Generic Platform for Advanced E-Health Applications

Elmar Zeeb*, Guido Moritz*, Wolfgang Thronicke†, Myriam Lipprandt‡, Andreas Hein‡ Frerk Müller‡, Jan Krüger§, Oliver Dohndorf§, Anna Litvina¶, Christoph Fiehe¶, Ingo Lück¶, Frank Golatowski* and Dirk Timmermann*
*Institute of Applied Microelectronics and Computer Engineering
University of Rostock, 18057 Rostock, Germany
{elmar.zeeb, guido.moritz, frank.golatowski, dirk.timmermann}@uni-rostock.de
†Siemens AG, SIS C-LAB, Fürstenallee 11, 33102 Paderborn, Germany
wolfgang.thronicke@siemens.com
‡OFFIS Institute for Information Technology, Oldenburg, Germany
{myriam.lipprandt, frerk.mueller, andreas.hein}@offis.de
§TU Dortmund University, 44221 Dortmund, Germany
{jan.krueger, oliver.dohndorf}@tu-dortmund.de
¶MATERNA Information & Communications, 44141 Dortmund, Germany
{anna.litvina, christoph.fiehe, ingo.lueck}@materna.de

***Abstract*— The demographic change and the cost pressure in the healthcare sector drive the need for efficient and secure medical homecare solutions which apply for the people who are elderly or in anastasis. As the complexity of such systems is rising, there is a need of a common foundation of components which can be reused to lower the implementation effort of such systems. The European ITEA2 OSAmI research project targets such a common foundation of basic components for a basic, widely applicable service-oriented component platform. The German OSAmI-D subproject develops a construction kit based on the OSAmI component platform with a particular focus on services for medical and E-Health applications. The results of the ongoing project and especially the foundation of reusable components are demonstrated in the scenario of home-based ergometer training during the rehabilitation of patients with cardiologic illnesses.**

I. Introduction

The complexity of flexible, efficient and secure medical homecare solutions requires services, platforms and basic components that can be reused in several systems and thus lower the implementation effort. Efficient design of IT-based solutions depends on a rich selection of reusable components and services. Especially in medical scenarios with very high security and safety requirements, a collection of approved components for a reliable software platform is a key enabler of reduced time-to-market cycles.

The European ITEA2 OSAmI research project targets a common foundation of basic components for a basic, widely applicable service-oriented component platform. The OSAmI project involves partners from several European countries. The German subproject OSAmI-D develops a construction kit with a particular focus on medical and E-Health applications. The results of the ongoing OSAmI-D subproject are demonstrated in the scenario of home-based ergometer training during the rehabilitation of patients with cardiologic illnesses. An overview of the OSAmI-D scenario and the resulting OSAmI-D application is given in [1].

The OSAmI platform is composed of cross-domain 'horizontal' and domain-specific 'vertical' services, provided from various project partners, mostly under an open source licence. Section II describes the specific requirements of the medical training scenario and thus for the basic components. Then the OSAmI basic platform and its component is described from section III to VIII. This is not the full description of all components of the OSAmI-D application but a set of basic components that are candidates for the OSAmI component foundation. Finally a conclusion is given in section IX.

II. Requirements on Communication between different Users of the Platform

A. User in a E-Health Application

The stakeholder of a generic platform for E-Health applications are on the one hand IT-professionals, developing new applications by composing the developed vertical and horizontal basic components to fulfill the needs of reusable and stable components for a generic platform. On the other hand, this platform is used by health professionals who employ this for professional purpose and patients who are the beneficiary group of persons who had an illness or are elderly. These divers stakeholders and their relation to each other cause special requirements on communication and interaction in this generic platform. A physician has the role of a professional who is in charge and has the responsibility for the accurate setting of health related function. The patient relies on the physicians' experience and gives aware feedback through communication. Activities like exercise or daily behavior can also be a part of the interaction between patient and physician. Through changes in behavior or changes of vital signs, a patient gives important health related information. The application scenario and the described functional application

requirements raise further conceptual and technical requirements as explained in the remainder of this section.

B. Requirements on Communication and Interaction

The ability to communicate between patient and physician is a basic requirement. Many ways of interaction and communication between the two users can be distinguished. This can be a real-time communication (audio and/or video) between two users, a uni-directional message with operation instructions triggered by the physician in, e.g., taking medication. Also a uni-directional communication from the patient to a physician can be established in case of an emergency situation [2]. Furthermore, a uni-directional interaction can be build through health related documents and plans with behavioral rules like, e.g., a training plan or a diet plan. Generally, a generic platform for E-Health application must fulfill a mechanism to collect different kind of data from a patient. This can be a sensor based data from mobile devices like ECG. To achieve this requirements the generic platform must collect health related data from a patient to send them to the physician.

C. Medical Documents

To fulfill the needs of interaction between physician and patient the challenge of interoperability must be addressed. Interoperability is from prime importance because various types of vital signs and other health related data need to be gathered and interpreted across applications, e.g., in a clinic or ambulant [1]. On the physician side, the data must be human-readable, processable, ensure authenticity and should be saved for years due to legal restriction. General purpose formats such as Portable Document Format (PDF) supports content that is only available to a human reader and further data processing is not possible. Therefore medical documents should be able to contain structured information and data like ECG recordings. Standardised medical document formats like DICOM Structured Reporting [3] or the HL7 CDA [4] are possible solutions. The CDA format seems to be partially applicable. It is possible to store patient based health related data in XML and additionally three levels of semantical encoded content can be choosen. On level one the content is just human-readable and futher processing is not possible. On level two the sections with text can be enriched with a code system like LOINC or SNOMED. In level three every textual entry like systolic blood pressure can be semantically derived via code systems. Especially CDA on level three makes data processing possible. The disadvantage on CDA ist the overhead given by XML. Documents with many vital signs can have over 100 MB that makes a transmission complicate and handling with a webbrowser for readability impossible.

III. Basic Integration Platform

In the project OSAmI, the *OSGi Framework* serves as the basis for software developments. OSGi is specified by the *OSGi Alliance* as an open, modular, and scalable service delivery platform [5], which runs in a *Java Virtual Machine*, and offers an intra-JVM *service-oriented architecture* (SOA). OSGi provides a standardized way of managing the lifecycle of software components, the so-called *bundles*, running in an OSGi platform. Bundles can be installed, started, updated, stopped and uninstalled at the platform's runtime, without the need to restart the entire system. They offer their functionality in the form of *services* to other bundles by means of a publish-find-bind mechanism. Therefor, a *service registry* is used in which services are registered and can be found by potential service consumers. Furthermore, the OSGi framework supports the *eventing* concept, allowing to receive events from other services or the framework itself.

Originally, OSGi was developed to serve as a local SOA, thus being restricted to the boundaries of a single JVM, not intending interaction of different OSGi platforms. Hence, *WS4D-DOI* [6] was developed, which integrates transparently OSGi and DPWS [7] and vice versa, thus allowing the use of OSGi services located in a different OSGi platform according to the paradigm of *Distributed Object Systems* [8]. Furthermore, it integrates native DPWS services transparently into an OSGi platform, allowing the use of those out of bundles running in that platform. So OSGi's capability of software lifecycle management, combined with WS4D-DOI's capability to enable the interaction of different OSGi platforms and the integration of native DPWS services forms an ideal basis for the development of applications in the E-Health domain which require distributed software solutions as well as the integration of medical hardware, e.g., sensors for pulse or ECG measuring.

IV. Patient Monitoring

The patient monitoring is not implemented as component but as the actual OSAmI-D application on top of the OSAmI component platform. It uses and instruments several basic and several application specific components to implement the training scenario as described in [1]. Some but not all basic components used by this application are described in this paper.

V. Device Integration

A flexible and modern device integration is essential for the success of an E-Health system. Such systems do not have to only integrate medical devices but also a wide range of devices from other application domains like home automation, consumer electronics, etc. Thus E-Health systems have to cope with the complexity coming from the device integration. This complexity results from the increasing number of standards and products that are described briefly in the following paragraph.

A. Technology Jungle

Beside several proprietary solutions or solutions by huge industry consortia, IP based device technologies and infrastructures have been developed.

Main scope of the IEEE 802.15 WPAN Task Group 4, which has brought forth the IEEE 802.15.4 specifications, is low power, low cost, and low data rate wireless communication.

Based on 802.15.4 on link layer, the ZigBee Alliance has developed further network and application layer protocols. The application layer protocols are organized in clusters, which can be combined to build a complete application profile.

In turn and in addition to the existing classic Bluetooth specifications, the emerging Bluetooth Low Energy (BTLE) technology was developed. Low energy link layer are defined, working under the existing L2CAP layer. This allows application of dual mode architecture, consisting of parallel running standard Bluetooth and BTLE stacks in one circuit. BTLE revised drawbacks of classic Bluetooth like piconet architecture and thus limited subnet size. Additionally, a broadcast mode is described, which leads to new application scenarios because of the absence of required direct pairing. In contrast to classic Bluetooth application protocols and profiles, BTLE is capable of the lightweight attribute protocol and attribute profiles.

Both ZigBee and Bluetooth Low Energy are chosen by Continua Health Alliance to provide wireless connectivity. Nevertheless, neither ZigBee nor Bluetooth Low Energy is able to communicate directly with higher valued services in other networks without intermediate devices. They require application layer gateways to map payload data in IP based network protocols. Other existing and emerging technologies and architectures are developed and extended to be applied in networking device infrastructures without need for application layer gateways. In accordance to the IPv6 specification, IETF has established the 6LoWPAN working group. The focus of 6LoWPAN is to compress IPv6 headers to be sent on top of 802.15.4-based technologies. 6LoWPAN establishes the basis for TCP and UDP data transmissions in Wireless Sensor Networks.

This brief overview is not complete and many standards are missing like DASH7, ISA100.11a, EnOcean, ANT, Z-WAVE, and Wi-Fi Direct, to name few only. Every standard has its dedicated application scenario and niche. In conclusion, there is a need for a basic component that eases the integration of devices in applications in a technology independent and manufacturer independent way.

B. OSAmI Device Integration

The OSAmI Device Integration is the central component in the OSAmI platform to cope with the large number of standards and devices in modern E-Health systems. As the OSAmI platform bases on OSGi the OSAmI Device integration tries to fit into the OSGi framework as close as possible. OSGi itself provides a mechanism that addresses device integration. This mechanism called OSGi Device Access is specified in the OSGi compendium specification [9] and thus an optional component in OSGi frameworks. OSGi Device Access solves the problem of automatic matching and loading of drivers at runtime for hotplugging technologies like USB. Figure 1 describes the sequence that is triggered by plugging in a new device to the OSGi platform with implemented Device Access mechanism. The OSGi Device Access mechanism can be compared to traditional dynamic device driver registries as present in operating systems like Windows, Mac OSX

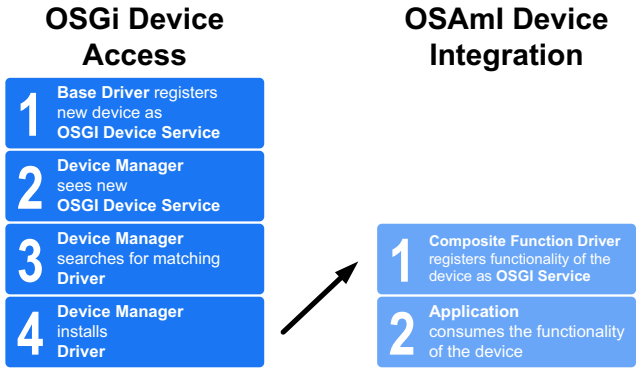

Fig. 1. Sequences in OSGi Device Access and OSAmI Device Integration

or Linux. But from the perspective of the OSAmI project, this mechanism does not decouple applications from device communication technologies and standards. As this is a main requirement for the OSAmI Platform there is a need for a more abstract device access mechanism.

This was a brief description why the mechanism provided by OSGi does not meet requirements of the OSAmI project and why there was a need for the OSAmI Device Integration component. The OSAmI Device Integration does not replace the OSGi Device Access but extends it.

OSGi Device Access defines several abstraction levels of device access and how this is represented in drivers. The specification defines low level device access to address low level features, high level device access to address protocol stacks and all levels in between. A typical networked device that is UPnP or DPWS capable could be accessed with several drivers on different levels like Ethernet, IP, HTTP, or the Application level protocol. There further device access levels are defined to address multifunction devices or gateway devices.

The OSAmI Device Integration basically defines a new level of device access inside the OSGi Device Access. The aim is to decouple applications from underlying device communication technologies, by applying principles derived from service-oriented architectures (SOA). These principles include loose coupling to the device access that can be achieved by using technology independent interfaces to access drivers. In general the OSAmI application does not care if the device is reachable by Bluetooth or wireless lan. The application wants to use the functionality of the device. This can be easily realized with the OSGi service mechanism. For example, applications need ECG functionality and use the ECG interface to access any device providing this functionality. Depending on the underlying technology, the interface can be provided by the driver or mapped to the service concept of the particular device communication technology.

At the moment the OSAmI Device Integration specification is implemented inside OSAmI project and consists of three main parts: Extensions to the OSGi Device Access specification, function level device access and design guides to design

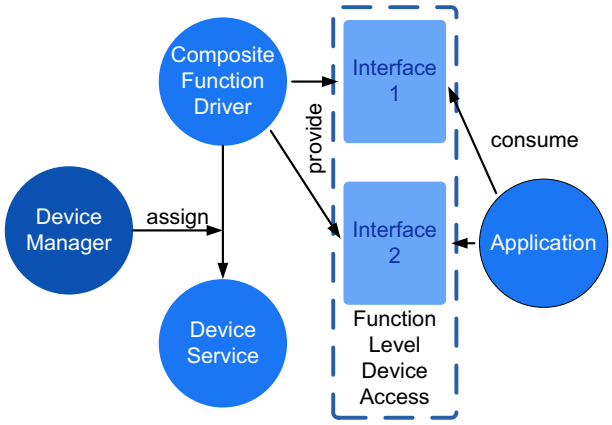

Fig. 2. Overview of function level device access

technology independent interfaces for function level device access.

The extensions to the OSGi Device Access consist of APIs to connect and discover devices. On the one hand there are technologies like RS-232 where the device integration must be triggered by applications as there is no way to detect devices. On the other hand there are modern technologies like UPnP, DPWS or USB that allows the enumeration or discovery of devices. In this case the device integration can be triggered by the application or device an

The second part of the OSAmI Device Integration defines the technology independent device access from the perspective of applications called function level device access (see figure 2). Therefor each function group of a device is represented as separate service in terms of SOA. Each of the service is represented as regular OSGi Service inside the OSAmI platform. If OSGi services represent similar functionality on devices, using different communication technologies, applications can use the same function on different devices. Thus the access to this function is independent of the underlying device communication technology.

This approach has two main advantages. The first advantage is the increased flexibility. Implementations of services and thus the devices can be exchanged without affecting the application. Therefore, application logic does not have to be adapted to new device communication technologies. The second main advantage is the seamless integration of modern device communication technologies that implement service oriented concepts. These technologies such as DPWS or UPnP can be integrated into the function level device access with a generic driver. This driver can integrate the functionality of the devices in a generic way without specific knowledge about the actual devices. Device communication technologies without service orientation require drivers that map the functionality of devices as services into the platform.

The third part of the OSAmI Device Integration consists of guidelines how to defines interfaces for the function level device access. As already mentioned before the function level device access bases on OSGi Services. OSGi Services are described with Java interfaces. The problem with Java interfaces is that they are very rich of features and not all features can be mapped to device communication technologies. So the OSAmI Device Integration specification defines a set of features common for device access. This includes simple, complex and custom data types, methods, exceptions, events, binary attachments, and data streams. This set of features was inspired by technologies like UPnP, DPWS and Jini. But this features can still be used with technologies like RS-232 or Bluetooth. In this case the missing features must be implemented in drivers. Of course a driver for a specific device only has to implement the features of the corresponding interfaces. So the OSAmI Device Integration addresses both technologies that offer service-oriented concepts themselves and technologies without such concepts. From the perspective of OSAmI applications, services on devices should only be represented by JAVA interfaces. The device driver has to implement these interfaces and communicate with the physical device.

VI. Management of E-Health Systems

Nowadays systems in the healthcare domain become quite complex and increasingly contain devices with limited resources. Furthermore they also have to adapt to the changing conditions and requirements, e.g., altering medical needs attributable to changing health state of a patient. Accordingly, a comprising and lightweighted management system is mandatory to provide those adaptions in a fast, but also predictable and reliable manner. This section introduces the management system, applied to the OSAmI software platform, on the basis of an exemplary application scenario from the healthcare domain.

The scenario demonstrates a cardiac patient who completed an ambulant rehabilitation program. He is supplied with the necessary equipment: an ergometer, an ECG, some sensors for pulse rate measurement and a so-called home gateway. The home gateway serves as an execution platform for software components to control the training and also connects the home equipment to the hospital servers, e.g., to perform a video conference for a live consultation.

Besides functional requirements from the medical domain, the scenario also involves several non-functional requirements. These are used to assure predefined quality criteria of the system, like costs, reliability or security. For instance, the sensible patient's personal data are handled, and therefore, protection objectives like confidentiality and integrity must be met. The management system ensures that both functional and non-functional requirements are taken into account. To provide this and to guarantee the requested lightweighted management, the concept of the model-based-management, as developed in [10], was adopted. This concept combines management policies and policy hierarchies [11] with a layered system model. The management policies build an additional control level above the actual programming code to "govern the

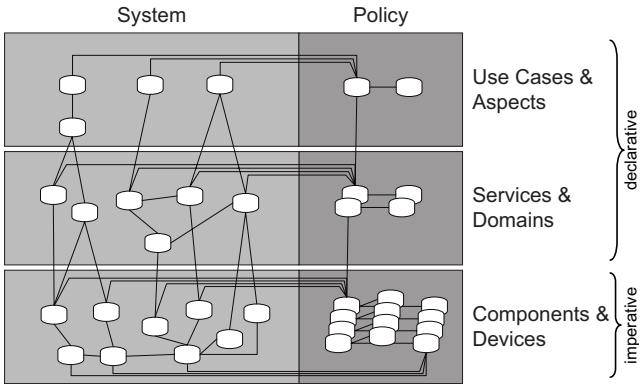

Fig. 3. Model Layers and Policy Refinement

choices in behaviour of a system" [12], thus performing adaption, correction and configuration functions.

The model-based management used is divided into two phases: the *design phase* and the *runtime phase*. In the design phase, the system to be managed is modeled on three layers with decreasing degree of abstraction. On the most abstract top layer, named "Use Cases & Aspects", technical details are backgrounded. Instead, existing use cases with their aspects relating to non-functional requirements and quality criteria of the system are modeled. On the middle layer, named "Services & Domains", the system is represented from a service-oriented point of view. This involves services, clients and dependency relations, assigned to so-called domains. On the lowest layer, named "Components & Devices", actual software components and devices existing at runtime are represented. A self-contained and independent model exists at each layer, whereas each model represents the complete system. Corresponding elements of adjacent models are associated using refinement relations. In order to be manageable, each component must declare a set of *management variables*: *status variables* to describe the management-relevant state of the component, and *configuration variables* to effect the behavior of a component. Furthermore, on each layer the policies applying to the model elements are attached. According to the approach of policy hierarchies, policies are also arranged in different degrees of abstraction. In the design phase, only abstract high-level policies on the upper levels are defined manually. Technical low-level policies are refined automatically based on the abstract policies, and the relations between the corresponding elements. For this refinement as well as for modeling the system, the tool *MoBaSec* (Model Based Service Configuration) (cf. [10]), developed in cooperation between TU Dortmund University and MATERNA, is used. The relation between the model layers and the refinement of declarative high-level into imperative low-level policies is depicted in Figure 3.

In the runtime phase, the management is realized by so-called *Component Managers*, implemented as OSGi bundles. A component manager provides some management services: a *configuration service* offering access to the management variables, a *policy service* providing policy-based decisions with respect to the overall system state, and a *binding service* allowing to establish and release interaction relationships (so called *bindings*) to other components. Each component is assigned to exactly one component manager, whereas a component manager might be responsible for more than one component. The allocation of the components to their manager as well as the assignment of the derived low-level policies, which are the only policies present in the runtime management system, is also planned in the design phase using MoBaSeC. In detail, low-level policies are distributed in form of efficiently executable Java byte code, and it can be distinguished between four policy types. *Policy Expressions* and *Policy Decisions* are terms defined on management variables, operations and constants, and are evaluated on demand of a component. A policy expression may have any return value, whereas a policy decision is restricted to return boolean value. Another type is the so-called *Policy Rule*, which represents an event-condition-action rule specifying actions to be performed when a certain event occurs, e.g., as a reaction on the change of some status variables. Finally, *Binding Requirements* are used to define non-functional requirements to be respected whenever a binding between two components should be established.

Figure 4 shows an excerpt of the healthcare scenario as modeled in the design phase. The left column shows the modeled system in the three layers of abstraction, the policies are depicted in the right column. According to the scenario, *Ergometer Training* as the main use case is modeled on the top "Use Cases & Aspects" layer. It is conducted according to the asset *Training Plan*, involves the actor *Cardiac Patient*, and makes for example use of the function *Ergometer Controlling*. Non-functional requirements for the self-contained model on this layer include aspects of *Availability* and *Security*, e.g., high integrity and high confidentiality. On the following "Services & Domains" layer, all services and client applications, that are necessary to provide the required functions of the layer above, are modeled. For instance, the training controlling function is provided by a *Training Control Application*, which itself needs a *Pulse Rate Analysis Service* and a *Ergometer Control Service* to govern the training. To fulfill security requirements regarding confidentiality the Authentication Service is essential as well. In this context, the abstract security aspects of the top layer have been refined to several *Security Service Level Objectives*, e.g., the provision to use credentials, or to use AES with a key of 256 bit for encryption. Furthermore, the management-specific services for performing *Binding*, *Configuration*, and *Policy* operations are represented at this model layer. Finally, the bottom layer "Components & Devices" contains all actual software components, resources, and devices existing at runtime. For instance, the *Ergometer Bundle* software component providing the ergometer control service is modeled here, as well as the *Ergometer Device*. Because the *Home Gateway Device* serves as execution platform for the software components, all of these are set in relation, as well as all devices which are physically connected to the home gateway, e.g., the ergometer. Moreover, the service level objectives of the layer above have been refined to appropriate

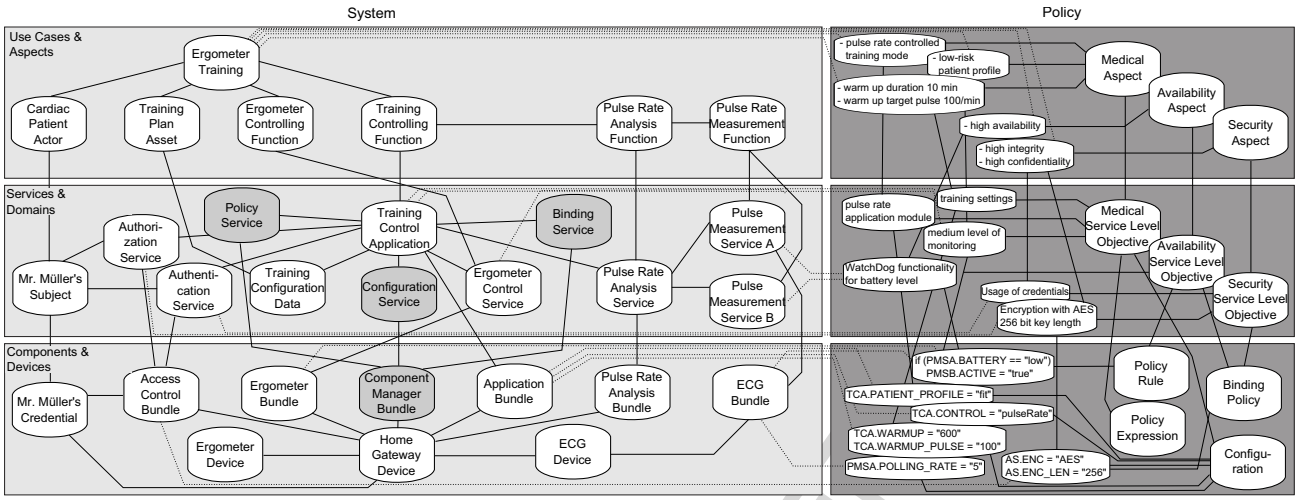

Fig. 4. Layered Model of the Application Example

technical low-level policies, e.g. to *Policy Rules*, or to *Binding Policies*. For instance, the security objective defining that AES with a 256 bit key length must be used has been refined to a binding policy which assigns the appropriate configuration variables.

VII. Autonomous Management

A. General concepts and design

The task of autonomous management (AM) is the detection or prediction of faulty system behaviour and its correction – ideally without necessary interaction from a human.

In this article the focus for AM is set on the service platform. The general operation cycle of AM applies here too:

1) Detect a management incident.
2) Diagnose system under consideration until the causing problem space of the incident is concise enough for a suitable autonomous handling.
3) Perform the ”treatment”[1].

The generic service platform sets specific constraints for AM partly due to the medical scenario and the actual service configuration. However, the autonomous management service is being designed as a configurable modular system which can be adapted to various scenarios.

The *detection of management incidents* is based on the evaluation of events. These events are named context events, and the component receiving, storing and distributing these events is the context store. The real difference to existing event frameworks is that events are persistent, thus a context event handler can consult not only one event, but also the trail of past context events in the system. If this information is pertinent to a management incident the AM core can be notified with the set of identifying information. With the history of selected context events proactive system management is viable by observing trends in context events. For example the discharge of storage batteries can be monitored in order to give an in-time alert for replacement to recharge, or the end-of-life of a component can be detected and a timely maintenance can be initiated.

The *diagnosis* is a – potentially iterative – step gathering additional information necessary for determining the correct management action.

After this phase the AM core executes the stored management actions on the system. For the current platform the proven flexibility of rule-based systems are considered.

Because of the connected nature of the systems the AM functionality can be distributed if necessary allowing to put complex management tasks on powerful back-end systems and only retaining basic ”first aid” measures on the system. This will be achieved by tapping into the distributed OSGi features.

B. Implementation details

The overall implementation follows the component view depicted in figure 5.

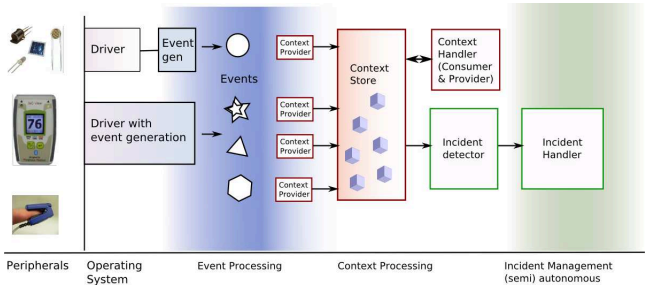

Fig. 5. OSAMI management components overview

As the platform is OSGi based the architecture exploits the service-orientation features for a modular and extensible solution. The dynamic installation and de-installation of service bundles in combination with the rich collection of OSGi services [5], [9] allow an on-demand ”reloading” of functionality and remote administration of the installed services. These

[1]the case of a not existing treatment usually routes the incident to a human specialist

features are used for temporary installation and use diagnostic routines or the installation of necessary updates or corrective bundles.

Management of a service is not limited to installation and complete replacement of the service. Usually a service has certain *service management variables* and *functions* made accessible through a *management interface* which allows to adapt the behaviour of a service. This is the starting point for the management routines of the AM.

In the current platform most context events will be provided using the policy management as event source. The problem of getting the right context events still remains in case the service provides no means to notify the management by design. In order to observe such services the programming paradigm of "aspect-orientation"[13] can be used. Using aspects the change of parameters and values and the call of methods can be intercepted and the necessary event context information extracted for the autonomous management. Additionally management routines could modify or even block service calls if necessary. Of course, it is crucial to the operation of any management system to understand the semantics of the managed services and functions and how to integrate them into the appropriate management actions.

VIII. Security

Security aspects are of special importance in E-Health applications. For instance, access to security relevant functions from inside the programming code (e.g., a service to access medical settings for a patient) has to be restricted. In case of a distributed application with sensitive data being transmitted over potential vulnerable channels, the communication needs to be secured. This is realized by means of the OSGi distribution solution WS4D-DOI [6], which's underlying DPWS implementation (JMEDS, [14]) allows to secure the communication using *Transport Layer Security* (TLS). This facilitates authentication of the participating communication partners as well as encrypting the actual communication. Due to the nature of the used distribution solution, this is transparent for the software developer. Necessary configurations for securing the communication are realized by the management applied to the system (cf. section VI).

To secure access to security relevant functions, a security concept regarding the special requirements resulting from the distribution of the OSGi platforms is developed. The use of existing OSGi security concepts (e.g., the OSGi Conditional Permission Admin [5]) was not possible, because these were not designed for distributed environments. For instance, they rely on the Java call stack, which is not present, or not present in the same context on a remote platform, respectively. The developed concept is based on the *Java Security Manager*, and the *Java Authentication and Authorization Service* (JAAS), and relies on granting or refusing permissions whenever security relevant programming code is about to be executed. It is implemented by means of an OSGi bundle, which performs some necessary adjustments to the Java Security Manager when it is started. Furthermore, it provides a service to authenticate users, for instance based on user name and password. In case of a successful authentication, the service returns a *Subject* (`javax.security.auth.Subject`), which is used to represent the *role(s)* of the current user. The adjustments to the security manager are, that the implementation of the *Java Security Policy* (`java.security.Policy`), which is the basis for permission checks performed by the security manager, is exchanged. The replacement security policy is modified in that way that it grants (or refuses) permissions for the execution of Java programming code encapsulated in a *Privileged Action* (`java.security.PrivilegedAction`) carried out on behalf of a subject, as it is returned by the authentication service. Hence, the Java programming code has to be developed *security aware*: parts of the code realizing security relevant functions must be encapsulated in privileged actions in order to ensure, that every time this code is about to be executed the permissions for this execution are checked. If an required permission is refused, the encapsulated code is not executed. Instead, an *Access Control Exception* (`java.security.AccessControlException`) occurs.

In the case of a remote service call, the subject on which's behalf this call should be performed is serialized and transferred to the remote platform. There, it is deserialized, and is used as the basis for granting permissions for the execution of security relevant programming code resulting from the remote call. The process of serialization and deserialization is performed by the OSGi distribution solution, thus being transparent for the software developer. Due to the fact, that the remote communication is secured as described before, transferring those subjects does not constitute a risk.

As mentioned, the subjects used as basis for granting permissions represent the role(s) of an authenticated user. This allows to define the permissions for each role. Each user is associated to one or more roles, therefore the permissions for all these roles are granted for this user (cf. *RBAC* [15]). This is a simplification especially in the E-Health domain, in which permissions to be granted are often structured in a hierarchical way. For instance, a nurse may have the permissions to view the medication settings of a patient, but may not have the permissions to change this settings. A physician of course needs to have the permissions to view and to change this settings. So users being physicians can be associated to the role "medical settings reader" as well as to the role "medical settings writer", whereas users being nurses are only associated to the role "medical settings reader". This simplifies the administration of the permissions granted for each user.

IX. Conclusion and future work

The complexity of modern E-Health systems requires platforms and basic components that can be reused and thus lower the implementation effort. In this paper the approach of the European ITEA2 OSAmI project and German OSAmI-D subproject to create a E-Health system based on components from a common foundation is described. The paper gives an overview of the requirements of the stakeholders of an E-Health system and describes basic components that can

be found in many E-Health systems. These components are developed for the common foundation of reusable components of the OSAmI Project and will be available as open source building blocks that can be reused in E-Health systems or even other application domains. This foundation should reduce the cost of development of E-Health systems and even advocate the convergence of computer systems from different application domains.

ACKNOWLEDGMENT

This work has been funded by German Federal Ministry of Education and Research (BMBF) under reference number 01IS08003.

REFERENCES

[1] M. Lipprandt, M. Eichelberg, W. Thronicke, J. Krüger, I. Drüke, D. Willemsen, C. Busch, C. Fiehe, E. Zeeb, and A. Hein, "OSAMI-D: An open service platform for healthcare monitoring applications," in *Proc. 2nd Conference on Human System Interaction HSI '09*, 2009, pp. 139–145.

[2] A. Helmer, M. Eichelberg, M. Meis, M. Gietzelt, O. Wilken, and A. Hein, "System zur eskalierenden Notruf- und Informationsweiterleitung im häuslichen Umfeld älterer Menschen," January 2010.

[3] D. A. Clunie, *DICOM Structured Reporting*. PixelMed Publishing, Bangor PA, 2000.

[4] R. H. Dolin, L. Alschuler, S. Boyer, C. Beebe, F. M. Behlen, P. V. Biron, and A. Shabo (Shvo), "HL7 Clinical Document Architecture, Release 2," *J Am Med Inform Assoc*, vol. 13, no. 1, pp. 30–39, 2006. [Online]. Available: http://www.jamia.org/cgi/content/abstract/13/1/30

[5] OSGi Alliance, *OSGi Alliance: OSGi Service Platform, Core Specification, Release 4, Version 4.2*. OSGi Alliance, 2009.

[6] C. Fiehe, A. Litvina, I. Lück, O. Dohndorf, J. Kattwinkel, F.-J. Stewing, J. Krüger, and H. Krumm, "Location-Transparent Integration of Distributed OSGi Frameworks and Web Services," in *Proceedings of the IEEE 23rd International Conference on Advanced Information Networking and Applications (AINA 2009)*. Bradford, UK: IEEE Computer Society, 2009, pp. 464–469.

[7] S. Chan *et al.*, "Devices Profile for Web Services (DPWS) Specification," 2006.

[8] A. S. Tanenbaum *et al.*, *Distributed Systems: Principles and Paradigms*. Prentice Hall PTR, 2001.

[9] OSGi Alliance, *OSGi Alliance: OSGi Service Platform, Service Compendium, Release 4, Version 4.2*. OSGi Alliance, 2009.

[10] S. Illner, H. Krumm, I. Lück *et al.*, "Model-based Management of Embedded Service Systems – An AppliedApproach," in *Proc. of the 20th Int. Conf. on Advanced Information Networking and Applications (AINA '06)*. IEEE Computer Society, 2006, pp. 519–523.

[11] R. Wies, "Policies in Network and System Management – Formal Definition and Architecture," *Journal of Network and System Management*, vol. 2, no. 1, pp. 63–83, 1994.

[12] N. Dulay, E. Lupu, M. Sloman, and N. Damianou, "A Policy Deployment Model for the Ponder Language," in *Proc. IEEE/IFIP International Symposium on Integrated Network Management (IM'2001)*, Seattle, May 2001, pp. 14–18.

[13] R. E. Filman, T. Elrad, S. Clarke, and M. Aksit, *Aspect-Oriented Software Development*. Addison-Wesley Professional, 2004.

[14] *WS4D JMEDS-Stack*, WS4D Initiative, http://www.ws4d.org/?page_id=14.

[15] National Institute of Standards and Technology, "Role Based Access Control and Role Based Security," http://csrc.nist.gov/groups/SNS/rbac/.