptation model are packaged into OSGi bundles. Then, on the one hand, the service has a public API that can be accessed by the other services in the adaptation model. On the other hand, it is possible to activate and deactivate the service according to the requirements of the system.

The interaction among services and applications is implemented by OSGi. Hence, they can be discovered and used by other bundles. Furthermore, the services and applications of the adaptation model can access to the context model as context producers or consumers (as described the previous section). Thus, a service can extract the needed contextual information (from the context model) and convert it to its own format. In the same way, the service can introduce the contextual information that has been generated.

Open Services Gateway initiative provides AmISim with a great flexibility making possible to introduce any adaptation mechanism into the bundles. Hence, it is feasible to design the adaptation logic by semantic SWRL rules, neural networks, reinforcement learning techniques, mining data stream techniques, machine learning methods using the historical contextual information, etc.

This design in the adaptation model supports other interesting features, for example, the services and applications of AmISim can be remotely controlled, managed and diagnosed without stopping the system. This characteristic is really useful for services that need some kind of offline learning. For example: obtaining offline data from the ContextHistory CAS, using this data for learning tasks, updating the information, and finally, activating the updated service.

3.4 Overall process

This section describes the overall process to test, verify and validate a SUT in an AmI system (see Fig. 3) using the architecture presented.

In the first step, the environment model is defined. This process involves defining the physical world and the domotic devices (sensors and actuators), which compose the real world where the SUT is deployed. As described in Sect. 3.1.1, Ubik is used in this work for this process. Ubik contains a Java3D editor to model the environment and users easily. Therefore, different structures of building can be simulated. Furthermore, the editor contains a wide range of objects available to incorporate to the simulated model.

The next step, step 2, includes the definition of typical user features and behaviours according to the real world. Therefore, if the real world is an office building, users will be workers of the office. However, if the real world is a hospital, the users will be patients, doctors or nurses. Again, for this process, this work uses Ubik. Ubik defines some default behaviours, which can be used; for example, movements in a building, answering a call, having a coffee, etc.

After the environment and user models are simulated, the context is defined in the step 3. This step depends on the context manager employed. Using OCP, the developer has to define the domain ontology and which components in the system are producers and consumer of context. This step is straightforward in OCP because only a few instructions are needed (see Sect. 3.2.1).

In step 4, the adaptation model is defined. This step is highly dependent on the kind of scenario and the requirements of the SUT. Therefore, according to the SUT to be tested,

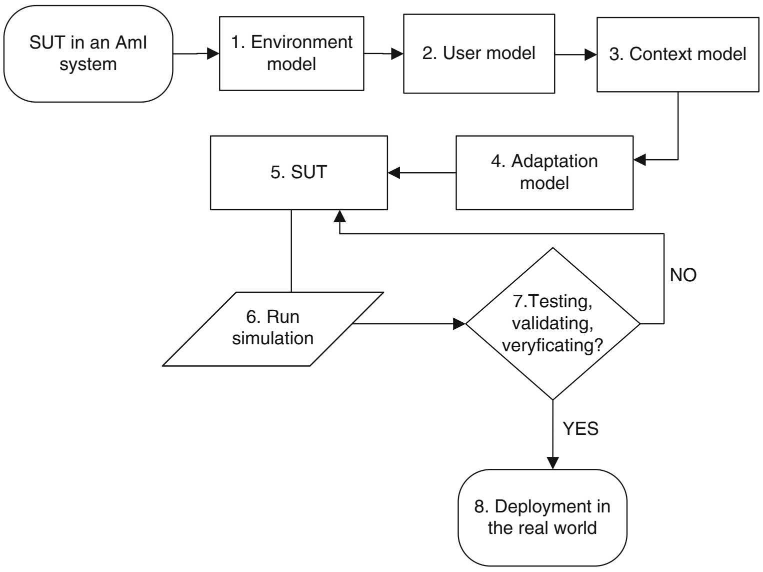


Fig. 3 Overall process

some services will be needed or not. For example, if the SUT is a system to control the air conditioning depending on the location of users, an intelligent service of location will be needed in this model. Finally, when the four models are defined, in the step 5 is defined the SUT and their initial conditions.

The step 6 starts the simulation. In this step, the simulator uses the configuration defined in the previous steps to obtain the simulations. This simulation provides the needed data and views to test, verify and validate the system. Then, the suitability of the SUT can be analyzed and compared with other alternatives in the step 7. If the results obtained from the simulation find faults in the SUT, the developer ought to go back to the step 5 in order to modify the SUT and repeat the next steps. Finally, in the step 8, the obtained SUT, free of faults in its interaction with the four models, can be deployed in a real-world scenario.

4 Example of AmI service in indoor environments: location service

In order to illustrate the effectiveness of the approach presented, an example of an indoor location service is built with AmISim. Location information is one of the most essential and used contextual information in context-aware systems (Weiser 1993). With information about location, systems can provide users with nearest features of interest and adapt to their requirements, i.e., Location-Based Services(LBSs). Users usually have some degree of regularity in their motion. Therefore, the system can predict their destination and acts proactively (Garcia-Valverde et al. 2009).

The LBS presented here can predict the location of workers in an intelligent office building. AmISim is able to simulate the building and sensors and actuators. Furthermore, the behaviour and movement of the workers in the building are also modelled and simulated. These components represent the complete environment and user models of this scenario to simulate and test the LBS.

Nevertheless, note that the context and the adaptation models are real. The context model stores the information about workers' positions and this information is used in the adaptation model to learn behaviours about users' motion and location prediction. Different algorithms and strategies can be built from the simulation to validate the service. When the LBS is tested, it can be deployed directly over the real-world environment. Hence, the use of AmISim reduces errors, problems, effort and, finally, because of that, it reduces costs in the final deployment.

The following sections describe this process and the four models of AmISim in the location service example.

4.1 Environment model

The environment and user models are simulated in this indoor location approach. Ubik bundle is used to simulate these models, as described in Sect. 3.1.1.

The environment model provides an office building that consists of a configurable number of floors, stairs, rooms, corridors, offices, etc. In each office, several workers agents have their desks with their own phones. Furthermore, there are some special rooms: a room to relax with a coffee maker, a warehouse and a meeting room.

There are plenty of technologies which are able to automatically identify and inform about the location of people and objects. One of the most widespread is RFID (Radio Frequency Identification). A RFID system consists of tags, which are usually passive wireless elements, attached to persons or objects. Then, fixed wireless readers, collocated within the environment, retrieve information from those tags, identifying people and objects as they are perceptible by them. Thus, when a person or an object passes near a RFID antenna, the location of this person or object is associated with that antenna.

Radio Frequency Identification is used in this scenario to build a location service. Nevertheless, not all places in a building are usually sensorized (for reasons of budget, privacy, etc.). In such situations, the information generated by the RFID antennas can be used to perform machine learning and predict user location more effectively. With this aim, this service predicts the location of the workers in an indoor environment. Each worker wears a RFID tag, which identifies him, and the building is equipped with RFID antennas in some places where locating people is interesting. When a worker passes near an antenna, a message is generated reporting her current position to the context model. Therefore, a list of the antennas that the worker traverses is generated, i.e., a trace.

The office building and their elements are designed using the graphical tool of Ubik, Ubik3D. The Fig. 4 shows some views of the environment model. Figure 4a shows a 2D view and Fig. 4b displays the same view but in 3D. The RFID antennas are represented in blue. The building is divided into three zones $\{Z1, Z2, Z3\}$. This division is made in order to show the fact that, in a real environment, the workers of the zone Zi usually will go to destinations in

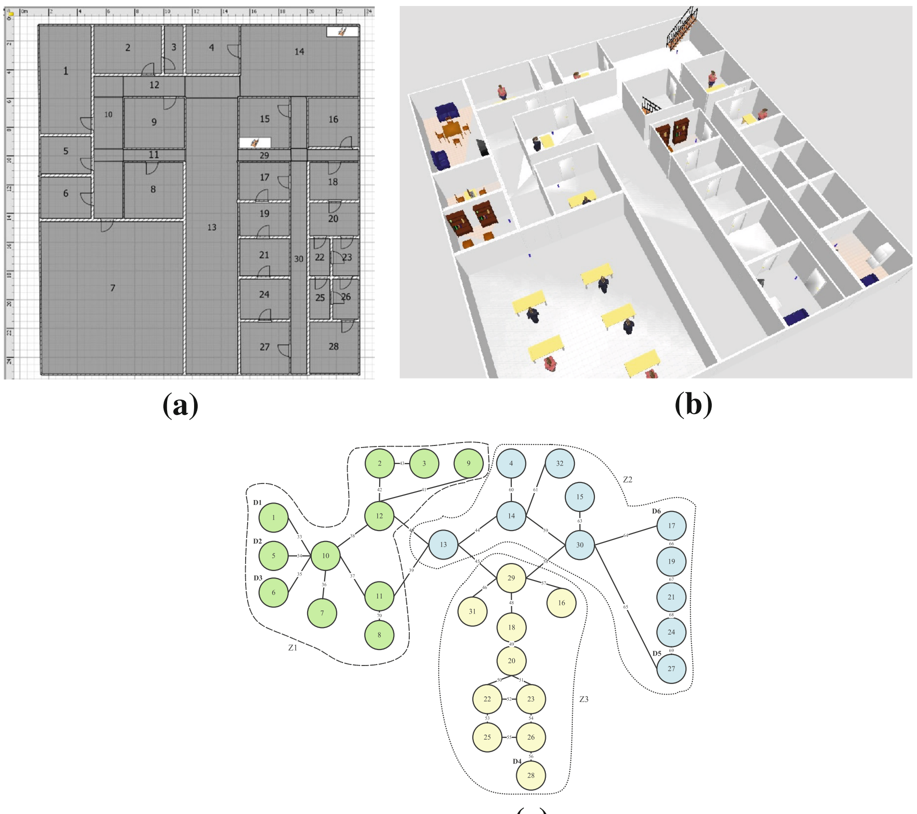


Fig. 4 **a** Office floor in 2D. **b** Office floor in 3D. **c** Graph of the floor

this zone, the nearest destinations. For instance, if the user has a desk in zone 1, she usually will go to the relaxation room located in zone 1.

A destination is every location where a worker needs to be located in the building and there is not any antenna. In this example, the destinations, $\{D1, D2, D3, D4, D5, D6\}$, are the special rooms and they are spread out over the three zones. D1, D4 and D5 are the relaxation rooms of the zones 1, 2 and 3, respectively (the relaxation rooms are areas with coffee-maker, rest-room and a terrace for smokers). In the destination D2, there is a meeting room and the other destinations, D3 and D6, are warehouses.

When the Ubik bundle is registered in OSGi, the elements of the environment model are registered in the context model. Then, the context model builds an abstract schema of the building as a graph (the construction of the graph is described below). The abstract schema represents the static context about the environment. This static contextual information is used to know the structure of the building and to calculate paths through the building. The Fig. 4c shows the graph of the building. The three zones of the building are represented in the graph. Z1 is the green zone, Z2 the blue zone and Z3 the yellow zone.

4.2 User model

As described above, the user model is a fundamental goal in this kind of simulator (focussed on users). We propose MABS, by Ubik, to achieve this goal. With the use of Ubik, the behaviours of the users are modelled and simulated by using probabilistic automata.

Since users are workers is possible to assume a basic state *go to work*. The worker remains in the basic state most of the time. This kind of behaviour is an Any time behaviours. The action of this basic state can be interrupted in order to go to other states and change the behaviour. These behaviours are of two types: Monotonous behaviours and Non monotonous behaviours. Monotonous behaviours, like *having a coffee* are usually in the same time slot, whereas Non Monotonous are not bounded to a concrete time slot, for example, *answering a phone call* or *going to the bathroom*.

Monotonous behaviours must be activated at specific time hours or within specific time intervals. For example, in the case of *having a coffee*, a new necessity of changing to such state will be generated at the moment of having coffee. In the case of *answering a phone call* or *going to the bathroom*, the necessity is not bounded to a specific time interval. The distribution that models these transitions is bounded in this time slot.

In order to generate transitions in real time, pdfs are used according to the type of the behaviour and its features. These distributions are member of the exponential family. It has been shown that these distributions are suitable to model these behaviours (Garcia-Valverde et al. 2010).

Each agent always starts its behaviour in the same point (its desk) and moves to different places in the building: relaxation room, meeting room and warehouse. These behaviours are made in a specific proportion and they depend on the zone where the agent has its desk. Users usually go to nearest destinations to their desks.

These behaviours imply movement through the office building. In order to solve the simulated movement of the users, the agents have between their beliefs the abstract schema of the map of the building (see Fig. 4c). Then, agents can access to the structure of the building and use it to calculate the best path for their movements using the Dijkstra algorithm (Dijkstra 1959). When the agent knows the best path to a particular destination, this path is added to its beliefs, in a similar way as in a real scenario. This service of paths is provided from the context model by two context advanced services (see following section).

As described below, thanks to the flexibility of the architecture of AmISim, changing the strategy of the calculation of the paths using other algorithms is simple and intuitive. However, note that these behaviours can not be completely realistic because it is impossible to predict entirely the real users' behaviours. Despite that, simulations help to test and reduce risks of errors in the final deployment.

4.3 Context model

The context model is managed by the OCP bundle, as described in Sect., 3.2.1. Moreover, in this example, OCP includes two context advanced services, the special ContextHistory CAS and a domain specific ontology. The context advances services are integrated by OSGi bundles and can communicate with the OCP bundle as context consumer/producer. The generic OWL-based ontology is refined with a domain specific ontology, which includes information about the building, the RFID devices and the workers. The ontology defines all the information regarding the environment and user model. Thus, it defines the elements of the building, the RFID antennas, their ranges and their locations, the workers and the RFID tags that identifies them.

As described above, the ontology defines the whole structure of the building. When the Ubik bundle is registered in OSGi, the elements of the building are registered in OCP and stored in the ontology. This includes the information about the floors, rooms, doors, the location of the RFID antennas, and the rest of the elements of the building (desks, phones, etc.).

There is a context advanced service bundle in the context model, which uses this contextual information to build the abstract schema of the building (see Fig. 4c). The abstract schema of the map of the building defines the structure of the building with areas big enough, as room, corridor, etc., but smaller areas like a cell of a grid are not considered because this would be very inefficient. This bundle builds the schema by an undirected graph with weights that models the structure of the building. Hence, the topology of the building is based on a graph $G = (V, E)$, where V is the set of rooms, corridors and stairs, E is the set of doors and crossings and $v, v' \in V$ are connected by the edge $e \in E$ if they are connected spaces in the building. Each edge has a weight X that defines the distance between the nodes that connects.

When the bundle has built the graph, the ontology is updated for setting the connections between the areas. For example, if there is an edge between $Room1$ and $Room2$ connected by $Door1$, the bundle will create a new concept $ConnectionAmongDwellings$ with the following properties: $ConnectionWith_1 = Room1$, $ConnectionWith_2 = Room2$, $WayOfLink = Door1$. Thus, the graph is stored in the ontology of OCP.

The other CAS bundle in the context model uses the stored graph to calculate the possible paths between all the rooms (nodes of the graph). The paths are calculated according to the shortest path between an origin and a destination using the Dijkstra algorithm. Hence, as described in the user model, the agents know the structure of the building and they are able to move from their desks to other destinations.

The construction of the abstract schema and the paths is divided in two bundles in order to provide flexibility to the system. Thus, it is possible to define other algorithms to calculate the paths in the building. For example, instead of the use of the shortest path, it is possible to define other strategies: the shortest path avoiding the $Room3$ (boss' desk), the path that never uses the stairs and so on. The modeller only needs to integrate a bundle with the desired strategy and this bundle will use the abstract schema for building the paths according to its strategy.

Table 1 Behaviour patterns of the users

Zones	Destinations					
	D1	D2	D3	D4	D5	D6
Z1	20	10	5	1	1	1
Z2	1	10	5	1	20	1
Z3	1	10	1	20	1	5

Finally, the ContextHistory CAS gives sets of data for learning tasks from the contextual information. When the environment and user models are simulated, the workers move in order to do their daily routines. Meanwhile, the activated antennas in the paths of the workers register their positions, i.e., the traces of the users in the building. Thus, the ContextHistory CAS is able to provide the set of traces of workers in order to train and validate offline techniques for location prediction in the adaptation model.

4.4 Adaptation model

In the adaptation model, three external services are proposed to location prediction: Hidden Markov Models (HMM) (Rabiner 1989), PART (Frank and Witten 1998) and C4.5 (Quinlan 2003) algorithms. The three methods model the users' movement patterns. Ubik is used to generate the synthetic data set that allows the model to be trained and validated. For this reason, the three external services need to use the ContextHistory CAS bundle in order to obtain the synthetic data from the historical context information, as described in the previous section.

Here, Ubik simulates workers at work. Thus, it simulates also movements of workers at the building. By using a priori knowledge about the location of their offices, appropriate trajectories between their offices and other places can be generated and transformed them into traces. This simulation of the movements is made according to the described user model (see Sect. 4.2). The frequency of the behaviour patterns for a user in a month are described in Table 1. Finally, users are uniformly spread between the zones.

As described in the environment model, there are special rooms: D1, D4 and D5 are relaxation rooms of the zones Z1, Z2 and Z3, respectively; D2 is a meeting room in zone Z1; and finally, D3 and D6 are warehouses of the zones Z1 and Z2. Therefore, according to the Table 1, the relaxation room D1 is visited 20 times by the users in the zone 1, the meeting room D2 is visited 10 times by these users and the warehouse D3 is visited 5 times. They visit only once the other destinations. In a similar way, the relaxation room of the zone 2 (D5) is visited 20 times by the users of the zone 2 and they goes to the warehouse D3 5 times, to the meeting room 10 times and to remaining destinations once. Finally, the behaviour in the zone 3 is similar: the users of zone 3 go to the meeting room 10 times, to the relaxation room (D4) 20 times, to the warehouse D6 5 times and once to other destinations.

Note that all places are not sensorized and, therefore, it can not be assured at any precise moment t the location of an user. In these lack of information situations, the only available data are the ones generated by the antennas when the workers go past near them. Data are compounded by traces. Traces are a list of antenna locations that the agent passes through until its final destination is reached. Each time the worker leaves the desk, she is supposed to go somewhere. The list of antennas that the worker traverses, generates a trace. Then, the set of traces of a simulation is represented as

$$\begin{array}{ll}\{ & \langle (t_{1,0}, a_{1,0}), (t_{1,1}, a_{1,1}), \ldots, (t_{1,n}, a_{1,n}, l_1) \rangle, \\ & \langle (t_{2,0}, a_{2,0}), (t_{2,1}, a_{2,1}), \ldots, (t_{2,m}, a_{1,m-1}, l_2) \rangle, \\ & \ldots \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \}, \end{array}$$

where $(t_{i,j}, a_{i,j})$ indicates that a user passed near $a_{i,j}$ antenna, at time $t_{i,j}$. Furthermore, l_k will be a concrete final location (e.g. a warehouse or a meeting room) if $t_{k,r} < \tau$, where τ is the threshold for a given location and $t_{k,r}$ the time that the user spends in a specific location. Such traces of antennas are analyzed to detect the most frequent destinations of users.

The ContextHistory CAS bundle classifies groups of traces to learn models according to the most frequent destination. The set of all users' traces in a month according to the above pattern makes the set of synthetic data. $2/3$ of the set is used for training and the other $1/3$ of data is used for validation. The ContextHistory CAS bundle is in charge of providing the data files with the $2/3$ of the set for training services.

The implementation of the HMM is carried out with JAHMM (Francois 2006), a Java implementation of HMMs related algorithms. An HMM is a doubly stochastic process with an underlying stochastic process that is not observable (it is hidden). The basic idea is to define a set of hidden states, where each state has an associate probability distribution over a set of observables states. In each hidden state, observable information is shown according to these probability distributions. The hidden states are interconnected by probabilistic transitions between them. The HMM parameters are estimated by using the Baum-Welch algorithm. Thus, learning the HMMs consists on finding the maximum-likelihood estimator of the parameters of the HMM given a set of observed feature vectors (group of training samples for a single destination).

The other two external services to predict location use the C4.5 and the PART algorithms by the Weka data mining tool. C4.5 algorithm has been implemented using the J48 implementation in Weka. C4.5 builds decision trees using the information entropy. Decision trees are a classic way to represent information and offer a fast and powerful way to express structures in data. The trees are built using the synthetic training sequences classified by the destination. C4.5 determines in each node the attribute that best divides the data set according to the information gain.

As well as the service that uses C4.5, the service that uses PART also is built using the implementation of PART in Weka. The PART algorithm is a mixing between the C4.5 and the RIPPER algorithms. PART builds a rule and removes the instances that this rule covers. Then, the algorithm continues recursively adding rules until there are not any instance.

For the three techniques, the models of users' movements are learnt by using the data set provided by the ContextHistory CAS bundle, i.e., the preprocessed groups of traces ordered by their destinations. The final learnt models by the different techniques are built in bundles in order to offer the models of the users' behaviours to the rest of the system.

A last bundle is built in the adaptation model in order to predict the destinations. This bundle uses the learnt models and the validation data set ($1/3$ of data set described above) to predict the destination in each trace and validate such prediction. Besides, in the context model, the bundles that model the users' behaviours are separated from this bundle for flexibility reasons. Therefore, it is possible the use of the users' models for other purposes, not only for location prediction.

Finally, in order to show the effectiveness of the proposed framework, the three learnt models are evaluated over the synthetic validation data set. The error rate and precision are shown in Tables 2 and 3. The error rate is the number of false negatives divided by true

Table 2 Error rates

Error rate	Destinations						
	D1	D2	D3	D4	D5	D6	Global
HMMs	20.09	65.664	47.26	1.58	25.17	24.84	30.77
PART	5.5	36.9	100	7.14	8.219	30.13	27.25
C4.5	6.78	27.45	100	0	7.30	28.76	23.65

Table 3 Precision

Precision	Destinations						
	D1	D2	D3	D4	D5	D6	Global
HMMs	0.565	0.573	0.318	0.997	0.926	0.756	0.689
PART	0.585	0.697	0	0.97	0.744	0.81	0.634
C4.5	0.604	0.683	0	1	0.906	0.765	0.659

positives plus false negatives. The precision is measured as true positives divided by the sum of true positives and false positives.

Note that the HMM approach is robust with respect to an unbalanced training data distribution. For example, the HMM for D_3 has a small training set in comparison with the others. However, the learning process is not affected by this fact. Actually, comparing HMMs approach with the other machine learning approaches, this particular issue can be observed. For example, when PART and C4.5 algorithms are used, the prediction of D_3 and D_6 gets worse. Nevertheless, other destinations clearly benefit from this (better error rates are obtained for them) because they never predict D_3 as destination. Such destinations, D_1 and D_2, directly compete against D_3.

Additional tests were performed with all destinations equally probable. In such case, error rates were similar for the three techniques. Nevertheless, the technique based on HMMs still gets a better precision. This gives to the system based on the models from HMMs a better adaptation to changes in the workers' pattern behaviours, which is common in a dynamic scenario such as the proposed here. Thus, the example presented illustrates the flexibility and effectiveness of this approach since using the architecture is easy to choose the algorithm most suitable according to the requirements of the specific system.

5 Conclusions and future work

This work presents the AmISim architecture. AmISim integrates the basic elements of AmI systems in order to provide a framework for testing, verifying and validating this kind of systems. Four models are basic in AmI systems: environment, user, context and adaptation model. The environment includes the physical surroundings of people interacting with the AmI system. This model also includes sensors and actuators. The user model represents the user in the simulation. In AmI systems, users are the key element and all the system is designed around them. The context model is needed to provide users with services that adapt to them in the adaptation model.

The general approach proposed in this work integrates simulation and real models to test, validate and verify services under development. The proposal attempts, on the one hand, to reduce the errors and effort usually involved in the final AmI deployments over the real world. Therefore, the cost of the deployment is also reduced. On the other hand, AmISim makes feasible to test scenarios whose testing would be impossible in real environments, like an environment with thousands of individuals or an emergency scenario. We have illustrated the use and advantages of the approach by its application to a predictive Location Based Service, which is able to predict the user locations to provide them nearest features and services adapted to their requirements.

Future work will focus on improving the simulated features in order to include more complex elements, like weather conditions, advanced devices and more complex user behaviours. Another promising future work is to adapt AmISim to implement an infrastructure for mobile environments. In this line, we are trying to integrate real mobiles devices in the simulation. This inclusion will offer more realistic models due to the injection of real features from the environment and the users. The flexibility of the AmISim architecture will allow us to carry out this future work to reduce the distance between the simulated and real worlds.

Acknowledgments This research work is supported by the Spanish Ministry of Science and Innovation under the grants AP2007-04269 and AP2007-04080 of the FPU program and in the scope of the Research Projects DIA++ (TRA2009_0141) and TIN2011-28335-C02-02, and through the Fundación Séneca within the Program “Generación del Conocimiento Científico de Excelencia” (04552/GERM/06).

References

Aarts E (2004) Ambient Intelligence: a multimedia perspective. IEEE MultiMed 11(1):12–19

Abowd GD, Dey AK, Brown PJ, Davies N, Smith M, Steggles P (1999) Towards a better understanding of context and context-awareness. In: HUC ’99: Proceedings of the 1st international symposium on Handheld and ubiquitous computing, Springer, London, vol 1707, pp 304–307

Barton JJ, Vijayaraghavan V (2002) Ubiwise: a ubiquitous wireless infrastructure simulation environment, tech. report hpl2002-303. HP Labs 2002

Brown PJ (1996) The Stick-e document: a framework for creating context-aware applications. In: Proceedings of EP’96, pp 259–272

Bylund M, Espinoza F (2002) Testing and demonstrating context-aware services with Quake III Arena. Commun ACM 45(1):46–48

Coporation V (1998) Half life. Half Life http://en.wikipedia.org/wiki/Half-Life_(video_game)

Dijkstra E (1959) A note on two problems in connection with graphs. Numerische Mathematik 1:269–271

Drogoul A, Vanbergue D, Meurisse T (2002) Multi-agent based simulation: Where are the agents? In: MABS, pp 1–15

Francois JM (2006) Jahmm-hidden markov model (hmm): an implementation in java. URL: http://runmontefioreulgacbe/francois/software/jahmm

Frank E, Witten IH (1998) Generating accurate rule sets without global optimization. In: Proceedings of the fifteenth international conference on machine learning, University of Waikato, Department of Computer Science, pp 144–151

Friedewald M, Da Costa O, Punie Y, Alahuhta P, Heinonen S (2005) Perspectives of Ambient Intelligence in the home environment. Telematics Inform 22(3):221–238

Gabel O, Litz L, Reif M (2005) NCS testbed for Ambient Intelligence. In: 2005 IEEE international conference on systems, man and cybernetics, IEEE, vol 1, pp 115–120

Garcia-Valverde T, Garcia-Sola A, Lopez-Marmol F, Botia JA (2009) Engineering Ambient Intelligence services by means of MABS. International conference on practical applications of agents and multi-agent systems

Garcia-Valverde T, Campuzano F, Serrano E, Botia J (2010) Human behaviours simulation in ubiquitous computing environments. In: Workshop on multi-agent systems and simulation at MALLOW

Garlan D, Siewiorek DP, Smailagic A, Steenkiste P (2002) Project Aura: toward distraction-free pervasive computing. Pervasive Comput 1(2):22–31

Gu T, Pung HK, Zhang DQ (2004) Toward an OSGi-based infrastructure for context-aware applications. IEEE Pervasive Comput 3(4):66–74

Hagras H, Callaghan V, Colley M, Clarke G, Pounds-Cornish A, Duman H (2004) Creating an ambient-intelligence environment using embedded agents. IEEE Trans Intell Syst 19(6):12–20

Hall M, Frank E, Holmes G, Pfahringer B, Reutemann P, Witten IH (2009) The WEKA data mining software: an update. ACM SIGKDD Explor Newsl 11(1):10–18

Horrocks I, Patel-Schneider P, Boley H, Tabet S, Grosof B, Dean M (2004) SWRL: a semantic web rule language combining OWL and RuleML. Tech. rep., National Research Council of Canada, Network Inference, and Stanford University

Intille S, Larson K, Kukla C (2000) House_n

Kidd CD, Orr R, Abowd G, Atkeson CG, Essa IA, MacIntyre B, Mynatt E, Starner TE, Newstetter W (1999) The aware home: a living laboratory for ubiquitous computing research. Lecture Notes in Computer Science, pp 191–198

Liu Y, OGrady MJ, OHare GMP (2006) Scalable context simulation for mobile applications, Lecture Notes in Computer Science, vol 4278, Springer Berlin Heidelberg, chapter 42, pp 1391–1400

Lu H, Chan WK, Tse TH (2006) Testing context-aware middleware-centric programs: a data flow approach and an RFID-based experimentation. In: SIGSOFT '06/FSE-14: Proceedings of the 14th ACM SIGSOFT international symposium on foundations of software engineering, ACM Press, New York, pp 242–252

Luck M, McBurney P, Shehory O, Willmott S (2005) Agent technology: computing as interaction (a roadmap for agent based computing). University of Southampton, Southampton

Luke S, Cioffi-Revilla C, Panait L, Sullivan K (2004) MASON: a new multi-agent simulation toolkit. In: Proceedings of the 2004 swarmfest workshop

Mangina E, Carbó J, Molina JM (eds) (2009) Agent-based ubiquitous computing, Atlantis Ambient and Pervasive Intelligence, vol 1. Atlantis Press, New York

Marples D, Kriens P (2001) The open services gateway initiative: an introductory overview. IEEE Commun Mag 39(12):110–114

Martin M, Nurmi P (2006) A generic large scale simulator for ubiquitous computing. Annual international conference on mobile and ubiquitous systems, vol 0, pp 1–3

Morla R, Davies N (2004) Evaluating a location-based application: a hybrid test and simulation environment. IEEE Pervasive Comput 3(3):48–56

Nieto I, Botía JA, Gómez-Skarmeta AF (2006) Information and hybrid architecture model of the OCP contextual information management system. J Univ Comput Sci 12(3):357–366

O'Neill E, Klepal M, Lewis D, O'Donnell T, O'Sullivan D, Pesch D (2005) A testbed for evaluating human interaction with ubiquitous computing environments. In: Testbeds and research infrastructures for the development of networks and communities, 2005. Tridentcom 2005. First international conference on, IEEE, pp 60–69

Pérez J, Arenas M, Gutierrez C (2009) Semantics and complexity of SPARQL. ACM Trans Database Syst (TODS) 34(3):1–45

Qiu F, Hu X (2010) Modeling group structures in pedestrian crowd simulation. Simul Model Pract Theory 18(2):190–205

Quinlan JR (2003) C4.5: programs for machine learning. Morgan Kaufmann, Waltham

Rabiner LR (1989) A tutorial on hidden markov models and selected applications in speech recognition. Proc IEEE 77(2):257–286

Railsback SF, Lytinen SL, Jackson SK (2006) Agent-based simulation platforms: review and development recommendations. SIMULATION 82(9):609–623

Reynolds V, Cahill V, Senart A (2006) Requirements for an ubiquitous computing simulation and emulation environment. In: InterSense '06, ACM Press, New York, p 1

Rudolph L (2001) Project Oxygen: pervasive, human-centric computing—an initial experience. In: CAiSE '01: Proceedings of the 13th international conference on advanced information systems engineering, Springer, London, pp 1–12

Salber D, Dey AK, Abowd DG (1999) The context toolkit: aiding the development of context-enabled applications. In: CHI '99: Proceedings of the SIGCHI conference on human factors in computing systems, ACM Press, New York, pp 434–441

Serrano E, Botia JA, Cadenas JM (2009) Ubik: a multi-agent based simulator for ubiquitous computing applications. J Phys Agents 3(2):39

Team R (2008) R: a language and environment for statistical computing. R Foundation for Statistical Computing Vienna Austria ISBN 3(10)

Tse TH, Yau SS, Chan WK, Lu H, Chen TY (2004) Testing context-sensitive middleware-based software applications. In: Computer software and applications conference, 2004. COMPSAC 2004. Proceedings of the 28th annual international, IEEE, pp 458–466

Wang XH, Zhang DQ, Gu T, Pung HK (2004) Ontology based context modeling and reasoning using OWL. Pervasive computing and communications workshops. Proceedings of the second IEEE annual conference on pp 18–22

Weiser M (1993) Hot topics-ubiquitous computing. Computer 26(10):71–72

Weiser M (1995) The computer for the 21st century. Sci Am 272(3):78–89