Component-based Development of Web-enabled eHome services

Michael Kirchhof and Sebastian Linz

Department of Computer Science III, Aachen University of Technology,
Ahornstr. 55, 52074 Aachen, Germany,
{kirchhof|linz}@i3.informatik.rwth-aachen.de

Abstract. The scope of Web-enabled eHome services covers both automated homes and automated industry facilities. Web-enabled eHome services provide a fully integrated view onto distributed systems comprising automated homes, back-end systems of providers, communication protocols, and services which make distribution aspects transparent. Future platforms should make the development and deployment as easy as achievable. Access should be possible by the way of all communication devices (e.g. desktop computers, PDAs, mobile phones) and all communication networks. Also, all appliances, ubiquitous devices, and their networking protocols have to be supported.
Generations of Web-enabled eHome services have been developed based on proprietary hard- and software. Today, an extensible and modular platform is required for forward-looking design and implementation of such services. One of the main requirements is, that the developed system is maintenance-free and the system brings itself in an operable condition. For setup tasks, both the end-user and a remote operator should be able to execute necessary steps.
It can be observed that services build up hierarchies. We propose a 3-layer system structure, which can be taken to account in system design. Software components grouped by service layers can then be realized in order to implement concrete services. Based on the OSGi platform, we have developed sample services. Gained experience is used for verification of our assumptions. Summarizing, we propose a cookbook for convenient development and deployment of services of the described nature.

1 Introduction

In this paper we will take a look at the interior of connected homes, which build up complex IT systems. The building blocks of such systems are electronic devices, networks, and *services*, which empower the user to interact with his environment. Objects in the environment can be the room itself, the building, other persons, and information from outside. Talking about services, we mean any piece of software, which is executed in a *networked* environment, making usage and administration of pervasive appliances easier.

The scenario is illustrated in figure 1: The connected home on the right-hand side of the drawing is equipped with a *residential gateway*, a hardware device, which provides access to connecting infrastructure (e.g., X.10, EHS, Lon, Jini) and acts as runtime

environment for the *service gateway*. The service gateway manages and runs certain software components. These services realize *Web-enabled eHome services*, which are these visible to the users. In our work, we focus on the domains of Security, Consumption, and Infotainment. The services in these domains are based on certain equipment, as some examples are shown in the figure: an alarm system depends on cameras, motion detectors, and lamps or sirens. Monitoring and optimization of energy consumption can be realized by the use of ammeters, photo sensors, thermometers, the heating systems and so on. Elements of the infotainment domain can be incorporated for audio-based and video-based interaction. The communication backbone is an IP-based platform, including a distributed extension. With this extension, internal and external communication can be handled equivalently. Beside direct interaction with devices in the house, interaction with the eHome system based on personal computers, PDAs, and mobile phones is realized. Service providers are connected to the systems via the distributed IP-based platform to provide digital content, applications, and services.

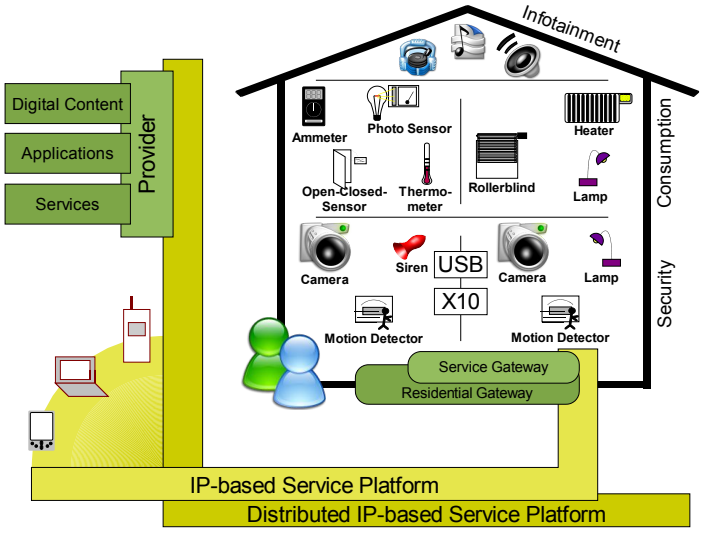

Fig. 1. Scenario

The extension of eHome systems to be connected to the outside world, is crucial for reasonable solutions. It has been shown, that many services profit from knowledge base systems that are maintained by service providers and offer immaterial goods [1]. These providers do not get directly into contact with customers. They merely communicate with one dedicated company in the role of the one and only service provider. Other companies interact and provide services, management data, and specific databases. They act as a virtual enterprise [2]. This stands in contrast to many other solutions: Though they provide network access, the supported functionalities is most often restricted to remote controlling single devices.

In this scenario, we make use of several technologies: First, we use communication networks for remote access. Second, we use a gateway to access devices *across* protocol boundaries, as we think of many devices working together in a networked environment. One often mentioned problem is the existence of many established home automation standards. While several papers address this topic, we use a standardized gateway as the nerve center of our solution, namely the standard defined by the Open Service Gateway Initiative (OSGI, [3]). The specification introduces a component-oriented approach, which becomes manifest in bundles. The usage of the open service gateway enables an abstract, almost protocol-independent view onto the different home automation protocols used. We are more interested in the mechanisms of *services*, rather than in the mechanisms of *serving*.

In this paper we will describe a new view on component-based development of Web-enabled eHome services. We target the following problem: There is an adequate framework for the development of state-of-the-art Web-enabled eHome services, but there is no knowledge about the system and service structure and its architecture in detail. We will propose an enhanced framework, which incorporates several established design ideas and show how this *cookbook* makes system architects' and developers' life easier.

At Department of Computer Science III at Aachen University of Technology, tools for supporting development processes have been built for software engineering [4], mechanical engineering, chemical engineering, process control, telecommunication systems, authoring support, and eHome systems. We derive tools automatically from specifications, using the PROGRES system [5] for specification development, a code generator for producing code out of the specification, and the UPGRADE visual framework environment [6] into which the code is embedded. The resulting tools are efficient demonstrators for proof of concept purposes. The basis for this work is our research on software architectures in the above application domains.

The paper is structured as follows: in section 2 we will discuss the general requirements for useful services that are accepted by customers and are therefore successful in market. Then related work is classified and the current state-of-the-art is shown. Section 4 gives an in-depth description of our approach. We will give insights of our service development, which is based on a project carried out within research and industry cooperation. The results of the different development cycles are discussed in section 5. In the last section, we summarize our results and give an outlook for future work.

2 General Requirements

Home automation promises new comfort and useful services in everyday life. These services will take place where today we find ubiquitous appliances. From users' point of view, services should be at least as easy in use. From developers' point of view, different *appliances and technologies* exist, which have to be integrated. This imposes specific requirements on the development of services in the area of home automation.

Home automation aims at mass markets. Therefore, technical background can not be expected from users. Mainly users will expect trivial *plug-and-play* from home automation appliances as they do from their legacy pendants. This leads to the demand

for *self-configuring* and *maintenance-free* systems, which seems to be an infeasible step. All details of a home, where each device is connected to each other, can not be specified in advance. So, some configuration and setup will always be a manual task done. On the other hand, home automation can not require technicians come to the user's home to integrate any kind of devices to home networks.

Talking about Web-enabled eHome services, we are talking about homes or facilities connected to external communication and data networks (at least temporarily). We recommend eHome services designed to be *remote-configurable*. This seems to be fairly more realistic, where configuration tasks can be left after installation. If a service does not install and configure itself automatically and a user can not do it on her own, he can call her service provider to remote access her eHome system and do necessary setup. Of course, this implies special mechanisms to keep Web-enabled eHome service's *security and integrity*. User should be able to determine and monitor who may virtually access her eHome system. Privacy of eHome must not be compromised by new comfort.

Connecting home area networks with communication and data networks provides potential for many service ideas. Up to now, service remote configuration is the most popular. Furthermore, user may access her eHome using different communication and data networks. Appliances can be controlled from any place (e.g., the office) using a browser and the Internet. The owner of a eHome may determine and change state of his alarm equipment with the Wireless Application Protocol (WAP [7]) and mobile communication networks using his mobile phone. Or in case of an alarm, the alarm equipment sends a multimedia message (e.g., MMS [8] or eMail) to the owner of an eHome, who will receive the message with his mobile phone wherever he is. But Web-enabled eHome services may also interact on their behalf with other data and communication network services making value-added services possible.

Summarizing, developing Web-enabled eHome services does not only mean to explore and realize consumers' needs and wishes. Realizing eHome services, special requirements have to be met. First, our eHome system should be connected with different communication and data networks. Second, as new needs arise by users, new appliances or new services emerge, our system should be extensible with services, respectively. Third, eHome appliances need a way to communicate. At present, home automation standards and technologies have been extremely fragmented. To cope with that, either, a standard has to be accepted, which encompasses all existing technologies, or we need a modular and extensible system to integrate existing technologies. The latter would facilitate developing applications, which use modules implementing existing protocols and appliances, which overcome the differences on syntactical and semantical level. Later on we will discuss this topic in more detail.

Decoupling of services from underlying infrastructure is based on abstraction from the infrastructure itself. A Web-enabled eHome service should rely on types of devices, instead of specific devices and their proprietary implementation. For example, an alarm system should be able to integrate any motion detector or device which is able to detect motion (e.g., cameras), instead of being tightly bound to a vendor-specific motion detector. Taking the back-end systems into account, a Web-enabled eHome service is not only a piece of software executed on the service gateway. A *Web-enabled eHome*

service is the *wholeness of services a user experiences*. The solution has to ease the realization of distributed systems, which do not impose any further burdens to users.

Following the approach of a modular and extensible system, we find useful concepts in *component-based software development*. Current component technologies rely on an infrastructure, which allows communication between registered components [9]. Applied to eHome service development, each component implements an appliance or network protocol. These components act as proxies, representing an appliance or a network protocol, respectively. This leads to the integration of existing and upcoming standards and technologies at a higher level, where appliances communicate by means of the component technology's infrastructure not using a specific protocol.

Beyond, component-based software development has advantages, which can be turned to account developing eHome services. We expect software engineering benefits and economic benefits from component-based software engineering mainly through software reuse. Previous paragraph described an example with a proxy, which would be a Web-server component. Other components could register their static or dynamic resources using the HTTP-proxy interface. Rigorous separation of interface and implementation and the non-existence of side-effects is a paradigm inherent in component technologies. Thus, components providing necessary implementation of an appliance or protocol from third-parties may be used reducing time-to-market of new eHome services.

Furthermore, new devices and protocols can easily be integrated. We can not figure out to which technology electric and electronic systems will converge. Most probably, actual situation will continue and different technologies and standards will coexist. But as new appliances or home automation technologies emerge, they may be incorporated using the approach of a modular and extensible system architecture. Either, a component has to be developed or a component can be used off-the-shelf, implementing the technology or appliance functionality, respectively. As new user needs and wishes will be explored, services satisfying these may be developed simply by *composing* existing service components.

Building eHome services, we use components' functionality through the interfaces provided. Any component captures and hides details of its knowledge about a special device or technology within its implementation. Thus, a component-based approach developing eHome services permits to concentrate on service logic, keeping details of used appliances or protocols unconsidered.

Based on the concept of residential gateway, the *open service platform* specified by OSGi [3] describes an approach, which supports extensible development of Web-enabled eHome services integrating existing and upcoming technologies. The open services gateway comes with an infrastructure that lets registered components retrieve other components and invoke their services. This infrastructure, sometimes referred to as a component model, is common among component infrastructure technologies (e.g., OMG CORBA [10], Sun EJB [11], Microsoft DCOM [12]). It builds the foundation for component-based software engineering. Thus, an approach based on the OSGi service platform enables component-based development of Web-enabled eHome services.

3 Related Work

A wide variety of different technologies competing for the emerging home networking market has been developed and evolved. While we are looking for a solution that enables communication of appliances and devices, to date, home networking technologies have been fragmented addressing particular application domains. Attempts exist to provide a generic framework that permits integration of different appliances and network protocols. We will give a brief overview in this section.

Different home networking technologies exist, e.g. Bluetooth, CEBus, EHS, LonWorks, HAVi [13–17]. They describe higher-level protocols built on top of a physical media and its lower-level protocol. Each addresses a specific domain. Using different physical media and protocols, it is obvious that devices may not inter-operate across technology boundaries. Thus, stand-alone solutions will not be successful.

Other technologies attempt to be more generic and independent of any physical media defining a common framework enabling all compliant devices to inter-operate. *Jini* [18] uses Java APIs to enable a device to join a Jini federation and propagate its service without any setup. Other devices in the Jini community can look up and discover the service. *UPnP* [19] uses IP networks to let devices discover, utilize and control other devices. The services a device provides and how they are controlled is described by XML-encoded messages. *AutoHAN* [20] has much in common with UPnP. Through the use of an XML-based service registry, it is not bound to any one API. These approaches impose particular hardware requirements to its devices, to run software modules of specific technologies. Some devices can not satisfy these particular hardware requirements, why a concept of proxy objects is introduced. Each resource-constrained device is thereafter connected with a performant device that runs the proxy object.

The *CableHome* specification [21] proposes an extension to cable modems to enable the execution of services with respect to IP-based communication, security aspects, and quality of service (QoS). While the extension of legacy installations (e.g., cable modems, telephone systems) is reasonable, the efforts of the specification are in the field of IP routing and ensurance of QoS. The integration of device drivers and applications is not tackled. CableHome technology could be used as the communication backbone for the distributed IP-based platform.

The open services gateway initiative (OSGi) specifies a set of open-standard software application interfaces (APIs) for building open-service gateways on top of *residential gateways*. The residential gateway describes an universal appliance that interfaces with internal home networks and external communication and data networks. Similar to a data network gateway, a residential gateway is equipped with interfaces to different physical media and provides conversion between protocols. It represents a concept, which lets different networks communicate transparently. This leads to a wide support of various standardized technologies, as depicted in figure 2.

The OSGi gateway (see figure 3) resides within a Java runtime environment, which offers the well-known features of Java. The core component is specified as the *OSGi service framework*, which acts as a container for service implementations. This environment includes a Java runtime with life cycle management, persistent data storage, version management and a service registry. Services are Java objects implementing a concisely defined interface. Services are packaged within *bundles*, which register zero,

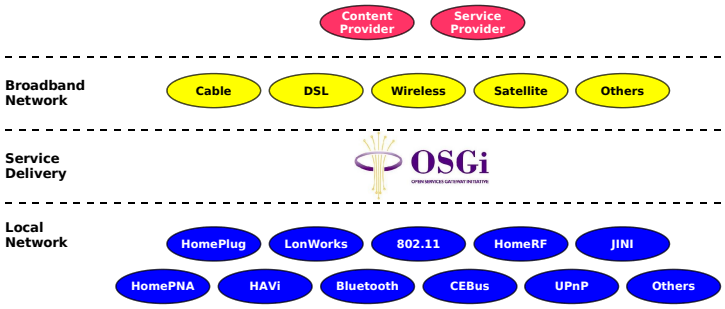

Fig. 2. OSGi and Related Standards

one or more services within the framework's service registry. Bundles contain services implemented in the Java programming language, a manifest file describing import and export aspects, and additional implementation-specific libraries. Bundles can be deployed and undeployed during runtime, while the information in the manifest file ensures the integrity of the system. Security aspects are handled as well. As the figure shows, a bundle is not restricted to rely on functionality offered by the framework, it can profit from every layer below, i.e. native functionality offered by the operating system and the hardware. This stands in contrast to layered approaches in software engineering, but allows the realization of bundles for any protocol. To ensure a minimal common set of functionality, certain bundles are standardized (e.g., the `Log` bundle for logging and the `HTTP` bundle for user interface purposes).

Central to the OSGi specification [3] is the *service framework* with a service registry. Components register their services at the service registry where other applications may retrieve and use them. Based on the concept of residential gateway, the open services gateway specification describes an approach that permits coexistence of and integration with multiple network and device access technologies. In addition, components may be added implementing new technologies as these emerge. Interaction is enabled via Java interfaces, without relying on proxy objects. While other systems -like Jini- are decentralized, the OSGi approach is a centralized system, which simplifies the maintenance of the system.

The specification provides a concise and consistent programming model for Java bundle development, simplifying the development and deployment of services by decoupling the service specification (i.e., the Java interface) from its implementation. Therefore, developers of applications and services only need to know about the services' interfaces. This approach is well known in component-based software development. By separating a component's interface from its implementation, we follow the goal of loosely coupled service components. Service implementations may be modified without taking effect on client applications as long as they obey to their interface specification.

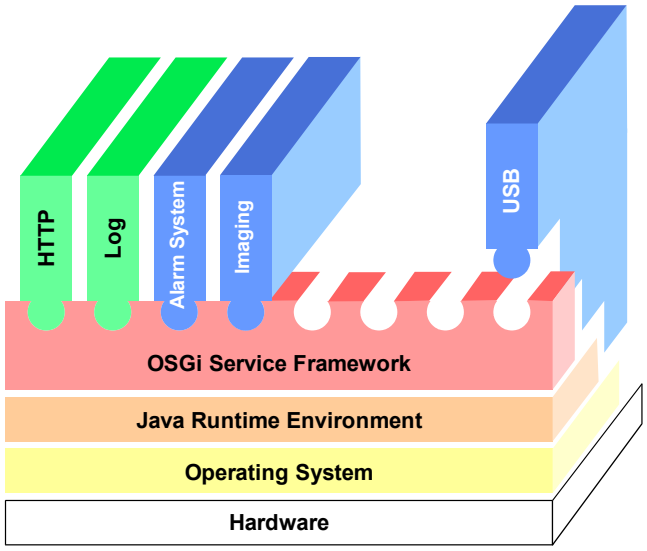

Fig. 3. OSGi system structure

As we have seen, to date, there is no adequate home automation standard, which meets our requirements (cp. sec. 2). Each proposed standard is focused on one application domain or dependent of a proprietary infrastructure. Not rarely, approaches impose certain hardware requirements on devices, which not always are able to meet that. Thus, an approach incorporating a central processing unit seems feasible. The features of the open service platform come quite close to our requirements. For this reason, we choose the open service gateway platform (OSGi) as a basis for our approach.

4 Basic Approach

This section deals with our proposed approach to overcome the posed problems. As stated, we use an OSGi-conform service gateway as a basis for our solution. This gives a quite complete component model of basic infrastructure services and makes an additional component-based development easy and naturally. Furthermore, the device access mechanism provided by OSGi allows effective usage of the service registry, because devices are represented by services, too, and are discovered and represented within the framework automatically.

Following the idea of OSGi, each and every Web-enabled eHome service resides within a bundle not crossing the bundle boundaries. The deployment is fail-safe, because each bundle comes with a manifest describing the dependencies and requirements, restrict to the direct neighbors (1-context). Also, the mechanisms of the service gateway can be used for deployment during operation. This does not interfere with executed services (services are hot-pluggable). After installation, new services are announced via the service gateway and its message subsystem.

Each bundle may provide its own configuration user interface (UI) utilizing the (specified) HTTP-service. Besides that, for configuration data get- and set-methods are provided in order to be direct accessible from other bundles and components. The configuration data is of course isolated within the bundle. A third way for configuration tasks can be realized: by implementing the interface ManagedService, bundles can be remotely configured by the module Configuration Admin, which is specified by OSGi, too. Configuration data are encapsulated in Property objects. Configuration Admin acts as a mediator. With that, basic configuration in terms of adjustments of properties can be realized.

Relevant for service and component development are two questions. Talking about component-based development, it is a design prerequisite to choose between (a) specialized components, which can not be used in other contexts, and (b) generic components, which can be used several times, but always with modifications necessary [22]. Applied to service development based on OSGi, the task is to strike the balance between insulation and sharing among bundles, which means that there is no reuse of basic components [23].

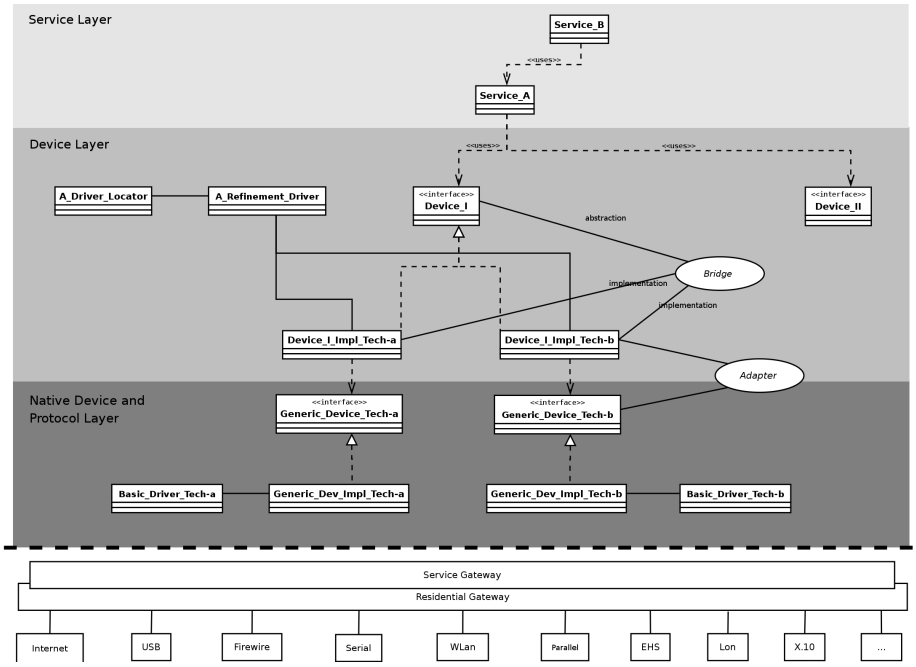

Fig. 4. Layered Service Architecture

In our dedicated approach for Web-enabled eHome service development, we propose a *3-layer system structure*, shown in figure 4. The lowest level comprises components mirroring functionality of native devices and network protocols. We named this

layer *native device and protocol layer*. The middle layer is called the *device layer*. There the representations of device and protocols as *services* reside. These components wrap details visible at native device and protocol layer. This is also known as an structural design pattern called *facade* [24]. From an outside point of view, we have now an abstract view on the basis, the device infrastructure. OSGi's device access technology is integrated into both lower and middle layer, enabling to reflect the dynamic aspects of the eHome system. Seamless integration of device access and protocol drivers provide means to reflects the *dynamic of the system*.

To meet the requirement of focusing on functionality instead of devices, we abstract from vendor-specific and device-specific implementations and used network infrastructure. We introduce interfaces for types of devices, which reflect a well-understood common functionality of devices. To incorporate existing technologies and devices, the specific interface has to be adjusted to the abstract interface. This enables the easy introduction of arbitrary devices and their implementations into the proposed system architecture.

The highest level provides room for sophisticated Web-enabled eHome services. These are the components developed by programmers, who now do not have to deal with device infrastructure details. They can concentrate on the abstract services, their functionality *and* requirements. Worth mentioning is, that services can not only be developed by turning lower levels to account, they can be build by composing high-level services, too. This brings developers in a quite comfortable position.

As an important design guideline a basic approach from component-based software engineering must be applied. Designing the architecture of eHome services must observe that dependencies between components may be described as a directed, acyclic graph. This enables services to be realized incrementally. Otherwise every part has to be realized to make it run. Beyond, the structure follows a logical design and becomes more easy to maintain.

In our approach, there is no need for generic bundles, which act as a container for traditional object-based realizations. Though there are approaches in this direction like Java Dynamic Management Kit [25], we think that this would be unnatural and makes the component-based basis nearly senseless. The separation of concerns is expressed by the bundle boundaries. Only basic services are shared among bundles, specialized tasks are captured within bundles.

The proposed layered system architecture describes a model for partitioning and integration of Web-enabled eHome services. With this model, a structure is introduced, which separates the application logic of Web-enabled eHome services from infrastructure details.

5 The Real World

Foregone we had a development cycle of Web-enabled eHome services based on a proprietary platform. Gathered experiences rose needs for a modular and extensible system. Therefore, extensibility had been identified as a must to integrate other home automation technologies and incorporate upcoming devices and services. Starting from the OSGi platform, the goal has been to gather experiences with a component-based ap-

proach redesigning services that had been developed based on the proprietary platform. Because of few experiences with component model, framework, and specified services of the OSGi service platform, a prototyping process has been chosen. The prototyping process has been applied to two development cycles. Within first development cycle, the service *PowerImage* has been developed. PowerImage is an imaging service, which provides access to imaging devices and offers additional functionality (e.g., an image archive). Within second development cycle *PowerSafe* has been developed. PowerSafe utilizes PowerImage, sensors (e.g., motion detectors), actors (e.g., lamps and sirens), and an eMail service to realize an alarm system. With two development cycles, information gathered from the first can be applied to the second. Possible design mistakes can be adjusted.

From external view, *PowerImage* and *PowerSafe* appear as two services. Following our basic approach, these two services consist of three components. Figure 5 illustrates the identified structure. The service *PowerImage* is described by a PowerImageDevice interface. This interface builds the facade to different picture grabbing devices, i.e. web-cams from different manufacturers and different device access technologies. There is no component PowerImage as might be expected. PowerImageDevice provides all functionality of the service *PowerImage*. The representation completes the service implementation.

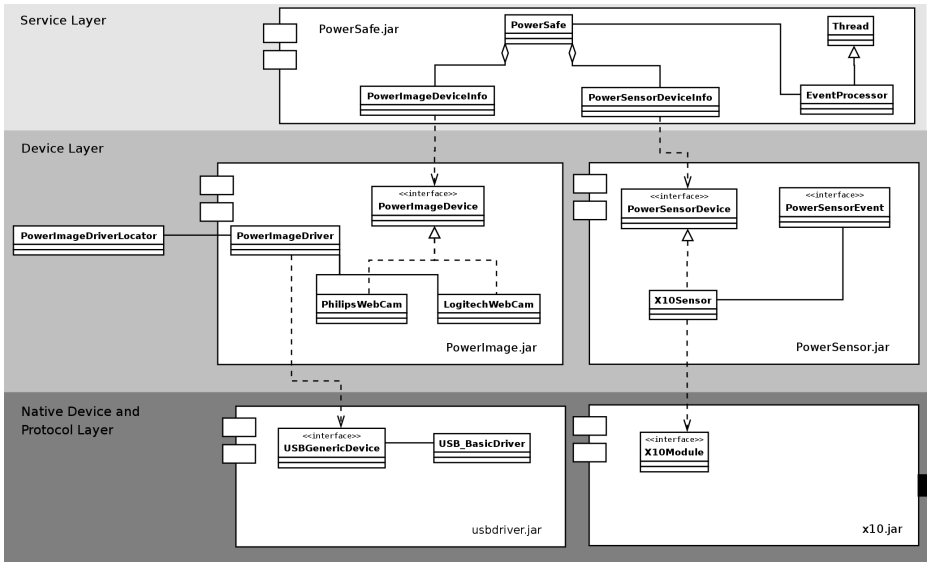

Fig. 5. Structure of PowerImage and PowerSafe according to Basic Approach

PowerSafe is the second component, that represents an alarm system. Different cameras can be connected to the alarm system. The component PowerSafe is placed at the service layer. It makes use of a third component called PowerSensorDevice. PowerSensorDevice is the second device facade at the device layer. It hides details of

underlying sensors (e.g., motion detectors, cullet sensors, and fire detectors). Different sensors may be connected to the alarm system and send notifications to it in case an event occurs.

The separation of PowerSafe and PowerSensorDevice reflects our approach of distinguishing device layer and service layer. PowerSafe represents an alarm system composed of different devices. In contrast, PowerSensorDevice represents a generic interface for different sensor devices. They may be used in other contexts than alarm systems, e.g. Service A (see figure 5) could be another service using PowerSensorDevice to switch on lights.

For development of PowerImage and PowerSafe a test bed was needed. USB web-cams have been chosen as PowerImageDevices. X.10 motion detectors have been chosen as PowerSensorDevices. A personal computer running Linux is used as residential gateway running an implementation of the OSGi service platform. The web-cams are directly connected at the USB-ports. X.10 devices are accessed through a X.10 module, that is plugged in a power socket and connected with the residential gateway's serial port.

For a deeper understanding of the three layers, we will describe them in detail in the next subsections.

5.1 Native Device and Protocol Layer

The introduction of device layers follows concepts of the OSGi platform. The OSGi *Device Access* specifies a common concept for management and automatic detection of devices in an OSGi environment. By the use of Device Access, a device service is automatically registered at or removed from the service registry as a device is attached to or disconnected from the residential gateway.

Base drivers provide the lowest-level representation of physical devices. Therefore, we associate them to the native layer. They detect if a device is connected to or removed from the residential gateway. Respectively, they register a generic device of the corresponding device access technology.

Device services registered by base drivers encapsulate implementation details of specific devices. They come with an API to access the functionality they provide.

Beyond device services, network services are located at the native device and protocol layer. Like device services, they provide an API hiding protocol details from their clients. For example, figure 5 shows USB as a device access technology and Lon, X.10, and EHS as home networking protocols at the native device and protocol layer.

5.2 Device Layer

OSGi Device Access specifies another category of drivers, the so called *refining drivers*. After a generic device service for a newly detected device has been registered, the OSGi environment searches for refining drivers. Each refining driver found is questioned, if it provides a refined service representation for the generic device. If the refining driver can match the generic device, it registers a refining device service in the OSGi environment. This refinement process is repeated until no other matching driver is found. To determine the process, a ranking is used to identify the best match.

Following the basic approach, refining drivers are located at the device layer. Base drivers are packaged within components of native device and protocol layer. Refining drivers are packaged together with interfaces and implementations they know from device layer (e.g., PowerImageDriver is bundled with the interface PowerImageDevice and the implementing class PhilipsWebCam).

We used a USB web-cam for development of *PowerImage*. Normally, a USB base driver is a component that could be used off-the-shelf. But no USB base driver bundle could be found that was applicable. Therefore, a USB base driver had been developed, wrapping an available Java USB API. This driver registers generic USB devices within the OSGi environment as new USB devices are plugged in at the residential gateway. After that, the refining drivers are invoked by Device Access, i.e. PowerImageDriver is invoked. This driver has knowledge about the class PhilipsWebCam, which is an implementation of the interface PowerImageDevice *and* at the same time is a refined view of a generic USB device. This way a PowerImageDevice is registered at the OSGi environment without requiring additional configuration.

The OSGi environment provides notifications to inform interested components about occurring environment events. OSGi specifies three kinds of events: FrameworkEvent, BundleEvent and ServiceEvent. The OSGi environment propagates ServiceEvents, when services are registered at and unregistered from the OSGi service registry and when service properties have changed. Base drivers remove the generic device, if the corresponding device is removed from the OSGi environment. Therefore, the event mechanism can be used to unregister refining devices, if the corresponding generic device is unregistered. PowerImageDriver listens for service events, so it can unregister PhilipsWebCam objects it registered before.

The component PowerSensor developed within the second development cycle uses X.10 motion detectors. We used an OSGi-based X.10 component that implements the X.10 protocol. X.10 devices do not provide a discovery mechanism. Therefore, we choose an alternative proposed by the OSGi specification: we developed a Web-frontend for management and configuration of X.10 sensors. Through the use of Web-based user interfaces, X.10 motion detectors can be registered at, configured for, and removed from the OSGi environment.

5.3 Service Layer

The service PowerSafe is located at the service layer. First, because it represents an abstract view of an alarm system and does not represent a device connected with the residential gateway. Second, it is a service build by composing components from device layer. With our realization of an alarm system, different PowerImageDevices and PowerSensorDevices can be connected.

In contrast to traditional library-based software environments, the OSGi service platform is rather a dynamic runtime environment [23]. During run-time, services used by other services may come and go. Especially, as PowerSafe is composed of services from device layer, these may appear and disappear during runtime, as devices are connected or disconnected. Therefore, the pattern *ServiceMonitor* [23] has been used and has been further extended.

ServiceMonitors encapsulate dynamics of an OSGi environment using the event mechanism. Clients within a component do not use services from other components directly. They use external services through a ServiceMonitor, which manages the corresponding service reference. Beyond, the utilization of ServiceMonitors has been extended to manage tasks within a component when specific events occur. For example, servlets generating the UI are automatically registered after the HTTP-service becomes available.

5.4 Lessons Learned

Basically, the OSGi service platform meets our basic requirements for development of Web-enabled eHome services. Beyond, its framework provides services useful for cooperating components. Developing PowerImage and PowerSafe, we gathered experiences with development of Web-enabled eHome services.

While some technologies provide a discovery mechanism, some do not. The latter does not allow to detect connected devices automatically. Thus, maintenance-free systems are rather unrealistic, if legacy home automation technologies should be integrated. We identified a feasible alternative by the use of browser-based HTML-user-interfaces. If user does not cope with setup, operators may access the system remotely and configure *PowerSensorsDevices* for example.

The *Configuration Admin Service* specified from OSGi describes how operators can set configuration data of deployed components. Configuration data are stored persistently so they can be retrieved after the OSGi runtime-environment is restarted. Unfortunately, it allows only character strings to be used. If necessary, this can be drawn aside by using Java serialization [26]. But the trade off between design advantages and an approved configuration admin service implementation has to be evaluated.

Regardless, the OSGi environment meets our requirements for development of Web-enabled eHome services. Beyond it, the OSGi service platform provides mechanisms that support our basic approach, while the layered architecture reaches software engineering goals.

6 Summary & Conclusion

We have shown, that by using our approach the required design efforts can be split into handy tasks. Once infrastructure questions are answered, the design and development process can go one step further. The abstract view of the component model and basic services gives not only room for free thinking, it contributes to the tasks of saving investments by making reliable, approved, and maintainable solutions possible. The described development of the services PowerImage and PowerSafe reaches the described goals.

As other approaches (e.g., Jini) identified the update and maintenance tasks as a severe problem, we do feel confident, that the proposed 3-layer system structure reflects reasonable levels of abstraction. These levels can be associated with different roles, e.g. the native device and protocol layer can be maintained by manufactures or organizations for standardization. Components at the device layer will usually be provided

by service gateway providers. Components at the service layer will be provided and marketed by service providers. Each layer has a clear-cut interface, so every component of a certain layer can be exchanged and new ones added without destroying the operable condition of the service gateway. The proposed architecture focuses on the a-priori design phase and the reegineering of Web-enabled eHome services. Third party services can be incorporated by the means of OSGi or the distributed IP-based service platform [27].

Thus, the described solution brings together the benefits of the OSGi open services gateway and of component-based software engineering. We proposed guidelines, how sophisticated systems of the described nature can be build. Following this idea, we expect that new abstract services can be composed by other service's implementations. Through the deployment features of the service gateway, installation and setup is nearly as easy as plugging in an appliance. We expect an eased realization of Web-enabled eHome services, which are (a) remote-accessible, (b) remote-configurable, (c) remote-maintainable, (d) safe and secure, (e) fail-safe, (f) affordable, and (g) incorporating existing devices and infrastructures.

Based on the current development, several areas of future work have been identified. First, basic services may be identified to provide designers and developers a complete set of instruments. It has be to clarified, which of them should be proposed as an enhancement of the OSGi specification. Second, there are security issues. Monitoring of access and associated evaluation, placement and extent of personal data and configuration data are topics of interest crucial for the success of such systems. Furthermore, concurrency and synchronization of data has to be researched. A more sophisticated approach for dealing with arbitrary data in such systems is currently being developed. Third, the dependencies between gateways and back-end systems have to be examined. Furthermore, it has to be evaluated, in which other domains the results gathered can be applied.

6.1 Acknowledgements

The work presented here is part of a project carried out at the Computer Science III group at the Aachen University of Technology, Germany. We are especially indebted to Mr. Peters (RWE AG, Essen, Germany) and Dr. Pritsch (Booz, Allen, and Hamilton, Düsseldorf, Germany) for their engagement in this project.

References

1. Becker, S., Kirchhof, M., Nagl, M., Schleicher, A.: EAI, Web und eBusiness: Echte Anwendungsintegration macht Aufwand! In: Proceedings of Online '02. Congress VI (2002) C630.01–C630.27
2. L. M. C. Matos, ed.: Collaborative Business Ecosystems and Virtual Enterprises . Volume 213 of IFIP International Federation for Information Processing. Kluwer Academic (2002)
3. Open Services Gateway Initiative: OSGi Service Platform. (http://www.osgi.org (13.11.2003))
4. Nagl, M., ed.: Building Tightly Integrated Software Development Environments: The IPSEN Approach. LNCS 1170. Springer, Heidelberg (1996) ISBN 3-540-61985-2.

5. Schürr, A.: Operationales Spezifizieren mit programmierten Graphersetzungssystemen. PhD thesis, RWTH Aachen (1991)
6. Böhlen, B., Jäger, D., Schleicher, A., Westfechtel, B.: UPGRADE: Building Interactive Tools for Visual Languages. In Callaos, N., Hernandez-Encinas, L., Yetim, F., eds.: Proceedings of the 6^{th} World Multiconference on Systemics, Cybernetics, and Informatics (SCI02). Volume I (Information Systems Development I)., Orlando, Florida, USA, IIIS (2002) 17–22
7. WAP-Forum: Wireless Application Protocol. (http://www.wapforum.org)
8. 3GPP: Multimedia Messaging Service, TS 22.140. http://www.3gpp.org (2002)
9. Brown, A.W., Wallmann, K.C.: The current state of CBSE. IEEE Software (1998) 37–46
10. Object Management Group, Inc.: The Common Object Request Broker: Architecture and Specification, Revision 2.6.1. (2002) http://www.omg.org (14.6.2002).
11. DeMichiel, L., Yalcinalp, L., Krishnan, S.: Enterprise Java Beans Specification, Version 2.0. (2001) Sun Microsystems, Inc.
12. Brown, N., Kindel, C.: Distributed Component Object Model Protocol DCOM. Microsoft Corporation (1998)
13. Bluetooth SIG, Inc.: Specification of the Bluetooth System. (2003) https://www.bluetooth.org/foundry/adopters/document/Bluetooth_Core_Specification_v1.2 (16.5.2004).
14. CEBus Industry Council: EIA 600 Specification. http://www.cebus.org/ (1996)
15. EHSA: Home Systems Specification (EHS). http://www.ehsa.org (2000) Release 1.3a.
16. Echelon Corporation: Cea-709.1-b: Control network protocol specification (2002)
17. HAVi Inc.: HAVi 1.1 Specification of the Home Audio/Video Interoperability Architecture (2001)
18. Waldo, J.: The Jini Architecture for Network-Centric Computing. Communications of the ACM **42** (1999) 76–82
19. UPnP Forum: UPnP Specification Documents. (2004) http://www.upnp.org/standardizeddcps/default.asp (16.5.2004).
20. Saif, U., Gordon, D., Greaves, D.J.: Internet Access to a Home Area Network. IEEE Internet Computing (2001) 54 – 63
21. Cable Television Laboratories, Inc.: CableHome 1.1 Specification. http://www.cablelabs.com/projects/cablehome/downloads/specs/CH-SP-CH1.1-I03-040129.pdf (2004)
22. Szyperski, C.: Component Software. 2 edn. Addison Wesley (2002) ISBN 0-201-74572-0.
23. Chen, K., Gong, L.: Programming Open Service Gateways with Java Embedded Server Technology. Sun Microsystems (2001)
24. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley (1995)
25. Sun Microsystems, Inc.: Java Dynamic Management Kit White Paper. http://www.sun.com/products-n-solutions/nep/software/java-dynamic/wp_jdmk40.pdf (2000)
26. Sun Microsystems, Inc.: Java Object Serialization Specification. http://java.sun.com (1999)
27. Kirchhof, M.: Distributed and Heterogeneous eHome Systems in Volatile Environments (2004)
28. Heineman, G.T., Councill, W.T.: Component-Based Software Engineering. Addison-Wesley (2001)
29. Gruhn, V., Thiel, A.: Komponentenmodelle DCOM, Javabeans, Enterprise Java Beans, CORBA. Addison-Wesley (2000) ISBN 3-827-31724-X.
30. Callaos, N., Hernandez-Encinas, L., Yetim, F., eds.: Proceedings of the 6^{th} World Multiconference on Systemics, Cybernetics, and Informatics (SCI02). Volume I (Information Systems Development I)., Orlando, Florida, USA, IIIS (2002)