TinySEP - A tiny platform for Ambient Assisted Living

Sebastian Wille, Ivan Shcherbakov, Luiza de Souza and Norbert Wehn

Microelectronic Systems Design Research Group,
University of Kaiserslautern,
Germany
{wille,shcherbakov,wehn@eit.uni-kl.de}, luiza.pad@gmail.com,
http://ems.eit.uni-kl.de/

Abstract. Many AAL software platforms developed in the last years are not widely accepted outside of their projects, because of their high complexity. Moreover, many successful software frameworks are based on the thin architecture approach: they implement only basic functionality, being more flexible in practical use cases. In this work we present the **Tiny S**mart **E**nvironment **P**latform. **TinySEP** is a compact platform, which makes use of two approved concepts of the software engineering. It combines the benefits of complex modular platforms and proprietary monolithic solutions.

Keywords: TinySEP, Tiny, Smart, Environment, Platform, AAL, framework, driver model, device, signal-slot model

1 Introduction

Health insurance companies, their customers and individual care-givers create numerous use cases for Ambient Assisted Living (AAL) systems. Moreover, the AAL systems are not limited to medical services. They can provide more comfort in everyday life (e. g. automatic light control), higher security (e. g. burglar or fire alarm) or automatic energy saving. The focus for health insurance companies is a manifold use in the medical area. For example, fall detection *and* supervision of daily medication *and* integration of tele-care services shall be supported at the same time. Furthermore, the AAL system has to adapt itself automatically to the given boundary conditions like already installed hardware of the actual housing situation. If other services are demanded in future or the boundary conditions change (e. g. structural measures), the system has also to readjust itself. From the developers point of view it shall be easy to integrate new hardware and new services at any time. Finally, AAL systems have to be reliable and cheap enough to be interesting for a wide customer segment.
A typical AAL system consists of several sensor/actor nodes (called AAL hardware in the following), a central computational unit and a central AAL software platform. The required AAL hardware is today available in form of wired solutions like KNX[1] or wireless solutions like Wireless Sensor Networks (WSN,

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

e. g. AmICA [8]). Many standards for wireless communication, like WLAN or ZigBee[2], are also available today. Netbooks or set-top boxes are common used as computational units.
As for 2011 there exists no standard for an AAL software platform. That is one of the reasons, why AAL systems are not wide spread despite of years of intense research work. Although a lot of different, modular AAL platforms were designed in research projects, for concrete, practical projects proprietary, monolithic systems are often developed. A lot of platforms are universal and flexible, but require significant time to understand and install the overall system. Moreover a platform typically requires to be adapted to realize special functions. Thereto, the complex entire structure has to be understood.
In many cases most of the complex functionality of a "heavy" universal framework remains unused in practice. The most famous example is the ISO/OSI layered model: the TCP/IP stack, which is used in reality, does not implement a lot of the proposed layers. In the area of Wireless Sensor Networks, TinyOS [7] as a flexible, thin and resource-saving operating system has established.
In this paper we present the **Tiny Smart Environment Platform "TinySEP"**. TinySEP bridges the gap between the two approaches used so far and combines the advantages of both. It makes use of two very successful and frequently used concepts of software engineering: the driver concept and the signal-slot model. TinySEP is designed for developers of AAL solutions and can be seen as the lowest common denominator. While giving the developers of AAL hardware, intermediate services and AAL services maximum freedom, TinySEP connects the single elements to one AAL system.
The paper is structured as follows: Next section shows the requirement of a modern AAL software platform. State of the art is presented in section 3. Base models of TinySEP and their design are depicted in section 4. Section 5 shows how the layered model is used in TinySEP. The practical use of TinySEP is explained in section 6. Section 7 gives a closer look at the openAAL and universAAL project. The last section summarizes the paper and gives an outlook how TinySEP can be used in future.

2 Requirement of a AAL platform

The following **functional requirements** are essential for developing an AAL platform:

- Hardware abstraction
- Open system interfaces
- Changes of hardware and software at runtime
- Sensor fusion and context management
- Mechanisms for self-configuration and -adaption

[2] http://www.zigbee.org/

These are the **non-functional requirements**:

- High re-usability of single components
- Internal system communication and ontologies
- High usability for the developer
- Low resource consumption
- Easy changeability and portability

	TinySEP	Monolithic systems	Platform-based systems
Hardware abstraction	+	−	+
Open system interfaces	+	−	+
Changes of hardware and software at runtime	+	−	+
Sensor fusion and context management	+	**o**	+
Mechanisms for self-configuration and -adaption	+	−	+
High re-usability of single components	+	−	+
Internal system communication and ontologies	+	−	+
High usability for the developer	+	+/−[1]	−
Low resource consumption	+	+	**o**
Easy changeability and portability	+	**o**	−

Table 1. Comparison of fulfilling the requirements of TinySEP, monolithic systems and modular AAL platforms; [1]only high for internal developers, who know the system structure

3 State of the art

Home emergency call and classic home automation systems can only be partly seen as AAL systems. Beside them, existing AAL software can be divided in two classes: proprietary, monolithic systems and modular AAL platforms. Both classes are described in the next two subsections. They are evaluated based on the criteria given in section 2. A summary is presented in Tab. 1.

3.1 Monolithic systems

Proprietary, monolithic systems are developed bottom-up for concrete projects and problem statements. The functional requirements as well as the boundary

conditions are well-defined from the beginning. As a result, these AAL platforms are optimally adapted to the problem statement. Developers can use their well-known paradigms resulting in a faster development cycle. The resource consumption is minimized by implementing only the required functionality. Thus, the underlying hardware can be very compact and energy-efficient.
The biggest disadvantage of monolithic systems is their low re-usability. Own concepts for hardware abstraction has to be developed, sensor-fusion is highly dependent on the algorithms and a context-management as well as ontologies are often not existing. Open system interfaces are very rare. Changes to the software require restarting, manual interference or even recompilation. Another downside is, that mechanisms for self-configuration and -adaption are often missing. As a result, those systems have to be manually reconfigured for every flat configuration and every time new services are introduced. The advantages in the development phase are often neutralized, because simulators and methods of early debugging (Virtual Platform concepts) are not available. In many cases, external developers cannot integrate own services, because those monolithic systems are often closed-source. Consequently, more and more monolithic platforms are developed each year, each requiring time and money.
To sum up, monolithic systems fit well for projects with a well-defined functionality, clear boundary requirement and low probability of late changes and extensions. If new functionality is needed or the system has to be used in different, but similar environments (other flat, other hardware etc.), more time is needed and higher costs rise up.
Examples for monolithic AAL systems are the EU project EMERGE[3] [5], the Assisted Living project PAUL[4] [3], which realizes a inactivity recognition based on motion sensors and the demo flat of the "Verband Sächsischer Wohnungsgenossenschaften e. V." in Burgstädt/Germany[5].

3.2 Platform-based systems

Modular AAL platforms are developed top-down. Most important are universal applicability, high flexibility and re-usability. Mechanisms for hardware abstraction and open system interfaces are integrated to connect with future AAL hardware and AAL services from third-party supplier. A running system can be changed without a re-start. Normally, AAL hardware as well as AAL services can be removed and added while the systems is running. Sensor fusion is used to generate general context information, which can be used by different AAL services. One example is the "Context Manager" of SOPRANO [6]. Together with other components, like the "UI Engine", the Follow-Me-User-Interface or the localization service of SerCHo [1], mechanisms for self-configuration and -adaption are realized. Encapsulation of functions leads to a higher re-usability. Many AAL platforms require knowledge of specific programming languages and

[3] http://www.emerge-project.eu/
[4] http://assistedliving.de/
[5] http://www.wbg-burgstaedt.de/news_details.php?Pos=23

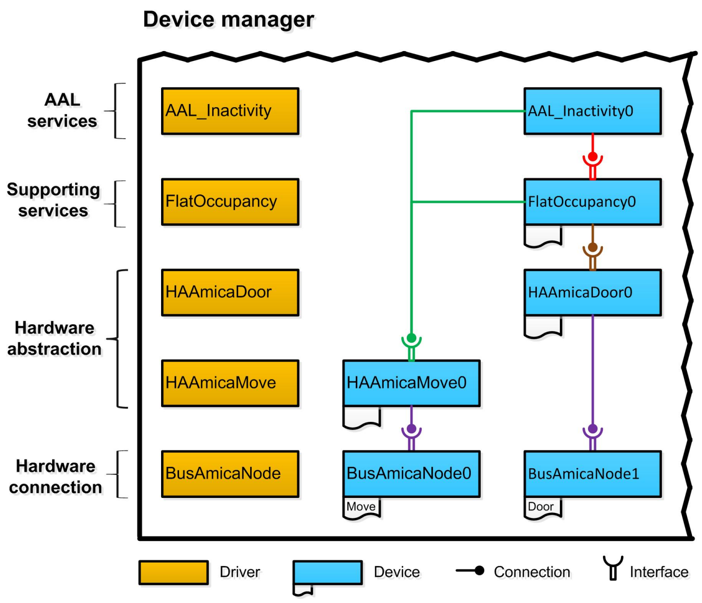

Fig. 1. TinySEP sample configuration

tools. For example, to work with SerCHo [1] one has to learn the Business Process Modelling Notation (BPMN). AAL solutions, which are already developed in other programming languages, have to be re-implemented and tested again. Changes and extensions of the platform itself are possible, but need a deep knowledge of the complex software. Beside that, there are practical barriers like urgent memberships. Examples are OSAmI[6] [2] or the Connected Living e. V.[7] with costs. Most of real installations require relatively simple hardware and have no need for most complex functionality provided by the frameworks. One example is the german BMBF project "TSA - Technisch-soziales Assistenzsystem" (TSA - technical-social assistance system)[8]. SerCHo is not able to exploit its UI Engine, if only *one* TV output is present. Modular AAL platforms offer often a *too big* functional, confusing range. That is the reason, why often monolithic systems are developed again, because this is seen as the "faster and easier" way. Other examples of modular AAL platforms are AMIGO, GENESYS, OASIS, MPOWER and PERSONA, which advantages shall be melted in the universAAL project[9] [4].

[6] http://www.osami-commons.org/
[7] http://www.connected-living.org/
[8] http://www.spellerberg-stadtsoziologie.de/forschung.php?id=15
[9] http://www.universaal.org/

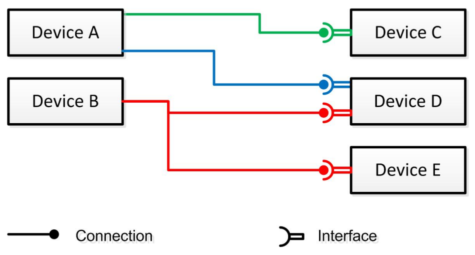

Fig. 2. TinySEP Interfaces

4 TinySEP

Ambient Assisted Living (AAL) systems are typically event-driven: for example, when someone enters a room, the movement detector registers it and the light is switched on. In parallel, the information is forwarded to the inactivity recognition system. Thus, TinySEP is also event-driven.
TinySEP was developed after the analysis of 12 typical services in the areas of healthcare monitoring, comfort, security and energy saving. Each AAL platform has to consist of at least four layers:

- Hardware connection
- Hardware abstraction
- Intermediate services
- AAL services

Each layer is reflected in TinySEP. The main focus during TinySEP development was to keep the complexity as low as possible while fulfilling all requirements for a AAL system described in section 2. For implementing the platform, we have adapted two concepts from software engineering world: the driver concept and the signal-slot model. With the help of these two models, all layers can be represented. Fig. 1 shows a exemplary system status.

4.1 Drivers and devices

The **driver model** follows the 4-layer model and is used in most modern operating systems. An operating system is able to access the hardware with the help of drivers. Further, a driver creates so called **device** objects, accessible by other drivers and devices. For example, a USB controller driver creates a new USB device, if a mouse is plugged into a the computer. This device provides raw data from the USB mouse. A USB mouse driver creates a second device that converts raw data into a common format (e. g. cursor moving instructions).
TinySEP utilizes a similar driver model. Each of the four layers listed in the previous section can be represented in TinySEP. Devices are objects that encapsulate certain functionality. For example, Devices can be used for hardware

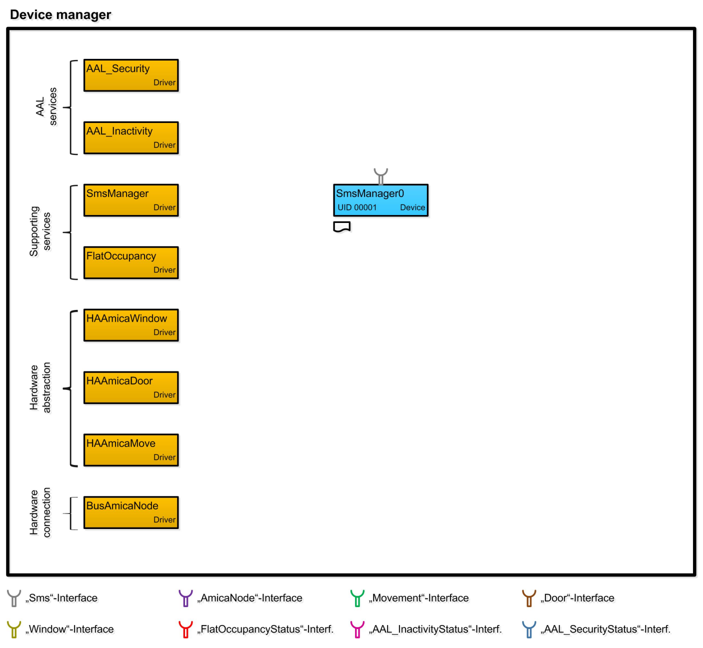

Fig. 3. TinySEP example scenario, part 1: after first startup

abstraction or for AAL intermediate services. Additionally, each Device object stores internal configuration information, which can be accessed from outside. If e. g. a door sensor is abstracted, the corresponding Device provides the information about the location of the door in the flat or if it is an inner or outer door. Other Devices need this information to decide, if they want to connect to the Device or not. Section 5.1 to 5.4 give practical examples.

4.2 Interfaces

In a event-driven system, objects can be connected with the help of the signal-slot model[10]. Signals are "messages", which are sent to slots. A signal can be sent to several slots. A slot can receive several different messages.
The signal-slot model is used in TinySEP to interconnect Devices. To enable this, the signal-slot model was extended in two ways. First, several signal-slot pairs are bundled to so called **Interfaces**. Second, always a bidirectional connection is established. Thus, Device A is able to send different signals to Device B. Device B is also able to send data back to Device A. An overview is shown in Fig. 2.

[10] First use in the QT library, see http://qt.nokia.com/

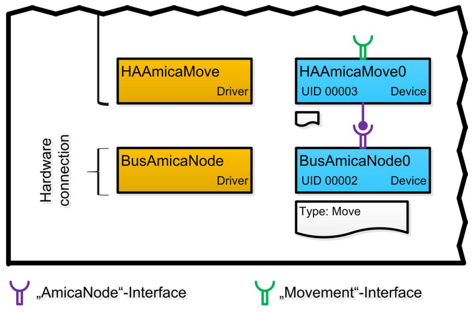

Fig. 4. TinySEP example scenario, part 2: after installation of the first AmICA WSN movement node

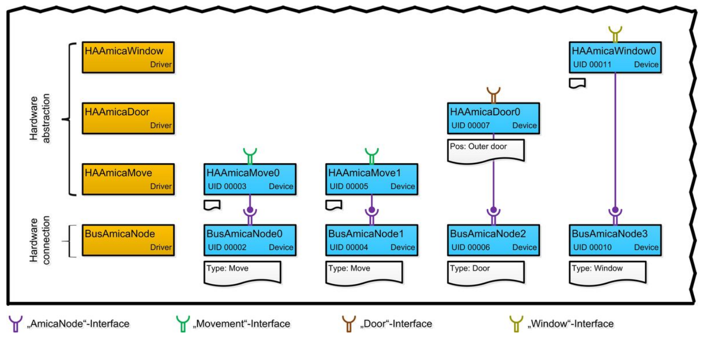

Fig. 5. TinySEP example scenario, part 3: after installing of all AmICA WSN nodes

4.3 Device manager

All Drivers, Devices and available Interfaces are administrated by the **Device manager**. New Drivers register them-self at the Device manager and tell him, about which new interface types they want to be informed. Devices also register at the Device manager and tell him, which interfaces they provide and with which interface types they want to connect. The Device manager implements only minimal functionality, while the intelligence of the system is realised with the help of Drivers, Devices and their Connection with each other. In Fig. 1 the components of TinySEP are shown exemplary.

5 Layered model of TinySEP

We demonstrate the role of TinySEP on two typical AAL service examples. An **Inactivity recognition** sends a SMS message to a mobile phone of a predefined person as soon as there was for a longer time no activity (movement, opening of doors/windows, etc.), although a person is at home. A **Security surveillance**

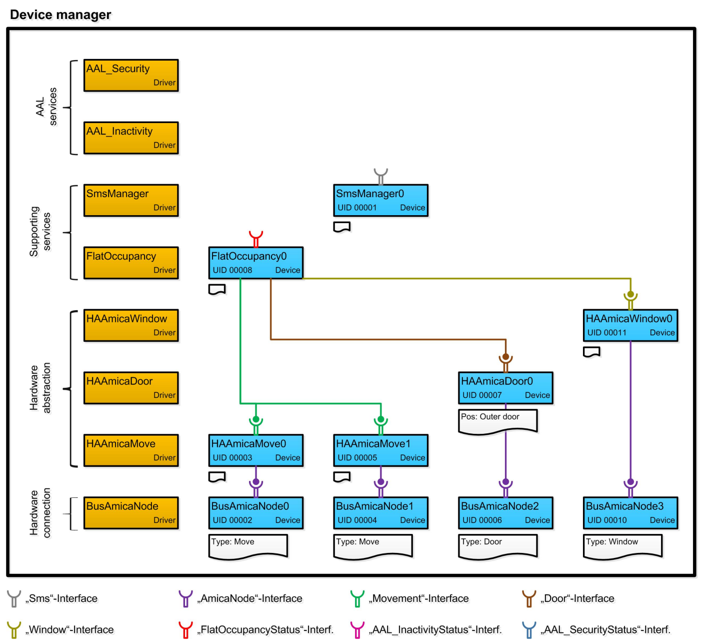
Device manager
AAL services
Supporting services
Hardware abstraction
Hardware connection
AAL_Security Driver
AAL_Inactivity Driver
SmsManager Driver
FlatOccupancy Driver
HAAmicaWindow Driver
HAAmicaDoor Driver
HAAmicaMove Driver
BusAmicaNode Driver
SmsManager0 UID 00001 Device
FlatOccupancy0 UID 00008 Device
HAAmicaWindow0 UID 00011 Device
HAAmicaDoor0 UID 00007 Device
Pos: Outer door
HAAmicaMove0 UID 00003 Device
HAAmicaMove1 UID 00005 Device
BusAmicaNode0 UID 00002 Device
BusAmicaNode1 UID 00004 Device
BusAmicaNode2 UID 00006 Device
BusAmicaNode3 UID 00010 Device
Type: Move
Type: Move
Type: Door
Type: Window
„Sms"-Interface
„AmicaNode"-Interface
„Movement"-Interface
„Door"-Interface
„Window"-Interface
„FlatOccupancyStatus"-Interf.
„AAL_InactivityStatus"-Interf.
„AAL_SecurityStatus"-Interf.

Fig. 6. TinySEP example scenario, part 4: after creation of the "FlatOccupancy0" Device

sends a SMS message as soon as the last inhabitant left the flat, but one or more windows are still open. Both services have to know, if anyone is in the flat and both have to be able to send SMS messages. This reusable functionality is outsourced in the two intermediate services "FlatOccupancy" and "SmsManager". All of the four layers – hardware connection, hardware abstraction, intermediate services and AAL services – are implemented with the help of the driver and extended signal-slot model.
For a better clearness, the functionality of the platform is illustrated for a flat with four AmICA WSN nodes: two for movement, one for the flat door status (open/closed) and one for a window status (open/closed).

5.1 Hardware connection

All available drivers are loaded, when TinySEP is started. Fig. 3 show the internal status of TinySEP after startup, when no AAL hardware was installed. In this example, the following drivers are loaded by default:

- BusAmicaNode (**hardware connection**)
- HAAmicaMove, HAAmicaDoor, HAAmicaWindow (**hardware abstraction**)
- FlatOccupancy, SmsManager (**AAL intermediate services**)
- AAL_Inactivity, AAL_Security (**AAL services**)

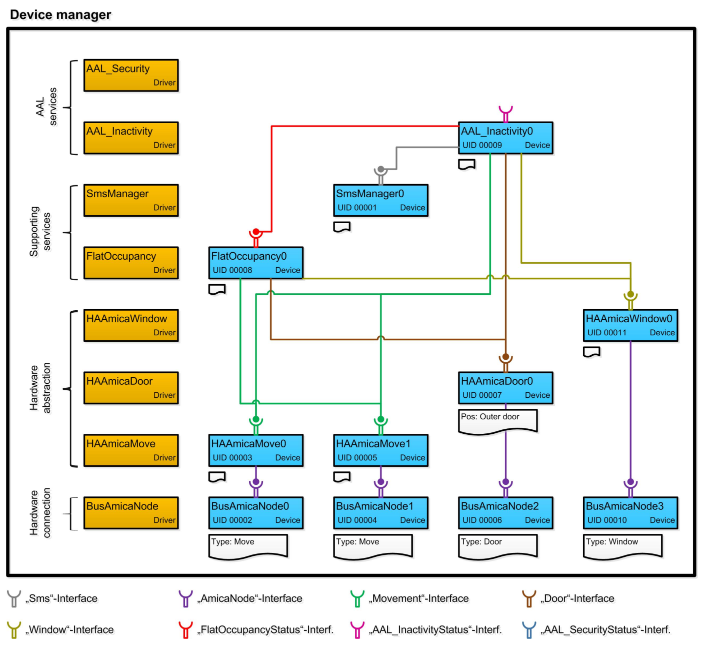
Device manager
AAL services
Supporting services
Hardware abstraction
Hardware connection
AAL_Security Driver
AAL_Inactivity Driver
SmsManager Driver
FlatOccupancy Driver
HAAmicaWindow Driver
HAAmicaDoor Driver
HAAmicaMove Driver
BusAmicaNode Driver
AAL_Inactivity0 UID 00009 Device
SmsManager0 UID 00001 Device
FlatOccupancy0 UID 00008 Device
HAAmicaWindow0 UID 00011 Device
HAAmicaDoor0 UID 00007 Device
Pos: Outer door
HAAmicaMove0 UID 00003 Device
HAAmicaMove1 UID 00005 Device
BusAmicaNode0 UID 00002 Device
BusAmicaNode1 UID 00004 Device
BusAmicaNode2 UID 00006 Device
BusAmicaNode3 UID 00010 Device
Type: Move
Type: Move
Type: Door
Type: Window
„Sms"-Interface
„AmicaNode"-Interface
„Movement"-Interface
„Door"-Interface
„Window"-Interface
„FlatOccupancyStatus"-Interf.
„AAL_InactivityStatus"-Interf.
„AAL_SecurityStatus"-Interf.

Fig. 7. TinySEP example scenario, part 5: after creation of the "AAL_Inactivity0" Device

Only the Driver "SmsManager" immediately creates a Device, because the Device is independent from other Devices. When the first AmICA WSN movement node is installed, it starts sending packets. As soon as the Driver "BusAmicaNode" receive the first packet of this new node, it creates a new Device of the type "BusAmicaNode" with the Interface "AmicaNode" and the information "Type: Move" (see also Fig. 3). With the help of this Device, other Devices can now access the node to receive raw data or to send configuration data to the sensor node.

5.2 Hardware abstraction

Drivers of the hardware abstraction layer are responsible for converting the raw data into usable data and vice versa. The new device "BusAmicaNode0" registers itself with its Interface at the Device manager. The three Drivers "HAAmicaMove", "HAAmicaDoor" and "HAAmicaWindow" are informed about the new Device, because they might be interested in connecting to its Interface. All three Drivers read out the information "Type: Move". Driver "HAAmicaMove" creates a new Device called "HAAmicaMove0". This Device analyses the raw data and abstract it for other Devices, which can connect to the Device via the Interface "Movement". The two other drivers do not create Devices, because the can only

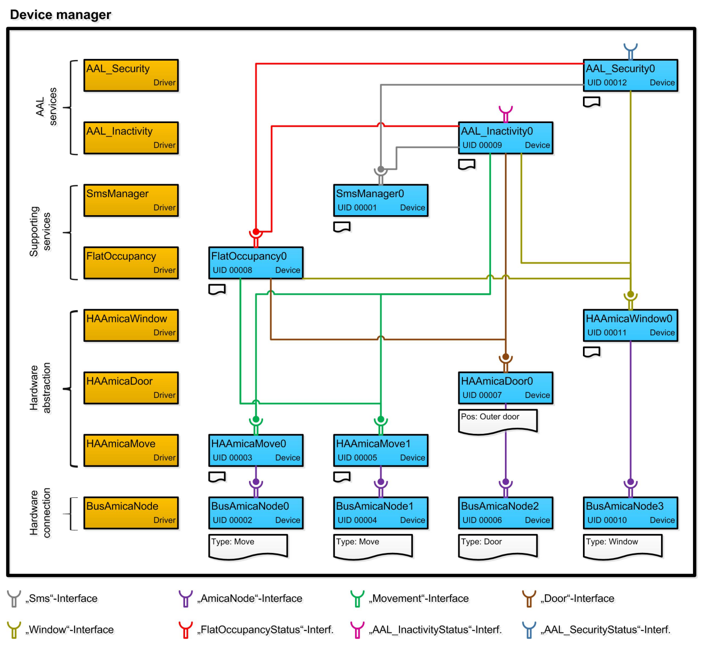
Device manager
AAL services
Supporting services
Hardware abstraction
Hardware connection
AAL_Security Driver
AAL_Inactivity Driver
SmsManager Driver
FlatOccupancy Driver
HAAmicaWindow Driver
HAAmicaDoor Driver
HAAmicaMove Driver
BusAmicaNode Driver
AAL_Security0 UID 00012 Device
AAL_Inactivity0 UID 00009 Device
SmsManager0 UID 00001 Device
FlatOccupancy0 UID 00008 Device
HAAmicaWindow0 UID 00011 Device
HAAmicaDoor0 UID 00007 Device
Pos: Outer door
HAAmicaMove0 UID 00003 Device
HAAmicaMove1 UID 00005 Device
BusAmicaNode0 UID 00002 Device
BusAmicaNode1 UID 00004 Device
BusAmicaNode2 UID 00006 Device
BusAmicaNode3 UID 00010 Device
Type: Move
Type: Move
Type: Door
Type: Window
„Sms"-Interface
„AmicaNode"-Interface
„Movement"-Interface
„Door"-Interface
„Window"-Interface
„FlatOccupancyStatus"-Interf.
„AAL_InactivityStatus"-Interf.
„AAL_SecurityStatus"-Interf.

Fig. 8. TinySEP example scenario, part 6: after creation of the "AAL_Security0" Device

convert raw data from other AmICA WSN node types. The actual system status is shown in Fig. 4.
When the second AmICA WSN movement node is connected, two more Devices ("BusAmicaNode1" and "HAAmicaMove1") are created. The AmICA WSN door and window nodes are similar. Additionally, in the case of the door sensor, the user is asked, if it is a inner or outer door. In the end, two more Interface types, "Door" and "Window", are available. The actual system status is depicted in Fig. 5.

5.3 Intermediate services

Intermediate services can also be implemented in the TinySEP driver model. There are two intermediate services in the example. "FlatOccupancy" recognizes, if at least on person is in the flat. "SmsManager" is able to send SMS messages for for other Devices. As described in section 5.1, the "SmsManager" Driver has already created a Device. Other Devices can connect to this Device via the Interface "SMS".
The "FlatOccupancy" Driver waits till the Device manager has informed it about at least one Interface of the type "Movement" and one of the type "Door" with the information "Pos. = Outer door". As soon as this is the case, the Device

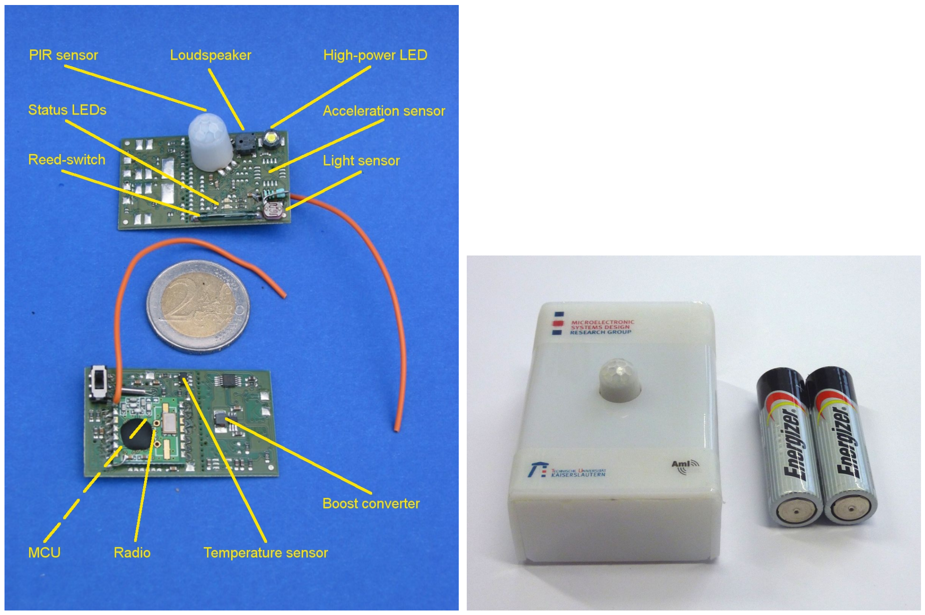

Fig. 9. AmICA WSN sensor node; without (left) and with (right) housing

"FlatOccupancy0" is crated. It connects with the necessary Interfaces. In turn, it provides "FlatOccupancyStatus" Interface for other devices. The actual system status of TinySEP is shown in Fig. 6.

5.4 AAL services

Similar to the intermediate services, AAL services can also be implemented using TinySEP driver model. In this example, an Inactivity recognition and a Security surveillance service were chosen. They are represented by the "AAL_Inactivity" and "AAL_Security" Drivers. Driver "AAL_Inactivity" creates a Device, which connects to the Interfaces "FlatOccupancyStatus", "Movement", "Door" and "Windows". The actual status is shown in Fig. 7.
The "AAL_Security" Driver creates a Device, which connects to the "FlatOccupancyStatus" and "Window" Interfaces. The final system status of TinySEP is depicted in Fig. 8.

6 Real-world setup with TinySEP

To evaluate TinySEP in real-world, two actual inhabited flats were equipped with AAL hardware. The two floor plans are depicted in Fig. 10. The flexible, easy-to-program and compact AmICA WSN platform was used [8]. In comparison to other WSN platforms, the AmICA sensor nodes support various on-board

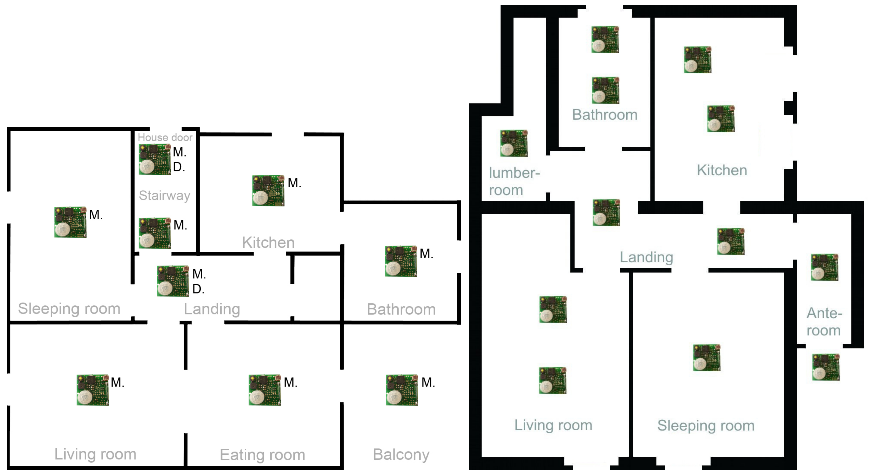

Fig. 10. Inhabited AAL test flats; flat A (left) and flat B (right)

sensors like different kind of movement detectors, light sensors, reed-switches or temperature sensors. Further, they save more energy than sensor nodes of other WSN platforms [9]. Especially that is important for AAL, because each battery change leads to maintainance costs.
Flat A is inhabited by a 87 year old, solitarily person. It is equipped with nine movement detectors and two door contacts. Over one million real data points were collected till now. Flat B is is inhabited by a 27 year old single person and equipped with 12 movement detectors. For a concrete evaluation, 12 representative services of the areas health-care monitoring, comfort, security and energy-saving were chosen. Fig. 11 depicts the data collected during one day and the state of the "FlatOccupancy" flag. The bottom row "Calculated Flat Occupancy" shows in blue the periods, where the person was at home. Orange shows the periods, the person was not at home. The algorithms of the Driver were developed and optimized based on the real data. At the moment, further AAL service are developed and evaluated with the help of the collected data. One example is a Driver for detecting burglars based on the installed hardware.

7 openAAL and universAAL

openAAL is a joint open source initiative by FZI Research Center for Information Technologies, Friedrich-Schiller-University of Jena and CAS Software AG. Amongst other it is based on the SOPRANO project. openAAL uses the service-oriented OSGi framework and a own-defined ontology. On top, there are the core component. The Context Manager collects all sensor information and user input, combines them and offers them to the other core components. The Procedural Manager manages from a concrete installtion independet workflows, which were

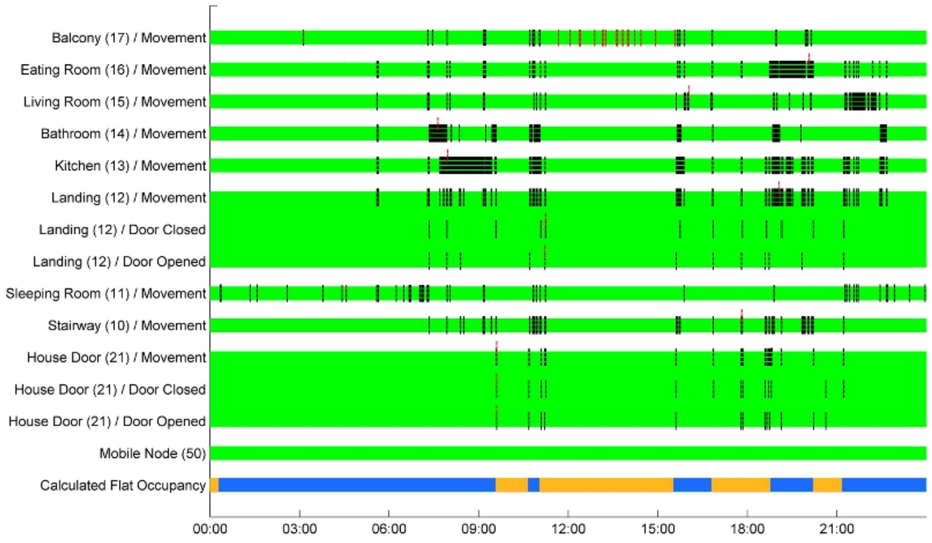

Fig. 11. Graphical representation of collected data in flat A of one day; green: function status of the WSN nodes; blue/orange: calculated periods, where a person was at home/was not at home

created with BPEL[11] processes. The Composer knows all availabe services in a concrete installation. He combines them adaptively in order to achieve the (abstract) service goals defined by the Procedural Manager.
TinySEP is not using an explicit ontology. Communication between Devices take part with pre-definfied interfaces. The centralized functions of the Context Manager, the Procedural Manager and the Composer are resolved in the single Devices. For example, in TinySEP is no central sensor fusion. Single Devices of the supporting service layer realize this function.
The EU project universAAL started in February 2010. It consolidates the most important AAL projects for the first time. The main results of the projects persona, MPOWER, SOPRANO, AMIGO, OASIS, GENESYS, VAALID and AALIANCE shall be melted. A kind of "AAL App Store" named uStore will be created. Also a so-called "universAAL developer depot" is planned. Interoperability is a overall goal.

8 Summary

TinySEP takes advantage of two successful concepts of the software engineering: the driver concept and the signal-slot model. Thereby, the high usability of the proprietary, monolithic systems and the high re-usability of encapsulated components of modular AAL software platforms are combined. TinySEP covers

[11] http://de.wikipedia.org/wiki/BPEL

all important software layers of an AAL platform. We see TinySEP as a starting point for a evolutionary process to develop a compact AAL platform. TinySEP shall be compared in detail with the openAAL and universAAL project in future work. In order to test TinySEP in real scenarios, two normally inhabited flats were equipped with AAL hardware. The AmICA WSN sensor nodes were chosen for the hardware. Currently, TinySEP is extended with methods of the Virtual Platform concept to enable early simulations under real conditions.

References

1. S. Albayrak, M. Blumendorf, S. Feuerstack, T. Kster, A. Rieger, V. Schwartze, C. Wirth, and P. Zernicke. Ein Framework fuer Ambient Assisted Living Services. In *2. Deutscher Ambient Assisted Living-Kongress, VDE-Verlag, Berlin, 2009*, pages 264–268, 2009.
2. M Eichelberg, F. Stewing, W Thronicke, K. Hackbarth, M. Seufert, C. Busch, A. Hein, H. Krumm, M. Ditze, and F. Golatowski. OSAMI Commons: Eine Softwareplattform fuer flexible verteilte Dienstesysteme ueber Geraete und eingebetteten Systemen. In *2. Deutscher Ambient Assisted Living-Kongress, VDE-Verlag, Berlin*, pages 269–272, 2009.
3. M. Floeck. *Activity Monitoring and Automatic Alarm Generation in AAL-enabled Homes.* PhD thesis, Logos-Verlag, Berlin, 2010.
4. S. Hanke, C. Mayer, O. Hoeftberger, H. Boos, R. Wichert, M. Tazari, and P. Wolf. universAAL eine offene und konsolidierte AAL-Plattform. In *4. Deutscher Ambient Assisted Living-Kongress, VDE-Verlag, Berlin*, 2011.
5. S Prueckner, C Madler, D Beyer, M Berger, T. Kleinberger, and M. Becker. Emergency Monitoring and Prevention EU Project EMERGE. In *1. Deutscher Ambient Assisted Living-Kongress, VDE-Verlag, Berlin*, pages 167–172, 2008.
6. A. Schmidt, P. Wolf, M. Klein, and D. Balfanz. SOPRANO Ambient Middleware: Eine offene, flexible und marktorientierte semantische Diensteplattform fuer Ambient Assisted Living. In *2. Deutscher Ambient Assisted Living-Kongress, VDE-Verlag, Berlin*, pages 259–263, 2009.
7. Werner Weber, Jan Rabaey, and Emile H.L. Aarts. Chapter TinyOS: An Operating System for Sensor Networks. In *Ambient Intelligence. Springer Verlag, ISBN 978-3-540-23867-6*, 2005.
8. Sebastian Wille, Norbert Wehn, Ivan Martinovic, Simon Kunz, and Peter Goehner. AmICA - A fexible, compact, easy-to-program and low-power WSN platform. In *Springer LNICST series (Mobiquitous 2010)*, http://ems.eit.uni-kl.de/uploads/tx_uniklwehn/AmICA_A_flexible_compact_easy-to-program_and_low-power_WSN_platform.pdf, dec. 2010.
9. Sebastian Wille, Norbert Wehn, Ivan Martinovic, Simon Kunz, and Peter Goehner. AmICA - Design and implementation of a flexible, compact, easy-to-program and low-power WSN platform. Technical report, http://ems.eit.uni-kl.de/uploads/tx_uniklwehn/AmICATechReport2010.pdf, oct. 2010.