In: Barbara Pernici and Jon Atle Gulla (ed.): Proceedings of the CAiSE'07 Workshops and Doctoral Consortium, Vol. 2, Ubiquitous Mobile Information and Collaboration Systems Workshop 2007 (UMICS 2007), Trondheim, p. 515-529, ISBN 978-82-519-2246-3. Tapir Academic Press, 2007.

Revised Version

# Frictionless Service Interaction in Protected Areas: Collaboration in eHomes

Ibrahim Armac, Michael Kirchhof, and Claus Richterich

Department of Computer Science 3 (Software Engineering)
RWTH Aachen University
Ahornstr. 55, 52074 Aachen, Germany
armac@i3.informatik.rwth-aachen.de
m.kirchhof@mm1-consulting.de
cluuse@i3.informatik.rwth-aachen.de

**Abstract.** eHomes are complex heterogeneous pervasive systems. Such systems consist of many diverse devices, their interconnection, and value-added services. The in-house interconnection is realized by a residential gateway. Usually, only one residential gateway is running in each eHome. This imposes problems in terms of service levels, connectivity, and availability. In this paper, we discuss the appropriateness of introducing multiple purpose-specific gateways in one eHome and present our OSGi-compliant solution that does not effect the implementation of services. This solution is called collaboration in eHomes and allows convenient development of eHome services which can communicate with other services transparently across different gateways.
We discuss interesting parts from the OSGi-specification in terms of life cycle management, service interaction, and connectivity with ad-hoc networks. Based on this, we derive our solution. The work is completed with discussions of dependencies in the component-based system, non-trivial parameter types, and distributed garbage collection.

## 1 Introduction

In this paper, we will take a look at the inside 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 executed in a *networked* environment, making usage and administration of pervasive appliances easier. These kinds of networked homes are called *eHomes*.

Usually, a *residential gateway*, a hardware platform, provides access to devices across protocol boundaries, as we think of many devices working together in a networked home environment. Running on the residential gateway, a service gateway acts as a runtime environment for eHome services. In our work, we use the standard defined by the Open Service Gateway Initiative (OSGI [1–3]). The specification introduces a component-oriented approach for service development.

In this paper, we discuss the problems of using only one residential gateway in an eHome. Thinking about many devices used in an eHome, the problem is that a pure centralized approach suffers from limited range of physical connections and limited ports of the residential gateway. Thus, the number and kinds of connectable devices is limited, which in turn are not sufficient for truly useful eHome services in pervasive home environments.

Based on this problem, we will introduce an approach to interlink two or more residential gateways in one eHome connected by a local network in an OSGi-conforming way. We call this kind of interaction between service gateways *collaboration*. This will enable eHome services to interact independently of their location. Because the current specification of OSGi does not handle the interaction of multiple service gateways in a local network, we developed a collaboration mechanism for OSGi gateways based on UPnP [4]. In doing so, we neither burdened the OSGi specification nor did we change the services' implementation.

In the following section, we do a detailed problem analysis before we describe our approach of the glue-less collaboration between service gateways. In the discussion, we show how to benefit from the OSGi specification and the concept of ad-hoc networks. After that, we give insights to important implementation aspects. Then, we discuss our approach in contrast to other approaches in the field of service provisioning in similar environments before concluding with a summary and an outlook to future work.

## 2 Problem Analysis

In this section, we give a detailed problem analysis. Before doing that, we describe first the area of eHomes. After that, we introduce OSGi briefly and describe the important aspects of service composition in OSGi. Finally, the problems of using multiple residential geteways in one eHome will be discussed.

### 2.1 The Structure of an eHome

The structure of eHome systems is illustrated in Figure 1. Today, the connected homes are usually equipped with one *residential gateway*, a hardware device which provides access to connecting infrastructure (e.g., X.10, EHS, EIB, Lon, Jini) and acts as a runtime environment for *service gateways*. In the figure, however, we already illustrate our vision of an eHome that is equipped with *multiple* residential gateways which are connected by a local network. The service gateway running on a residential gateway manages and runs the software components, which realize *eHome services* and are visible to the users [5]. The residential gateways can be assigned for example to the domains of Security, Consumption, and Infotainment. In our approach, there is one dedicated gateway, which is crucial for the functioning of the eHome. This residential gateway handles the communication with the outside world and is responsible for the services of the Security domain. Other residential gateways of diverse nature are located at other

places in the home, e.g. in the home office or the living room. Gateways of the second type do not need to be online always.

The services in the three domains are based on certain equipment, some of which are shown in the Figure 1: an alarm system depends on cameras, motion detectors, and 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. Even when the devices are connected to only one gateway, their functionality might be required by services running on other gateways. We give an example for that in the next paragraph. As a result of that requirement, the devices within the eHome should be available to all eHome services. Thus, there has to be a way to share functionality within the eHome among the different gateways. This will be achieved by the *collaboration* of service gateways, as mentioned in the introduction.

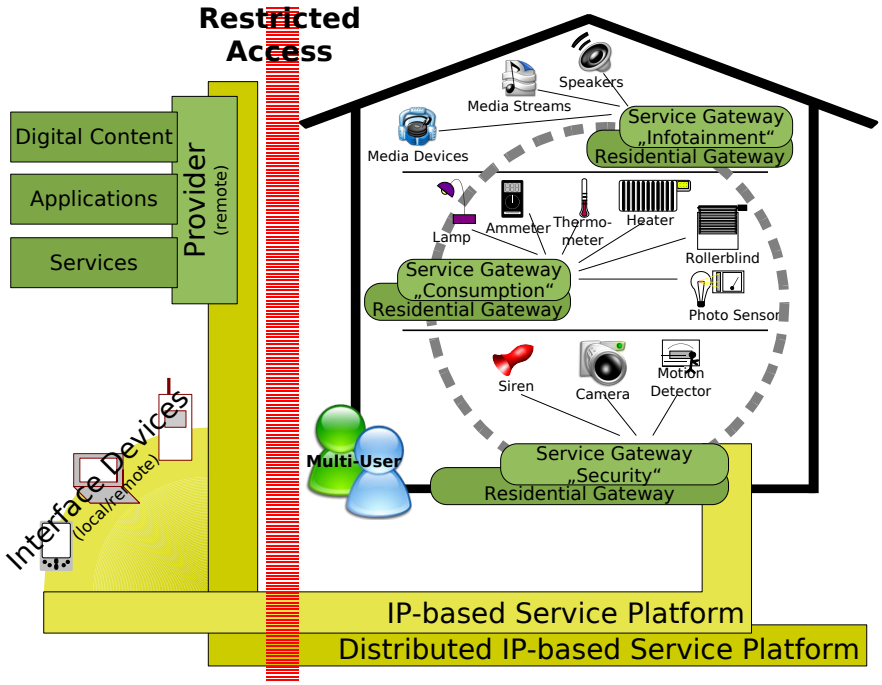


**Fig. 1.** Structure of eHome Systems

One example of an eHome service is a modular alarm system. The alarm system consists of cameras, sensors, an outbound connection for alarm messages (e.g., email facility, SMS), and some power switches. All devices required for its functionality are connected to the dedicated residential gateway. The procedure is

the following: when some sensors, for example motion detectors, detect something worth mentioning, a predefined subset of available switches are activated and selected cameras take pictures of the location and store them. The house-owner is informed in order to take adequate measures based on the kind of the event and the pictures obtained from within his house. Possible actions might be to call the police in emergency cases or to discard the event, such as when the cat raised the alarm accidentally. The provider conserves evidence which is crucial in case the local recording device is stolen, too. In addition, the provider can be authorized to assess the current situation on behalf of the house owner and to trigger police or security service, if necessary. The system is extensible by additional functionality. For example, the alarm system setup could be tweaked by the user in terms of which rooms should be monitored. Or, the way to alarm the owner is flexible.

There are also some extensions to the alarm system possible and reasonable that which us to the requirement of the collaboration of service gateways. For example would it be thinkable that some music-stream is played on the speakers, which are connected to the infotainment gateway as depicted in Figure 1. That way, the system can simulate the presence of inhabitants at home in case of an detected intrusion. The same is possible by scrolling the roller blinds, which are connected to the consumption gateway. Considering another service that is running on the consumption gateway with the objective of optimizing the energy consumption, we can give a further example for the gateway collaboration: The consumption optimizing service has also to observe the rooms and switch on or off the lights depending on the location of the inhabitants in the eHome. Thus, it needs to have access to the motion detectors connected to the security gateway.

### 2.2 OSGi Characteristics

One often mentioned problem is the existence of many established home automation standards, which results in a *heterogeneity* problem. We use an OSGi-compliant gateway as the nerve center of our solution. The usage of the open service gateway enables an abstract, almost protocol-independent view onto the different home automation protocols used. OSGi specifies a component-based framework for the service gateway. The usage of an OSGi-compliant gateway solves the heterogeneity problem. In the following paragraphs, we will give a brief description of the OSGi characteristics important for our collaboration approach.

OSGi components, called *bundles*, are Java-based and have a defined *life cycle*. Each bundle can implement one or more services. In this paper, we usually assume that each bundle implements only one service. The bundle life-cycle defined by OSGi is depicted in Figure 2. After a bundle is installed and the gateway has resolved all the imports of the bundle, it can be started. While the bundle is active, it can register/unregister services on/from the OSGi service registry. If a bundle is stopped, it reaches the state resolved from which it can be updated or uninstalled.

Each bundle is a jar-file with a defined structure. It consists of Java class-files, resources like HTML-files and the above-mentioned *manifest-file*. This

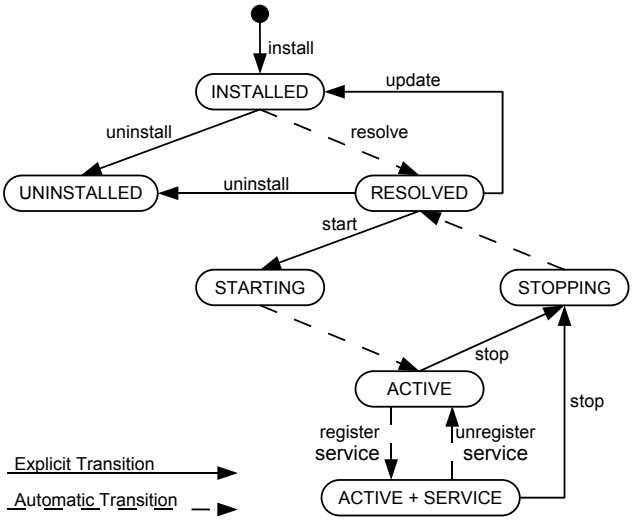


**Fig. 2.** OSGi Life Cycle for Bundles

file describes the bundle by a set of attributes. The attribute `export-package` declares the packages which the bundle exports to the service gateway. These exported packages are loaded to the gateway when installing the bundle and will remain there until the bundle is uninstalled. The attribute `import-package` declares the packages which the bundle needs to import. The attribute `bundle-activator` contains the full name of the class which will be instantiated first when the bundle is started. Each bundle has its own *bundle-classloader*, which is totally insulated from the bundle-classloaders of other bundles. The only possible way to exchange classes between bundles is to import and export packages via the service gateway. The information explicitly given by these attributes determine the scope of resources needed for *service composition* between different bundles and the starting point of a bundle's life cycle. Because the the manifest-file is created at bundle development time, the import and export of packages are statically determined [3]. As we will describe later, the attributes `export-package`, `import-package`, and `bundle-activator` are important for the collaboration of OSGi service gateways.

The OSGi specification defines the composition of bundles in terms of service interaction. Here, a *service* does not stand for an eHome service, as mentioned in the introduction, but for an instance of a class implementing a functionality. The OSGi service gateway manages a *service registry* to enable the interaction. We call a bundle providing a service to other bundles by registering it in the service registry with the name of the service and additional properties a *service-providing bundle*. As the counterpart, a *service-using bundle* looks up the required services in the service registry using criteria that matches to the properties of the registered services. It has to import the package which is exported by the service-providing bundle including the class that implements the required service

in order to use it. So, service interaction in OSGi depends on the package which includes the interface class of the service and the service itself. The availability of both the imported package and the service is determined by the current life cycle state of the service-providing bundle.

### 2.3 Problem Definition

As already described by the alarm system, distributed eHome services incorporate devices, and thus functionality, not only in close physical distance, but rather they depend on functionality which is available at some place within the eHome. However, today eHome solutions consider a central residential gateway to which all devices are connected. The problem is that a pure centralized approach suffers from limited range of physical connections and limited ports of the residential gateway. Thinking about a big eHome with many devices, it is obvious that on the one hand the wired connection of devices will not work for big distances for all technologies, such as USB. On the other hand, the different kinds of residential gateways are being build with different levels of capability. There are also gateways that are considered only for a small number of device connections. The advantage is that these specific gateways are both low priced and small-sized. Thus, they can be installed in deployed in multiple rooms within one eHome. This aspect of gateway interconnection can be called the *problem of connectivity*.

Another important issue is the field of application of one service gateway. Taking the different kinds of eHome services into account, a new dimension of heterogeneity comes into play. This heterogeneity is caused by different areas of services and their associated service level agreement. This leads to purpose-build service gateways. For example, the security service is expected to be very robust. To minimize side-effects and for prevention of attacks, the security service will be executed on a specially protected gateway, which is likely to be installed in an insular place. The installed software is certified for uninterruptible 24/7 operation. In contrast, the multimedia service is not security relevant and is not in all-day use. So, the requirements of the execution environment can be expected to be lower. In addition, the physical characteristics are crucial for good audio and video quality. This can be called the *problem of purpose-specificity*.

In our example, the alarm system resides on the protected service gateway. In order to benefit from other devices in the eHome, this gateway is not directly connected to the multimedia equipment, but it communicates with the multimedia gateway which offers alarm sounds to simulate activity in the eHome (to scare the intruder), and similar. In the same fashion, the lighting service is not directly connected to the isolated gateway. Rather, the gateway which hosts the lighting service is contacted in the case of an alarm. Thus, two types of gateways come into play: the first order gateways are crucial for service execution, the second order gateways can be included if they are present. The advantage is twofold: First, if one of these second order gateways fails to execute, the main service will not be burdened. Second, the complexity of the physical connection is lowered to a manageable level.

While the usage of an OSGi-compliant gateway solves the heterogeneity problem, it imposes a drawback in terms of distributed eHome services. The problem of gateway-spanning service composition, as it is required in our vision of multiple connected residential gateways in one eHome, is that the OSGi specification does not consider such a distributed service composition. Neither does it describe how to import packages of a remote bundle nor how to use a remote service. In the next section, we describe how we used the concept of an ad-hoc network to enable the composition of eHome services across multiple OSGi service gateways within a local network.

## 3 Approach

In this section, we describe our approach of collaboration between OSGi service gateways. First, we give a brief description of how we made gateway-spanning service composition possible and how we realized the necessary inter-gateway communication. Then, we give a short overview of how we put our approach into practice.

### 3.1 Collaboration between Service Gateways

To solve the above described *problem of connectivity* and at the same time to benefit from the OSGi specification, it is necessary to introduce more than one service gateway in one eHome. By doing this, the limited connectivity of one gateway can be overcome. The additional service gateways can be selected according to the purpose of the executed services, and thus this solves the *problem of purpose-specificity*. For example, there is one fail-safe service gateway in a utility area for the security service and the communication service, another service gateway in the living room for the multimedia service offering hi-fi audio and brilliant high-resolution video, and a third service gateway in the home office for print, scan, fax jobs and high-volume storage.

With this setup, a new problem arises: how can services interact arbitrarily without being burdened by distribution aspects? So, out of the solution of the first two problems a new problem arises, namely the *problem of a-posteriori, minimal-invasive, distributed computing*.

A new solution to this problem is necessary, because the current specification of OSGi does not handle the interaction of multiple service gateways in one home. To tackle this problem, neither the specification of service gateways should be burdened, nor the services' implementation. Our approach is to interlink two or more gateways in an OSGi-conforming way, such that the services can interact independently of their location. We call this kind of interaction between service gateways *collaboration*. It is achieved by using the life cycle of OSGi bundles and services, the known set of interaction dimensions in a component-based system, and an ad-hoc network. We will discuss these building blocks and show how we implemented the collaboration non-disturbingly. With our solution, services can arbitrarily interact within a community of OSGi service gateways transparently.

### 3.2 Putting the Idea into Life

Since the composition of services in OSGi is determined by the life cycle of a bundle, our concept of collaboration for enabling gateway-spanning service composition is based on the *imitation of the life cycle* of the service-providing bundle on remote gateways. Doing the imitation by a special bundle called *proxy-bundle*, which is installed on all remote gateways available in the local network, it is possible to resolve gateway-spanning *composition dependencies*. Because the proxy-bundle is also a bundle, our solution is OSGi-compliant. To make the proxy-bundle equivalent to the imitated bundle regarding the composition, the proxy-bundle consists of all exported packages of the imitated bundle, a special bundle-activator called *collaboration-activator*, and an adjusted manifest file. As exported packages of the imitated bundle are statically determined, it is possible to generate the proxy-bundle at install time or even at runtime. Because the proxy-bundle and the imitated bundle have the same exported packages, *package dependencies* can be resolved by the proxy-bundles so that remote service composition and interaction work correctly.

Each time the imitated bundle registers a new service on the local gateway, the collaboration-activator of the corresponding proxy-bundle registers an accordant *proxy-service* on the remote gateway that imitates the original service. As the proxy-service is registered with the same name and the same properties, it is indistinguishable from the imitated service. As a result, the *service dependencies* are also resolved by the proxy-bundle. All interactions with the proxy-services will be ascribed to the imitated services, respectively. Thus, bundles on the remote gateway can be composed using proxy-bundles instead of the original bundles.

For gateway-spanning composition of bundles, the gateways have to communicate with each other. Because the OSGi specification does not specify any direct communication between service gateways, we decided to use an ad-hoc network to realize the communication. The use of an ad-hoc network instead of using a client-server architecture has three advantages: First, an ad-hoc network needs less configuration. Second, it is possible to communicate simultaneously with multiple gateways. Third, OSGi specifies the integration of the two ad-hoc networks *UPnP* and *Jini* from which we decided to use UPnP for our purposes. In UPnP, each element in the UPnP network is made available for all participants of the network and removed if not existent any longer. The idea is to use these mechanisms for the publication of bundles and services over the network in an OSGi-compliant way.

We realized this concept by one single bundle called *PowerCollabo* that is runnable on all OSGi-compliant service gateways. As a result, the collaboration is not limited to one special implementation of the OSGi specification. Each time a new bundle is installed on the local gateway, PowerCollabo will generate a corresponding proxy-bundle and distribute it over the ad-hoc network. On the remote gateway, PowerCollabo fetches this proxy-bundle and installs it. Each time a new service is registered on the local gateway, PowerCollabo will distribute it also over the ad-hoc network. The corresponding proxy-bundle on the remote gateway will generate and register a proxy-service which imitates the

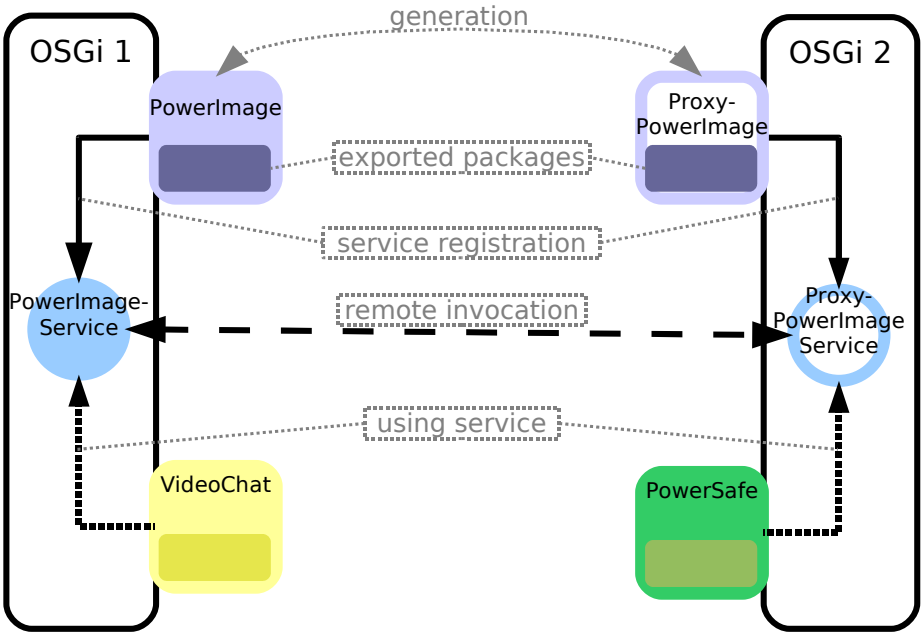


**Fig. 3.** Concept of collaboration

original service. Because the proxy-services are realized as Java *proxy-objects*, all interactions with them can be controlled by the proxy-bundle and are forwarded to the original services over the ad-hoc network. This solution is OSGi-compliant and the developers of eHome services do not need to care about the gateway-spanning composition because PowerCollabo will publish all services on all gateways available in the local network automatically.

Figure 3 illustrates the concept of collaboration by the example of the bundles PowerImage and PowerSafe, which are installed on different service gateways. For simplification, neither PowerCollabo nor any communication is shown. The PowerImage service provides access to video devices such as web cams. PowerSafe is the security bundle, whose service sends eHome pictures to the occupant's mobile phone or e-mail address, in case an alarm is activated. On the left of the figure, local service interaction is shown. The bundles PowerImage and VideoChat are both installed on OSGi 1. PowerImage has registered the PowerImage-service, which is used by VideoChat. In the middle and the right of the figure, the functioning of the collaboration is shown. Because PowerSafe is installed on OSGi 2, it is not able to use the PowerImage service without collaboration. If PowerImage is installed, PowerCollabo on OSGi 1 will generate Proxy-PowerImage, which will be installed by PowerCollabo on OSGi 2. If PowerImage registers its service, the corresponding Proxy-service will be also registered by Proxy-PowerImage on OSGi 2. Now, PowerSafe can use this service. If PowerSafe invokes a method

of the Proxy-service, this invocation will be transmitted by PowerCollabo to the PowerImage service on OSGi 1. Thus, powerCollabo enables PowerSafe to interact with PowerImage even though they are installed on different gateways.

## 4 Implementation using UPnP

In this section, we will give a deeper insight into the realization of PowerCollabo as our solution for the collaboration of OSGi service gateways. First, we explain how the in-house communication between multiple gateways is mapped to UPnP. Next, we describe in detail how PowerCollabo works. We complete this section by depicting some runtime issues, like the provided distributed garbage collector.

### 4.1 In-house Communication

As mentioned in the last section, the remote communication is done by an ad-hoc network. In our implementation, we use *UPnP*, which, however, can be replaced by any other ad-hoc network. Instead of sending messages, the communication is realized by *UPnP-devices*. Each bundle and each service is mapped to an UPnP-device with a special *device type* for bundles and services. Thus, the amount of UPnP-devices in the network represents all bundles and services on all gateways which participate in collaboration. By using UPnP, we can use a bundle implementation of the *UPnP-base-driver*, which is part of the OSGi specification and integrates each UPnP-device as a special service in the gateway. So, each UPnP-device representing a bundle or service will be integrated automatically into the gateway. As a result, the whole communication is reduced to service interactions.

### 4.2 Functioning of PowerCollabo

Responsible for the realization of the collaboration is the bundle PowerCollabo, which is divided in two areas: the *providing-area* and the *using-area*. The providing-area is required on the local gateway, whereas the using-area is required on the remote gateway. The providing-area listens to bundle and service events by registering a bundle- and a service-listener on the local gateway. So, it will be notified each time a bundle changes its state in its life-cycle or a service is registered or unregistered. The using-area on the remote gateway registers a service-listener on services which represent a UPnP-device so that it will be notified each time a UPnP-device is added to or removed from the network. In particular, it will be notified if an UPnP-device which represents a bundle or service is added or removed.

If a bundle changes its life cycle state, the providing-area will generate or destroy a UPnP-device and a proxy-bundle depending on the events causing the state changes. The proxy-bundle is accessible by the UPnP-device. This UPnP-device is automatically integrated at the remote gateway by the UPnP-base-driver, so that the using-area will be notified. Then, the using-area fetches the proxy-bundle and installs it.

As mentioned above, a proxy-bundle is generated on the local gateway by the providing-area for the bundle it imitates. All exported packages from the imitated bundle are copied into the proxy-bundle. Also, a customized version of the manifest file is packed into the proxy-bundle. The customization consists of replacing the value of `bundle-activator` with the full name of the collaboration-activator and appending the UPnP package name to the import-package value. If a proxy-bundle is installed by the using-area, it will register a service-listener on services which represent UPnP-devices.

Analogously, if a service is registered on the local gateway, the providing-area will generate a corresponding UPnP-device for this service. All methods of the service are accessible by this UPnP-device. The UPnP-device will be automatically integrated at the remote gateway by the UPnP-base-driver, so that the corresponding proxy-bundle will be notified adequately. After that, this proxy-bundle registers a proxy-service which imitates the original service.

Proxy-services are generated by a proxy-bundle, depending on the information fetched from the corresponding UPnP-device. This information includes the interface class name of the service that has to be imitated. The proxy-bundle creates a Java proxy-object from this interface class and registers it as an OSGi service. The invocation handler of this proxy-object forwards all method calls by UPnP-actions of the corresponding UPnP-device to the imitated service. The return values of these method calls are also forwarded by the corresponding UPnP-device. The actual method calls are done by Java reflection.

### 4.3 Runtime Issues

Interacting with remote services means in particular interaction with remote Java objects. The outcome of this are remote method calls, which need passing of objects from one gateway to another in the form of parameters or return values. But for all objects except the immutable objects, it is not possible to exchange them between gateways, because each object has a state which is mutable. If an object is passed from one gateway to another, there will be two equal objects on both gateways without any conjunction. So, if the state of one of these objects changes, both objects will not be equal anymore. Therefore, PowerCollabo does not exchange the actual objects between the gateways but exchanges *distributed-objects* which represent the actual objects. This representation consists of three properties: the unique `gateway id`, the unique `object id`, and the `name of the object class`. Each time a mutable object has to be passed from one gateway to another as a return value or parameter, PowerCollabo generates the accordant distributed-object and passes this to the remote gateway. On the remote gateway, PowerCollabo creates a Java proxy-object according to the class given by the distributed-object. The invocation handler of this proxy-object forwards all method calls by UPnP-actions to the original object. Therefore, PowerCollabo provides a UPnP-device which allows calling methods of objects that have been distributed using Java reflection.

PowerCollabo establishes a *distributed garbage collector* for distributed memory management over the gateways. The functioning of the distributed garbage

collector is illustrated in Figure 4. On the left side, the situation is shown in which the local gateway has passed an object to the remote gateway. The local PowerCollabo references the passed object by a strong reference, so this object will remain in the memory. The remote PowerCollabo has generated a corresponding proxy-object and references this by a *weak reference* [6]. As a result, this proxy-object will only remain in the memory as long as it is referenced by other objects via strong references. On the right side of the figure, a process sequence is shown which starts (1) if all strong references to the proxy-object have been deleted. Afterward, (2) the remote PowerCollabo deletes its weak reference to the proxy-object and (3) sends an event to the local PowerCollabo notifying that it does not require the object anymore. If no other remote PowerCollabo requires this object, (4) the local PowerCollabo will delete its strong reference to the object. Due to this, the garbage collector is able to remove this object from the memory.

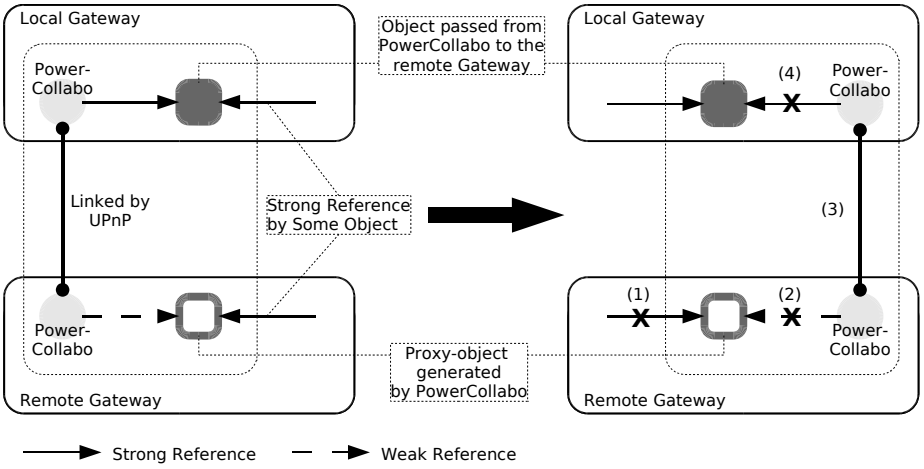


**Fig. 4.** Functioning of the distributed garbage collector

## 5 Related Work

In this section, we will give a review about related work which could be used for the interaction of multiple service gateways. First, we will discuss RMI and XML-RPC, which focus on remote method invocation. Next, we will describe an approach which is used for the realization of remote service interaction at a higher level as the aforementioned ones: JXTA. Finally, DSF will be described as an approach which can be used for remote protocol-independent service communication between heterogeneous platforms.

### 5.1 Remote Interaction Approaches

There are approaches which enable the remote interaction of services based on remote method invocation. One of them is the Java *Remote Method Invocation (RMI)* [7], which is part of the Java specification. Because OSGi is Java-based, RMI can also be used by OSGi services. RMI enables the interaction of remote Java objects by providing a client-server architecture. A similar client-server architecture is presented by *XML-Remote Procedure Calling (XML-RPC)* [8]. In this approach, the remote communication between services is XML-based and HTTP is used as the underlying protocol. Thus, XML-RPC is not Java-specific and the communication is synchronous and stateless.

In both cases (RMI and XML-RPC), the services are aware of the remote communication. This means that the service implementations must be explicitly modified so that RMI or XML-RPC can be used. This leads to an implementation effort which has to be made in addition to the implementation of the service logic. In contrast, our solution PowerCollabo is implicitly used, and the services do not need to be modified to enable the collaboration of OSGi-Gateways. Furthermore, the client-server approach requires extra configuration effort. So, the automatic mapping of the services' life cycle onto remote gateways, as in PowerCollabo, is not possible.

### 5.2 JXTA Approaches

*JXTA* [9] is a *peer-to-peer (P2P)* protocol which is independent of the underlying network and programming language. Peers are arranged in *peer groups*, whereas each peer can belong to multiple peer groups at the same time. Furthermore, peers can dynamically create virtual networks, so that any peer can interact with others in the same peer group independent of the networks the peers are on. Peers are connected by *pipes*, which are similar to UNIX-pipes. *Advertisements* are used by peers to offer and solicit JXTA-resources, which can be peers, peer groups, pipes, and advertisements. By default, JXTA is not integrated into OSGi. However, there are approaches that combine JXTA functionality with OSGi: For example *Jadabs* [10] is an OSGi-based runtime environment for smart devices. Nevertheless, the implementation is restricted to J2ME, which does not fully implement the OSGi specification. Another example for the combination of JXTA and OSGi is provided by the *P2PComp*-framework, whose aim is to "ease and support the development of pervasive computing applications based on spontaneous interaction of mobile peers" [11]. To enable the collaboration of remote OSGi service gateways, a Ports concept is introduced. A *Portsmanager* service is installed on every gateway to provide a transparent communication between client and server services. Therefore, each service may have *uses* and *provides* ports, which are communication endpoints between services.

Because JXTA is not integrated into OSGi by default, the client services must access the Portsmanager to obtain a reference to the required server service. Thus, this concept does not support an entirely OSGi-compliant service implementation, like PowerCollabo does. Furthermore, to allow dynamic class loading by remote

gateways in P2PComp, the OSGi service gateway must implement a proprietary interface. This is another modification of the OSGi standard. In PowerCollabo, exporting the classes in proxy bundles to the remote gateways is done without standard modification: Neither the bundles must care about the loading of remote classes, nor the gateways. Another mentionable difference between P2PComp and PowerCollabo is that PowerCollabo focuses on Collaboration of *in-house* gateways, that is gateways running in a local home network. In contrast, P2PComp focuses on *mobile* P2P applications where explicit collaboration may be reasonable for management purposes.

### 5.3 Distributed Services Framework

The aim of the *Distributed Services Framework (DSF)* [12] is to enable communication between services running on heterogeneous platforms like J2EE and OSGi. The introduced *connector-acceptor* mechanism allows services to dynamically select an appropriate communication protocol for each connection and overcomes, thus, the problems by integrating arbitrary communication protocols. Furthermore, DSF deals with problems related to non-permanent Internet connections between remote platforms.

DFS focuses on wider collaboration aspects whereas PowerCollabo's aim is to provide implicit collaboration for homogeneous OSGi service gateways in a local in-house network. So, the abstraction from a non-permanent Internet connection as well as from heterogeneous platform types (e.g., J2EE and OSGi) would produce an overhead for the goals presented in this paper. Furthermore, also DSF is based on a client-server model. This results in an overhead of configuration effort because each connection has to be configured separately.

## 6 Summary & Outlook

In this paper, we introduced PowerCollabo, an approach providing *a-posteriori minimal-invasive distributed computing* in OSGi-based eHome systems, which build up complex pervasive systems. PowerCollabo enables the interaction of eHome services running on different gateways, and thus solves the problems of *connectivity* and *purpose-specificity*. As a result, services can access remote services and devices not physically connected to the local gateway to provide more possibilities for service composition. In doing so, the collaboration of gateways is *standard-compliant*; neither the OSGi specification is burdened, nor the services' implementation.

The main intention of PowerCollabo is the *in-house collaboration* of multiple OSGi-based service gateways. Up to now, PowerCollabo provides a basic control mechanism for ad-hoc collaboration: Either a gateway can be part of the collaboration in the eHome, or not. By default, proxy bundles are installed on *all* available remote gateways in the *local* network. Future work could be seen in the management of the installation process of proxy bundles to save gateway resources and make services available only on required and authorized gateways.

This issue will also be of interest in the domain of *virtual eHomes*. Virtual eHomes extend the physical eHome by integration of different environments, like home, car, work place, hotel etc. Thus, the question of where to make which services available should be addressed. Up to now, PowerCollabo does not care about *security* and *privacy* issues because it focuses on in-house collaboration. By realizing the virtual eHomes concept, however, security and privacy have to be ensured. So, future work has to be done for *secure service migration in virtual eHome systems*.

## References


1. The OSGi Alliance: OSGi Service Platform Core Specification. `http://www.osgi.org/osgi_technology/download_specs.asp#Release4` (August 2005) Release 4.
2. Gong, L.: A Software Architecture for Open Service Gateways. IEEE Internet Computing **5**(1) (January 2001) 64–70
3. Chen, K., Gong, L.: Programming Open Service Gateways with Java Embedded Server Technology. Addison-Wesley Professional (2001)
4. UPnP-Forum: UPnP Device Architecture (June 2000)
5. Kirchhof, M., Linz, S.: Component-based Development of Web-enabled eHome Services. Personal and Ubiquitous Computing Journal **9**(5) (2005) 323–332
6. Simmons, R.: Hardcore Java. O'Reilly (March 2004)
7. Sun Microsystems, Inc.: Java Remote Method Invocation (RMI) Specification. Specification (2005) `http://java.sun.com/j2se/1.5.0/docs/guide/rmi/spec/rmiTOC.html` (10.08.2006).
8. Dave Winer: XML-RPC Specification (June 2003)
9. Sun Microsystems, Inc.: Project JXTA: An Open, Innovative Collaboration (April 2001)
10. Information and Communication Systems Research Group - ETH Zürich: Jadabs (June 2005)
11. Ferscha, A., Hechinger, M., Mayrhofer, R., Oberhauser, R.: A Light-Weight Component Model for Peer-to-Peer Applications. In: ICDCSW '04: Proceedings of the 24th International Conference on Distributed Computing Systems Workshops - W7: EC (ICDCSW'04), IEEE Computer Society (2004) 520–527
12. Kirchhof, M.: Distributed and Heterogeneous eHome Systems in Volatile Environments. In Weerawarana, S., ed.: Proceedings of Forum at $2^{nd}$ International Conference on Service Oriented Computing (ICSOC 2004). Number RA221 W0411-084 in IBM Research Report, IBM (2004) 123–131 Refereed Papers.