FEDERAL UNIVERSITY OF RIO GRANDE DO SUL
INFORMATICS INSTITUTE
BACHELOR OF COMPUTER SCIENCE

LUIZA DE SOUZA

# Conception and Implementation of a Tiny Smart Environment Platform

Graduation Thesis

Prof. Dr. Cláudio Fernando Resin Geyer
Advisor

PhD Student Valderi Reis Quietinho Leithardt, Sebastian Wille
Coadvisor

Porto Alegre, January 2013

**CIP – CATALOGING-IN-PUBLICATION**

de Souza, Luiza

Conception and Implementation of a Tiny Smart Environment Platform / Luiza de Souza. – Porto Alegre: Graduação em Ciência da Computação da UFRGS, 2013.

99 f.: il.

Graduation Thesis – Federal University of Rio Grande do Sul. BACHELOR OF COMPUTER SCIENCE, Porto Alegre, BR–RS, 2013. Advisor: Cláudio Fernando Resin Geyer; Coadvisor: Valderi Reis Quietinho Leithardt, Sebastian Wille .

1. TinySEP. 2. Smart environment platform. 3. Ambient assisted living. 4. Ambient intelligence. 5. Monolithic systems. 6. Platform-based systems. 7. Driver model. 8. Signal-slot model. I. Geyer, Cláudio Fernando Resin. II. , Valderi Reis Quietinho Leithardt, Sebastian Wille. III. Título.

*“ The most profound technologies are those that disappear.
They weave themselves into the fabric of everyday life until
they are indistinguishable from it.”*

— MARK WEISER

# ACKNOWLEDGEMENTS

I would like to thank my parents. They are the base of my life. They have always believed in me even when I did not. Thank you so much for always invest in my education, for all support and love. The day is coming when you can proudly say that all your children are graduated. For sure a part of this achievement is yours. I would like to thank my brothers for always being an example for me, I have always looked up to you. I would like to thank to all my uncles and aunts, specially my aunt Maria Alice, my second mother, for all the support and love.

I would like to thank the Federal University of Rio Grande do Sul (UFRGS), specially Informatics Institute, for the excellence with which you conduct everything in the educational process. I would like to acknowledge the Profa. Dra. Taisy Weber and the Prof. Dr. Valter Roesler for being more than professors to me, thank you for being my friends. I would like to thank the Prof. Dr. Cláudio Fernando Resin Geyer, my advisor, and the Dipl.-Inf. Valderi Reis Quietinho Leithardt, my co-advisor, for the important contributions and the helpfulness in this work since I arrived back in Brazil.

I would like to thank, the Prof. Dr.-Ing. Norbert Wehn and Martina Jahn for giving me the outstanding opportunity of studying in their excellent research group, in Germany. I would like to, specially, thank the Dipl.-Ing. Sebastian Wille, my co-advisor, who had lots of patience and dedication to help me in every detail of this work, being not only an co-advisor, but a friend.

# CONTENTS

# LIST OF ABBREVIATIONS AND ACRONYMS

| | |
|---|---|
| DB | Database |
| IP | Internet Protocol |
| AAL | Ambient Assisted Living |
| AmI | Ambient Intelligent |
| DLL | Dynamic Link Library |
| GUI | Graphical User Interface |
| IDE | Integrated Development Environment |
| LED | Light-Emitting Diode |
| PNG | Portable Network Graphics |
| SEP | Smart Environment Platform |
| UDP | User Datagram Protocol |
| UID | Unique Identifier |
| USB | Universal Serial Bus |
| WSN | Wireless Sensor Network |
| XML | Extensible Markup Language |
| MCA2 | Modular Controller Architecture Version 2 |
| SANET | Sensors-Actuators NETwork |
| TinySEP | Tiny Smart Environment Platform |

# LIST OF FIGURES

# LIST OF TABLES

# ABSTRACT

In the last years many research and development have been made on *Ambient Assisted Living* (AAL). However, as for 2012, there exist no definition of the AAL platform requirements, neither a standard for an AAL system. The solutions already available are no widely accepted outside their projects and can be divided in two groups. The first of them is composed of the universal and flexible platforms, however they are complex and difficult to understand. The second group includes the monolithic systems, which were developed for a specific problem statement and therefore implement only basic functionalities.

Due to the problem of finding a platform that is at the same time, easy to understand and sufficiently flexible and universal, it was decided to design and implement a new solution, which was named as *Tiny Smart Environment Platform* (TinySEP). TinySEP is a compact platform for AAL that makes use of two very successful and frequently used concepts of software engineering: the driver concept and the signal slot model. It allows a single platform to be capable of adapting itself to different scenarios and the needs of each of its users without having to do any reprogramming neither having to restart the hole system. One of the main advantages of this platform is that besides providing the support expected of an AAL system, it also allows new hardware and software to be integrated to the system at runtime in a fast and easy way. Because of this ease of integration, both hardware and software developers can make use of TinySEP as a means of validating their solutions. TinySEP can be seen as a starting point of an evolutionary process to develop a compact platform, which makes use of the high usability of the monolithic systems and the high reusability of encapsulated components of the universal platforms.

**Keywords:** TinySEP, smart environment platform, ambient assisted living, ambient intelligence, monolithic systems, platform-based systems, driver model, signal-slot model.

# RESUMO

Nos últimos anos, tem-se realizado muita pesquisa e desenvolvimento na área de *Ambient Assisted Living* (AAL). Entretanto, até 2012, não existia nenhuma definição dos requisitos de sistema para AAL, muito menos um padrão para as plataformas desenvolvidas. As soluções que já foram desenvolvidas não são amplamente aceitas fora do escopo de seus projetos e podem ser divididas em dois grandes grupos. O primeiro deles é composto por plataformas universais e flexíveis, entretanto essas mesmas plataformas são muito complexas e de difícil compreensão. O segundo grupo engloba os sistemas monolíticos, que foram desenvolvidos para um problema específico e por esse motivo implementam apenas funcionalidades básicas.

Devido ao problema de se encontrar uma plataforma que fosse, ao mesmo tempo, de fácil compreensão e suficientemente flexível e universal, se optou pela concepção e desenvolvimento de uma nova solução, que foi chamada de *Tiny Smart Environment Platform* (TinySEP). TinySEP é uma plataforma compacta que utiliza dois conceitos de engenharia de software: driver model e signal slot model. Ela permite que um único sistema seja capaz de se adaptar aos diversos cenários e às necessidades de cada um dos seus usuários, sem ter de realizar nenhuma reprogramação muito menos reiniciar o sistema. Um dos principais diferenciais dessa plataforma é que além de ela fornecer o suporte esperado de um sistema AAL, ela ainda permite que novos hardwares e softwares sejam introduzidos ao sistema de uma forma rápida e fácil. Devido a essa facilidade de integração, tanto desenvolvedores de hardware quanto software podem fazer uso do TinySEP como forma de validação de suas soluções. TinySEP pode ser vista como um ponto de partida de um processo evolutivo para desenvolver uma plataforma compacta, que faz uso da alta usabilidade dos sistemas monolíticos e a elevada possibilidade de reutilização dos componentes encapsulados das plataformas universais.

**Palavras-chave:** TinySEP, smart environment platform, ambient assisted living, ambient intelligence, monolithic systems, platform-based systems, driver model, signal-slot model.

# 1 INTRODUCTION

The constant miniaturization of computer technology has allowed the integration of tiny microelectronic processors and sensors into everyday objects. More and more people cease to communicate with the traditional computer input and output media, instead they communicate directly with the daily objects (Weiser 1999).

More than 20 years ago, Mark Weiser published the influential article "The Computer for the 21st Century" (Weiser 1999). In this paper, he created a vision of omnipresent computers that would serve people in their everyday lives at home and work, functioning invisibly and unobtrusively in the background (Coroama et al. 2004). This view has been increasingly adopted by Ambient Intelligent (AmI). An Ambient Intelligent system, also referred as Smart Environment Platform (SEP) and as Ambient Assisted Living (AAL), aims to improve the quality of people's life by making everyday activities more convenient and enjoyable with digital media. Technically, AmI refers to the presence of a digital environment that is sensitive, adaptive, and responsive to the presence of people (Doorn e Vries 2006). A SEP interconnects and controls various appliances and services, as for example TV's, lights, stoves, doors, refrigerators and some others that we had never dream about, reconfiguring them on the fly, responding as your needs and desires change (Hedberg 2000). While there are a number of different technologies involved, the goal of a SEP is also to hide their presence from the user by having the computer "disappear" from the users' perception and providing them with implicit, unobtrusive interaction paradigms (Streitz 2007).

A typical AmI system requires for its operation an AAL software platform, a Central Processing Unit (CPU) and Sensors-Actuators NETworks (SANETs). The basic idea is that SANETs provide a communication infrastructure while gathering and distributing information, such as speed, pressure, temperature, brightness and localization. Obviously, SANETs are heterogeneous networks having widely differing sensor and actuator node characteristics (Akyildiz e Kasimoglu 2004). In this text the term sensor node is defined as a device that account with at least one sensor and may include actuators as well, while an actuator node is a device composed with at least one actuator, but no sensors. Sensors are responsible for measure the physical property and quantity of an observation, while actuators are devices that can carry out an action, e.g. a physical response such as turn on a light, in response to a certain stimulus caused by an input signal (Gómez et al. 2010). In some cases, the same elements of a sensor network can be used to activate actuators in order to perform certain tasks in response to an event or to exceed a predefined threshold for a certain parameter. To better understand how does this interaction works, imagine the situation represented on the Figure 1.1, where there is a smart environment equipped with sensors nodes, which are capable of detecting movement and brightness, and actuators nodes, which can control the room light and TV. When a person arrives on this ambient, the movement sensor node detects this motion and sends this information to the AAL software. While that, the other sensor node also sends data related to the room brightness to the platform. When the platform receives all this information, it knows that there is someone on the room due to the movement information and the light is off, because of the brightness data. So it sends a message to the light actuator node in order to turn the light on and an other one to the TV actuator node, because there is someone on the room and this person usually watch TV at this time.

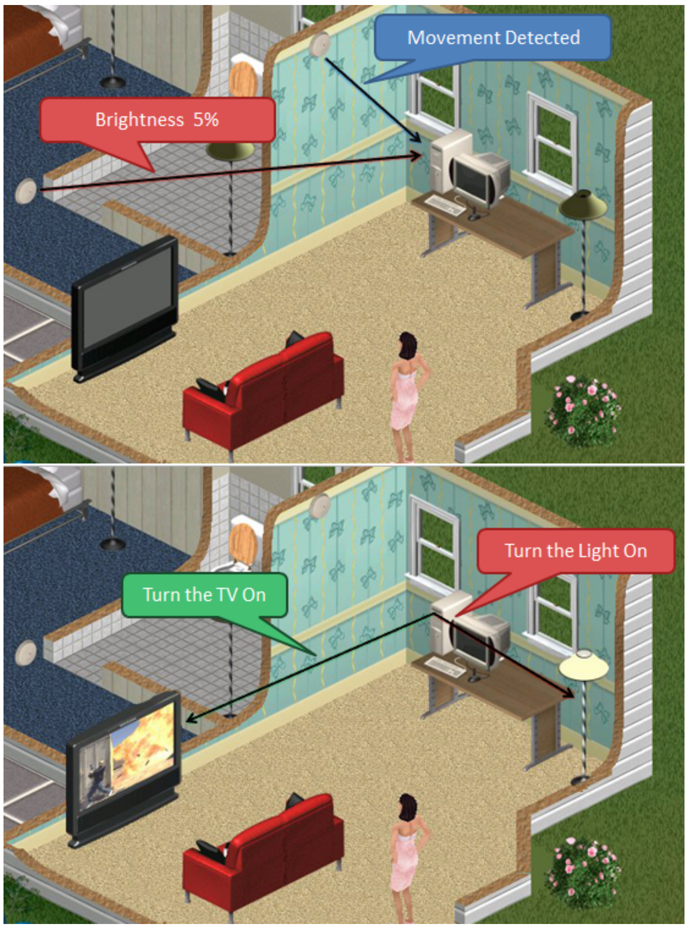


Figure 1.1: Interaction between sensor nodes and actuator nodes on a smart environment.

Currently the market offers the SANETs in two different ways. The first one is given in the form of wired solutions like KNX[1]. It is apparent that in a wired sensor network, all the sensor nodes are connected via a wired network. In spite of its simplicity, a wired sensor network is a very expensive means of sensory data transmission due to the wiring costs, particularly in environments with massive numbers of sensors. Moreover it is hard to change the position of a sensor node and maintenance is not only costly as difficult as well (Lee et al. 2010). The second one is given in the form of Wireless Sensor Network (WSN) like AmICA (Wille et al. 2010; Wille et al. 2010). In a WSN, there is no need for any wired infrastructure. Sensory devices accompanied by their wireless modules can be deployed anywhere in an ambient intelligence environment. Wireless sensor networks, in comparison with wired sensor networks, are more flexible in terms of the deployment, more easy to maintain the required infrastructure

[1]http://www.knx.org/

of the network in AmI environments. Power consumption is the most important concern in WSN because sensory devices and their wireless modules are usually powered by batteries (Aboelaze e Aloul 2005; Akyildiz et al. 2002). Recent advances in wireless communications and electronics have enabled the development of low cost, low power, multifunctional sensor nodes that are small in size and communicate untethered in small distances (Akyildiz et al. 2002). The cost of production of a single node has been reduced to less then 1 dollar, paving the way for large scale of deployments (Tynan et al. 2005).

## 1.1 Motivation

The ageing of the world population is a phenomenon to which even the richest and most powerful countries are still trying to adapt. There are at least two important reasons for people living longer. First of all, healthy life styles in terms of improved nutrition, hygiene, and a growing emphasis on physical and mental exercise have contributed a big deal to greater longevity. Another crucial factor has been medical progress, which has reduced infant and maternal mortality rates significantly (Germany 2011). In Brazil, the index ageing indicates changes in the population age structure. In 2008, for every 100 children aged 0 to 14 years old there was 24,7 elderly aged 65 years old or more. In 2050, those values will change and for every 100 children aged 0 to 14 years old there will be 172,7 elderly. Studies from the Brazilian Institute of Geography and Statistics (IBGE), as shown on the Figure 1.2, indicates that in 2030 elderly will represent 18,7% of the Brazilian population (ESTATÍSTICA 2008). Those values are even higher on the first world countries. Studies of the federal statistical office from Germany shown that there will be approximately 39,3% more elderly people living in Germany in 2030 (Germany 2011).

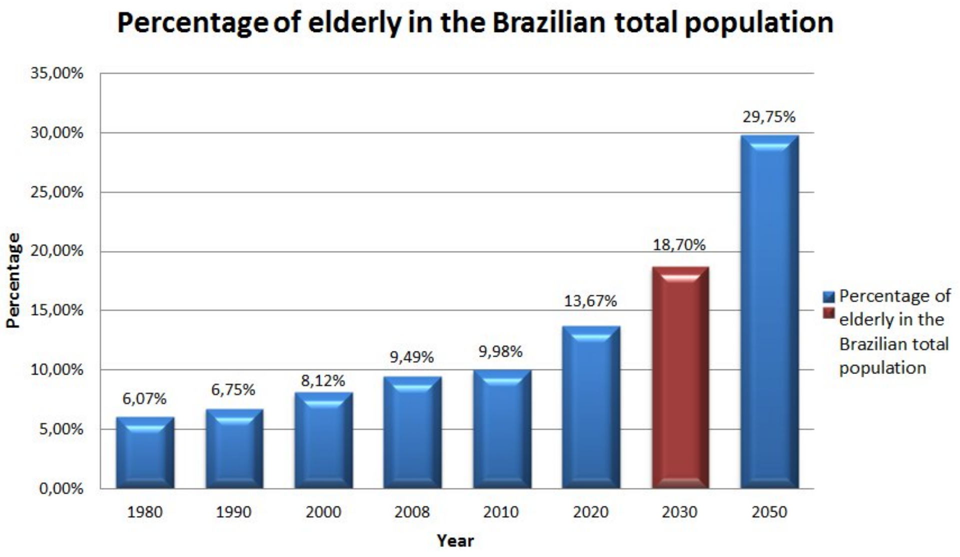


Figure 1.2: The percentage of elderly on the Brazilian total population.

Studies of Counsel and Care in UK revealed that elderly would prefer to live in their own home, however they need support to remain independent (Counsel e Care 2005). And that's exactly the goal of an Ambient Assisted Living platform. AAL aims to prolongate the time people, especially elderly and those who need special care, can live in a decent way in their own familiar environment by increasing their autonomy and self-confidence (Ras et al. 2007). To achieve this purpose it is necessary to provide assistance to carry out daily activities, health and activity monitoring, enhancing safety and security, getting access to social, medical and emergency systems, and facilitating social contacts. AAL also has the potential to reduce the costs associated with elderly care, because it reduces the need of caretakers and personal nursing. A study conducted in Finland has suggested a reduction in the order of 50% (Steg et al. 2006). Therefore, there is a twofold goal of AAL: a social advantage and an economic advantage.

## 1.2 Goal

Much research is being carried out on building AmI systems around people. As for 2012 there exists no standard for an AAL software platform. That is one of the reasons, why smart environment platforms are not wide spread despite of years of intense research work. Most of the AAL platforms that are already available, which will be discussed in detail in Chapter 2, contains some limitations that preclude their widespread use. The first obstacle encountered is related to the platform supported hardware, because many of the systems where developed in research projects as a means of validating a specific hardware, thereby precluding the use of hardware from different suppliers. Thus the cost of deploying these systems becomes much more expensive, because even having a house equipped with various sensor nodes, the user will have to purchase the specific sensor nodes for that platform. The platforms that can be adapted to support different hardware are so complex and difficult to understand that the simple task of adapting the system to perform a special function requires a long time from the developer (Wille et al. 2012).

Another very common problem is the lack of system customization according to the needs and desires of the end user. Many platforms are static, so in order to make any modification to a supported application it is necessary to reprogram the system. That is not acceptable on a AAL software, because each user is different from each other and consequently their wants and needs are as well. Furthermore the needs and desires of a user may change, so it is not feasible to have to change a platform every time the user want to add or remove a certain functionality. Ideally, these updates should be done without even having to reboot the platform (Ziefle et al. 2011).

In this work the conception and implementation of a **Tiny** **S**mart **E**nvironment **P**latform (TinySEP) is presented. TinySEP is a compact platform for AAL that makes use of two very successful and frequently used concepts of software engineering: the driver concept and the signal slot model. It allows the easy integration of hardware and software from different vendors, without having to completely understand the platform. In one easy way TinySEP connects all the single elements to one AAL system, which can be customized according to the needs of its different users, without having to do any reprogramming. All the features available can be customized at run time, without having to restart the system. In other words, TinySEP bridges the gap between the approaches used so far, combining their advantages (Wille et al. 2012).

The results obtained with this work have already been published. The first publication was in the proceedings of the AAL-Kongress, that happened in Berlin, Germany in 2012 (Wille et al. 2012). The second publication was in the proceedings of the Ubirobots 2012 Workshop conducted at the 14th International Conference on Ubiquitous Computing, that happened in Pittsburgh, USA in 2012 (Wille et al. 2012). This paper was also selected to submit to a special issue of the Robotics and Autonomous Systems journal. Furthermore, due to the ease of integration of different hardware and software components, TinySEP is already being used by other international research groups as a way of validating their projects.

## 1.3 Text Structure

The rest of this document is organized as follows: Chapter 2 discusses the state of art of the ambient assisted living platforms, and also presents two simulation tools, that can be used as ways to validate smart environment platforms. Chapter 3 presents a model for a new smart environment platform that may be seen as a starting point for a evolutionary process to develop a compact ambient assisted living system. Chapter 4 shows how the model proposed on Chapter 3 was prototyped and the developed applications provided in the first moment. Chapter 5 analyses the platform performance and the supported hardware. Chapter 6 presents the concluding remarks and future works.

# 2 STATE OF ART

Developed societies are getting older at an unprecedented rate. This ageing leads to new challenges for the provision of healthcare and elderly care. In response of this need, many research and development have been made on Ambient Assisted Living (AAL). An Ambient Intelligent (AmI) system is comprised of SANETs, a CPU and an AAL software, which receives and process the sensory input information in order to determine the action that should be performed (Eichelberg et al. 2010). The AAL systems already available may encompass a wide range of applications such as recognition of adverse events, monitoring of vital parameters, home automation and biometric readers (Hein et al. 2009). However, as for 2012, there exist no standard for a SEP nor a definition of its requirements. So before making an analysis about the AmI solutions already developed, the requirements of a SEP, for which they will be assessed, will be presented.

After the presentation of the requirements of an SEP that are going to be used in this work, some of the AAL solutions that were already developed will be analysed. Those platforms will be divided in two groups: the monolithic platforms and the platform-based systems. The monolithic platforms, presented in chapter 2.2.1, were developed for a specific problem statement and therefore implement only basic functionalities. The platform-based systems, discussed in the section 2.2.2 are composed of the universal and flexible platforms, which are complex and difficult to understand. Once the characteristics of the platforms that compose those two groups were already presented, a comparison of fulfilling the requirements previously defined will be analysed in the section 2.3.

Testing AAL platforms is a difficult task, specially due to the lack of the provision of a readily smart environment. Therefore some simulation tools were developed as an attempt to provide some ways to validate the platforms and their algorithms. In this work the robotic frameworks, MCA2 and SimVis3D, and the context simulator Siafu are going to be used to perform some validations and thus they will be presented, respectively, in the sections 2.4.1, 2.4.2 and 2.4.3.

## 2.1 Requirements of an AAL Platform

The requisites of an AmI system, which are going to be used on this work analysis, was based on the following sources:

- TinySEP - A tiny platform for Ambient Assisted Living (Wille et al. 2012)
- The GAL Middleware Platform for AAL (Eichelberg et al. 2010)
- The PERSONA Service Platform for AAL Spaces (Tezari et al. 2009)

- Promises and Challenges of Ambient Assisted Living Systems (Sun et al. 2009)
- SOPRANO - An extensible, open AAL platform for elderly people based on semantical contracts (Wolf et al. 2008)

The following list summarizes requirements derived from the sources previously presented. Those selected prerequisites are generic, ie, they are not specifically focused on a select group, as for example people with impaired vision, loss of hearing, mobility difficulties or with specific illness such as the Parkinson or the Alzheimer's disease. Since each of these groups has different needs, which would make unfeasible a more comprehensive analysis of the current state of the art of the SEPs.

- **Hardware abstraction**: It should be possible to easily add new sensor or actuator nodes, which may come from different vendors, to the SEP while enabling their easy adoption and usage by the rest of the logical or physical components.
- **Open system interfaces**: Guarantee flexibility in the distribution of functionalities and facilitate the integration of different kinds of hardware and software in the systems.
- **Changes of hardware and software at runtime**: The user by himself must be capable to disconnect the services at all times, without having to restart and reconfigure the hole system.
- **Sensor fusion and context management**: The devices used for building up the ambient infrastructure should be, as must as possible, invisible to the users perception, aiming to avoid situations as represented in the Figure 2.1. Furthermore the system should always take into account the current context of the user for the sake of ensure coherent service provision.
- **Mechanisms for self configuration and adaptation**: The system should support personalization and context-awareness in all layers of the system and it must be capable of adapt itself to those changes.
- **High re-usability of single components**: Serving explicit service requests that cannot be resolved directly but by an intelligent strategy using simple services available as building blocks to compose the original service request.
- **Internal system communication and ontologies**: The platform should support different communication patterns, such as event-based and call-based, and ontologies, in order to give an explicit specification of a conceptualization and to define aspects of all species that are at the same level of abstraction.
- **High usability for the developer**: The platform and all the features and resources it offers should serve the developer also as an easy way to validate applications and ideas, that may range from a new kind of hardware to a new software application, such as one for fall detection.
- **Low resource consumption**: One of the goals of an ALL system for a developer should be to avoid to re-inventing the wheel. It means that should be possible to adopt the solutions already available, in order to spare time, effort and money. Moreover the cost of a AmI platform for the end user must be acceptable so it can be widely adopted.

- **Easy changeability and portability**: It should be possible to easily develop new functionalities and services to a system, which will be integrated to those already available without having to understand the implementation details.

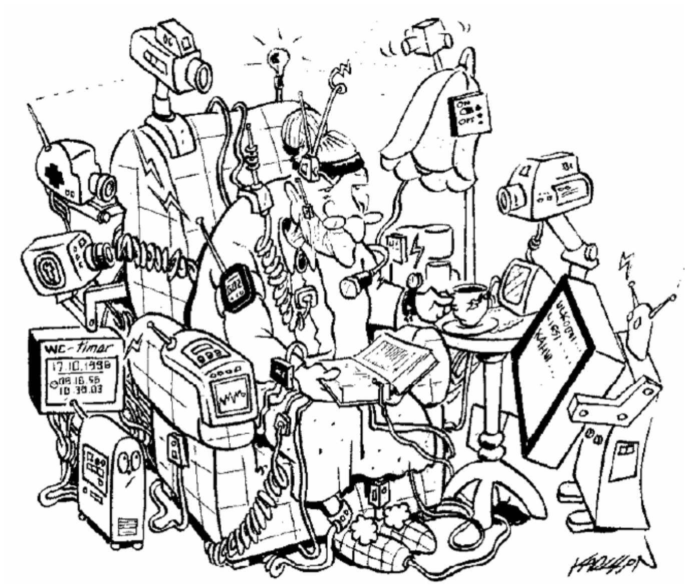


Figure 2.1: Side Effect of Over-Using Assistive Devices (Karlsson 1996).

## 2.2 Related Works

If an analysis on the existing AAL platforms is carried out, it is possible to observe that they do not offer suficient means to be adapted. According to (Wolf et al. 2008), that might be one of the reasons why they did not have reached a high market. Basically those platforms can be divided in three different groups: the home automation, the monolithic and the platform-based AAL systems. However the group composed by the home automation solutions, such as the In Home project (Vergados et al. 2008), can be only seen as a part of a SEP. Due to the fact that they were developed by sensor providers and network experts, those solutions focus primarily on the advantages of the use of certain sensors and their communication protocols. In order to validate these solutions, automated homes are developed in research laboratories, where all devices are integrated, however they often lack services for their specific users, ie the user must adapt to the system, not the opposite and that is against all the design principles. The author from (Sun et al. 2009) remarked that "The assistive devices are not useful if not combined with services and formal or informal support and help" and that's the reason why this group can not be considered a fully AAL solution. The other two classes of AAL solutions are described hereafter together with their drawbacks.

### 2.2.1 Monolithic Systems

The monolithic platforms are usually developed by research groups as a solution for concrete projects and problem statements, in which all the functional requirements are well defined since the beginning and have a low probability of late changes and extensions. Consequently, their applicability is usually restricted to the problem scope, what allows them to be developed in a short time and be efficient, since only the required functionalities will be implemented. They are usually developed using the programming languages which are better known by their developers, who rarely makes use of ontologies and concepts of software engineering. Due to those peculiarities, these systems are used to be characterized by autonomously gloomy "brain", that is responsible for making all the decisions for the devices based on those highly-customized algorithms. The monolithic platforms commonly are developed for a specific hardware, on one hand the underlying hardware can be very compact and energy-efficient, but on the other hand the sensor-fusion is highly dependent on the algorithms and the concepts for hardware abstraction has to be developed.

The biggest disadvantage of monolithic systems is their low re-usability, because they are not use to be separated as components and open system interfaces are very rare, consequently, it is very hard to extend these systems at all. An other drawback is that the changes made on the software require restarting or even recompilation. Moreover they frequently do not offer mechanisms for self-configuration and self-adaptation, therefore they have to be manually reconfigured every time new features are installed. In many cases, developers from other projects cannot integrate their own services to a monolithic system, because they are often developed for specific hardware and they usually are closed-source. Consequently, more and more monolithic platforms are developed each year, each requiring time and money.

Research projects from this area are the EU project EMERGE(Prueckner et al. 2008), which aims to utilize ambient intelligence technologies for detecting emergency situations in elderly people's homes and for alerting emergency response personnel; the EMBASSI project (Herfet et al. 2001), which is focused at enhancing the interaction with living rooms, automobiles and terminal systems by providing intelligent assistance and multimodal interaction; the MAP consortium (Fischer et al. 2003), whose purpose is to set up a multimedia workplace environment for the integration of user systems, network and supporting services, to evaluate multimodal interactions for delegation and assistance tasks; the Assisted Living Project PAUL (Floeck 2010), which is focused on inactivity recognition based on motion sensors; the Verity (Winkley et al. 2012), which integrate sensing devices within their clothing in order to determine their internal health-state.

### 2.2.2 Platform-based Systems

The platform-based systems, also referred as agent-based systems, have an universal applicability. They are characterized by a set of devices, each one having its own intelligence, which together comprise the system intelligence. They differ significantly, in this aspect, from monolithic systems, because while the decisions of monolithic systems are made in a centralized manner typically by an algorithm highly-customized, in agent-based platforms such decisions are taken in a decentralized way by one or more agents. This decentralization allow these systems to have a high flexibility and re-usability. Usually it is possible to make changes on the system without having to restart. They also use to be composed by mechanisms for hardware abstraction, sensor fusion and open system interfaces, which allows the integration of hardware and AAL services from different suppliers. Extending the platform by the creation of new devices is possible, but need a deep knowledge of the complex platform, which commonly requires knowledge of specific programming languages, frameworks and tools. For example, the SerCHO (Albayrak et al. 2009) requires the developer to learn and use the Business Process Modelling Notation (BPMN). This complexity is the most significant disadvantage of the platform based systems, because the simple task of adapting the system to perform a special function becomes almost impossible and demands a long time from the developer. That is the main reason why often the agent-based systems are developed again, because in the view of the developer it is much more easier and faster to develop a new platform than to adapt one already implemented. An other drawback from the agent-based platforms is the cost charged for a membership in order to use a determined solution, as for example the OSAmI platform (Eichelberg et al. 2009).

Some other research projects in this area are the PERSONA (Tezari et al. 2009), which relies on the bus-based architecture to integrate a set of intelligent devices; SOPRANO (Wolf et al. 2008), which is based on a combination of ontology-based techniques, semantic contracts and a service-oriented device architecture; GAL (Eichelberg et al. 2010), which is a service oriented architecture focused on the integration of different assistive systems; Amigo (Janse et al. 2007), which is composed by a service oriented architecture with service composition strategies and those strategies must be implemented by each service; OASIS (Kehagias et al. 2008), which is based on an Ontology-driven, Open Reference Architecture and System, which allows an interoperability, seamless connectivity and sharing of content between different services and ontologies; MPower (Mikalsen et al. 2009), which is composed by interoperability services based on patterns, service-oriented architectures, web services and XSDL transformations; SENSACTION-AAL(SENSACTION-AAL 2007), which is a wireless on-body system that enables daily activities monitoring, real-time active control of physical performance and fall detection and management; I-Living (Wang et al. 2006), which is focused on the integration of different hardwares and softwares from distinct suppliers in order to construct a better environment.

## 2.3 Comparison of Fulfilling the Requirements

Given the criteria presented on the subsection 2.1 Requirements of an AAL Platform, a comparison between the existing AAL platforms, presented on the subsection 2.2 Related Works, and TinySEP, which is going to be discussed in detail in the next sections, was made and systematized in the Table 2.1. As can be seen, TinySEP bridges the gap between the approaches used so far and combines the advantages of both in one single platform (Wille et al. 2012).

## 2.4 Simulator

Evaluate the AAL systems is not an easy task, because due to the heterogeneous nature of the data sources, which makes the context gathering and processing difficult. Furthermore, the lack of readily available infrastructure makes the process of collecting large data sets unfeasible. The limitation if those infrastructures makes the process of evaluation from those AAL platforms be limited to a small subset of users and sensor nodes. Moreover, those tests usually are performed in a short time.

As a way to solve this problem of AAL systems evaluation, simulation tools were developed. Those tools attempt to give suitable abstraction for the problem domain. The main idea from those simulators is to allow the test of functionalities of group of services and applications before performing the user evaluations.

In this work 2 simulators are going to be used to perform the evaluation of the applications from the AAL platform that are going to be presented in the next chapter. The first of them is composed by robotic frameworks, the Modular Controller Architecture

Table 2.1: Comparison of fulling the requirements of an AAL platform of TinySEP, monolithic systems and platform-based systems. Given the criteria presented on the subsection 2.1 Requirements of an AAL Platform the tokens represents the level of satisfaction, where + represents high, - assume the role of low, 0 correspond to dissatisfied (Wille et al. 2012).

| | TinySEP | Monolithic systems | Platform-based systems |
|---|---|---|---|
| Hardware abstraction | + | - | + |
| Open system interfaces | + | - | + |
| Changes of hardware and software at runtime | + | - | + |
| Sensor fusion and context management | + | 0 | + |
| Mechanisms for self configuration and adaptation | + | - | + |
| High re-usability of single components | + | - | + |
| Internal system communication and ontologies | + | - | + |
| High usability for the developer | + | +/-[2] | - |
| Low resource consumption | + | + | 0 |
| Easy changeability and portability | + | 0 | - |

Version 2 (MCA2) and SimVis3D. Both are discussed, respectively, in the sections 2.4.1 and 2.4.2. The results obtained using from both robotic frameworks will be presented in the section 5.4.2. The second one is an open source context simulator, named as Siafu. Siafu will be presented in the section 2.4.3 and its usage is going to be discussed in the section 5.4.3.

### 2.4.1 MCA2

The Modular Controller Architecture Version 2 (MCA2) is used as a main control system of a variety of robots including the Robust Autonomous Vehicle for Off-road Navigation RAVON (Schaefer et al. 2009), the Mobile Autonomous Robotic Vehicle for Indoor Navigation MARVIN (Schmidt et al. 2006), the Autonomous Robot for Transport and Service ARTOS, that has also been used for several experiments for the elderly care (Berns e Mehdi 2010). Additionally, it has been used as the control system of CROMSCI, the Climbing Robot with Multiple Sucking Chambers for Inspection tasks (Hillenbrand et al. 2008), the Robothuman interaction machine ROMAN (Hirth et al. 2007) and several other projects. It also serves as the original framework that forms the base of the popular Integrated Behaviour-Based Control iB2C (Proetzsch et al. 2010).

MCA2 was developed using a set of small modules written in C++. All the modules are interconnected allowing the communication between them. It is very easy to connect modules that run in separate processes on the same machine, but also to connect mod-

[2]Only high for internal developers, who know the system structure

ules that run on entirely different machines. Each module process data in two different methods, the Sense-method and the Control-method. The Sense-method processes sensor inputs and generates sensor outputs while the Control-method processes the controller inputs and generates the controller outputs. Another feature to cope with complexity is the concept of groups. MCA2-groups have basically the same interface as modules, but instead of containing program logic, as the modules do, they are used to encapsulate a bunch of modules or even other groups, exposing only the external controller/sensor inputs/outputs.

To allow the usage of this framework to perform simulations of AAL systems, more specific with the prototype that is going to be presented in the chapter 4, new modules were created. Their structure is illustrated by the Figure 2.2. The inner group, which is illustrated in light gray, simulates the behaviour of the sensor nodes. The sensor nodes that are simulated by this module are equipped with movement sensor and reed-switch sensor. The module SimulatedMedium processes the simulated sensor node information and feeds the AmicaUDPTransmitter module that sends this information to the AAL platform.

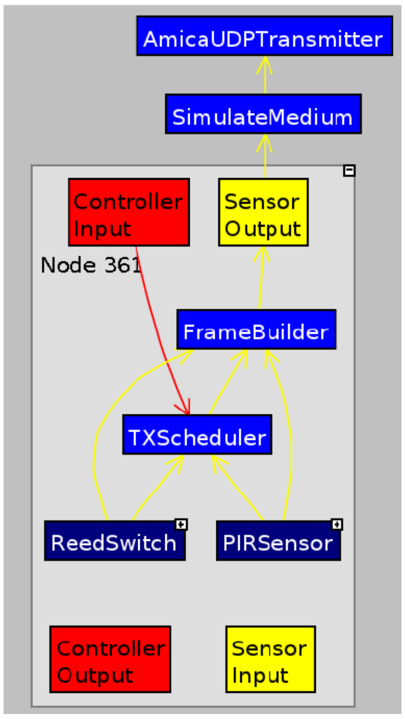


Figure 2.2: Example structure of MCA2 modules and groups.

### 2.4.2 SimVis3D

The SimVis3D framework (Braun et al. 2007) is a primary simulation and visualization framework for the 3D sceneries of all the robots described in the section 2.4.1. It is an open and flexible and customizable solution which has been released under the terms of the GNU General Public License (GPL)[3].

SimVis3D serves two main tasks: first the simulation of robots by providing the possibility to simulate the sensor systems as well as the actuation systems of the robots. Additionally, all kinds of static objects as well as other dynamic entities such as people or animals can be simulated. The second task is the visualization of the current state of a

[3]http://rrlib.cs.uni-kl.de/software/simvis3d/

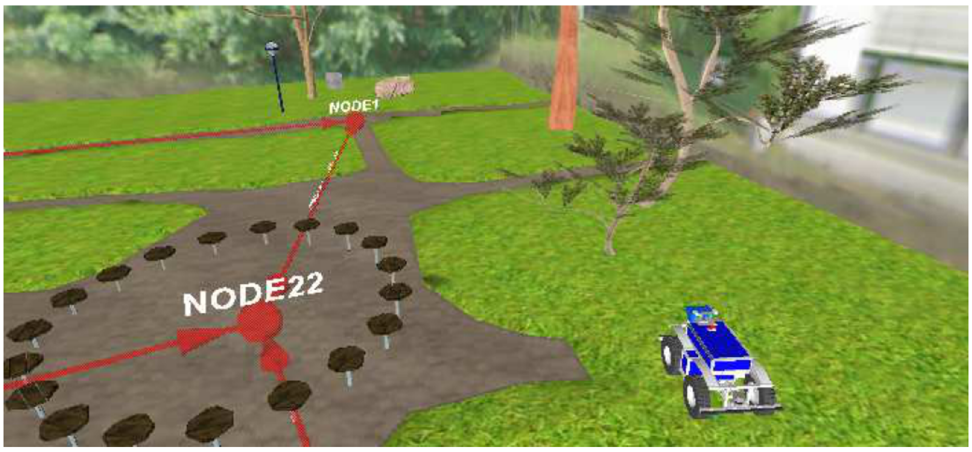
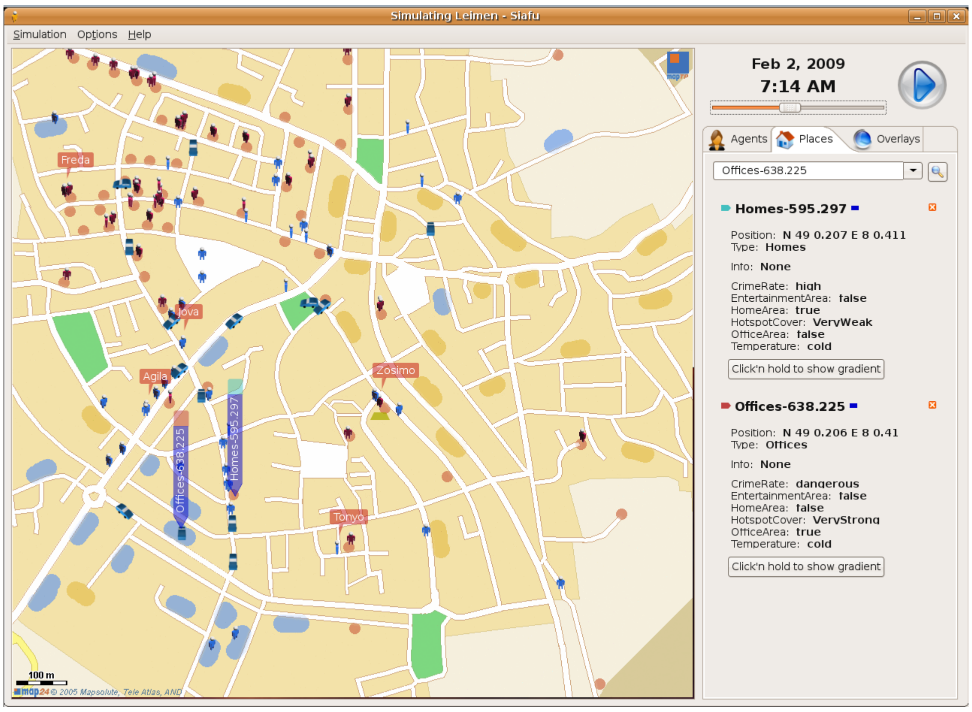


Figure 2.3: 3D-Visualization of Ravon with overlaid navigation graph (Hirth et al. 2007).

(real) robot. This is usually done by displaying a 3D model of the robot and additionally visualizing the state of the world, as illustrated by the Figure 2.3.

## 2.4.3 Siafu

Siafu is an open source, versatile, large scale context simulator written in Java. It was developed by the NEC Europe Networks Labs from Heidelberg, Germany. Siafu has been designed to be generic, flexible and applicable in a wide range of scenarios, as shown in Figure 2.4. The main information sources from the simulator is split in 3 parts: the agent model, the world model and the context model (Martin e Nurmi 2006).



Homes-595.297

Position: N 49 0.207 E 8 0.411
Type: Homes

Info: None

CrimeRate: high
EntertainmentArea: false
HomeArea: true
HotspotCover: VeryWeak
OfficeArea: false
Temperature: cold

Click'n hold to show gradient

Offices-638.225

Position: N 49 0.206 E 8 0.41
Type: Offices

Info: None

CrimeRate: dangerous
EntertainmentArea: false
HomeArea: false
HotspotCover: VeryStrong
OfficeArea: true
Temperature: cold

Click'n hold to show gradient

Figure 2.4: A snapshot of Siafu being used to simulate a city.

The agent model is responsible for modelling the behaviour of individual agents, ie, the agent model decides what an agent should do given the current environment and context. Therefore this model must change the proprieties of an agent, as for example change the current agent's activity from sleeping to working. In order to perform these changes, the simulator designer must specify those changes. To facilitate this process, Siafu provides some templates that can be used to define the agent's behaviour. Those agents provides an random behaviour to the agent (Martin e Nurmi 2006).

The world model handles the environment. This model consists of three parts, the environment to be simulated, the places of interest for the simulation and a global event model that handles global events such as citywide festivals. This model is specified through a set of Portable Network Graphics (PNG) images. The base set of image files consists of a background map and a wall map (see Figure 2.5). The background map defines the places where the simulation is performed, while the wall map define the regions from the background map where the agents can walk and others where they can not (Martin e Nurmi 2006).

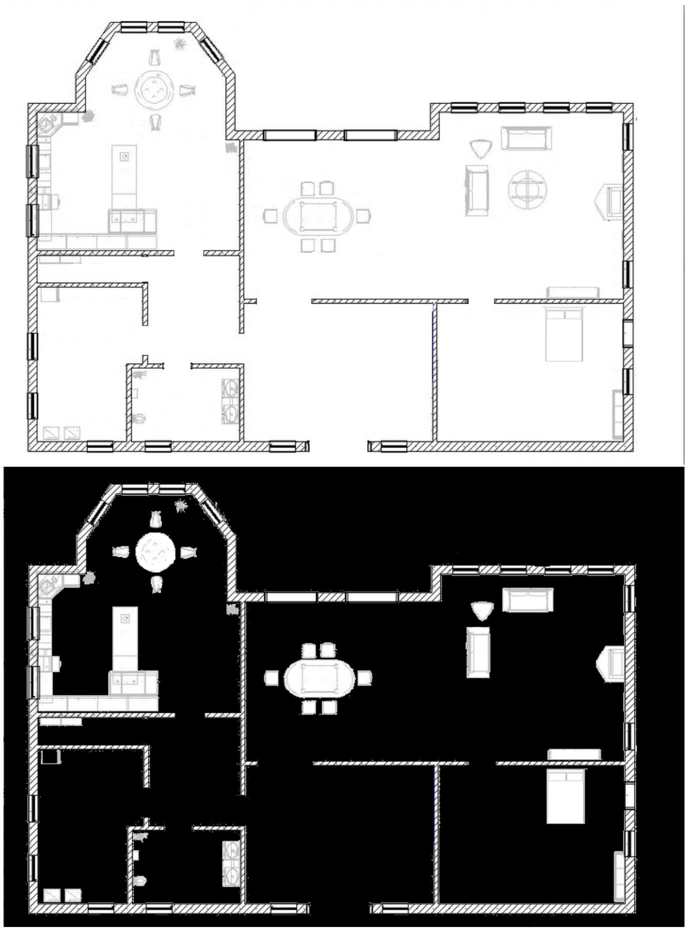


Figure 2.5: The upper figure illustrates a background map, while lower figure represents the wall map.

The context mode is used for defining how the context data is simulated, ie, it manages the context variables that are used in the simulation. As an example, consider a context model with 2 variables that are temperature and Wi-Fi hotspot coverage. For the temperature, this model can specify the temperature in specific areas, while the Wi-Fi hotspots can specify the signal strength of the base stations. In order to simulate the sensors behaviour, this context can be used to specify the signal range from the sensor nodes (Martin e Nurmi 2006).

# 3 TINYSEP: MODEL

In this chapter a model for an event-driven AAL system, named as Tiny Smart Environment Platform (TinySEP), will be presented. The main functionalities of TinySEP are to provide all the necessary support expected of a SEP, while allowing an easy and fast integration of hardware and software from different vendors, enabling a high usability for the developer and even greater freedom to its users; and permitting a single system to adapt itself to several scenarios and different needs of their users, without having to reprogram a single line of code and without having even to reboot the system.

TinySEP might be seen as a starting point for a evolutionary process to develop a compact platform, that makes use of the high usability of the proprietary monolithic systems, presented in the section 2.2.1, and the high re-usability of encapsulated components of the platform-based systems, described in the section 2.2.2.

TinySEP model makes use of two concepts of software engineering: the driver model and the signal slot model. The driver concept, which is going to be presented in detail in the section 3.1, allows an easy encapsulation and integration of new components of software and hardware. In this model each software component is encapsulated in a driver and devices that might be accessed from the other devices and drivers. The signal slot model from TinySEP that will be discussed in section 3.2 enables the communication between the several drivers and devices. This concept is extremely important to event-driven applications like TinySEP.

In order to manage all the devices and drivers, TinySEP has its own device manages that will be explored in section 3.3. In this section an example scenario, where devices are created and removed, is presented. Through this example it is also possible to have a better understanding of how each driver and device reacts to events.

TinySEP model supports runtime changes and an easy integration of hardware and software. Those two characteristics are discussed, respectively, in the sections 3.4 and 3.5. Before proceeding to this sections it is important to have understood the concepts previously presented, once this two features are directly interconnected with them.

One of the requirements of the TinySEP model is to allow a single software to run in different environments. With the view to enable that a new component, called as House Information, was projected. The goal of the House Information is to allow the other components to access the environment informations, such as the rooms name, room type and which room is connected with an other room. All the details about the House Information will be discussed in the section 3.6.

During the process of installation the user to might need to provide a large amount of information, because some components need specific inputs from the user, as for example, the user might need to inform the room in which a sensor node is installed. In order to avoid the reconfiguration of all those informations, the TinySEP model provides a system backup that automatically stores all the needed information and reconfigure the system when it is restarted. This feature is presented in the section 3.7.

In the section 3.8 a sketch from this model is presented. This section discuss the basic functionalities that the graphic interface should support.

At the end of this chapter the hardware that will be first supported will be assessed. In order to validate and evaluate the TinySEP model, at least two types of hardware from different vendors should be selected. The AmICA and the Sun SPOT sensor nodes were selected and the reasons and characteristics of each of them will be encompassed in this chapter.

## 3.1 Driver Model

The driver model is used in most of the modern operating systems as an easy way of integrating a computer's software with its hardware. Basically, this concept enables the encapsulation of software applications from the hardware, in order to protect each other from changes. (Wille et al. 2012; Lemon e Rossi 1995) The operational system may access the hardware with the help of drivers, which can create devices, that may be accessed from other drivers or devices. As for example, if a mouse is plugged into the computer, the computer USB controller driver creates a new USB device, that will be responsible to handle and provide the raw data from the hardware, in this case the USB mouse. The USB driver also creates a second device that convert the raw data from the device previously created into cursor coordinates (Wille et al. 2012).

TinySEP makes use of a similar driver model. In the same way as described before the drivers can create devices, which can be accessed from others devices or drivers. The drivers can create a device given an input, which may range from a hardware connection to the creation of a specific device. In fact the main goal of a driver is to manage it's devices. The devices are objects that encapsulate a certain application, as for example, the devices can be used to make an abstraction from a hardware or to verify if the flat is occupied or not. Each device stores it's own internal configuration information, some of which can be accessed externally. For example, a device that correspond to the abstraction of a door sensor node can provide information about the location of the door, if the door is an outer or inner door and if the door is opened or closed. Other devices need this information, in order to decide if they want to establish a connection with this device or not. It is important to note, that a device can not create an other device, the device creation can be performed only by it's driver. Therefore a TinySEP application is seen as the combination of a driver and its devices.

## 3.2 Signal Slot Model

A SEP is typically an event-driven system, in which observed events cause reactions in the system (Hinze et al. 2009). For example, reconsider the situation represented by the Figure 1.1, when someone enters in the room, the movement detection sensor node notices this motion and forwards this information to the event-driven system. As a reaction to this event the light is automatically switched on. The event-driven applications consist basically of three features, sense, analysis and respond, as shown in Figure 3.1. The sensing property is responsible for collecting the information, ie, observe the external system events. In the situation described previously, the monitoring component would be symbolized by the movement detection sensor node, which noticed someone's motion and transmit this information to the system (Chandy 2006).

The analysis part is responsible for analyzing the data transferred to the system by sensing component and for performing event notification. The event notification may be performed in three different ways. In the first, the producers disseminate the information to the consumers. This approach is often called push-based. In the second one, denominated pull-based, the consumers must request the information to the producer. The third one, referred as publish/subscribe notification, may be seen as the combination of the two previous approaches, since the consumer need to register itself to a producer to receive notifications, but that same consumer can make requests for information (Muhl et al. 2006).

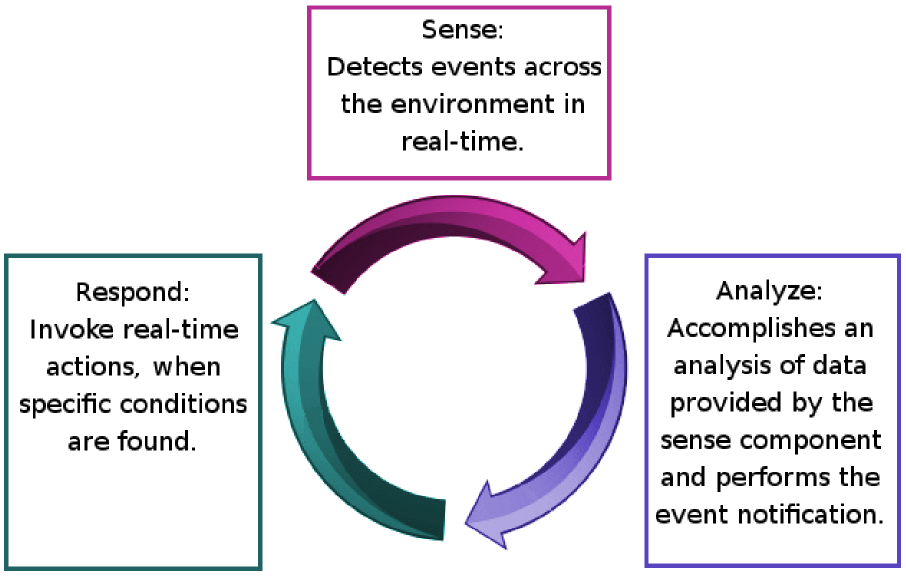

Sense:
Detects events across the environment in real-time.
Analyze:
Accomplishes an analysis of data provided by the sense component and performs the event notification.
Respond:
Invoke real-time actions, when specific conditions are found.

Figure 3.1: The three basic functions that compose the main characteristic of event-driven applications

The respond component is triggered by the corresponding events. If an determined condition is met, then a action is performed. This action may be the sending of a warning, the start of another application or the performance of an operation by actuators nodes, as occurred in the previous example, in which the light was turned on.

In an event-driven system, the objects, producers and consumers, can be interconnected with the help of the signal slot model. In this model, the signals are messages, which may represent an event, that are sent to one or more slots. The same slot may receive several different signals. TinySEP makes use of the signal slot model to interconnect primarily devices. However the signal slot model was modified, in order to allow a publish/subscribe event notification. This decision was made, because a publish/subscribe notification was evaluated as the best possible approach when compared to the pull-based and the push-based notification. In a push-based concept the consumers become extremely dependent on producers, which disseminate the information to all consumers, even those who will not make use of it. In the pull-based approach, the problem of the data notification to the uninterested parties is solved, although the events notification no longer occurs in real time, because it becomes necessary to request the information producer. If such requisitions are made in a very large space of time, the consumer will be only informed of an event a long time afterwards it has occurred, which may even cause the loss of information from an intermediate event. On the other hand if the period between this requests is too short, then the producer will be overloaded with requisitions, especially if it has established connections with a large number of consumers. In a publish/subscribe approach none of these problems occurs, because a consumer signs up to receive information only from the producers that interest it and, consequently, the producers send notifications only for consumers which sign them. Consumers can also make requisitions to producers whenever they need, without necessarily rely solely on those producers notification to obtain the information needed.

The signal slot model from TinySEP was modified in two ways. First, the several pairs of signals and slots were bundled into so-called interfaces. It is important to clarify that, in this work, whenever the word interface is referenced, it will make mention of a signal slot pair. Second, in order to implement a publish/subscribe approach, the connection must be bidirectional. To better understand this concept, the interfaces from Figure 3.2 will be explained one by one.

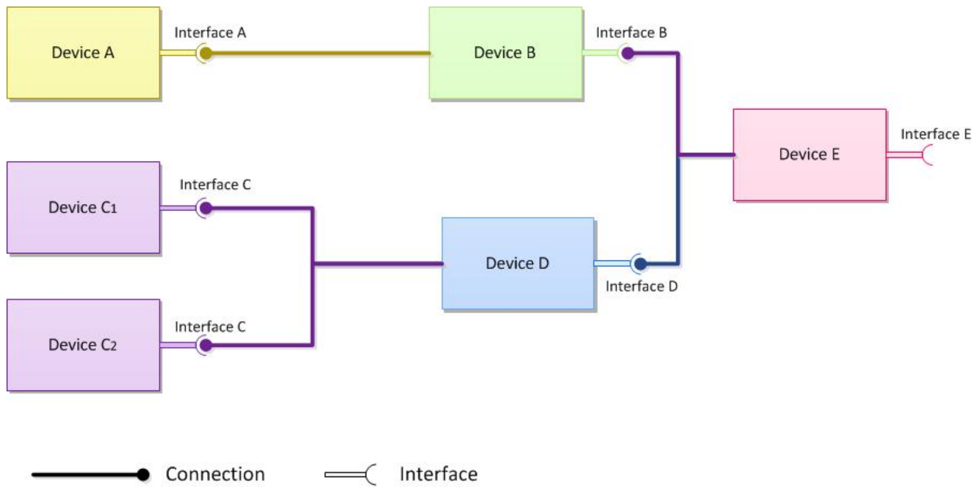


Figure 3.2: TinySEP Interfaces

The Device A is able to send different signals to the Device B through the Interface A. The Device B is also capable to send data to the Device A. The Device C1 and C2 have the same Interface C, through which a bidirectional connection is established with the Device D. This illustrates that a single interface may be available from more than one device. Two other interesting characteristics can be observed from Device E. The first one is that it is connected at the same time to the Device B, through the Interface B, and to the Device D, through Interface D, ie, the same device can be connected to more than one interface simultaneously. The second characteristic is that an interface do not need necessarily to have an established connection to exist, such as the Interface E, which is provided by the Device E.

## 3.3 Device Manager

All the drivers, devices and available interfaces are administrated by the device manager. In order to become available and active, the drives must register themselves at the device manager. Once registered, they start to receive notifications from it whenever a new interface is found or removed. When a driver receives a notification from a new interface encountered, it can choose to create a device to communicate with this interface or not. This decision is made individually by the driver and the whole logic that rules its actions. In this scope the device manager works only as a way to establish a connection.

The devices also must register themselves at the device manager to become active and available. Once the registration is performed, all the drivers and devices so far installed are informed about the interfaces provided by this device. Upon receipt of such notifications of new interfaces found, a device may choose to connect to this interface or not, but it will never be able to create another device. Once more this decision will be taken solely by device, in other words, once created, the device starts to establish its connections without the aid of the driver that originated it. Since a connection was set among the two devices via an interface, all communication shall be made directly between them, ie, at no point the device manager or the drives are informed about the data or signals forwarded via this interface. This decentralization of event communication is an extremely important factor, because the device manager does not need to be overwhelmed with transmission of messages. Imagine if that does not happen, ie, to send a signal from the Device A to the Device B, it would be necessary first to send it to the device manager, which will be transferred this signal to its destination. Clearly this approach is not efficient and it would end up to overload the device manager as far as the amount of devices increases.

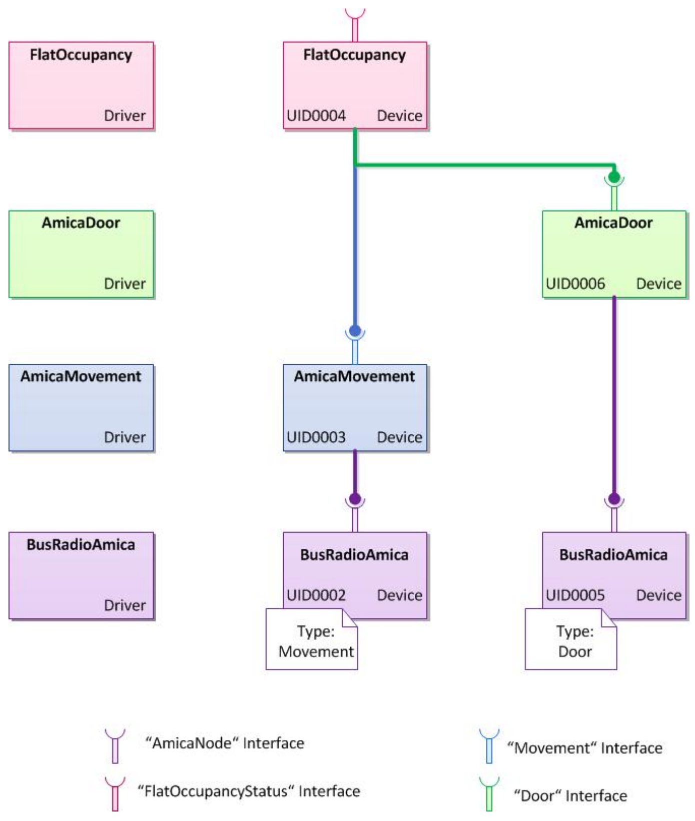


Figure 3.3: TinySEP example scenario: after creation of the "FlatOccupancy" Device.

To better understand the dynamics of the devices creation and communication, the installation process that led to the scenario illustrated by the Figure 3.3, will be described step by step. Initially are loaded and registered at the device manager the drivers "BusRadioAmica", which makes a connection with the sensor nodes; "AmicaMovement", which abstracts the raw data supplied by a movement sensor node and informs if there were an occurrence of a movement or not; "AmicaDoor", which abstracts the raw data from a door sensor node and reports if the door is open or closed; and "FlatOccupancy", which informs whether the house is occupied or not.

When the first movement sensor node is installed, it starts sending packets. As soon as the first packet from this node is received by the "BusRadioAmica" driver, a "BusRadioAmica" device, with the "AmicaNode" interface and "Type: Movement" is created. The "BusRadioAmica" interface allows the other devices to receive the raw data or to send configuration data to this sensor node. Having completed the build process, this device registers itself at the device manager to become active. Once this registration is done, the device manager will inform all the drivers and devices already registered, in this case the drivers "BusRadioAmica", "AmicaMovement", "AmicaDoor" and "FlatOccupancy", about the new device found, because they might be interested in connecting with its "AmicaNode" interface. The 4 drivers receive this information, but only the "AmicaMovement" and the "AmicaDoor" have the interest on a "AmicaNode" interface. Therefore, both read out the "Type: Movement" information. The "AmicaMovement" driver decided create a device to connect with this "BusRadioAmica" device, through the "AmicaNode" interface. This device analyses the raw data and abstract it as movement information to the other devices via a "Movement" interface. The "AmicaDoor" driver does not create a device, because they can only convert raw data from "Type: Door".

Once created this "AmicaMovement" device, it registers itself at the device manager as an activation process. By doing this the device manager sends a message to the four drivers and to the "BusRadioAmica" device, notifying about the creation of this device, which provides the "Movement" interface. Upon the receipt of this information, the "FlatOccupancy" driver creates a device, in order to stablish a connection with the "Movement" interface. This device, verifies if the flat is being occupied or not and deliver this information through the "FlatOccupancyStatus" interface. This device will register at the device manager, which in turn will notify all drivers and devices currently active concerning this new device and its "FlatOccupancyStatus" interface. However, none of the active drivers and devices have interest on using this interface, therefore, any connection is established, neither a new driver is created.

A door sensor node, which until then was inactive, is turned on and starts to send data packets. As soon as the first packet arrives at the "BusRadioAmica" driver, a new device is created. This device will provide the "AmicaNode" interface and "Type: Door". Once registered, the driver manager will send a message to all the drivers and device current available informing that a new device, which supplies the "AmicaNode" interface and "Type: Door", was found. This time the "AmicaMovement" driver does not create device, because despite the interest in the interface, its type is different from the supported. However, the "AmicaDoor" driver creates a new device, which will make a connection with this new "BusRadioAmica" device. Through the "Door" interface, the "AmicaDoor" device abstracts the raw data into door informations and turn them available to the other devices.

Once created, the "AmicaDoor" device is registered to then become active. Once again, the device manager will inform all drivers and devices about the existence of this new device and its "Door" interface. This time the "FlatOccupancy" driver does not create a new device, because it has already created one, which is all it needs. However, the "FlatOccupancy" device wants to receive the data from the "Door" interface, therefore it establish a connection through the "AmicaDoor" device.

As it was possible to observe through the detailed explanation, the creation of a device is performed automatically, without needing to specify one by one how the connections between the devices will be made. Once established these connections, the communication between the devices is done automatically, without the help of the device manager. As for example, given this scenario, when the person arrives at home, he will open the door of his home. By doing this, the door sensor node will send data packets with this information. Once the "BusRadioAmica" device receives this message, it will abstract this raw data into door information, ie, it will observe that the door was opened and it will forward this information to all the devices to it connected, that have the interest on this information, in this example only the "FlatOccupancy" device. By receiving this notification, the "FlatOccupancy" device, for example, may initiate the process of checking the house status. The action by it performed will depend on the logic by which it was programmed.

The device manager is also responsible for reporting all the active drivers and devices about the removal of a specific interface. Thus, if a particular device has an established connection via this interface, it can terminate it. Moreover, if a certain device loses all the necessary connections for its operation, it will also uninstall itself, thus avoiding the system to have devices that can not operate.

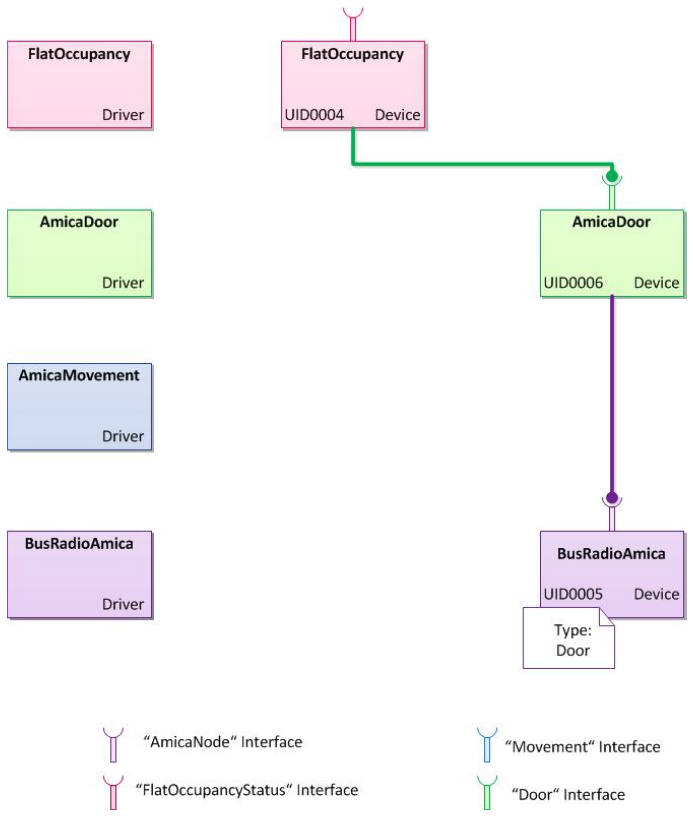


Figure 3.4: TinySEP example scenario: after removal of the "BusRadioAmica" Device "Type: Movement".

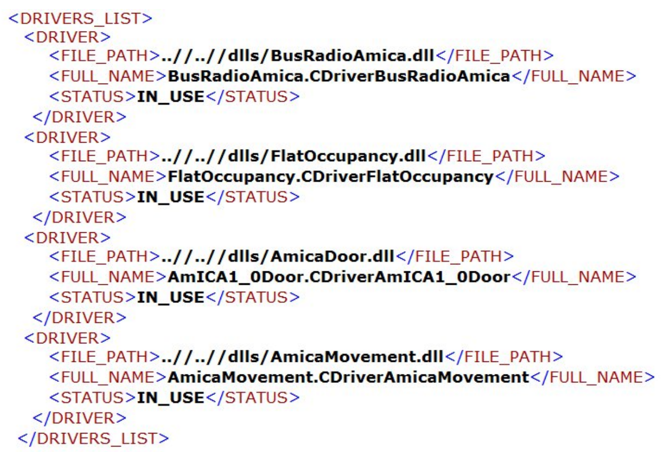
Given the scenario from the Figure 3.3, imagine that the user, displeased with the movement sensor node, decides to remove it. Then, he simply removes this "BusRadioAmica" device, which was performing a connection with this sensor node. By doing this, the device manager will be notified about the unregister of this device. Thus, it will inform all the devices and drivers about this. The "BusRadioAmica" driver, which has created this device, will remove this device from its device list, in order to allow the same movement sensor node to be switched back and a new device created. Additionally the "AmicaMovement" device, which held a connection to that device through the "AmicaNode" interface, decides to uninstall itself, because this connection was vital for its operation. The other drivers and devices do nothing when they receive this information, because there was no connection among them.

Once the device manager is informed that the "AmicaMovement" device is unregistering, it informs all the drivers and devices that occurred. The "AmicaMovement" driver removes this devices from its devices list. The "FlatOccupancy" device, which has a connection with this device through the "Movement" interface, removes it. But unlike the "AmicaMovement" device, it does not uninstall itself, because it still have a essential connection through the "Door" interface. At the end of this removal, the resulting scenario is represented by the Figure 3.4.

If the user opts to remove a driver, then all its devices will be also removed, one by one. Consequently, all devices and driver will be informed of each removal. It is interesting to note that it is up to each device to decide whether to unregister when they lose a connection or not. This decision will be based on the logic by which it was programmed.

## 3.4 Runtime Changes

One of the requirements presented in the section 2.1 was that it should be possible to perform software and hardware modifications at run time. In fact, TinySEP is capable of accomplish this, due to the combination of its driver model, its signal slot model and its device manager. According to what was presented in section 3.3, the installed drivers and devices, which compose TinySEP, are able to adapt themselves to the appearance and removal of other drivers and devices. However it is still necessary to explain how those new drivers and devices can be introduced to the system at runtime.



```
<DRIVERS_LIST>
  <DRIVER>
     <FILE_PATH>..//..//dlls/BusRadioAmica.dll</FILE_PATH>
     <FULL_NAME>BusRadioAmica.CDriverBusRadioAmica</FULL_NAME>
     <STATUS>IN_USE</STATUS>
   </DRIVER>
  <DRIVER>
     <FILE_PATH>..//..//dlls/FlatOccupancy.dll</FILE_PATH>
     <FULL_NAME>FlatOccupancy.CDriverFlatOccupancy</FULL_NAME>
     <STATUS>IN_USE</STATUS>
   </DRIVER>
  <DRIVER>
     <FILE_PATH>..//..//dlls/AmicaDoor.dll</FILE_PATH>
     <FULL_NAME>AmICA1_0Door.CDriverAmICA1_0Door</FULL_NAME>
     <STATUS>IN_USE</STATUS>
   </DRIVER>
  <DRIVER>
     <FILE_PATH>..//..//dlls/AmicaMovement.dll</FILE_PATH>
     <FULL_NAME>AmicaMovement.CDriverAmicaMovement</FULL_NAME>
     <STATUS>IN_USE</STATUS>
   </DRIVER>
 </DRIVERS_LIST>
```

Figure 3.5: TinySEP drivers XML file.

The combination of a driver and its devices comprises a TinySEP feature, therefore it is possible to encapsulate the code of this feature into a Dynamic Link Library (DLL). The major advantage of this encapsulation is that these DLL files may be explicitly loaded at runtime. So basically everything the device manager must do is to open this DLL file and install its driver, once the devices are dynamically created and installed by the driver. To make this possible, the device manager must have the knowledge of this DLL file path and the full name of driver. The easiest way to store this information and make it available to the device manager is through an Extensible Markup Language (XML) file. So whenever the user desires to add a new driver that was not being used so far, he just have to add a new entry in this XML file and request the TinySEP to update its drivers list. To avoid DLL to be reopened unnecessarily, a new field in this XML was included. This new field is managed by the device manager. The Figure 3.5 illustrates the XML file from scenario presented in the Figure 3.3.

To accomplish the removal of a TinySEP feature, it is not necessary to modify this XML file. Only the driver of this application must be removed, because it will automatically uninstall its devices. However if after this removal, the user desires this application to never be reinstalled, then it will be necessary to remove the entry of this application in the XML file.

This approach also enables the user to change the hardware being used at runtime. In order to do this, the user must include the DLL files relating to this hardware, which should be provided by its vendor. If the user wishes to completely remove his old hardware from the system, he just uninstall the drivers and remove them from the XML file.

## 3.5 Hardware and Software Integration

The requirements "Hardware Abstraction" and "Easy changeability and portability", presented in section 2.1, request that it should be possible to easily add new hardware and software to the SEP while enabling their easy adoption. TinySEP makes this possible through the use of its interfaces, which enables a given device to look for connections based in interfaces and not on the devices by themselves. Thus, the same device can establish connections through a particular interface, which is provided by different devices that may also be originated from different vendors.

In order to exemplify this concept, review the Figure 3.3. There the hardware used was from a certain type of sensor nodes, which communicated with the platform through the "BusRadioAmica" driver and its devices. Now consider that the user wishes to install an additional hardware from another supplier, for example another type of motion sensor. So all the user need to do is to perform the installation of the DLL files, as described in the section 3.4. In this example, the hardware vendor has made available two new DLL files. The first one has the "BusRadioSunspot" driver, which together with its devices carries out a connection with that supplier hardware. The second DLL file is composed of the "SunspotMovement" driver, whose devices are responsible for converting the raw data from this sensor nodes into movement information.

Once the sensor node is turned on, it starts to send data packet. As soon as the first packet is received by the "BusRadioSunspot" driver, it will create a device with the "SunspotNode" interface and "Type: Movement". Upon the registration of this device, the device manager will inform all the devices e drivers already actives about this new device, which provides the "SunspotNode" interface and "Type: Movement". The "SunspotMovement" driver has the interest on receive the data from this interface with the "Type: Movement", therefore it creates a new device. This device offers the "Movement" interface, which is the same provided by the "AmicaMovement" device. Upon its registration to become active, the device manager will inform all the driver and devices about the appearance of this "SunspotMovement" device and its "Movement" interface. The "FlatOccupancy" device, given its interest on the "Movement" interface, establish a connection with this new device. At the end of this interaction the resulting scenario is depicted in Figure 3.6.

Besides the easy integration of hardware and software from different vendors, another advantage of this approach is the compatibility with older software. In the example dis-

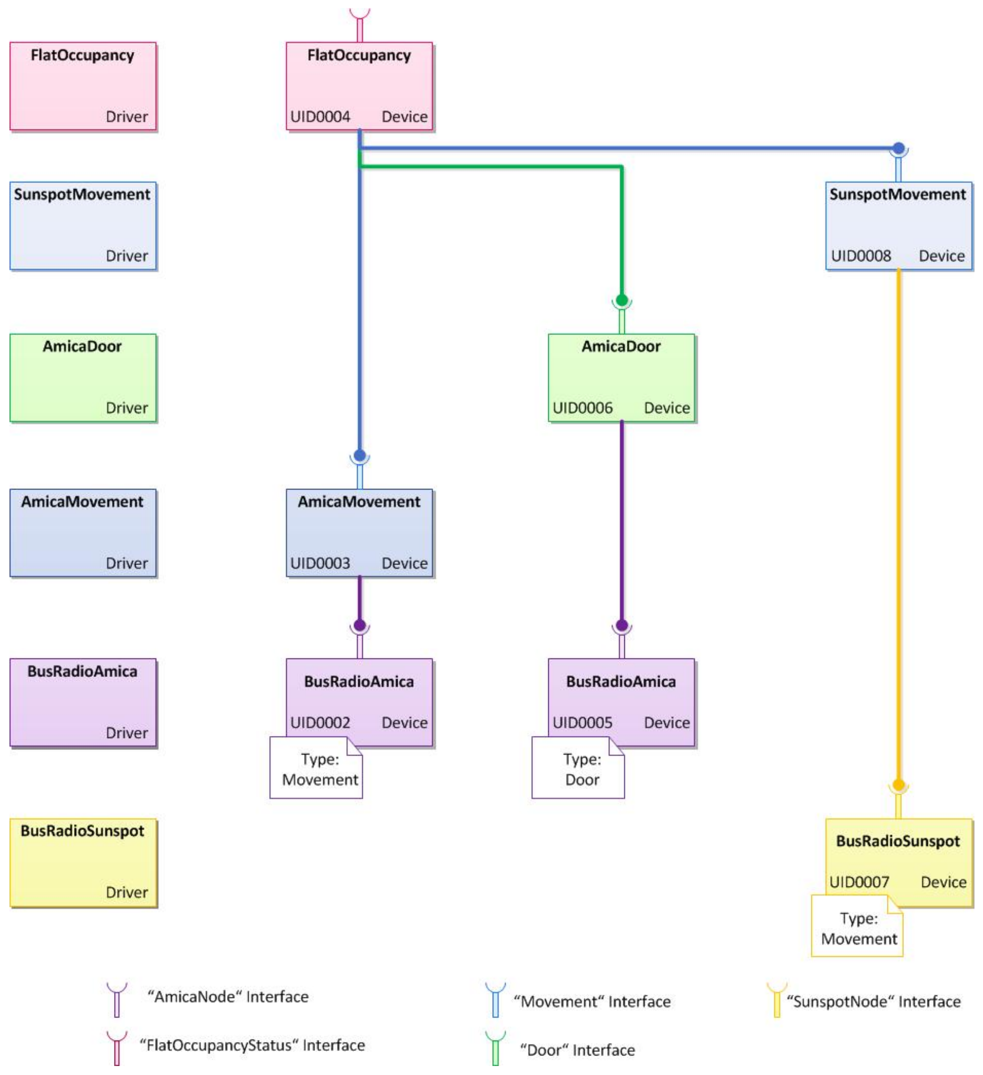


Figure 3.6: TinySEP example scenario: after the integration of new hardware.

cussed before, the "FlatOccupancy" driver and its devices could have been developed long time before the "SunspotMovement" and yet they remain compatible.

Note that in the previous example a new interface was introduced into the system, the "SunspotNode" interface. This interface, although not previously known by the drivers installed on the system, was integrated without any trouble. This was possible because if a particular driver or device is informed about an interface that it does not known, it simply considers as not being of interest. Therefore in the previous example, when the "AmicaMovement" driver and device, the "BusRadioAmica" driver and devices, the "AmicaDoor" driver and device and the "Flat Occupancy driver and device were notified about the "SunspotNode" interface, which they were not aware, they merely ignored, ie, they did not try to establish any form of communication with the device that offered it. This treatment allows different developers to develop their own solutions independently, without even knowing the other interfaces, giving greater freedom of choice to its users.

If a developer needs, he can extend an interface in order to include new functionalities. For example, consider that the "Movement" interface permits only to inform if a movement was detected or not and the exact instant that the detection was made. This interface may be perfect to the "AmicaMovement" device, however the "SunspotMovement" device can provide an additional information, as for example the exact coordinate from where the motion was detected. If the device just ceases to implement the "Movement" interface and starts to implement another interface, it will lose compatibility with the "FlatOccupancy" device. So what should be done is to deploy a second interface, such as "MovementWithCoordinates", which is an extension of the "Movement" interface. This new interface besides providing all information from "Movement" interface, it still make available the further data. Thus, with the availability of these two interfaces, the "SunspotMovement" device still compatible with the old and new solutions.

## 3.6 House Information

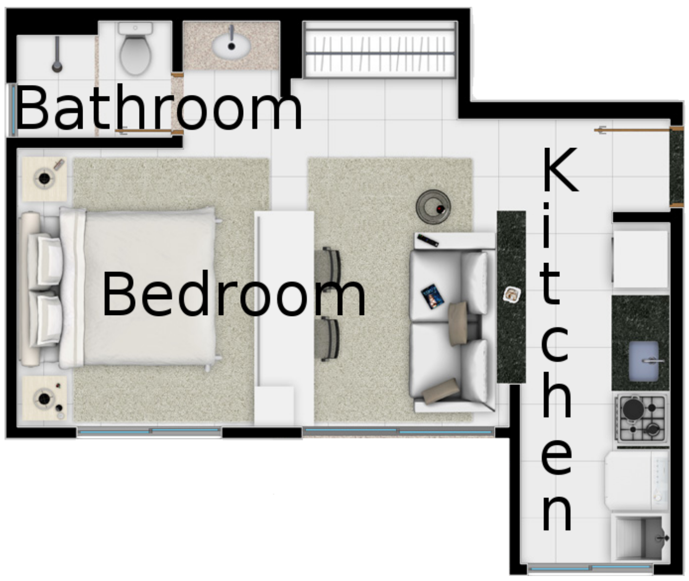
One of the main challenges of an AAL system is how to allow a single platform to be reused in several different homes, without performing any reprogramming. The set of residents who lives in each home are different from each others, hence the applications installed on the SEP should also be. According to what was presented in the sections 3.4 and 3.5 TinySEP allows in an easy and practical manner the integration of different software and sensor and actuator nodes, while at the same time it enables a system customization performed by the user. Therefore this problem is solved, however there is still a second one, the mapping of the structure of the house into the system

It is of upmost importance to the system to know how the rooms in the house are outlined and which sensor and actuator nodes are found in each of them. Only with those information the system will be able to make the right decisions. Imagine a feature responsible for turning the lights automatically on when someone enters in a room and off when he leaves. To facilitate the explanation, this application will be called as "Follow Me Light". If the SEP does not know in which room each light sensor is located, in the best case the "Follow Me Light" will not work. In the worst case the system will continuously turn all the house lights on and off, because it does not know how to proceed.

This example shows the importance of knowing the room that each sensor node is located, but it has not yet justified the importance of knowing the rooms interconnection. To accomplish this justification, now consider the following example. There is an application responsible for remembering the house resident that it is time to take his medicine, so whenever it is the time to taking a certain medication, a little robot brings it to the resident. Order for this robot to be able to move inside the house it must first know where the resident is right now and the path it must follow to find him. This requires that somehow the robot can access this information through the AAL system.

The alternative adopted by TinySEP was to save this information in a XML file. This XML file basically schematizes the main data of the house, such as the name and type of room and the other rooms that are interconnected with this one. Using this information it is possible to get an idea of the house plan, even if it is not possible to know the precise size of such rooms. The Figure 3.7 shows how the house plan on the left can be systematized in a XML file. It is interesting to note that for each room an Unique Identifier (UID) was included, because some of the rooms might have the same name or type, hence the best way found to differentiate them was through the use of an UID, which is created automatically by the system.

Despite the home information has been already structured into a XML file, it is still necessary to somehow turn those informations available to all the TinySEP applications. The first idea would be that the device manager provide such information to all applications that request it. However, there is a problem in this approach, because once defined all the fields that this XML file, it would be too costly to extend this solution, since it



```xml
<HOUSE_ROOMS>
  <ROOM>
     <UID>decb2a7f-88e6-4238-9d09-9fc2e932d4ec</UID>
     <NAME>Master Bedroom</NAME>
     <TYPE>Bedroom</TYPE>
    <NEIGHBORS>
       <ROOM_UID>0cf52171-0d00-486f-bbb2-e6ef950012b5</ROOM_UID>
       <ROOM_UID>9ffbd81c-9ab3-4bf7-8b13-8ab31634ebf5</ROOM_UID>
     </NEIGHBORS>
   </ROOM>
  <ROOM>
     <UID>0cf52171-0d00-486f-bbb2-e6ef950012b5</UID>
     <NAME>Bathroom</NAME>
     <TYPE>Bathroom</TYPE>
    <NEIGHBORS>
       <ROOM_UID>decb2a7f-88e6-4238-9d09-9fc2e932d4ec</ROOM_UID>
     </NEIGHBORS>
   </ROOM>
  <ROOM>
     <UID>9ffbd81c-9ab3-4bf7-8b13-8ab31634ebf5</UID>
     <NAME>Kitchen</NAME>
     <TYPE>Kitchen</TYPE>
    <NEIGHBORS>
       <ROOM_UID>decb2a7f-88e6-4238-9d09-9fc2e932d4ec</ROOM_UID>
     </NEIGHBORS>
   </ROOM>
 </HOUSE_ROOMS>
```

Figure 3.7: The systematization of a house plan into a XML file.

would be necessary to modify the hole platform. Ie if, for example, now it was necessary to include the size of each of the rooms, then it would be necessary to reprogram the platform in order to support this new information. The solution found and adopted was to create a new driver and its devices, named as "HouseInformation", that would serve as controller for those house informations. This approach in addition to facilitating the extension of the home information, also enables a parallel communication without the need of an intervention from the device manager. To allow this communication, the "HouseInformation" interface was developed. Through this interface the devices can get the house information as each of them need.

Once this driver is installed, it automatically creates a device, which is responsible for controlling all this information and make it available through the "HouseInformation" interface. The device is automatically created, because it does not have any pre-requisite for its operation and the information provided by it must always be available for the other drivers and devices. Those other devices that need to receive those informations, as for example the "Follow Me Light", can establish a connection through the "HouseInformation" interface. For example, the "AmicaMovement" device can make use of those informations to define the room, in which the movement was detected. So the name of all those rooms can be displayed to the user in order to allow it to choose one of them. The integration of the "HouseInformation" driver and device to the scenario from the Figure 3.6 result in the scenario shown by Figure 3.8.

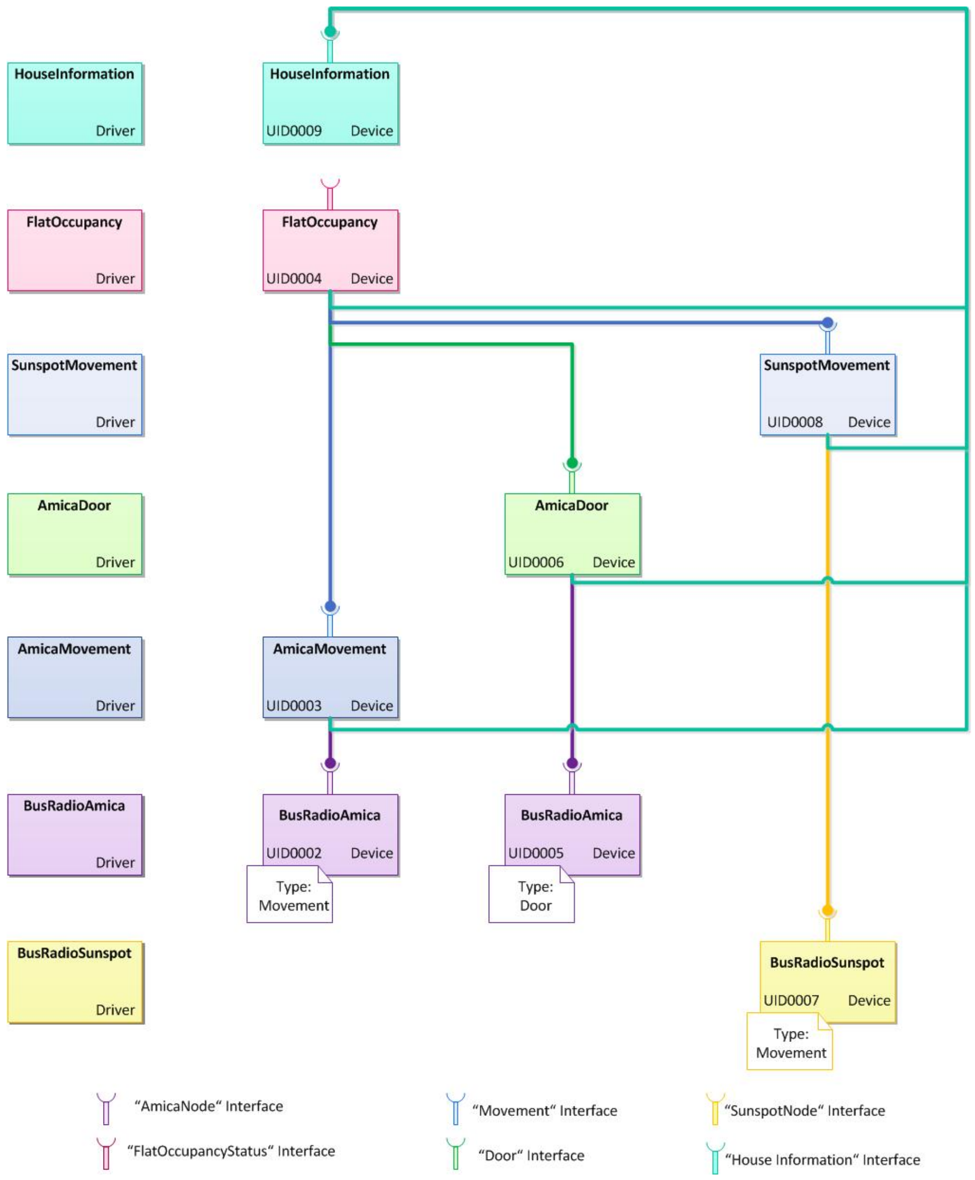


Figure 3.8: TinySEP example scenario: after the integration of the "HouseInformation" device.

The required updates should be made direct on the "HouseInformation" driver and device. Thus this update can be performed at runtime by the removal of this driver and installation of its new version, as described in section 3.4.

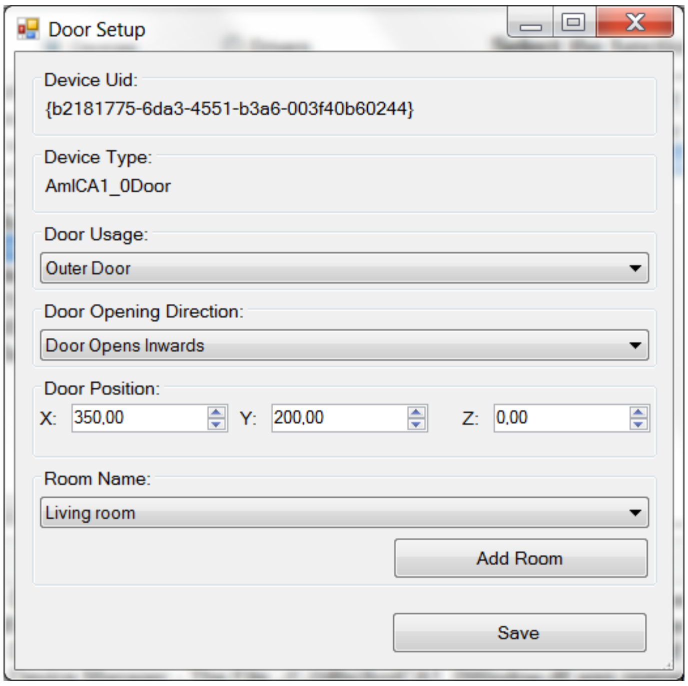
## 3.7 System Backup

TinySEP requires the user to provide a large amount of information during system installation. Those configuration may change according to the application that is being installed, because each of their specific informations. For example, the "AmicaDoor" device, that was first presented in the section 3.3, is capable of access informations such as the coordinates of sensor node, if the door makes connection with the outside of the house or if it is an internal door, the door opening direction and in which room is this door located. However, once the system is setup, the user will not have to perform any extra configuration, unless he wants to make system change, such as modifying the location of a sensor node, which will reflect on the alteration of such information in the system.

In order to facilitate this configuration, each device must provide a Graphical User Interface (GUI). Through this GUI , the users will be capable to performing all the requested configurations. If an application does not need the user input, then, off course, it would not need to provide a GUI. In the case of the "AmicaDoor" device, one feasible configuration GUI would be the one represented by Figure 3.9, where each of the parameters to receive a user input can be intuitively filled. In this example the parameter "Room Name" is populated according to the rooms provided by the "House Information" device, more specific through the "HouseInformation" interface.

However to avoid having to redo all of those settings, it is important that the SEP somehow saves those informations in case of the system is turned off, as in the case where for some reason the electricity supply is cut. TinySEP provides a system backup, in which all drivers and devices are saved along with their settings. Therefore, when the user restarts the platform, the system will automatically reconfigure it as it was when it was turned off.

Each driver and each device has its own data, but it is still necessary that the device manager save them all and somehow minister its reconfiguration. The solution adopted by TinySEP was to save the settings of each component in a XML file. Each driver and each device is responsible for managing its backup informations. Once concluded the backup, this XML is transferred to the device manager that performs the management of all these files. The device manager also makes a mapping of each of those files to its respective component. Therefore in order to reboot the system, the device manager will search all XML files saved to disk and send them to their respective owners, who will make use of this file to restore the previous settings.


Door Setup

Device Uid:
{b2181775-6da3-4551-b3a6-003f40b60244}

Device Type:
AmICA1_0Door

Door Usage:
Outer Door

Door Opening Direction:
Door Opens Inwards

Door Position:
X: 350,00 Y: 200,00 Z: 0,00

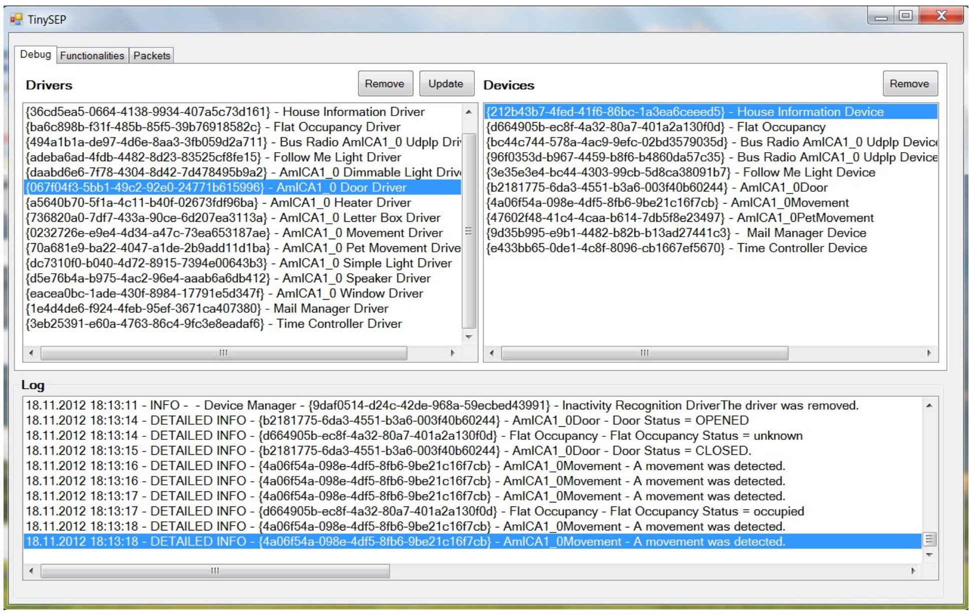
Room Name:
Living room

Figure 3.9: The configuration GUI from the "HouseInformation" device.

## 3.8 Sketch of the TinySEP GUI

As was presented in the section 3.7, some of the TinySEP applications can provide their own GUI, through which the user can make the necessary settings. However it is still necessary to somehow make available a GUI, whereby the user can perform the inclusion and removal of the driver and devices. Besides these two essential features, this GUI still need to allow the user to choose a certain device to be reconfigured, remembered that this setup GUI will be provided by the chosen device.

It is important to note, that once the platform is configured, the system is capable of operating without the user interaction. Meanwhile, there is some additional information that the GUI must provide, as an event log and a log for data packets received and sent. This event log will visually display all the occurred events, such as, if a new device has been created, if a door has been opened and if a motion has been detected. Together with those events, it is necessary to display two more informations, the exact time when this event was performed and which driver or device originated the log message. The data packets log will store and display the time at which a packet was received or sent and the data of this packet. Those logs should be saved by the system in order to maintain a history. Using those informations, the user can also discover why a particular application is not working properly. For example, the user realizes that the "FlatOccupancy" device is no longer functioning, ie, it always reports that the flat is not occupied status. Through the analysis of those logs, the user can identify that the door sensor node is out of battery, therefore the system does not recognize when he opens the door of his house. Hence, the "FlatOccupancy" device is not informed of this event.



```
{36cd5ea5-0664-4138-9934-407a5c73d161} - House Information Driver
{ba6c898b-f31f-485b-85f5-39b76918582c} - Flat Occupancy Driver
{494a1b1a-de97-4d6e-8aa3-3fb059d2a711} - Bus Radio AmICA1_0 Udplp Dri
{adeba6ad-4fdb-4482-8d23-83525cf8fe15} - Follow Me Light Driver
{daabd6e6-7f78-4304-8d42-7d478495b9a2} - AmICA1_0 Dimmable Light Driv
{067f04f3-5bb1-49c2-92e0-24771b615996} - AmICA1_0 Door Driver
{a5640b70-5f1a-4c11-b40f-02673fdf96ba} - AmICA1_0 Heater Driver
{736820a0-7df7-433a-90ce-6d207ea3113a} - AmICA1_0 Letter Box Driver
{0232726e-e9e4-4d34-a47c-73ea653187ae} - AmICA1_0 Movement Driver
{70a681e9-ba22-4047-a1de-2b9add11d1ba} - AmICA1_0 Pet Movement Drive
{dc7310f0-b040-4d72-8915-7394e00643b3} - AmICA1_0 Simple Light Driver
{d5e76b4a-b975-4ac2-96e4-aaab6a6db412} - AmICA1_0 Speaker Driver
{eacea0bc-1ade-430f-8984-17791e5d347f} - AmICA1_0 Window Driver
{1e4d4de6-f924-4feb-95ef-3671ca407380} - Mail Manager Driver
{3eb25391-e60a-4763-86c4-9fc3e8eadaf6} - Time Controller Driver
```

```
{212b43b7-4fed-41f6-86bc-1a3ea6ceeed5} - House Information Device
{d664905b-ec8f-4a32-80a7-401a2a130f0d} - Flat Occupancy
{bc44c744-578a-4ac9-9efc-02bd3579035d} - Bus Radio AmICA1_0 Udplp Device
{96f0353d-b967-4459-b8f6-b4860da57c35} - Bus Radio AmICA1_0 Udplp Device
{3e35e3e4-bc44-4303-99cb-5d8ca38091b7} - Follow Me Light Device
{b2181775-6da3-4551-b3a6-003f40b60244} - AmICA1_0Door
{4a06f54a-098e-4df5-8fb6-9be21c16f7cb} - AmICA1_0Movement
{47602f48-41c4-4caa-b614-7db5f8e23497} - AmICA1_0PetMovement
{9d35b995-e9b1-4482-b82b-b13ad27441c3} -  Mail Manager Device
{e433bb65-0de1-4c8f-8096-cb1667ef5670} - Time Controller Device
```

```
18.11.2012 18:13:11 - INFO -  - Device Manager - {9daf0514-d24c-42de-968a-59ecbed43991} - Inactivity Recognition DriverThe driver was removed.
18.11.2012 18:13:14 - DETAILED INFO - {b2181775-6da3-4551-b3a6-003f40b60244} - AmICA1_0Door - Door Status = OPENED
18.11.2012 18:13:14 - DETAILED INFO - {d664905b-ec8f-4a32-80a7-401a2a130f0d} - Flat Occupancy - Flat Occupancy Status = unknown
18.11.2012 18:13:15 - DETAILED INFO - {b2181775-6da3-4551-b3a6-003f40b60244} - AmICA1_0Door - Door Status = CLOSED.
18.11.2012 18:13:16 - DETAILED INFO - {4a06f54a-098e-4df5-8fb6-9be21c16f7cb} - AmICA1_0Movement - A movement was detected.
18.11.2012 18:13:16 - DETAILED INFO - {4a06f54a-098e-4df5-8fb6-9be21c16f7cb} - AmICA1_0Movement - A movement was detected.
18.11.2012 18:13:17 - DETAILED INFO - {4a06f54a-098e-4df5-8fb6-9be21c16f7cb} - AmICA1_0Movement - A movement was detected.
18.11.2012 18:13:17 - DETAILED INFO - {d664905b-ec8f-4a32-80a7-401a2a130f0d} - Flat Occupancy - Flat Occupancy Status = occupied
18.11.2012 18:13:18 - DETAILED INFO - {4a06f54a-098e-4df5-8fb6-9be21c16f7cb} - AmICA1_0Movement - A movement was detected.
18.11.2012 18:13:18 - DETAILED INFO - {4a06f54a-098e-4df5-8fb6-9be21c16f7cb} - AmICA1_0Movement - A movement was detected.
```

Figure 3.10: A possible sketch of the TinySEP GUI.

The TinySEP GUI should be simple and at the same time enable all previous information to be available to the user. Basically it can be divided into four parts. The first one is composed by the events log, which is one of the main features of the system and therefore should be always visible to the user. This log is depicted in the Figure 3.10. The second one enables the user to control all the devices and drivers, ie, the user can remove them or add new drivers. It is represented in the Figure 3.10. The third part allows the user to modify the drives and devices settings, ie, it provides an interface through which the user can select the component he wants to reconfigure. The last one is composed by the log for data packets received and sent.

## 3.9 The Hardware Behind TinySEP

Before prototyping the TinySEP model, it is necessary to define the hardware that will be initially supported by the platform. Through the choice of this hardware it will be possible to validate and evaluate the applicability of the model previously proposed. Since one of its goals is to enable the easy adoption and integration of new sensor and actuator nodes, which may come from different vendors, then it is necessary to choose at least two hardware components from different suppliers.

The hardware that will be used should give in the form of Wireless Sensor Network (WSN), once its implementation cost is much smaller then those that are given in the form of wired sensor network. Furthermore, it is much easier to perform modifications on the location of a WSN sensor node. Even with the power consumption problem, the advantages from the WSN sensor nodes are still more numerous then the ones found on the wired sensor nodes. Among the wide range of WSN nodes available on the market, the AmICA and the Sun SPOT were chosen as the initially supported hardware. The reasons that led to this choice will be presented in detail within this section.

### 3.9.1 AmICA

The AmICA sensor nodes were developed by Sebastian Wille, my co-adviser. Hence the first advantage of adopting this harware is the unrestricted access to the AmICA sensor nodes and its documentation. Beyond that, given the necessity of a variety of sensors and actuator nodes Sebastian developed some new sensor nodes, as for example the speaker and the heater sensor nodes. With the integration of this new hardware it was possible to develop more complex scenarios that include a significant variety of sensor nodes.

The AmICA sensor nodes, see Figure 3.11, are small in size and support various sensors and actors, as for example, temperature, light, movement, reed-switch and acceleration sensors and LED actuators. They have very low power consumption in sleep mode, self-wake-up capabilities, a fast wake-up time and a self-programming capability. They can be powered through two AA batteries or through a small lithium-polymer battery (Wille et al. 2010; Wille et al. 2010).

A sensor node transmission range depends on frequency band, modulation, and the physical properties of the environment. The AmICA sensor nodes uses a FSK modulation in the 433, 868 or 915MHz. In most countries at least one of these frequency bands is licence free, as for example the 868MHz frequency can be used in Europe. One single AmICA node can send up to 21 818 packets within one hour, however this value can be programmed and each sensor node can send those information in a different time (Wille et al. 2010; Wille et al. 2010).

In order to perform the evaluation of the TinySEP prototype the following AmICA sensor nodes will be used:

- Movement sensor node - This sensor node is capable o detecting a motion in 280 meters range. It sends a movement packet every 2 seconds.

- Reed-switch sensor node - It can be used for the detection of opened or closed door or window. It sends a packet every second.

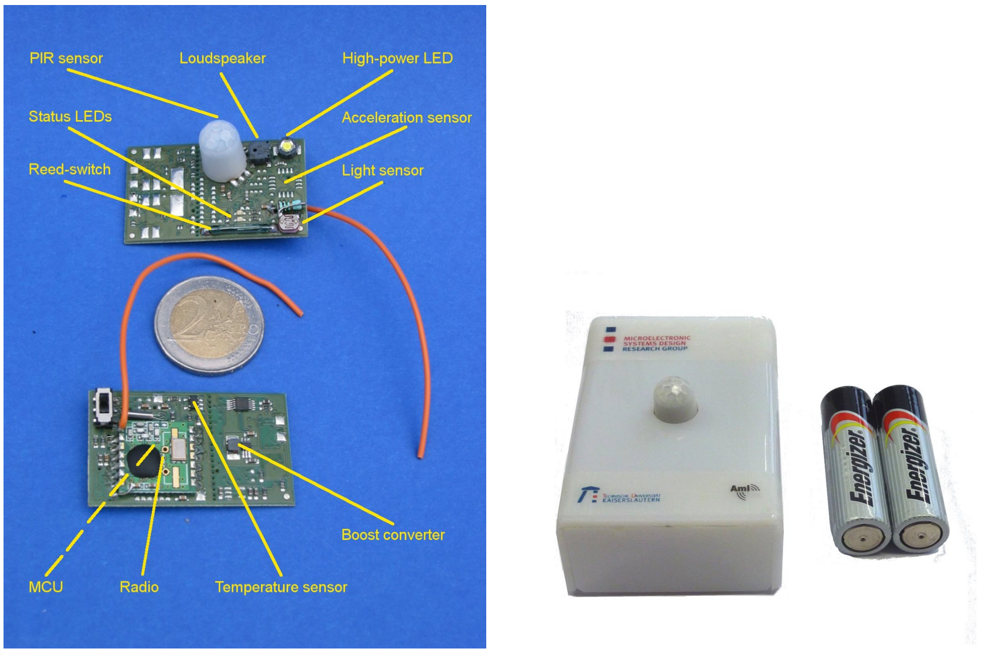


Figure 3.11: AmICA WSN sensor node; without (left) and with (right) housing.

- Heater sensor node - It allows the configuration of the heater's potency. This sensor node send a packet every 10 seconds with the current heater potency.
- Light sensor node - It sends a packet with the light brightness every 10 seconds. It also has a actuator that allows the light brightness to be changed.
- Speaker sensor node - It sends volume information in a packet every 10 seconds. The volume can also be changed with the help of a actuator.
- Pet collar sensor node - It is used to identify a pet motion and it sends this information every 2 seconds.
- Modified AmICA sensor node - Provides the communication between the computer and the sensor nodes, see Figure 3.12.

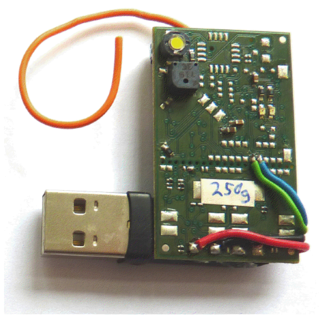


Figure 3.12: A modified AmICA sensor node, that was built on a board with a USB port for power supply.

It is important to note that one single node can be from one or more types, ie, one AmICA sensor node can be a movement and a reed-switch sensor node. In the previously list, those combinations were omitted, because it would be redundant to present them since their functionalities are the same presented in each type.

Table 3.1: Payload description of the AmICA node protocol

| Meaning | Payload | Usage |
|---|---|---|
| **Keep Alive** | 82 0 **A B C D F G H I J K L M N O**<br>Seconds since start up =<br>**A** * $256^3$ + **B** * $256^2$ + **C** * 256 + **D**<br>Battery Statys in mV =<br>(**F** * 256 + **G**)/ 5242880<br>Vendor ID = **H** * 256 + **I**<br>Product ID = **J** * 256 + **K**<br>Serial Number =<br>**L** * $256^3$ + **M** * $256^2$ + **N** * 256 | The keep alive payload is sent by every AmICA sensor node. It is used in order to give some informations, such as the battery status, the seconds since the sensor node was turned on and the node serial number, product ID and supplier. Through this payload it is possible to know the sensor type, ie, if it is a movement sensor node or a reed-switch or any of the types previously listed. |
| **Reed-Switch Status** | 100 3 0 **A**<br>Closed if **A** = 0 and<br>Opened if **A** = 1 | The reed-switch status payload is sent by a reed-switch sensor node in order to inform the current status of a door, window or letter box. |
| **Light Control** | 5 2 0 **A**<br>**A** = light brightness value in percentage (%) | The light control payload is sent only by the light sensor nodes. Through this payload it is possible to known the current light brightness and to reconfigure this value, once this sensor node is also equipped with an actuator. |
| **Movement Detection** | 100 5 0 **A**<br>If **A** = 1 a motion was detected, otherwise **A** = 0 | The movement detection payload is only sent by the motion sensor node. |
| **Speaker Control** | 5 4 0 **A**<br>**A** = volume in percentage (%) | The speaker control payload is sent by a speaker sensor node. This payload is used to inform the current volume and to change its value. |
| **Pet Movement Detection** | 100 5 0 **A**<br>If **A** = 1 a motion was detected, otherwise **A** = 0 | The pet movement detection payload is only sent by the pet collar sensor node. |
| **Heater Control** | 5 8 0 **A**<br>**A** = the heater potency in percentage (%) | Through the heater control payload it is possible to know the current heater potency and to reconfigure its value. This payload is sent only by a heater sensor node. |

The AmICA sensor nodes have their own communication protocol, named as AmICA node protocol. It ensures a basic communication functionality between the AmICA sensor nodes and the computer, through the modified AmICA sensor node. The structure of the protocol is simple and a coordinator node is not needed. In this protocol, the packets transmitted between the sensor nodes are composed by a packet size identifier, a header and a payload, as illustrated in the Figure 3.13. The packet size identifier is used to inform the packet beginning and its size. The header consists of seven bytes, which are, respectively, a header value, the source address, the destination address, the netgroup, the sequence number, the checksum and the payload length. The combination of the netgroup and the source address identifies the sender node. In the same way the combination of the destination address and the netgroup identifies the receiver. Thus it is possible to have 65536 different sensor nodes, once each one of those fields may range from 0 to 255. The payload can contain up to 255 bytes and each of them has a specific meaning (Wille et al. 2010; Wille et al. 2010). A list with all the recognized payloads is presented in the Table 3.1.

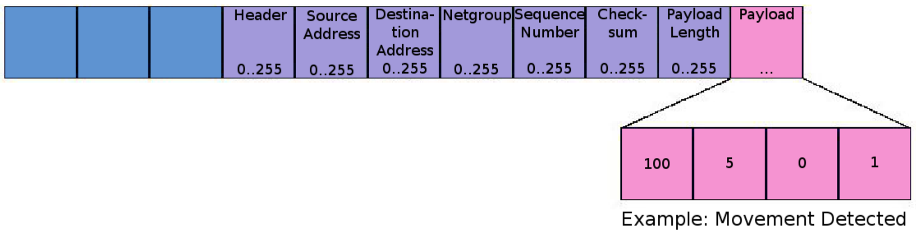


Figure 3.13: Packet description of the AmICA node protocol.

### 3.9.2 Sun SPOT

The Sun SPOT sensor nodes were developed by the Sun Microsystems. Unlike the AmICA nodes, the Sun SPOT sensor nodes are commercialized. Thus, the first advantage on their use is that they are produced in a large scale, therefore it is easier to buy the Sun SPOT sensor nodes then the AmICA sensor nodes. The second advantage of this sensor nodes is that beyond the complete documentation, they also provide an emulator capable of running Sun SPOT software just the way the physical Sun SPOT sensor nodes does.

The Sun SPOT sensor nodes, see Figure 3.14, are Java programmable embedded devices. Each node has accelerometer, temperature and light sensors, radio transmitter, eight multicolored LEDs, 2 push-button control switches, 5 digital I/O pins, 6 analog inputs, 4 digital outputs, and a rechargeable battery. The power consumption is extremely low during deep sleep, however if the device sensors and radio are turned full on and left to run continuously, the battery will only last a day.

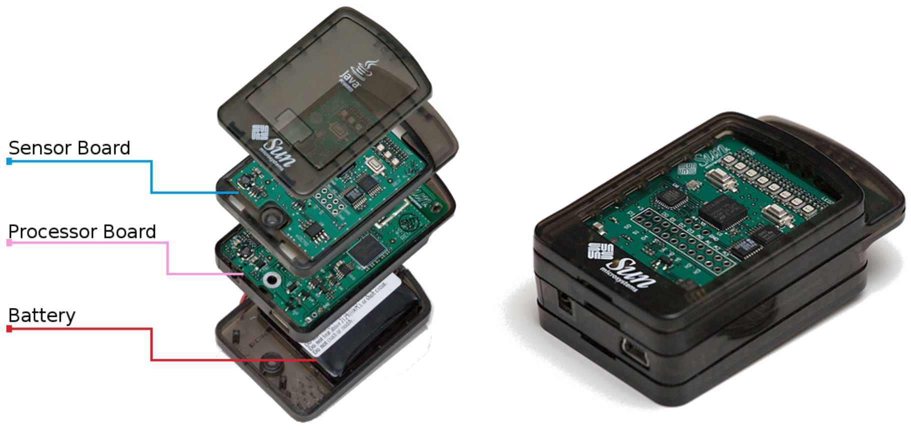


Figure 3.14: Sun SPOT sensor node.

Programming the Sun SPOT is surprisingly easy, because of the Sun SPOT Software Development Kit, that can be easily integrated to the IDE NetBeans. In order to perform the evaluation of the TinySEP prototype the following types of Sun SPOT nodes were programmed:

- Movement sensor node - Through the accelerometer sensor it is possible to detect if the sensor was moved. This sensor node sends a movement packet every 2 seconds.
- Temperature sensor node - Every 10 seconds this sensor node sends a packet with the temperature detected by the temperature sensor.
- Light sensor node - A packet with the brightness detected by the light sensor is sent every 10 seconds. This sensor node does not have any actuator, therefore it is not possible to set the light.
- Pet collar sensor node - It is used to identify a pet motion through the accelerometer sensor that should be fixed on the pet collar. This information is sent every 2 seconds.
- Basestation - Provides the communication between the computer and the sensor nodes.

Each Sun SPOT node can implement one of the features described previously or more. This combination depends of the intended usage, as for example, it make no sense to have a sensor node equipped with a pet collar and at same time with the temperature, because it would be not possible to know which room this temperature refers, since the pet can move around the house.

Table 3.2: Payload description of the Sun SPOT node protocol

| Meaning | Payload | Usage |
|---|---|---|
| **Movement Detection** | 2 1 0 **A**<br>If **A** = 1 a motion was detected, otherwise **A** = 0 | The movement detection payload is only sent by the motion sensor node. |
| **Temperature Status** | 3 1 **A B**<br>The bytes **A** and **B** generates the temperature value in degree Celsius (ºC) | The temperature sensor node sends a packet with the temperature value in degree Celsius (ºC). |
| **Light Status** | 4 1 0 **A**<br>**A** = light brightness value in percentage (%) | The light status payload informs the light brightness in percentage (%). This payload is only sent by the light sensor node. |
| **Pet Movement Detection** | 5 1 0 **A**<br>If **A** = 1 a pet motion was detected, otherwise **A** = 0 | The pet movement detection payload is only sent by the pet motion sensor node. |

In order to allow an easy communication between the Sun SPOT sensor nodes and the TinySEP prototype, a communication protocol was created. This protocol was named as Sun SPOT node protocol and it is quite simple, once it merely schematizes how the packages will be. In this protocol, each packet has a packet size identifier, a header and a payload, as illustrated in the Figure 3.15. The packet size identifier, in the same way as in the AmICA node protocol, is used to inform the packet beginning and its size. The header consists of eleven bytes, which are, respectively, the source address (8 bytes), the sensor type, the checksum and the payload length. The payload can contain up to 255 bytes and each of them has a specific meaning. A list with all the recognized payloads is presented in the Table 3.2.

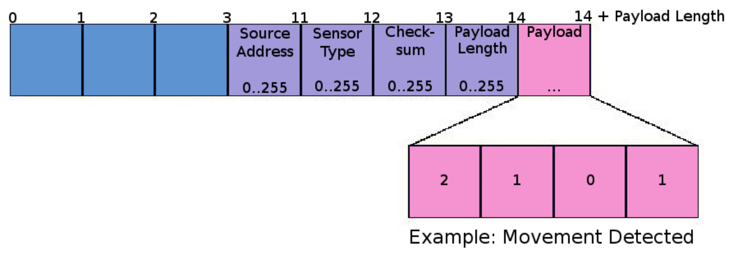


Figure 3.15: Packet description of the Sun SPOT node protocol.

# 4 TINYSEP: PROTOTYPE

In this chapter a prototype for the TinySEP model, which was discussed in detail in the chapter 3, will be presented. The TinySEP prototype was developed using the IDE Microsoft Visual Studio 2010 and the programming language C#. All the graphical user interfaces were developed using the Windows Forms. The programming language C# was chosen to make this implementation, because differently from other programming languages, like Java, in C# by passing a complex object as parameter, in fact the object reference is being passed, ie, the parameters in C# are passed by reference and not by value as in Java. Thus, when the object referred on a function parameter is modified, the original object is being modified. Another advantage from this programming language is the possibility to dynamically open DLL files to the program at run time. This DLL inclusion was extremely important, because it allows more freedom and scalability to the system. Another important factor for this choice was the whole background that I had with the development of software using this programming language and this IDE. On the other hand, when those choices were made, it became clear that this prototype would be limited to computers equipped with Windows operating system. However even with this disadvantage the C# was still the best choice.

The TinySEP prototype accurately follows the model proposed in the previous chapter. In order to implement the driver model concept, 2 programming interfaces were defined, the IDriver, which abstracts the functionalities that should be provided by the driver, and the IDevice, whose methods define the device features. The driver model prototype, will be discussed in the section 4.1. Programming interfaces were also used to implement the interfaces proposed in the section 3.2. Each interface is composed by two programming interfaces, the Listener and the Provider. This topic is going to be explained in detail in the section 4.2.

As presented in the section 3.3, the device manager plays an important role on the management of the drivers and devices. To allow this management a bidirectional communication between the device manager and the drivers and devices is established. At the same time this device manager also establish the communication with the TinySEP GUI. The explanation of the device manager prototype is presented in the section 4.3.

In order to validate this platform it is important to define the initial group of applications that will be supported. In total 20 different applications were developed. To better explain the applications, they were slip into 4 layers, the Hardware Connection, the Hardware Abstraction, the Intermediate Services and the Ambient Assisted Living Services. Those layers together with their applications will be discussed in detail in the section 4.4.

Table 4.1: Methods defined in the IDriver programming interface.

| Method | Meaning |
|---|---|
| **Guid getDriverUid();** | Method used to obtain the driver UID. It is widely used by the device manager in order to build log messages. |
| **string getDriverName();** | This function returns the driver name and it is also deeply used by the device manager to display to the user the informations from this driver. |
| **ArrayList getDevicesList();** | Method that returns a list with all the devices that were created by this driver. |
| **void newInterfaceFound( object newInterface);** | Method used by the device manager to inform the driver that a new interface was found. |
| **void interfaceRemoved( object interfaceRemoved);** | The device manager makes use of this function to inform the driver that a specific interface was removed. |
| **int installDriver( IDeviceManager deviceManagerRef);** | Method used by the device manager to perform the driver installation. |
| **void installDriverFromBackup( IDeviceManager myDeviceManager, XmlDocument backupFile);** | Function which installs a driver from its previous settings that were saved in a backup. |
| **Dictionary<string, string> getAllFunctions();** | Function that provides all the additional features of this driver, such as a GUI to perform certain configurations. |
| **void loadFunctions(string functionName);** | Method used to execute one of the driver functionalities that was provided by the "Dictionary<string, string> getAllFunctions()" method. |

## 4.1 Driver Model

In the TinySEP model, it was proposed that each application should be encapsulated into a DLL and that each of them should be composed by the combination of a driver and its devices. Therefore every application must have at least a class that implements the driver functionalities and another one that implements the device functionalities, see Figure 4.1. Those functionalities that each one of those classes must support are defined through programming interfaces, however, how each of them will be programmed is a decision from the developer. It is important to observe that an application may have other classes inside its project, as for example a GUI to perform configurations and auxiliary classes.

A programming interface that every driver must support was named as IDriver. The IDriver interface defines some basic features that allow the device manager to install a driver, to inform that a new interface was found or removed and to obtain some information from this driver, as its name, its UID and a list with its devices. Furthermore, the driver should allow the user to access its information to perform reconfigurations or just to analyse the current configurations. The Table 4.1 summarizes the methods defined by

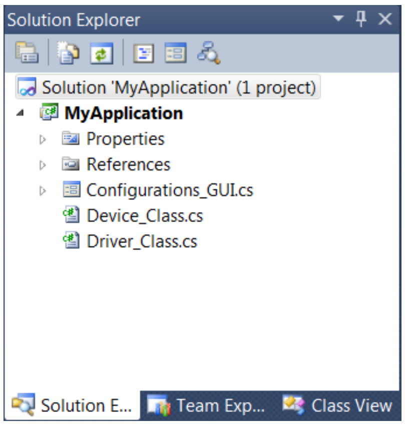


Figure 4.1: The project structure from a TinySEP application. This project is composed by 3 classes, one for the device, another for the driver and the last one is for the configuration graphical user interface.

the IDriver interface and the meaning from each of them.

In a similar way to the drivers, the devices are also characterized according to a programming interface, referred as IDevice. This interface defines methods used to create a device, to establish a connection with other devices, to obtain informations from this device, such as its configuration parameters, its UID and its type. This interface also allows the user to access additional functionalities, as for example a GUI that displays the current configuration from this device. The Table 4.2 shows the methods defined by the IDevice interface.

If a particular application does not implement all the features described by the programming interface, then it does not implements it, even if it is only one of the methods described. Ie, a class that abstracts a device need to implement all the functions described by the IDevice programming interface. Likewise, a driver must implement all the methods described by the IDriver programming interface.

## 4.2 Signal Slot Model

As presented in the section 3.2, the signal slot model from TinySEP allows a publish/-subscribe event notification, which is made through the interfaces. In order to implement these interfaces defined in the TinySEP model into the prototype, the programming interfaces were used. Each interface is composed by two programming interfaces, referred as Listener and Producer. The programming interface Producer is implemented by the device that provides the interface, while the Listener is implemented by the device that desires to receive the notifications from the events.

To better understand how the signal slot model was prototyped, see the Figure 4.2, in which the Device A provides the IMovement interface, which is used by the Device B to establish a connection. The IMovement interface is composed by the programming interface IMovementProvider, which is implemented by the Device A. The IMovement-Provider allows the Device B to obtain some information, as for example, when the last movement was detected and if this movement happened inside or outside the house. On the other hand, the IMovementListener is used by the Device A to inform the Device B about an event of motion detection. Therefore at the same moment that the Device A detects a motion, it forwards it to the Device B, through the programming interface

IMovementListener.

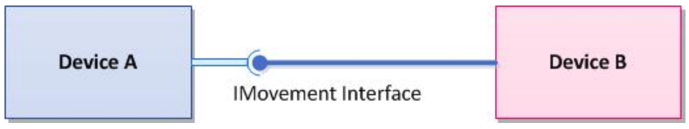


Figure 4.2: Connection establishment between a Device A and a Device B using the IMovement interface.

It is important to observe that those programming interfaces, which were used to prototype a TinySEP interface, only defines the methods that must be implemented, not how they should be implemented, because this implementation may change according to the application. A list with all the TinySEP interfaces already defined is presented in the Appendix A.

Table 4.2: Methods defined in the IDevice programming interface.

| Method | Meaning |
|---|---|
| **Guid getDeviceUid();** | Method used to obtain the device UID. |
| **string getDeviceType();** | This function returns the device type, which will be the same for every device created by the same driver. |
| **int connectDeviceToOtherDevice( object myReference);** | Method used by another device to inform this one that it wants to establish a connection between them. |
| **int connectToDevice( object deviceReference);** | Function used by the driver or by this device to indicate that it must establish a connection with the device referred in the function parameter. |
| **void newInterfaceFound( object newInterface);** | Method used by the device manager to inform the device that a new interface was found. |
| **void interfaceRemoved( object interfaceRemoved);** | The device manager makes use of this function to inform the device that a specific interface was removed. |
| **int createDevice( IDeviceManager myDisposer);** | The driver makes use of this method to perform the device creation. |
| **int createDeviceFromBackup(IDeviceManager myDeviceManager, XmlDocument backupFile);** | The driver makes use of this function to create the device from it previous configuration, which was saved in a backup. |
| **Dictionary<string, string> getAllFunctions();** | Function that provides all the additional features of this device, such as a GUI to perform certain configurations. |
| **void loadFunctions(string functionName);** | Method used to execute one of the device functionalities that was provided by the "Dictionary<string, string> getAllFunctions()" method. |
| **Dictionary<string, object> getDeviceParameters();** | This function returns all the configuration parameters from this device. |
| **void setDeviceParameters(Dictionary<string, object> newDeviceParameters);** | Method used to edit the device configuration parameters. |

## 4.3 Device Manager

As presented in the section 3.3, the TinySEP model has a device manager, which is responsible by the management of drivers and devices. With the help of the device manager other drivers and devices are informed about the installation or removal from a specific interface. This communication, provided by the device manager, performs an important role, because it is due to it that new devices can be created or removed. However, the device manager does not interfere on the internal event communication, once this is made direct between the devices. In parallel to the devices and drivers communication, the device manager is also responsible to perform an interconnection with the Graphical User Interface (GUI), whose sketch was defined in the section 3.8. The device manager can be seen as a central piece of the TinySEP prototype.

The device manager was prototyped in a class named as CDeviceManager which implements two programming interfaces, the IDeviceManager and the IGuiDeviceManager. The IDeviceManager defines methods through which the driver and devices can communicate with the device manager. Those methods allow drivers and devices to register themselves in the device manager and to obtain a list with all the devices already installed and or a device from a specific UID. They also enable the drivers and devices to add a log message or to save, restore or remove a backup file from a driver or from a device. This communication between the device manager and the drivers and devices is not made on an unidirectional way, ie, in the same way that the drivers and devices can request some capabilities from the device manager through the programming interface IDeviceManager, the device manager can also make requisitions from the drivers and devices, respectively, through the programming interfaces IDriver and IDevice.

A programming interface IGuiDeviceManager allows the user to perform modifications on the system currently installed through the system GUI, as for example, remove a driver or a device form the system, or remove the whole applications. Furthermore, this programming interface also enables the GUI to send events to the device manager, as for example one event to notify that the application is being closed by the user, another one to update the list of drivers already installed and one event to inform the device manager that the user want to start the system using the backup. In the same way that the GUI can communicate with the device manager, it can also communicate with the TinySEP GUI through the programming interface IGui, which is implemented by the TinySEP GUI. This programming interface provides methods through which the device manager can use to show a list with all the drivers, another list with the devices and functionalities to add the log messages.

## 4.4 Applications

The model proposed in the chapter 3 does not specify which applications must be developed by the prototype. In fact, those definitions were not made, because the idea behind this prototype is that it can support a wide range of applications, which can supply different capabilities, as for example one that connects a specific hardware with the platform and another that can controls the house lights.

In this prototype 20 different applications were developed. To facilitate the explanation, they will be divided in 4 layers, as illustrated by the Figure 4.3. The 4 layers, Hardware Connection, Hardware abstraction, Intermediate Services and Ambient Assisted Living Services will be presented in detail together with their applications in the next sections.

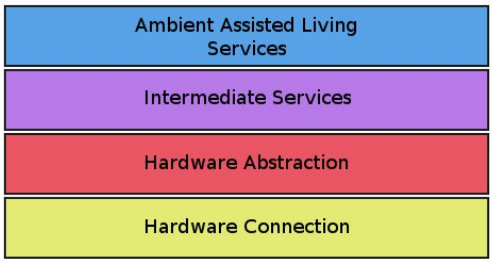


Figure 4.3: TinySEP applications layers.

### 4.4.1 Hardware Connection

The Hardware Connection layer is composed by applications that are responsible to establish the connection between the hardware and the software. Through those applications the software can communicate with the hardware and vice versa. To each hardware from a different supplier introduced into the system, a new application must be provided in order to enable this communication, which can be made through a serial port or through the Internet.

As was defined in the section 3.9, this prototype will initially support the AmICA sensor nodes and the Sun SPOT sensor nodes. Therefore, at least one application responsible to perform the communication between the TinySEP and the AmICA hardware and another one between the TinySEP and the Sun SPOT hardware must be provided. In the Hardware Connection layer, 3 different applications where developed, the Bus Radio AmICA, the Bus Radio AmICA UDP/IP and the Bus Radio Sun SPOT UDP/IP.

#### *4.4.1.1 Bus Radio AmICA*

The Bus Radio AmICA makes the connection between the AmICA sensor nodes and the software. The Bus Radio AmICA driver receives the data supplied by the modified AmICA sensor node (see Figure 3.12) through a serial port. This driver can also send packet data to the AmICA sensor node through the same serial port.

Each Bus Radio AmICA device abstracts a sensor node. Ie, in a house equipped with 10 AmICA sensor nodes, at least 10 Bus Radio AmICA devices will exist. Thus every time the driver receives a packet, whose header indicates a device already created, the driver will forward the payload encapsulated on this packet to the device referred on the message header. However, if the packet is from the type "Keep Alive" and its header indicates a device that does not exist, then the driver will create a new device to abstracts this sensor node.

All Bus Radio AmICA devices provide the IBusRadioAmICA interface. Through this interface the device can inform other devices connected to it about the receipt of new data packets. Through this same interface, the other devices can obtain the AmICA sensor node identifier and can also send data to this sensor node. It is interesting to note that this device does not make any analysis on the payload content, it just forward this payload to the devices that desires to receive those informations.

#### *4.4.1.2 Bus Radio AmICA UDP/IP*

The Bus Radio AmICA UDP/IP is also responsible to establish a connection between the TinySEP and the AmICA sensor nodes. The only difference between this application and the Bus Radio AmICA, previously presented, is that this application does not receive the data through the serial port. It recives them through the Internet using an UDP/IP connection. This application was developed primarily with the goal to allow the realization of simulations from the behaviour of the AmICA sensor nodes, as will be discussed in detail in the chapter 5.

#### *4.4.1.3 Bus Radio Sun SPOT UDP/IP*

The Bus Radio Sun SPOT UDP/IP is an application that makes a connection between the Sun SPOT sensor nodes and the software. The Bus Radio Sun SPOT UDP/IP driver is capable to receive packets through the Internet with an UDP connection. Differently from the Bus Radio AmICA UDP/IP, this application is not used only with the purpose of make simulations. It is also used to receive real data from the Sun SPOT sensor nodes.

The Sun SPOT sensor nodes have a complete Software Development Kit for Java, however the TinySEP prototype was developed using the programming language C#. The solution adopted to allow this integration was to develop an application using the programming language Java, which forwards those data packets through UDP/IP. Those data packets are received by the Bus Radio Sun SPOT UDP/IP. This program in Java, makes a connection with the basestation through a serial port. Once this program receives an information from the sensor nodes, it just encapsulate it into an UDP datagram and sends it to the Bus Radio Sun SPOT UDP/IP.

The whole process of creation from the Bus Radio Sun SPOT UDP/IP devices is made in a similar way as the Bus Radio AmICA and the Bus Radio AmICA UDP/IP, however the Bus Radio Sun SPOT UDP/IP driver can create a new device from any data packet that it receives, ie, it does not need to wait for a data packet from with a "Keep Alive" payload. All the Bus Radio Sun SPOT UDP/IP devices provide the IBusRadioSunSPOT interface.

### 4.4.2 Hardware Abstraction

The applications of the hardware abstraction layer are responsible for converting the raw data into usable data and vice versa, as for example, convert the sensor raw data into a movement data. A device from this layer receives this raw data from the devices from the Hardware Connection layer, through the IBusRadioAmica interface or through the IBusRadioSunSPOT interface.

For every device from the Hardware Connection layer, there is one or more devices from the Hardware Abstraction layer. For example, for a sensor node that is equipped at the same time with a movement sensor and light actuator, a Hardware Connection device will be created and two Hardware Abstraction devices, one for the movement sensor and another one for the light actuator.

#### *4.4.2.1 AmICA Movement*

The AmICA Movement application is responsible for converting the raw data from the AmICA movement sensor node into movement data. In order to make this conversion, the AmICA Movement device analyses the payload information received from the Bus Radio AmICA device or from the Bus Radio AmICA UDP/IP device, through the IBusRadioAmICA interface. The movement information is provided by the AmICA Movement device through the IMovement interface.

#### *4.4.2.2 AmICA Door*

The AmICA Door converts the raw data from the AmICA reed-switch sensor node into door data. In order to make this conversion, its device analyses the payload information received from the Bus Radio AmICA device or from the Bus Radio AmICA UDP/IP device, through the IBusRadioAmICA interface. Every AmICA Door device implements the IDoor interface, through which the door data is turn available.

#### *4.4.2.3 AmICA Window*

The AmICA Window application is responsible for converting the raw data from the AmICA reed-switch sensor node into window data. In order to make this conversion, the AmICA Window device analyses the payload information received from the Bus Radio AmICA device through the IBusRadioAmICA interface. The window information provided by the AmICA Window device through the IWindow interface.

#### *4.4.2.4 AmICA Simple Light and AmICA Dimmable Light*

The AmICA Simple Light and AmICA Dimmable Light are responsible for converting the raw data from the AmICA light sensor node into light data and also responsible for sending information to this sensor node, as for example a message to turn the light on. To make this conversion, a AmICA Light device analyses the payload information received from the Bus Radio AmICA device or from the Bus Radio AmICA UDP/IP device, through the IBusRadioAmICA interface. Using the same interface the AmICA Light device also sends the information to change the light status. The light information is turned available by the AmICA Simple Light device through the ISimpleLight interface, while the AmICA Dimmable Light device provides it through the IDimmableLight interface. The ISimpleLight interface is recommended to light sensor nodes that can just turn the light on or off, while the IDimmableLight interface allows regulation of the light brightness.

#### *4.4.2.5 AmICA Heater*

The AmICA Heater application converts the raw data from the AmICA heater sensor node into heater data and vice versa. The AmICA Heater device can also receives raw data from the Bus Radio AmICA device or from the Bus Radio AmICA UDP/IP device, through the IBusRadioAmICA interface. It is also capable of sending the configuration data to the Bus Radio AmICA device, as for example a message to change the heater potency. The Bus Radio AmICA application forward the message to the AmICA heater sensor node. The heater information is provided by the AmICA Heater device through the IHeater interface.

#### *4.4.2.6 AmICA Speaker*

The AmICA Speaker application is responsible for converting the raw data from the AmICA speaker sensor node into speaker data. In order to make this conversion, the AmICA Speaker device analyses the payload information received from the Bus Radio AmICA device through the IBusRadioAmICA interface. This device is also capable to send raw data to its sensor node in order to perform some configurations, as for example to change the speaker volume. Every AmICA Speaker device provide the speaker information through the ISpeaker interface.

#### *4.4.2.7 AmICA Pet Movement*

The AmICA Pet Movement converts the raw data received from the AmICA pet movement sensor node into pet movement data with the help of the IBusRadioAmICA interface. Every AmICA Pet Movement device implements the IPetMovement interface, through which the pet movement data is turn available.

#### *4.4.2.8 Sun SPOT Movement*

The Sun SPOT Movement application is responsible for converting the raw data from the Sun SPOT movement sensor node into movement data. To perform this conversion, the Sun SPOT Movement device analyses the payload information received from the Bus Radio Sun SPOT UDP/IP device through the IBusRadioSunSPOT interface. The movement information is provided by the Sun SPOT Movement device through the same IMovement interface found on the AmICA Movement application.

#### *4.4.2.9 Sun SPOT Temperature*

The Sun SPOT Temperature converts the raw data received from the Sun SPOT temperature sensor node into temperature data with the help of the IBusRadioSunSPOT interface. Every Sun SPOT Temperature device implements the ITemperature interface, through which the temperature data is turn available.

#### *4.4.2.10 Sun SPOT Light*

The Sun SPOT Light is responsible for converting the raw data from the Sun SPOT light sensor node into light data. Differently from the AmICA light sensor node, the Sun SPOT light sensor node is not equipped with a actuator, therefore the Sun SPOT light device is not capable to adjust the light brightness. Ie, it can only receive the raw data from the light sensor node. In order to receive this raw data, the device connects itself with the Bus Radio Sun SPOT UDP/IP device through the IBusRadioSunSPOT device. The Sun SPOT Light device provides the light information through the ILightBrightness interface.

#### *4.4.2.11 Sun SPOT Pet Movement*

The Sun SPOT Pet Movement converts the raw data received from the Sun SPOT pet movement sensor node into pet movement data with the help of the IBusRadioSunSPOT interface. Every device from this application implements the IPetMovement interface, through which the pet movement data is turn available. This interface is the same provided from the AmICA Pet Movement interface.

### 4.4.3 Intermediate Services

The Intermediate Services layer is composed by applications that provide services, which may be used by other applications. The devices from this layer can, optionally, make use of the services provided by the applications situated in the lower layers, Hardware Abstraction layer and Hardware Connection layer. In this prototype, 4 applications for the Intermediate Services layer were developed, the Flat Occupancy, the House Information and the Mail Manager.

### *4.4.3.1 Flat Occupancy*

The Flat Occupancy is an application that monitors the house occupation. It is capable to verify if there is someone at house or not. In order to perform this verification, a simple algorithm, which is presented on the Figure 4.4, was developed. This algorithm is capable to detect if there is someone at the flat, but it can not inform the exact number of people inside the house. It is interesting to observe that the goal of this work is not to develop the most efficient and precise algorithm to detect a flat occupation, the goal is to provide a smart environment platform that in addition to providing all the support necessary for an AAL system, also allows an easy integration of hardware and software from different vendors. Ie, this application was developed only with the goal to validate the platform and not to publish an innovative algorithm.

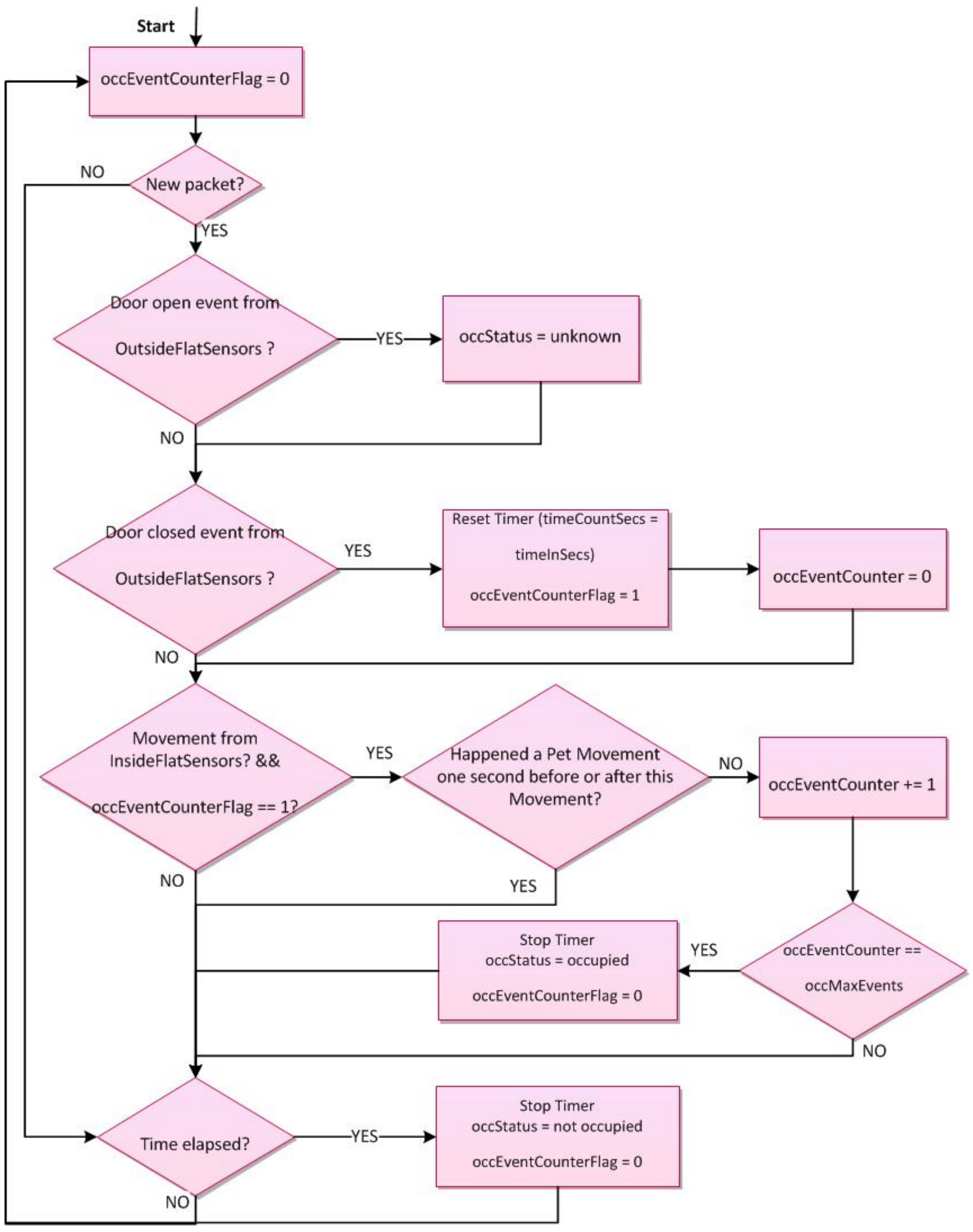

Start
occEventCounterFlag = 0
NO
New packet?
YES
Door open event from OutsideFlatSensors ?
YES
occStatus = unknown
NO
Door closed event from OutsideFlatSensors ?
YES
Reset Timer (timeCountSecs = timeInSecs)
occEventCounterFlag = 1
occEventCounter = 0
NO
Movement from InsideFlatSensors? && occEventCounterFlag == 1?
YES
Happened a Pet Movement one second before or after this Movement?
NO
occEventCounter += 1
NO
YES
Stop Timer
occStatus = occupied
occEventCounterFlag = 0
YES
occEventCounter == occMaxEvents
NO
Time elapsed?
YES
Stop Timer
occStatus = not occupied
occEventCounterFlag = 0
NO

Figure 4.4: The flat occupancy algorithm used by the Flat Occupancy device to verifies if there is at least someone at the house or not.

The Flat Occupancy driver creates only one device. This device implements the algorithm from the Figure 4.4. The Flat Occupancy device is interested on the IDoor, IMovement, IPetMovement and ITimeController interfaces, because the information provided by them is used on its algorithm. The information related to the flat occupancy status is turn available through the IFlatOccupancy interface.

#### *4.4.3.2 House Information*

The House information application is the solution for the problem previously presented in the section 3.6. The House information device manages all the necessary information from the house and turn it available to the other devices through the IHouseInformation interface. The house information device is created automatically when the driver is installed, because this device does not need any other device to perform its functionalities. Ie, this device was created only to allow its information to be used by the other devices.

#### *4.4.3.3 Mail Manager*

The Mail Manager application, when requested can send an email to a list of addresses. The email message and the subject are informed by the device that want to forward it. The Mail Manager device allows the other devices to make this request through the IMailManager interface. In the same way as the House Information application, the Mail Manager driver creates instantaneously a device, which may receive connection requests from other devices.

#### *4.4.3.4 Time Controller*

The Time Controller was developed specially to allow the realization of simulations. Some of the simulators that were used to perform the platform evaluation may take more time to make its graphic renderization. Therefore, while the TinySEP prototype runs in real time, the simulator may be running in another time. The solution to that problem was to develop a time synchronizer that will communicate with the TinySEP through the Time Controller application. Another advantage of this approach is that it is possible to simulate all the events that happened in a house for a whole day in a much shorter time, increasing the number of simulations performed.

The TinySEP prototype does not need necessary to use the Time Controller application, ie, if this application was not installed, the platform continues to operate using the real time. Once the Time Controller is installed, the user can still choose to use it or not and these changes may be made at run time.

Once Time Controller driver is installed, it automatically creates a device. This device is responsible for making the whole time control. The time information is turned available through the ITimeController interface, which is provided by the Time Controller device.

### 4.4.4 Ambient Assisted Living Services

The Ambient Assisted Living Services layer is composed by applications that might be used by other devices. The main difference between the applications from this layer to those from the Intermediate Services layer is that the ones from the AAL Services layer can make use of the services provided by the applications situated on the Intermediate Services layer. In fact the applications from this layer can make use of any service provided by any of the four layers.

In this prototype 2 applications for the AAL Services layer were developed, the Inactivity Recognition and the Follow Me Light. Both applications make use of the services provided by the Intermediate Services and by the Hardware Abstraction layers.

#### *4.4.4.1 Inactivity Recognition*

The Inactivity Recognition application detects if the person suffers an accident that makes him unable to ask for help. If this situation is detected the application automatically calls for help. The main idea of its algorithm, which is illustrated by the Figure 4.5, is that if the person stops to make activities, as for example, movement or open a door, for a specific time, then a inactivity was detected. Once again the idea of this application is not to validate a new and precise algorithm but to evaluate this platform and its applicability.

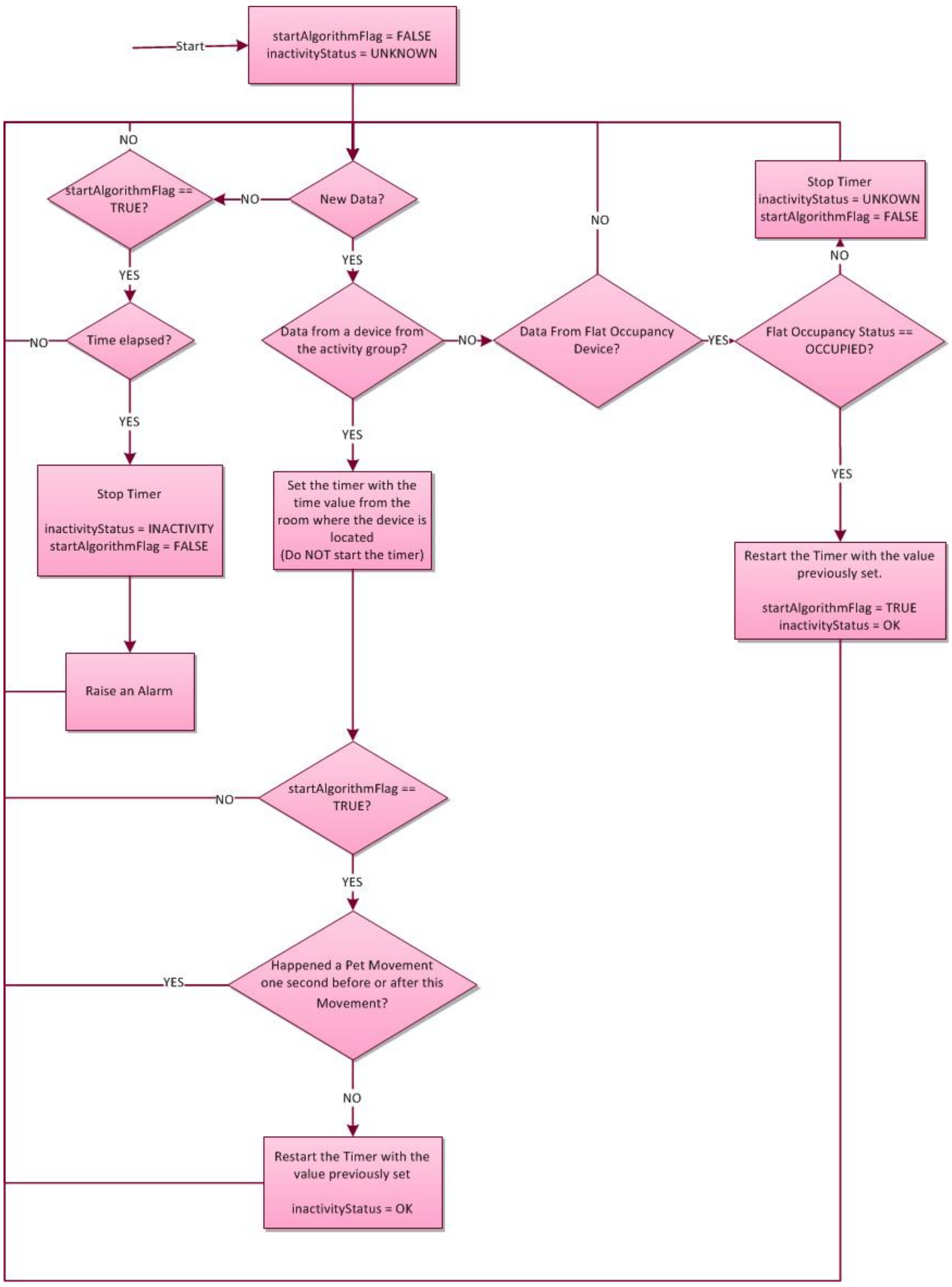

Start
startAlgorithmFlag = FALSE
inactivityStatus = UNKNOWN
New Data?
NO
startAlgorithmFlag == TRUE?
NO
YES
Time elapsed?
NO
YES
Stop Timer
inactivityStatus = INACTIVITY
startAlgorithmFlag = FALSE
Raise an Alarm
YES
Data from a device from the activity group?
NO
Data From Flat Occupancy Device?
NO
YES
Flat Occupancy Status == OCCUPIED?
NO
Stop Timer
inactivityStatus = UNKOWN
startAlgorithmFlag = FALSE
YES
Restart the Timer with the value previously set.
startAlgorithmFlag = TRUE
inactivityStatus = OK
YES
Set the timer with the time value from the room where the device is located
(Do NOT start the timer)
startAlgorithmFlag == TRUE?
NO
YES
Happened a Pet Movement one second before or after this Movement?
YES
NO
Restart the Timer with the value previously set
inactivityStatus = OK

Figure 4.5: The algorithm used by the Inactivity Recognition device to detects if the person suffers an accident and need help.

This version of the Inactivity Recognition algorithm (see Figure 4.5) only works for one person, because if there is another one at home and this person continues to make activities, then the algorithm will continue to work without detecting that the other one had an accident and needs help. Another problem from this algorithm is that the time used for each room must be informed by the user and this algorithm does not learn with previous examples, so if the times are not good enough, then the user must set them manually.

The Inactivity Recognition application receives the activities information through the IMovement, IDoor and IWindow interfaces. The IFlatOccupancy interface is used in order to verify if there is at least one person at home, because it does not make any sense to verify an inactivity recognition if there is no one at the house. The IMailManager interface also play an important role, because it is used to notify the occurrence of an inactivity. Another important interface for this algorithm is the IHouseInformation, that provides all the environment information. If the ITimeController interface is available, then this time provided by this interface will be used, otherwise the real time is going to be used on the timer.

#### *4.4.4.2 Follow Me Light*

The Follow Me Light application controls the house whole illumination. As the person moves inside the house, the lights follow this person movements, for example, if the person is at the living room and he walks to the kitchen, then the living room lights are turned off and the kitchen light are turned on. This application only works if there is at least one person inside the house, because otherwise it would not make any sense to control the house illumination if there is nobody on it.

If there is more then one person at home, this application will not turn off the lights from where one of the users is, only because the other one is moving. It will wait for some seconds to verify if there is someone else at one room and then turn the light off, if there is nobody there, or maintain it on, if there is someone there.

This application has one peculiarity. If in this house there is a pet and if this pet moves around the house, then the house lights will also follow this pet movements. However, this will only happen while there is someone else inside the house, because if the flat is occupied only by the pet, then the flat occupancy status will be not occupied and therefore this application will not work.

The Follow Me Light driver creates only one device, which makes the control from the house illumination. The Flat Occupancy device is interested on the ISimpleLight, IDimmableLight, IDoor, IMovement, IFlatOccupancy and ITimeController interfaces, because the information provided by them is used on the light control. The Follow Me Light device does not provide any interface, because the service provided by it can not be used by other devices and it can not give any special information.

# 5 PLATFORM EVALUATION

In order to demonstrate the applicability of the prototype developed in this work, several tests were made. Some of those tests were performed in real environments, others were made only with the help of simulators and the others were performed with the help of simulator and real sensor nodes.

The first tests, discussed in the section 5.1, were performed with the aim to make a performance analysis. Basically, measurements were made to verify the use of resources by the prototype in different scenarios. The sensor nodes that were chosen for this prototype were analysed in the section 5.2, specially, according to the signal range of each of them. Some tests were also performed in real environments. In fact 3 flats were used to evaluate the usage of the prototype and the advantages and disadvantages of the usage of this prototype will be presented in the section 5.3. The end tests, discussed in the section 5.4, were performed with the help of 3 different simulators. Those simulators allowed the creation of more complex tests, which were capable to analyse each application provided by the platform and their internal communication.

## 5.1 Performance Analysis

The TinySEP prototype was designed to run on personal computers or even on notebooks, in which the user can continue to use all the features from his machine while the AAL system runs. Some tests with the goal to simulate some expected scenarios for this platform were performed in order to analyse the platform resources usage. In total 5 different scenarios were developed. Each of them seeks to include all hardware and software resources defined in previous chapters, ie, all the applications, which were presented in the section 4.4, were used. In those tests the information provided by the AmICA sensor nodes an by the Sun SPOT sensor nodes was used, once one of the goals from this platform is to provide an easy integration of hardware and software components from different suppliers. However due to the fact that some of those scenarios requires more sensor nodes then the ones available, some sensor nodes were simulated with the help of the simulators that are going to be discussed in the section 5.4.

In each one of those developed scenarios the number of installed devices were increased. The Scenario 1 is composed by 10 devices, while the Scenario 2 has 20 installed devices. In fact the Scenario 2 is composed by the same 10 installed devices from the Scenario 1 and more 10 other devices. Using the same logic, the Scenario 3 has 30 devices and the Scenario 4 and 5 have, respectively, 40 and 50 devices. However, only the number of devices found in each scenario does not give much information about their complexity. In order to have a better visualization about those scenarios and their complexity, the total number of packets received by the platform in 30 minutes, 1 hour, 1 hour and 30 minutes and 2 hours were analysed. The obtained results are presented in the Table 5.1.

Table 5.1: Total number of received packets by the prototype in each one of the 5 scenarios.

| Scenario | 30 Minutes | 1 Hour | 1 Hour and 30 Minutes | 2 Hours |
|---|---|---|---|---|
| **1** | 1.077 | 2.155 | 3.233 | 4.313 |
| **2** | 3.768 | 7.545 | 11.325 | 15.102 |
| **3** | 13.292 | 26.581 | 39.900 | 53.217 |
| **4** | 18.351 | 36.704 | 55.052 | 73.406 |
| **5** | 24.110 | 48.228 | 72.344 | 96.461 |

Once the scenarios were already defined, it was possible to make some performance evaluations, in which the CPU and memory usage were analysed. All the tests were performed in a Sony VAIO notebook, model VPCEB4X1E, with the processor Intel® Core$^{TM}$ i5-480M, with the processor speed 2.66 GHz with Turbo Boost up to 2.93 GHz and with a memory size from 4 Gb. Each test was redone 4 times and the results that are going to be presented are originating from the average values obtained in each execution. In order to maintain the result most precise as possible, after the conclusion of each test, the notebook was restarted and the devices installation was always performed in the same order. Each execution lasted one hour and the values were recorded every 10 minutes. The results of this performance test are summarized in the Table 5.2 and illustrated by graphics form the Figures 5.1 and 5.2.

Table 5.2: The average of the CPU and memory usage values obtained in each scenario.

| Scenario | CPU (%) | Memory Usage (K) |
|---|---|---|
| **1** | 0,50% | 21276 |
| **2** | 1,25% | 21884 |
| **3** | 1,50% | 22608 |
| **4** | 1,50% | 23140 |
| **5** | 1,50% | 23674 |

Through the results illustrated by the graphic from Figure 5.1, it is possible to observe that increasing the number of devices, the rate of CPU usage did not change much. In fact this rate was the same obtained for the Scenarios 3, 4 and 5. This test has also shown that given a machine with the same configurations used in this test, it is completely feasible to maintain the prototype running along with other applications.

The graphic from the Figure 5.2 allows the achievement of similar conclusion to the ones found with the previous graphic. Once again it is possible to note that increasing the device number did not change significantly the memory usage. In fact, with the addition of 10 devices, the memory usage value was increased by approximately 600K. Once again the obtained results have shown this prototype feasibility. However those results are not enough, since this platform was developed to operate 24 hours a day, 7 days per week. For this reason it is necessary to check whether the platform is able to run for a long

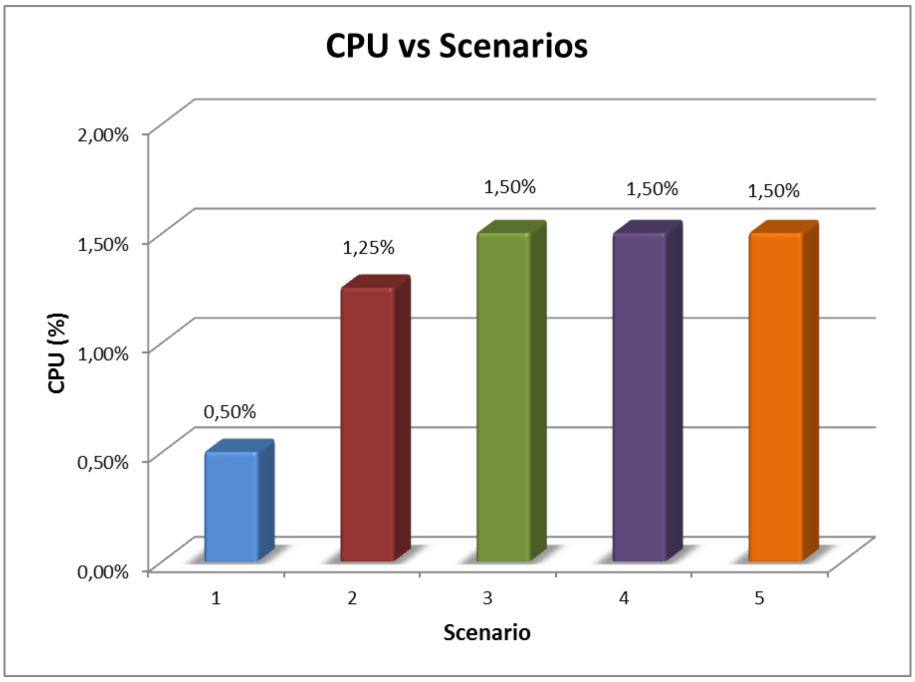


Figure 5.1: Graphical representation of the CPU average rate used to perform the tests in each scenario.

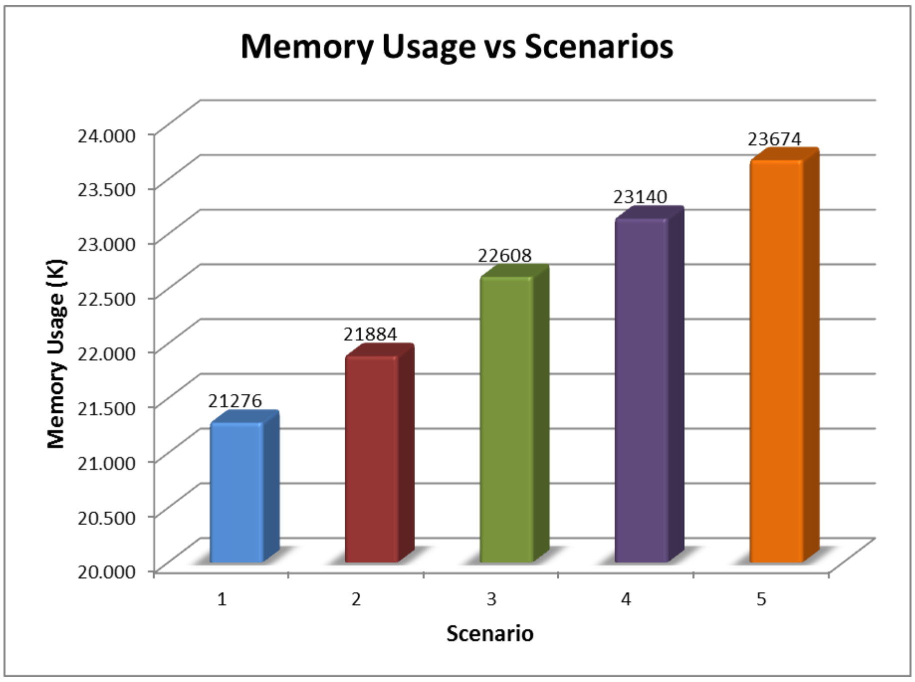


Figure 5.2: Graphical representation of the memory usage average value used to perform the tests in each scenario.

time. This test can also be used to verify if there is any memory leak in the prototype implementation. This test was executed using the Scenario 5 for 72 hours and the CPU and memory usage values were recorded every 6 hours. The obtained results are presented in the Table 5.3 and Figures 5.3 and 5.4.

It is interesting to observe that the battery from the Sun SPOT sensor nodes did not resist to the test. In fact the battery that was completely charged ate the beginning of the

Table 5.3: Results obtained with the execution of the Scenario 5 for 72 hours.

| Runtime (in hours) | CPU (%) | Memory Usage (K) |
|---|---|---|
| 0 | 1,50% | 23.680 |
| 6 | 1,50% | 23.676 |
| 12 | 2% | 23.796 |
| 18 | 1,50% | 23.536 |
| 24 | 1,50% | 23.572 |
| 30 | 2% | 23.616 |
| 36 | 2% | 23.828 |
| 42 | 2% | 23.944 |
| 48 | 1,50% | 23.624 |
| 54 | 1% | 23.632 |
| 60 | 2% | 23.652 |
| 66 | 1,50% | 23.596 |
| 72 | 1,50% | 23.672 |

test, last approximately a day. It was not possible to determine the exact time that the battery lasted, because this was only noticed after the first 24 hours. However even with the Sun SPOT sensor nodes not sending any packet data to the platform, the devices that were responsible to abstracts this sensor nodes have continued to be installed. Ie, they were not removed from the application, because if the sensor node were recharged and turned on, the application would return to receive the information provided by the sensor and any other configuration would be necessary.

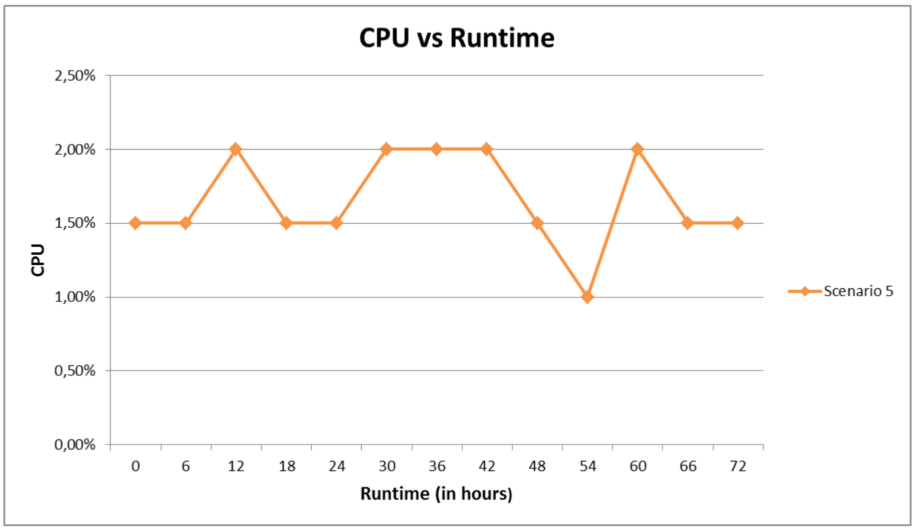


Figure 5.3: Graphical representation of the CPU rate obtained with the execution of the Scenario 5.

Analysing the results from this test it is possible to observe that no memory leak was found. Both CPU and memory usage values oscillated during the tests. The reason for this oscillation is that in the moment that those measurements were made the platform probably was executing a step on the algorithm that might has required more resources.

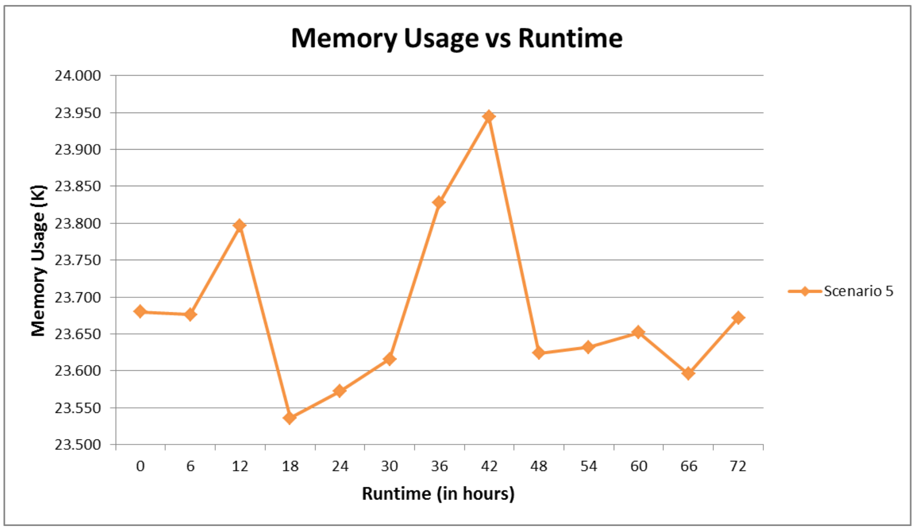


Figure 5.4: Graphical representation of the memory usage obtained with the execution of the Scenario 5.

## 5.2 Sensor Nodes Signal Range Analysis

As was presented in section 3.9, this prototype was developed to support initially the Sun SPOT sensor nodes and the AmICA sensor nodes. Since those sensor nodes are going to be used in real environments, it is necessary to verify their signal range. This test was divided in 2 parts. In the first of them, which will be assessed in the section 5.2.1, the signal range from the AmICA sensor nodes were analysed. In the second test, presented in the section 5.2.2, the same tests were performed, but this time using the Sun SPOT sensor nodes.

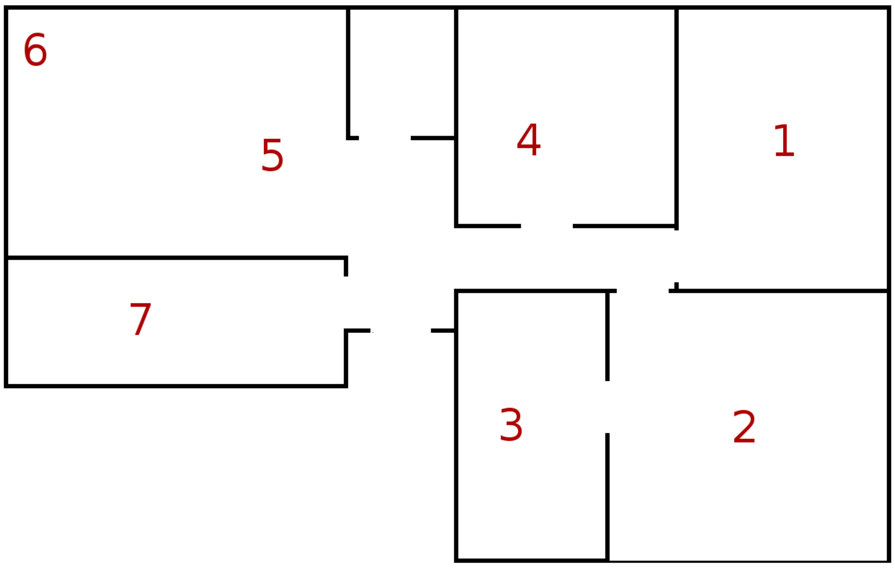


Figure 5.5: Illustration of the flat that was used to perform the signal range analysis.

Those tests were performed in a real house, whose house plan is depicted in the Figure 5.5. To each one of the numbered points from 1 to 7 the base station, which received the sensor packet data through the radio, was situated. Once the base station position was defined, the verification process started. This test consisted in verify if the packets sent by the movement sensor were received by the base station. Thus the movement sensor node was positioned in each one of the 7 points that were indicated in the Figure 5.5 and during 5 minutes the total number of packets that were received by the base station were verified. Since the sensor node that was being used was a movement one, in 5 minutes 150 movement packets should be received, once the movement sensor nodes send a movement packet every 2 seconds. This test was redone 4 time to each position and the results of the average from the obtained values will be presented in the sections 5.2.1 and 5.2.2.

### 5.2.1 AmICA Sensor Nodes

The Table 5.4 summarizes the results that were obtained with the AmICA sensor nodes, in which 100% is equivalent to 150 packets received. Those results have shown that it is possible to use the AmICA sensor nodes in an environment without having much troublesome with the sensor node signal range. However, those results has also shown that it is very important to chose carefully the sensor node position, because this choice directly affects the number of packets lost. This was exactly the problem that originated approximately 40% of packets loss in the position 6. If several sensors used in this house are found at the same conditions as the one from the position 6, then the functioning of the prototype and of all applications on it installed will be compromised.

Table 5.4: The results obtained with the analysis of the signal range from the AmICA sensor nodes.

| | AmICA Movement Sensor Node Position | | | | | | |
|---|---|---|---|---|---|---|---|
| **Basestation position** | **1 (%)** | **2 (%)** | **3 (%)** | **4 (%)** | **5 (%)** | **6 (%)** | **7 (%)** |
| **1** | 100 | 100 | 100 | 100 | 95 | 57 | 99 |
| **2** | 100 | 100 | 100 | 100 | 99 | 61 | 93 |
| **3** | 100 | 100 | 100 | 100 | 97 | 58 | 98 |
| **4** | 100 | 100 | 100 | 100 | 100 | 83 | 100 |
| **5** | 98 | 97 | 97 | 100 | 100 | 100 | 100 |
| **6** | 64 | 58 | 61 | 83 | 100 | 100 | 100 |
| **7** | 100 | 92 | 99 | 100 | 100 | 99 | 100 |

### 5.2.2 Sun SPOT Sensor Nodes

The results obtained with the Sun SPOT sensor nodes, which are presented in the Table 5.5, were worse than expected. It was expected that their values would be close to those obtained with the AmICA sensor nodes, however they were much lower. The tests have shown that the Sun SPOT signal range is much shorter then the one found in the AmICA sensor nodes. Therefore the packet loss value has increased and in some situations this lost was so high that the base station practically stopped to receive the sensory data. With those results, it is possible to note that the Sun SPOT sensor nodes are not suitable to cases in which the sensors are at a long distance from the base station. For the same reason, it is possible to conclude that they are not suitable to be used as the only hardware in a house, even because of the resources that it offers and the battery durability.

Table 5.5: The results obtained with the analysis of the signal range from the Sun SPOT sensor nodes.

| | **Sun SPOT Movement Sensor Node Position** | | | | | | |
|---|---|---|---|---|---|---|---|
| **Basestation position** | **1 (%)** | **2 (%)** | **3 (%)** | **4 (%)** | **5 (%)** | **6 (%)** | **7 (%)** |
| **1** | 100 | 100 | 79 | 98 | 62 | 2 | 43 |
| **2** | 100 | 100 | 100 | 97 | 42 | 5 | 31 |
| **3** | 77 | 100 | 100 | 47 | 25 | 0 | 32 |
| **4** | 97 | 99 | 51 | 100 | 100 | 15 | 99 |
| **5** | 55 | 43 | 20 | 100 | 100 | 100 | 100 |
| **6** | 8 | 3 | 0 | 17 | 100 | 100 | 79 |
| **7** | 43 | 25 | 28 | 100 | 100 | 71 | 100 |

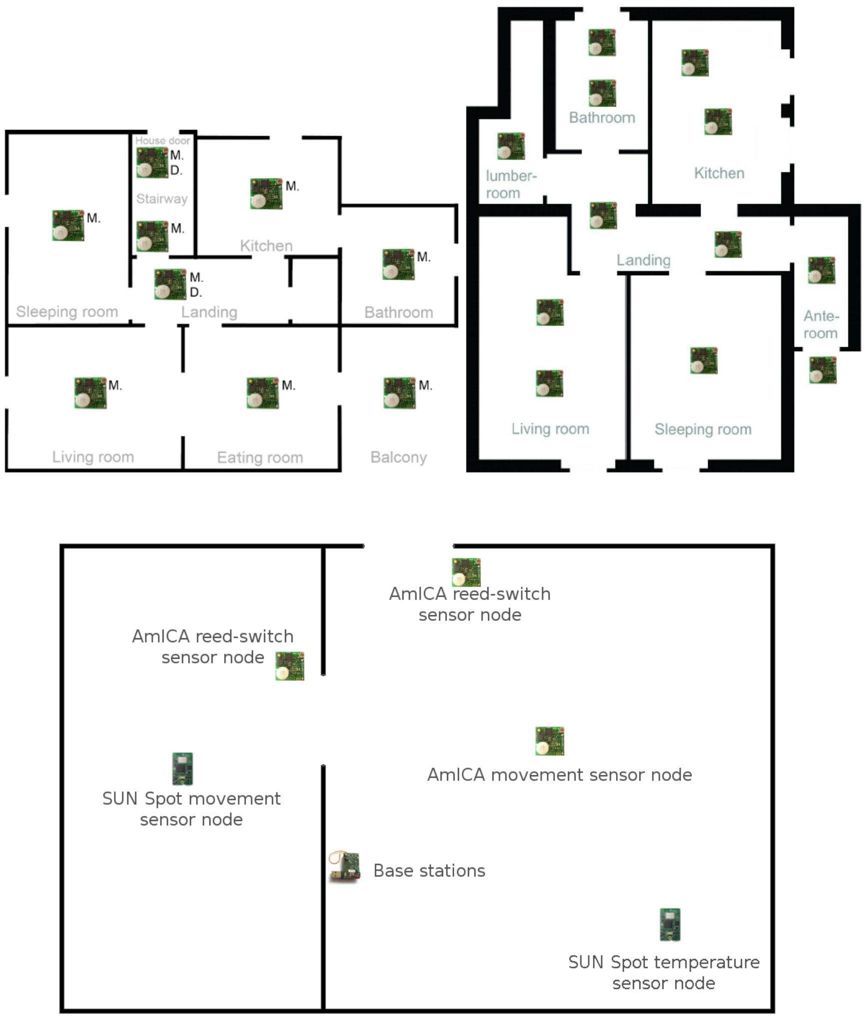


Figure 5.6: Illustration of the three inhabited flats used to perform the real-world evaluations; flat A (left), flat B (right) and flat C (bottom).

## 5.3 TinySEP in the Real-world

In order to evaluate the platform in the real-world, 2 inhabited flats were equipped with the AmICA sensor nodes and a third flat equipped with the AmICA sensor node and the Sun SPOT sensor nodes. This three flats are illustrated by the figure 5.6.

The flat A is equipped with 7 AmICA movement sensor nodes and 2 AmICA movement and reed-switch sensor nodes. This flat is inhabited by a 88 years old person. The flat B is equipped with 12 AmICA movement sensor nodes and it is inhabited by a 28 years old single person. The flat C is equipped with 1 Sun SPOT temperature sensor node, 1 Sun SPOT movement sensor node, 2 AmICA reed-switch sensor nodes and 1 AmICA movement sensor node. This flat is inhabited by a 23 years old single person.

The results obtained with the flat C has shown that it is really possible to accomplish the integration of these two hardware in the platform, whose applications take benefit of the data provided by both. However they have also shown some disadvantages on the use of this prototype and the hardware that it compounds. Those disadvantages are found specially with the usage of the Sun SPOT sensor nodes. The battery durability have proved to be a big problem, because their battery must be recharged every day. Related with the Inactivity Recognition application, the configuration of the ideal times used by the algorithm were difficult to determine, specially because at night when the person is sleeping, he does not perform any movements for a long and therefore the algorithm detected an inactivity when in the fact the person was just sleeping. Those results have also shown the necessity of an algorithm that would be capable to automatically adapt the its times according to the user and the time of the day.

## 5.4 System Simulations

The results from the tests performed so far were not enough, since it was not possible to make a precise evaluation of the platform applications, specially the ones found in the Intermediate Services layer and in the Ambient Assisted Living layer. Moreover, the tests made so far did not allow the realization of tests from adverse situations, because they were really inhabited. To overcome this problem, 3 different simulation tools were used. With the help of those simulators, it was possible to verify precisely the behaviour of each prototype application.

### 5.4.1 Simple UDP/IP Simulator

The Simple UDP/IP simulator was developed in C# with the goal to analyse the precise behaviour of each prototype application to certain sensor inputs. This simulator is illustrated in the Figure 5.7. It simulates the flat A, whose house map was previously presented in the Figure 5.6, but for this simulation the flat is equipped with more sensor nodes then the ones found on the real flat. In total this simulator is composed by 20 AmICA sensor nodes and 3 Sun SPOT sensor nodes, which may or not be activated during the test. At same time that this simulator sends the sensory data to TinySEP, it also receives information from the prototype, such as a packet data informing the light actuator that the light should be turned off. All the received data can be analysed through the log box found on the bottom of the Figure 5.7.

The first results obtained with this simulations have shown that the packets received by the applications from the Hardware Connection layer were forwarded to the right devices, ie, the devices that were responsible to abstracts the sensor node functionalities.

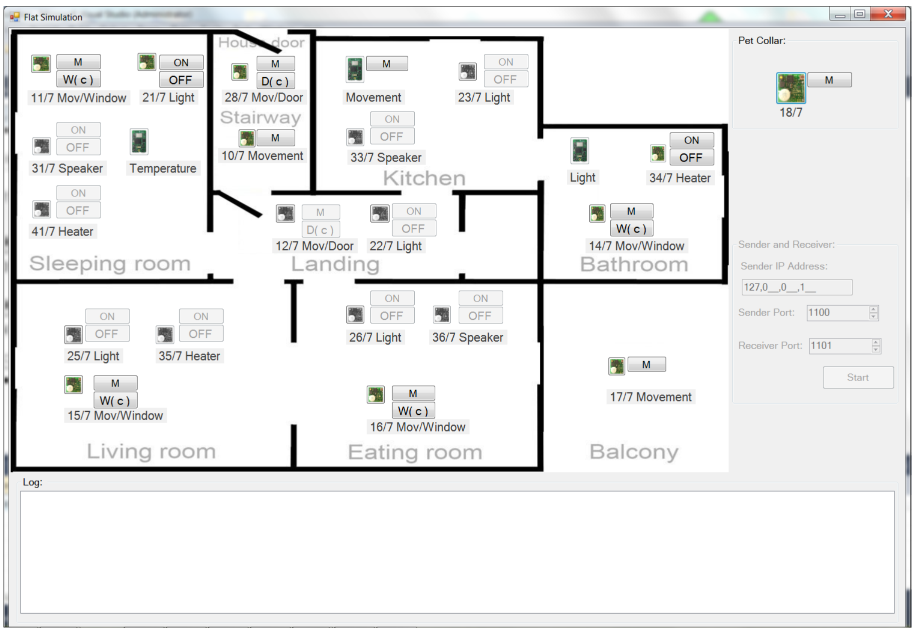
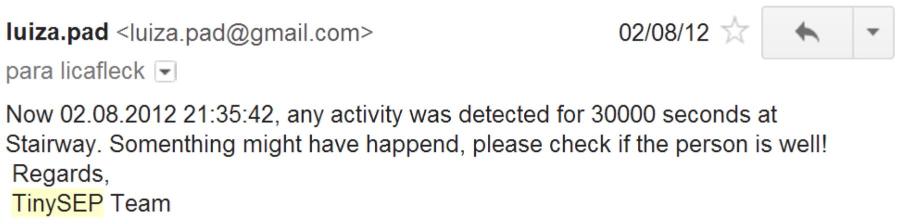


Figure 5.7: The Simple UDP/IP simulator.



**luiza.pad** <luiza.pad@gmail.com> 02/08/12

para licafleck

Now 02.08.2012 21:35:42, any activity was detected for 30000 seconds at Stairway. Somenthing might have happend, please check if the person is well!
Regards,
TinySEP Team

Figure 5.8: Example of a mail sent to inform that an inactivity was detected.

Those devices from the Hardware Abstraction layer were also performing the correct abstraction from the sensor nodes raw data. Once it was guaranteed the correctness from the devices found on both layers, it was possible to make a basic analysis of the Flat Occupancy, Inactivity Recognition and Follow me Light applications. The tests started with the Flat Occupation algorithm, because the other applications depend of the results of the Flat Occupancy application. It was possible to conclude that the Flat Occupancy algorithm was responding as the expected. Moreover, it was possible to observe that when its status has changed, this event was being forwarded to the right devices. For the Inactivity Recognition application it was possible to obtain similar results, ie, the timer used in the algorithm were being setted to the right value, that was stipulated according to the room, in which the user was performing the activities. Once an inactivity was detected the Mail Manager application was used to inform this event, as the Figure 5.8 illustrates. However this simulator was not adequate to evaluate the Follow Me Light application, because it does not allow the straight visualization of the situation being performed. Ie, in order to discover which light was being turned on and which was being turned off, it was necessary to analyse the packets displayed in the log provided by the simulator window. This analyses were not natural, in fact it was confuse and consumes a lot of time. Another disadvantage of this simulator is that the sensory information was only sent when the tester clicked on a button, ie, the information were not being sent in the times that were defined on the sensor nodes specification. So it can not be used to simulate real scenarios.

### 5.4.2 MCA2/SimVis3D

The MCA2 and SimVis3D platforms, first presented in the section 5.4.1, were developed by the Robotics Research Lab from the University of Kaiserslautern, Germany. The idea of this simulation is that the robotics frameworks MCA2 and SimVis3D simulate a smart environment with one simulated person, one pet and several AmICA sensor nodes placed in the flat, illustrated by the Figure 5.9.

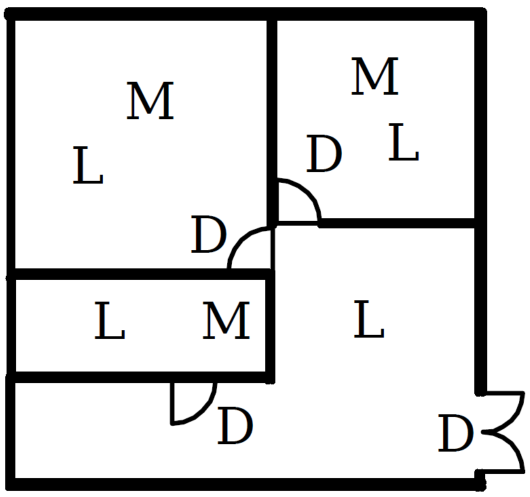


Figure 5.9: Layout plan of the flat with sensor and actuator positions. “M” indicates a movement sensor, “D” a door sensor (reed switch) and “L” a light.

This flat is equipped with 8 AmICA sensor nodes. Four of them were composed by movement sensors and reed-switch sensors (used as doors) and four of them were AmICA light sensor nodes. The simulator sensor nodes sends informations to TinySEP, which receives them through the Bus Radio AmICA UDP/IP application. The communication established between them is bidirectional, because at the same time that TinySEP receives information from the robotic frameworks it can also send data to them.

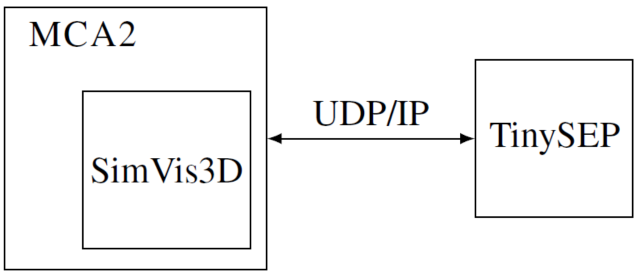


Figure 5.10: High-Level interconnection diagram of the robotic frameworks (MCA2 and SimVis3D) and TinySEP.

During the simulations, both frameworks where running on two different computers, one with the robotic frameworks and another with the TinySEP. The overall situation of the interconnected frameworks is shown in Figure 5.10. Through this image it is possible to note that SimVis3D resides as a module inside of the MCA2 framework, therefore there is no explicit connections drawn in the image.

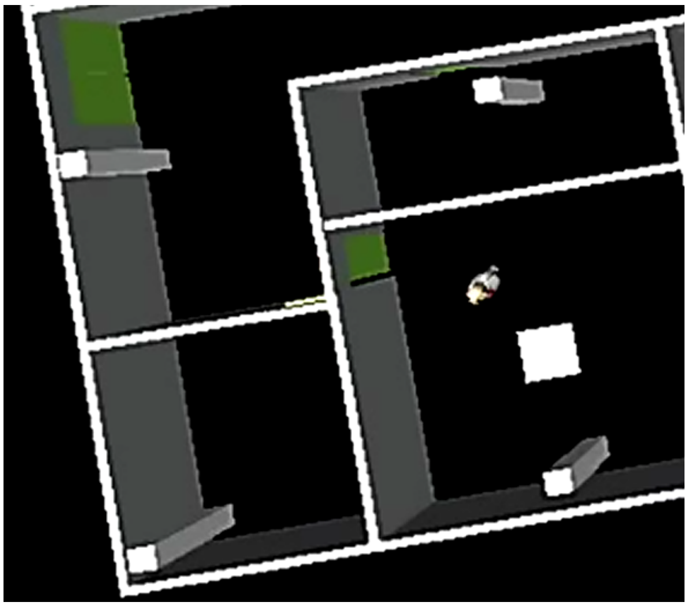


Figure 5.11: The flat equipped with the TinySEP framework as seen in the SimVis3D visualization.

In the first moment, only one simulated person was used. With it, differently from the one obtained with the previous simulator, it was possible to visually analyse the behaviour of the Flat Occupancy, Inactivity Recognition and Follow Me Light applications. The first two applications were already validated by the previous simulator, however with this one it was not only possible to visualize the simulated scenario, but this time the sensory data was sent in the time stipulated by their specification. The behaviour provided by the Follow Me Light algorithm was also validated. The Figure 5.11 illustrates the visualization provided by the robotic frameworks.

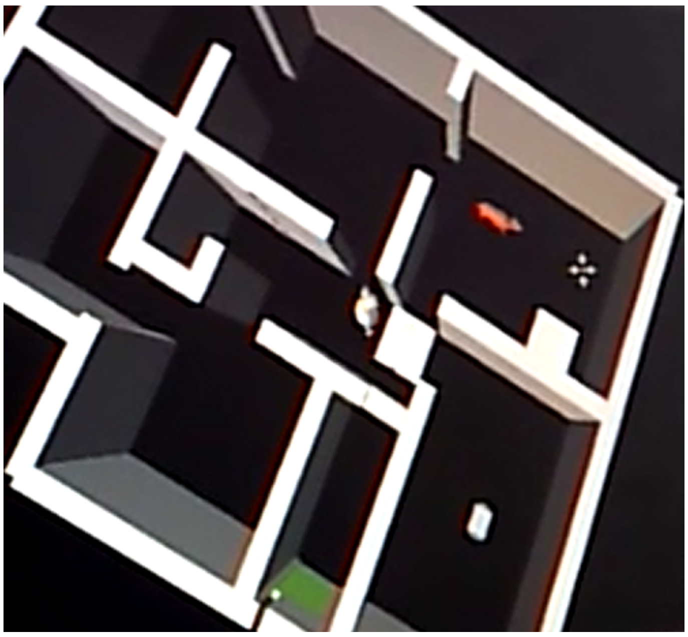


Figure 5.12: Example scenario including a pet and a person provided by the robotic frameworks.

In a second moment, a pet was included in the simulations, as illustrated by the Figure 5.12. This pet was equipped with a AmICA pet collar sensor node and therefore it was possible to analyse the influence of the pet on the algorithms. The results have shown that the Follow Me Light responds negatively to the inclusion of the pet. The problem provided by the pet is that the lights are also controlled according to the pet movements. This was also stipulated by the algorithm but the visualization of the scenarios has proved not to be the best solution to be adopted.

As a result of the combination of those frameworks a paper was published in the Proceedings of the Ubirobots 2012 Workshop conducted at the 14th International Conference on Ubiquitous Computing. This paper was also selected to submit to a special issue of the Robotics and Autonomous Systems journal.

### 5.4.3 Siafu

The Siafu, as presented in chapter 2, is an open source context simulator that allow the simulation of environments that range from an office up to a huge city. For the simulations from the prototype the Siafu TestLand simulator, which provides an office as an environment, was used. The flat used is illustrated by the Figure 5.13. This flat was equipped with several simulated AmICA sensor nodes. This simulator sensor nodes sends information to TinySEP, which receives them through the Bus Radio AmICA UDP/IP application. The TinySEP can also send configuration data to the simulator.

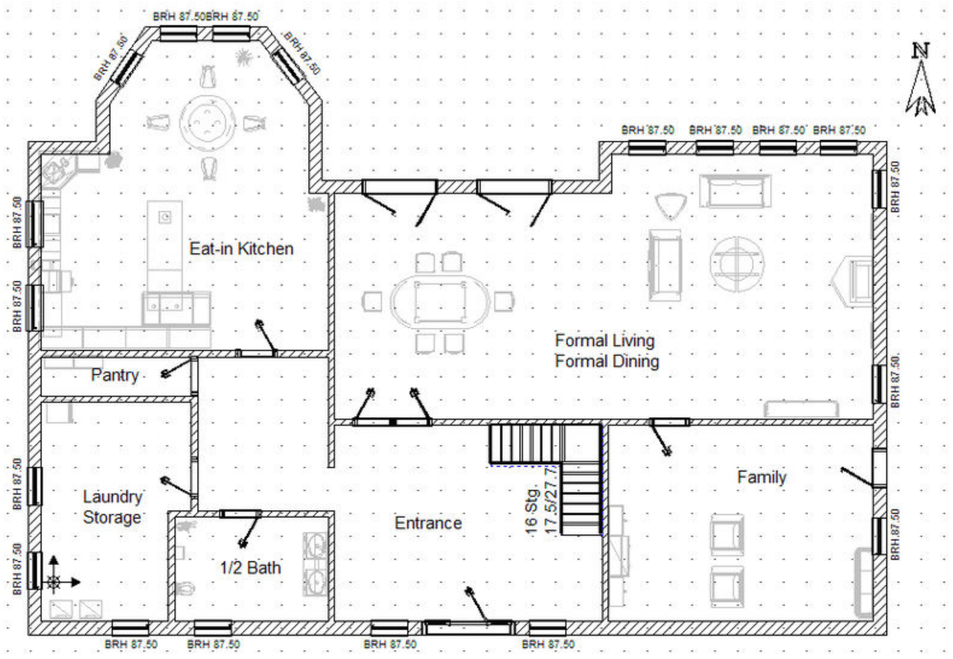


Figure 5.13: Illustration of the flat used in the simulations with the Siafu simulator.

The Siafu simulator does not run in real time, therefore it was necessary to develop the Time Synchronizer application. The idea behind the Time Synchronizer is that both applications must control their time with the time provided by this synchronizer (see Figure 5.14). Initially the Time Synchronizer informs both applications about the current time, when they receive this information they make all the processing they need. Once this processing is finalised, they send a message to the Time Synchronizer informing that they finished doing all the processing required and can advance to the next second. Once the Time Synchronizer receives the confirmation message from both applications, it advances to the next second and sends a message to the applications to inform that the time has advanced. Once Siafu receives this message informing that it can forward to the next second, it advance one step in its interaction. By doing this next step, the simulated user, consequently, moves in the scenario and at the same time the simulated sensor nodes detect this motion, they send a packet data with this information to TinySEP. By finishing this interaction step, Siafu sends a new confirmation message to the Time Synchronizer in order to advance to the next step. TinySEP receives the time information through the Time Controller application. By receiving this information, TinySEP performs all the processing required and then in sends a message to Time Synchronizer requesting that the application forwards to the next second. With the usage of the Time Synchronizer it was possible to guarantee that the application were running using the same time.

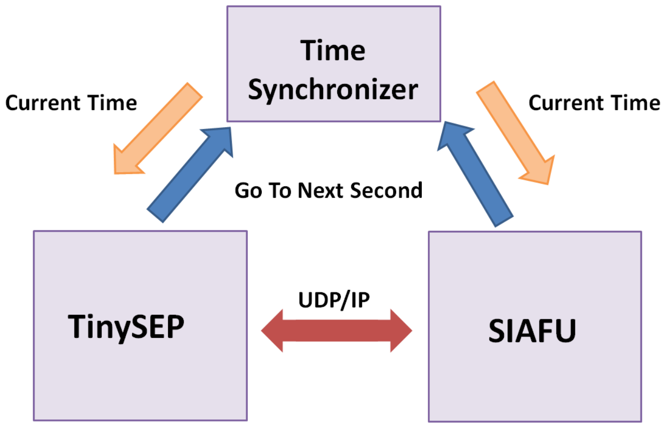
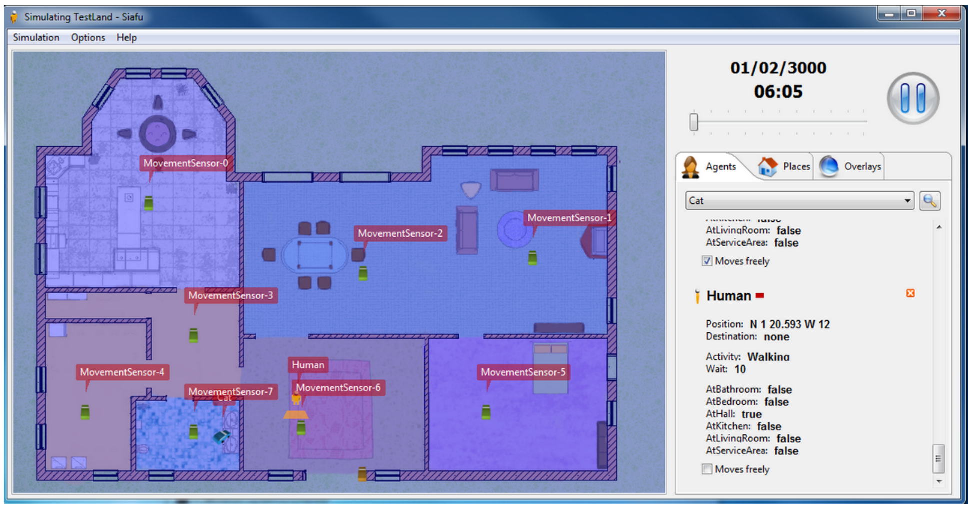


Figure 5.14: Relational structure from TinySEP, Siafu and the Time Synchronizer.



01/02/3000
06:05

Cat

AtLivingRoom: false
AtServiceArea: false
[✓] Moves freely

Human

Position: N 1 20.593 W 12
Destination: none

Activity: Walking
Wait: 10

AtBathroom: false
AtBedroom: false
AtHall: true
AtKitchen: false
AtLivingRoom: false
AtServiceArea: false
☐ Moves freely

Figure 5.15: Example scenario of the usage of Siafu to perform evaluations from the prototype.

The main advantage of the usage of the Siafu simulator comparing with simulators previously presented is that it allows the usage of more then one simulated person. This simulator also allows the usage of many pets and the usage of different houses, once it is not fixed to a specific environment. The results obtained with the usage of more users has proved the problem on the detection of an inactivity, once the second user continue to perform activities and therefore the inactivity is not detected. This simulator is perfect to continue to analyse the improvements that can be done on this algorithm.

It is important to clarify that all the simulation performed with the Siafu were developed together with the computer science student from the Federal University of Rio Grande do Sul Vitor Fortes Rey. Therefore this work to enable the simulation of smart environments that use the TinySEP framework is not only the result of my work.

# 6 CONCLUSION

This work presented the design and implementation of a Tiny Smart Environment Platform (TinySEP) for Ambient Assisted Living. Relevant concepts, requirements, related works and simulation tools that may be used to validate complex scenarios were presented. The model and the implementation of the TinySEP were discussed and the tests used to evaluate the platform were shown.

Developed societies are getting older at an unprecedented rate. This ageing leads to new challenges for the provision of healthcare and elderly care. In response of this need, many research and development have been made on Ambient Assisted Living (AAL). The solutions already provided are not wide spread, specially because of the lack of a standard for an AAL software platform.

The platform presented in this work combines the high usability of the proprietary, monolithic systems and the high re-usability of encapsulated components of the platform based systems. TinySEP takes advantage of the driver concept model and signal slot model. In one easy way TinySEP connects all the single elements to one AAL system, which can be customized according to the needs of its different users, without having to do any reprogramming. Its prototype was developed using the IDE Microsoft Visual Studio 2010 and the programming language C#. Among with the wide range of sensor nodes already available, the AmICA sensor nodes and the Sun Spot sensor nodes were chosen as the initially supported hardware. In total 20 different applications were developed and each of them was discussed.

In order to evaluate the applicability and the performance of the prototype, several tests were conducted. Those tests ranged from the analysis of the signal range from the sensor nodes to the simulation of complex scenarios performed by 3 different simulation tools. The results provided by those tests have proved that the platform is capable of providing all the features expected by an AAL system. However the results have also shown that the algorithm used to perform the Inactivity Recognition should be improved in order to allow an automatically adaptation of its parameters.

Due to the ease of integration of different hardware and software components, TinySEP is already being used by other international research groups as a way of validating their projects, such as a kind of sensor nodes. Furthermore, this work has already been published. The first publication was in the proceedings of the AAL-Kongress, that happened in Berlin, Germany in 2012 (Wille et al. 2012). The second publication was in the proceedings of the Ubirobots 2012 Workshop conducted at the 14th International Conference on Ubiquitous Computing, that happened in Pittsburgh, USA in 2012 (Wille et al. 2012). This paper was also selected to submit to a special issue of the Robotics and Autonomous Systems journal.

Even with the results already achieved there is still much work to be done. New applications are being developed, specially the ones more focused in the healthcare and social integration. The platform is being integrated with new hardware components. Other projects are also being made in the integration of mobile devices. A web visualizer is being developed in order to allow a remote visualization. Much research is being made in order to create new scenarios to validate the platform and all its new applications.

# REFERENCES

[Aboelaze e Aloul 2005]ABOELAZE, M.; ALOUL, F. Current and future trends in sensor networks: A survey. In: *Proceedings of the Second IFIP International Conference on Wireless and Optical Communications Networks*. IEEE Computer Society, 2005. (WOCN 2005), p. 551–555. ISBN 0-7803-9019-9. Available from Internet: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=1436087>.

[Akyildiz e Kasimoglu 2004]AKYILDIZ, I. F.; KASIMOGLU, I. H. Wireless sensor and actor networks: Research challenges. *Elsevier Ad Hoc Network Journal*, v. 2, n. 4, p. 351–367, 2004. Available from Internet: <http://www.sciencedirect.com/science/article/pii/S1570870504000319>.

[Akyildiz et al. 2002]AKYILDIZ, I. F. et al. A survey on sensor networks. *IEEE Communication Magazine*, v. 40, n. 8, p. 102–114, 2002. Available from Internet: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1024422&tag=1>.

[Albayrak et al. 2009]ALBAYRAK, S. et al. Ein framework fuer ambient assisted living services. In: *Proceedings of the Second Deutscher Ambient Assisted Living-Kongress*. VDE-Verlag, 2009. (AAL-Kongress 2009), p. 264–268. Available from Internet: <http://www.vde-verlag.de/proceedings-en/453138063.html>.

[Berns e Mehdi 2010]BERNS, K.; MEHDI, S. A. Use of an autonomous mobile robot for elderly care. In: *Proceedings of the Advanced Technologies for Enhancing Quality of Life*. IEEE Computer Society, 2010. (AT-EQUAL 2010), p. 121–126. ISBN 978-1-4244-8842-1. Available from Internet: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5663614>.

[Braun et al. 2007]BRAUN, T. et al. A customizable, multi-host simulation and visualization framework for robot applications. In: *Proceedings of the 13th International Conference on Advanced Robotics*. VDE-Verlag, 2007. (ICAR 2007), p. 1105–1110. Available from Internet: <http://www.vde-verlag.de/proceedings-en/453138063.html>.

[Chandy 2006]CHANDY, M. K. Event-driven applications: Costs, benefits and design approaches. In: *Presented at the Gartner Application Integration and Web Services Summit*. [S.l.: s.n.], 2006.

[Coroama et al. 2004]COROAMA, V. et al. Living in a smart environment – implications for the coming ubiquitous information society. In: *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*. IEEE Computer Society, 2004. (ICSMC 2004, v. 6), p. 5633–5638. ISBN 0-7803-8566-7. Available from Internet: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1401091>.

[Counsel e Care 2005]COUNSEL; CARE. Community care assessment and services. In: . [S.l.: s.n.], 2005.

[Doorn e Vries 2006]DOORN, M. V.; VRIES, A. P. D. Co-creation in ambient narratives. In: *Ambient Intelligence in Everyday Life*. [S.l.]: Springer, 2006. p. 103–129.

[Eichelberg et al. 2009]EICHELBERG, M. et al. Osami commons: Eine softwareplattform fuer exible verteilte dienstesysteme ueber geraete und eingebetteten systemen. In: *Proceedings of the Second Deutscher Ambient Assisted Living-Kongress*. VDE-Verlag, 2009. (AAL-Kongress 2009), p. 269–272. Available from Internet: <http://www.vde-verlag.de/proceedings-de/453138064.html>.

[Eichelberg et al. 2010]EICHELBERG, M. et al. The gal middleware platform for aal: A case study. In: *Proceedings of the 12th IEEE International Conference on e-Health Networking Applications and Services*. IEEE Computer Society, 2010. (Healthcom 2010), p. 1–6. ISBN 978-1-4244-6374-9. Available from Internet: <http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=05556589>.

[ESTATÍSTICA 2008]ESTATÍSTICA, I. B. D. P. E. Projeção da população do brasil: população brasileira envelhece em ritmo acelerado. In: *Comunicação Social*. [S.l.: s.n.], 2008.

[Fischer et al. 2003]FISCHER, K. et al. Multimedia workplace of the future. In: *Proceedings of the Human Computer Interaction Status Conference*. [s.n.], 2003. p. 97–113. Available from Internet: <http://publica.fraunhofer.de/documents/N-58659.html>.

[Floeck 2010]FLOECK, M. Activity monitoring and automatic alarm generation in aal-enabled homes. In: *PhD thesis*. [S.l.]: Logos-Verlag, 2010.

[Germany 2011]GERMANY. Statistisches bundesamt: Demografischer wandel in deutschland - bevolkerungs- und haushaltsentwicklung im bund und in den ländern. In: . [S.l.: s.n.], 2011.

[Gómez et al. 2010]GóMEZ, C. et al. Sensors everywhere: Wireless network technologies and solutions. In: . [S.l.]: Fundación Vodafone España, 2010. v. 1.

[Hedberg 2000]HEDBERG, S. R. After desktop computing: A progress report on smart environments research. In: *Proceedings of the IEEE Intelligent Systems and their Applications*. IEEE Computer Society, 2000. v. 5, p. 7–9. ISSN 1094-7167. Available from Internet: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=889101>.

[Hein et al. 2009]HEIN, A. et al. A service oriented platform for health services and ambient assisted living. In: *Proceedings of the International Conference on Advanced Information Networking and Applications Workshops*. IEEE Computer Society, 2009. (WAINA 2009), p. 531–537. ISBN 978-1-4244-3999-7. Available from Internet: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5136702>.

[Herfet et al. 2001]HERFET, T. et al. Embassi - multimodal assistance for infotainment and service infrastructures. In: *Proceedings of the Workshop on Universal Accessibility of Ubiquitous Computing: Providing for the Elderly*. [s.n.], 2001. p. 41–50. Available from Internet: <http://www.sciencedirect.com/science/article/pii/S0097849301000863>.

[Hillenbrand et al. 2008]HILLENBRAND, C. et al. Cromsci - a climbing robot with multiple sucking chambers for inspection tasks. In: *Proceedings of the 11th International Conference on Climbing and Walking Robots*. [s.n.], 2008. (CLAWAR 2008), p. 311–318. Available from Internet: <http://agrosy.informatik.uni-kl.de/fileadmin/Literatur/Hillenbrand08a.pdf>.

[Hinze et al. 2009]HINZE, A. et al. Event-based applications and enabling technologies. In: *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*. IEEE Computer Society, 2009. (DEBS 2009), p. 1–15. ISBN 978-1-60558-665-6. Available from Internet: <http://dl.acm.org/citation.cfm?id=1619260>.

[Hirth et al. 2007]HIRTH, J. et al. Emotional architecture for the humanoid robot head roman. In: *Proceedings of the IEEE International Conference on Robotics and Automation*. IEEE Computer Society, 2007. (ICRA 2007), p. 2150–2155. ISBN 1-4244-0601-3. Available from Internet: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4209403>.

[Janse et al. 2007]JANSE, M. et al. Amigo architecture: Service oriented architecture for intelligent future in-home networks. In: *Proceedings of the Constructing Ambient Intelligence – AmI 2007 Workshops*. Springer, 2007. v. 8, p. 371–378. ISBN 978-3-540-85378-7. Available from Internet: <http://link.springer.com/chapter/10.1007

[Karlsson 1996]KARLSSON, R. Printed in helsingin sanomat. In: . [S.l.: s.n.], 1996.

[Kehagias et al. 2008]KEHAGIAS, D. et al. Preliminary architecture of the oasis content connector module. In: *Proceedings of the International Conference Ambient Intelligence System of Agents for Knowledge-based and Integrated Services for E&D users*. [s.n.], 2008. p. 1–12. Available from Internet: <http://server-5.iti.gr/diok/publications/03.confs/OASIS_Content_Connector_Module-ASK-IT-Conf.pdf>.

[Lee et al. 2010]LEE, H. B. et al. Interactive remote control of legacy home appliances through a virtually wired sensor network. In: *Proceedings of the IEEE Transactions on Consumer Electronics*. IEEE Computer Society, 2010. (TCE 2010, v. 56), p. 2241–2248. ISSN 0098-3063. Available from Internet: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5681096>.

[Lemon e Rossi 1995]LEMON, S.; ROSSI, K. An object-oriented device driver model. In: *Proceedings of Compcon: Technologies for the Information Superhighway*. IEEE Computer Society, 1995. (CMPCON 1995), p. 360–366. ISBN 0-8186-7029-0. Available from Internet: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5681096>.

[Martin e Nurmi 2006]MARTIN, M.; NURMI, P. A generic large scale simulator for ubiquitous computing. In: *Proceedings of the 3rd Annual International Conference on Mobile and Ubiquitous Systems - Workshops*. IEEE Computer Society, 2006. (MOBIQW 2006), p. 1–3. ISBN 0-7803-9791-6. Available from Internet: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4205246>.

[Mikalsen et al. 2009]MIKALSEN, M. et al. Interoperability services in the mpower ambient assisted living platform. In: *Proceedings of MIE: The XXIInd International*

*Congress of the European Federation for Medical Informatics*. IOS Press, 2009. p. 366–370. ISSN 0926-9630. Available from Internet: <http://dx.doi.org/10.3233/978-1-60750-044-5-366>.

[Muhl et al. 2006]MUHL, G. et al. Distributed event-based systems. In: . [S.l.]: Springer, 2006. p. 1290–148.

[Proetzsch et al. 2010]PROETZSCH, M. et al. Development of complex robotic systems using the behavior-based control architecture ib2c. In: *Robotics and Autonomous Systems*. [s.n.], 2010. p. 46–67. Available from Internet: <http://www.sciencedirect.com/science/article/pii/S092188900900116X>.

[Prueckner et al. 2008]PRUECKNER, S. et al. Emergency monitoring and prevention eu project emerge. In: *Proceedings of the Erste Deutscher Ambient Assisted Living-Kongress*. VDE-Verlag, 2008. (AAL-Kongress 2008), p. 167–172. Available from Internet: <http://www.vde-verlag.de/proceedings-de/563076032.html>.

[Ras et al. 2007]RAS, E. et al. Engineering tele-health solutions in the ambient assisted living lab. In: *Proceedings of the 21st International Conference on Advanced Information Networking and Applications Workshops*. IEEE Computer Society, 2007. (AINAW 2007), p. 804–809. ISBN 978-0-7695-2847-2. Available from Internet: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4224204>.

[Schaefer et al. 2009]SCHAEFER, H. et al. Ravon — the robust autonomous vehicle for off-road navigation. In: *Proceedings of the IARP International Workshop on Robotics for Risky Interventions and Environmental Surveillance*. [s.n.], 2009. (RISE 2009), p. 1–28. Available from Internet: <http://agrosy.informatik.uni-kl.de/fileadmin/Literatur/Armbrust09.pdf>.

[Schmidt et al. 2006]SCHMIDT, D. et al. Autonomous behavior-based exploration of office environments. In: *Proceedings of the 3rd International Conference on Informatics in Control, Automation and Robotics*. [s.n.], 2006. (ICINCO 2006), p. 235–240. Available from Internet: <http://agrosy.informatik.uni-kl.de/fileadmin/Literatur/Schmidt06a.pdf>.

[SENSACTION-AAL 2007]SENSACTION-AAL. Sensaction-aal project: Sensing and action to support mobility in ambient assisted living. In: *6th Framework Programme of the European Union*. [s.n.], 2007. Available from Internet: <http://www.sensaction-aal.eu>.

[Steg et al. 2006]STEG, H. et al. Europe is facing a demographic challenge ambient assisted living offers solutions. In: *Technical Report*. [S.l.: s.n.], 2006.

[Streitz 2007]STREITZ, N. A. From human-computer interaction ti human-environment interaction: Ambient intelligence and the disappearing computer. In: *Universal Access in Ambient Intelligence Environments*. [S.l.]: Springer, 2007. p. 3–13.

[Sun et al. 2009]SUN, H. et al. Promises and challenges of ambient assisted living systems. In: *Proceedings of the 6th International Conference on Information Technology: New Generations*. IEEE Computer Society, 2009. (ITNG 2009), p. 1201–1207. ISBN 978-1-4244-3770-2. Available from Internet: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5070788>.

[Tezari et al. 2009]TEZARI, M. et al. The persona service platform for aal spaces. In: *Handbook of Ambient Intelligence and Smart Environments*. [S.l.]: Springer, 2009.

[Tynan et al. 2005]TYNAN, R. et al. Agents for wireless sensor network power management. In: *Proceedings of the International Conference Workshops on Parallel Processing*. IEEE Computer Society, 2005. (ICCPW 2005), p. 413–418. ISBN 0-7695-2381-1. Available from Internet: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1488723>.

[Vergados et al. 2008]VERGADOS, D. et al. An intelligent interactive healthcare services environment for assisted living at home. In: *Proceedings of the Second International Conference on Pervasive Computing Technologies for Healthcare*. Springer, 2008. (PervasiveHealth 2008), p. 329. ISBN 978-963-9799-15-8. Available from Internet: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5446172>.

[Wang et al. 2006]WANG, Q. et al. I-living: An open system architecture for assisted living. In: *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*. IEEE Computer Society, 2006. (SMC 2006, v. 5), p. 4268–4275. ISBN 1-4244-0099-6. Available from Internet: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4274570>.

[Weiser 1999]WEISER, M. The computer for the 21st century. *ACM SIGMOBILE Mobile Computing and Communications Review - Special issue dedicated to Mark Weiser*, v. 3, n. 3, p. 3–11, 1999. Available from Internet: <http://dl.acm.org/citation.cfm?id=329126>.

[Wille et al. 2010]WILLE, S. et al. Amica - a flexible, compact, easy-to-program and low-power wsn platform. In: *Mobile and Ubiquitous Systems: Computing, Networking, and Services*. Springer, 2010. (Mobiquitous 2010), p. 381–382. ISBN 978-3-642-29153-1. Available from Internet: <http://link.springer.com/chapter/10.1007

[Wille et al. 2010]WILLE, S. et al. Amica - design and implementation of a flexible, compact, and low-power node platform. In: *Technical Report*. [s.n.], 2010. Available from Internet: <https://kluedo.ub.uni-kl.de/frontdoor/index/index/docId/2807>.

[Wille et al. 2012]WILLE, S. et al. Combining robotic frameworks with a smart environment framework: Mca2/simvis3d and tinysep. In: *Proceedings of the Ubirobots 2012 Workshop conducted at the 14th International Conference on Ubiquitous Computing*. ACM Digital Library, 2012. (UBICOMP 2012), p. 818–825. ISBN 978-1-4503-1224-0. Available from Internet: <http://dl.acm.org/citation.cfm?id=2370405>.

[Wille et al. 2012]WILLE, S. et al. Tinysep - a tiny platform for ambient assisted living. In: *Proceedings of the AAL-Kongress 2012 - Advanced Technologies and Societal Change*. Springer, 2012. (AAL-Kongress 2012), p. 229–243. ISBN 978-3-642-27490-9. Available from Internet: <http://link.springer.com/chapter/10.1007

[Winkley et al. 2012]WINKLEY, J. et al. Verity: an ambient assisted living platform. In: *Proceedings of the IEEE Transactions on Consumer Electronic*. IEEE Computer Society, 2012. (TCE 2012, v. 58), p. 364–373. ISSN 0098-3063. Available from Internet: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6227435>.

[Wolf et al. 2008]WOLF, P. et al. Soprano - an extensible, open aal platform for elderly people based on semantical contracts. In: *Proceedings of the Third Workshop on Artificial Intelligence Techniques for Ambient Intelligence*. [S.l.: s.n.], 2008. (AITAmI 2008).

[Ziefle et al. 2011]ZIEFLE, M. et al. A multi-disciplinary approach to ambient assisted living. In: *E-Health, Assistive Technologies and Applications for Assisted Living: Challenges and Solutions*. IGI Global, 2011. p. 804–809. ISBN 978-0-7695-2847-2. Available from Internet: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4224204>.

# APPENDIX A TINYSEP INTERFACES

In this appendix the programming interfaces from all the TinySEP interfaces used to build the prototype are presented.

## A.1 IBusRadioAmICA Interface

```
public interface IBusRadioAmICA1_0Listener
{
    int receiveData(byte[] payload,
                    DateTime timestamp);
}

public interface IBusRadioAmICA1_0Provider
{
    int sendData(byte[] payload);
    string getSensorId();
}
```

## A.2 IBusRadioSUNSpot Interface

```
public interface IBusRadioSunSPOTListener
{
    int receiveData(byte[] payload, DateTime timestamp);
}

public interface IBusRadioSunSPOTProvider
{
}
```

## A.3 IMovement Interface

```
public struct SLastMovement
{
    public DateTime timestamp;
    public int movementHappened;
}

public interface IMovementProvider
{
    SLastMovement getLastMovement();
    int getLastMovementType();
}

public interface IMovementListener
{
    void movementHappened(object device,
                          SLastMovement movement);
    void movementTypeChanged(object device,
                             int newMovementType);
}
```

## A.4 IWindow Interface

```
public interface IWindowProvider
{
    int getWindowStatus();
    SPosition getWindowPosition();
}

public interface IWindowListener
{
    void windowStatusChanged(object device,
                             int newWindowStatus,
                             DateTime timestamp);
    void windowPositionChanged(SPosition windowPos);
}
```

## A.5 IDoor Interface

```
public struct SPosition
{
    public float xAxis;
    public float yAxis;
    public float zAxis;
}

public interface IDoorProvider
{
    int getDoorStatus();
    float getDoorAngle();
    int getDoorOpeningDirection();
    int getDoorType();
    SPosition getDoorPosition();
    DateTime getLastChangedTimestamp();
}

public interface IDoorListener
{
    void doorStatusChanged(object device,
                           int newDoorStatus,
                           DateTime timestamp);
    void doorTypeChanged(object device,
                         int newDoorType);
    void doorPositionChanged(SPosition DoorPosition);
}
```

## A.6 ISimpleLight Interface

```
public interface ILightBrightnessProvider
{
    int getLightBrightnessValue();
}

public interface ILightBrightnessListener
{
    void lightBrightnessChanged(object device,
                                int newLightStatus,
                                DateTime timestamp);
}
```

## A.7 IDimmableLight Interface

```
public interface IDimmableLightProvider
{
    int getDimmableLightStatus();
    void changeLightStatus(DateTime timestamp,
                           int newDimmableLightStatus);
}

public interface IDimmableLightListener
{
    void dimmableLightStatusChanged(object device,
                                    int newLightStatus,
                                    DateTime timestamp);
}
```

## A.8 IHeater Interface

```
public interface IHeaterProvider
{
    int getHeaterStatus();
}

public interface IHeaterListener
{
    void heaterStatusChanged(object device,
                             int newHeaterBoxStatus,
                             DateTime timestamp);
}
```

## A.9 ISpeaker Interface

```
public interface ISpeakerProvider
{
    int getSpeakerStatus();
}

public interface ISpeakerListener
{
    void speakerStatusChanged(object device,
                              int newSpeakerStatus,
                              DateTime timestamp);
}
```

## A.10 ITemperature Interface

```
public interface ITemperatureProvider
{
    int getTemperatureStatus();
}

public interface ITemperatureListener
{
    void temperatureStatusChanged(object device,
                                  int newTemperature,
                                  DateTime timestamp);
}
```

## A.11 ILightBrightness Interface

```
public interface ILightBrightnessProvider
{
    int getLightBrightness();
}

public interface ILightBrightnessListener
{
    void lightBrightnessChanged(object device,
                                int newLightStatus,
                                DateTime timestamp);
}
```

## A.12 IFlatOccupancy Interface

```
public interface IFlatOccupancyProvider
{
    int getFlatOccupancyStatus();
}

public interface IFlatOccupancyListener
{
    void flatOccupancyStatusChanged(int newStatus,
                                    DateTime timestamp);
}
```

## A.13 IHouseInformation Interface

```
public struct roomInfo
{
    public string roomName;
    public string roomType;
    public List<roomInfo> roomNeighbors;
}

public interface IHouseInformationProvider
{
    Dictionary<Guid, roomInfo> getHouseInformation();
    void setHouseInformation
            (Dictionary<Guid, roomInfo> newHouseInfo);
    roomInfo getRoomWithUid(string uid);
    void addNewRoom();
}

public interface IHouseInformationListener
{
    void houseInfoChanged
             (Dictionary<Guid,roomInfo> newHouseInfo);
}
```

## A.14 IMailManager Interface

```
public interface IMailManagerProvider
{
    void sendMailMessage(string mailBody,
                         string mailSubject);
}

public interface IMailManagerListener
{
}
```

## A.15 ITimeController Interface

```
public interface ITimeControllerProvider
{
    DateTime getCurrentTime();
    void getNextSecond();
}

public interface ITimeControllerListener
{
    void currentTimeChanged(DateTime currentTime);
}
```

## A.16 IInactivityRecognition Interface

```
public interface IInactivityRecognitionProvider
{
    int getInactivityRecognitionStatus();
}

public interface IInactivityRecognitionListener
{
    void inactivityRecognitionStatusChanged
                             (int newStatus,
                              DateTime timestamp);
}
```