PROCEEDINGS

55. IWK

Internationales Wissenschaftliches Kolloquium

International Scientific Colloquium

13 - 17 September 2010

Crossing Borders within the ABC

Automation,

Biomedical Engineering and

Computer Science

Faculty of
Computer Science and Automation

www.tu-ilmenau.de

TECHNISCHE UNIVERSITÄT
ILMENAU

Home / Index:
http://www.db-thueringen.de/servlets/DocumentServlet?id=16739

Impressum
Published by

Publisher:	Rector of the Ilmenau University of Technology Univ.-Prof. Dr. rer. nat. habil. Dr. h. c. Prof. h. c. Peter Scharff
Editor:	Marketing Department (Phone: +49 3677 69-2520) Andrea Schneider (conferences@tu-ilmenau.de) Faculty of Computer Science and Automation (Phone: +49 3677 69-2860) Univ.-Prof. Dr.-Ing. habil. Jens Haueisen
Editorial Deadline:	20. August 2010
Implementation:	Ilmenau University of Technology Felix Böckelmann Philipp Schmidt

USB-Flash-Version.

Publishing House:	Verlag ISLE, Betriebsstätte des ISLE e.V. Werner-von-Siemens-Str. 16 98693 Ilmenau
Production:	CDA Datenträger Albrechts GmbH, 98529 Suhl/Albrechts
Order trough:	Marketing Department (+49 3677 69-2520) Andrea Schneider (conferences@tu-ilmenau.de)

ISBN: 978-3-938843-53-6 (USB-Flash Version)

Online-Version:

Publisher:	Universitätsbibliothek Ilmenau ilmedia Postfach 10 05 65 98684 Ilmenau

Home / Index:
http://www.db-thueringen.de/servlets/DocumentServlet?id=16739

A MODULAR SYSTEM FOR BUILDING AUTOMATION

Andreas Günther, Thomas Meier, Chris Richter

Hochschule für Telekommunikation, Leipzig
Gustav-Freytag Str. 43-45
04277 Leipzig

ABSTRACT

This paper will introduce a modular system for building and home automation that can by applied to a wide range of scenarios, for example, energy management and ambient assisted living. Because of the modular structure, our system can be used in a lot of different scenarios. For example, it is possible to implement a central as well as a decentral management of the system. Beside the modular structure, one of the project main goals was to develop a generic system to integrate sensors and actuators already in existence. Therefore, we have implemented a gateway to connect a wide range of proprietary devices from different manufacturers with our system independent of their specific communication interface (for example ZigBee, WLAN or USB). The task of the gateway is to translate these different technologies to a uniform interface (UPnP). This interface is used by the controller to interact with the sensors and actuators. The controller is responsible for the execution of the system logic, for example, a heating control system or an alarm system. The controller implements a web service interface to facilitate the configuration of the controller by a various number of different clients, such as Home Control Interface, Management Server or a mobile device.

1. MOTIVATION

More and more, the role of technologies and computers is growing to improve and simplify everyday life. Also in the scope of the home control there are already many individual solutions existing to automatize and simplify different tasks. For example, the central heating control module as it is present in nearly each home, solutions for the automatically light control, shutters or sunblinds can be controlled automatically. Even though there are already a lot of such single components to automatize the home control and also different manufacturers offer proprietary solutions, there is not solution at the moment which integrates all these components or the isolated applications into one system.

It is the aim of this project to create a system to combine different technologies of the home control. An important point is the use of open standards like UPnP [1] or Web Services which have a wide propagation in industry at the moment.

The advantage of the migration of different products from different manufacturers to one system is the simplification for the end user. Therefore, he can control all the elements of the home control in a transparent way with one system and add new elements simply per plug and play to the system. This induces the facilitation of the daily tasks.

Beside the termed criterions — the use of open standards and the migration of existing products and solutions — the project's focus is upon the conception and development of a scalable, inexpensive, error-tolerant and extensible solution. In particular, the scalability should facilitate a manifold use of the system and include various and also different use cases. On the one hand, the central management of several costumers as a service or the management of several company locations is conceivable, and on the other hand the management of a single household. For instance, derived use cases from this are the implementation of an alarm system or to automatically read off the meter reading like power, water or gas as a service. But also the configuration and optionally the monitoring of the own household with a mobile device should be possible.

This document is divided into four main parts. Section 2 gives an overview about the whole system including the particular components with a description of their tasks and functions. Subsequent to this, section 3 provides a short presentation of the used hard- and software-platform for the Gateway and Controller. The last two sections 4 and 5 give a detailed description of these two components.

2. SYSTEM DESCRIPTION

The basic concept for the whole system is a combination of central and peripheral control of operation sequences according to passed criterions and parameters. The overall picture as shown in figure 1 contains six important components of the whole system:

- Management Server,
- Gateway,

- Controller,
- Home Control Interface (HCI),
- clients,
- sensors and actuators.

Figure 1 shows the schematic structure of the system.

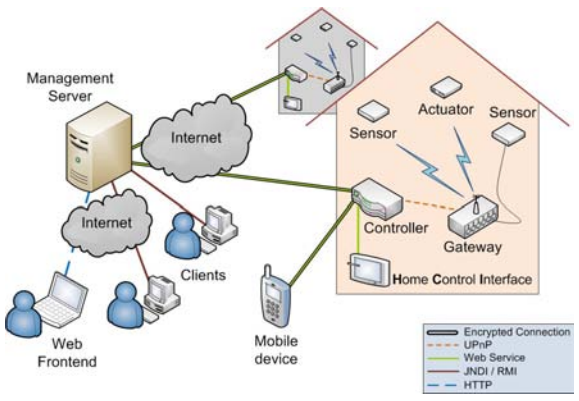

Fig. 1. Schematic structure of the system

The Management Server's main task is the management of one or more Controllers. This includes the configuration of the Controller as well as the analysis of the sensors' and actuators' values which are managed by the Controller. A Web Service interface is planed for the communication between the Management Server and the Controller. The use of the Management Server's functions should be realized by a web-based front-end or clients which are either connected locally or by the Internet to the Management Server.

As shown in Figure 1, the Controller has two communication interfaces – a Web Service interface to a Management Server and a Home Control Interface and a UPnP interface to one or more Gateways. The task of the Controller is the management of the sensors and actuators which are connected by the Gateways. This also includes complex use cases like the automatically control of the temperature in a room or a building. Beside the configuration of the Controller by the Management Server the configuration by the Home Control Interface and mobile clients should be possible. The Home Control Interface is an optional component which can be used together with the Management Server or instead of the Management Server for configuration.

The Gateway is supposed to be the connection point for the different sensors and actuators. On the one hand, it sends the values of the sensor and actuators to the Controller and on the other hand, it delivers the commands of the Controller to the corresponding actuators. Therefore, the Gateway has to support different transport technologies for the communication to the sensors and actuators. These transport technologies are for example ZigBee or WLAN. Beside the support of the different transport technologies, a generic interface is planned for the several drivers of the connected sensors and actuators. In fact, the sensors and actuators are the functional units of the home or building which had to be controlled. Some possible sensors and actuators for a building automation could be motion and air pressure sensors as well as light switches and locking systems.

After the conception of the whole system, the next step of the project was the development of the main components Gateway and Controller. This also included the choice of a suitable platform for these two components.

3. PLATFORM

The first step was to decide which platform is right for the Controller and Gateway based on investigations on hardware as well as on software components. From the beginning of the project, Linux was selected as the operating system for both devices. This enhances the scalability of the whole system by only replacing the hardware of Controller or Gateway. Additionally, the system could be easily adapted to meet the concrete requirements where it will be installed. An adaption of the software is not necessary. The wide support for Linux allows the use of various hardware platforms, also of embedded devices.

As the requirements to the hardware platform depend on the point of action and the size of the whole system, the right platform for all scenarios does not exist. But it is possible to define general requirements that should be paid attention to concerning the choice of the platform?

Asset costs
: To minimize the costs of the whole system, the costs for the hardware should be as small as possible because the price of the system is an important marketing argument. Thus features that are not needed for the system and which make the hardware unnecessarily expansive should be ignored.

Performance
: This is not only the clock frequency of the CPU but also the size of working memory and storage. Because of missing experiences, no exact data could be defined for this criterion. The requirements depend on the size of the system. If the platform is too powerful, this could have negative influence on the power consumption.

Power consumption
: As the devices of the system run all day long through out the year, the power consumption is an important criterion because high costs could arise with too wasteful devices.

Interfaces
Particularly for the Gateway, the configuration of the hardware with many various interfaces is important to integrate as much as possible sensors and actuators from different manufacturers. Beside the standard interfaces like USB or RS-232, different wireless technologies like WLAN, Bluetooth or ZigBee are desirable.

Linux compatibility
Because of the decision to use Linux as operating system, the Linux compatibility is a basic assumption for the selected hardware platform. This has the above mentioned advantage of scalability and the price advantage because of the missing royalties.

Robustness
This criterion contains the adaption of the components to environmental conditions such as temperature, dust or humidity. Particularly the case of Gateway or Controller is affected but also inner components like a hard drive, if the device is installed in an environment with a high risk of agitation.

During the development of the Gateway and the Controller, the authors agreed to concentrate on the scenario of a household. Therefore, the following requirements to the hardware platform are defined.

- 500 MHz to 1 GHz CPU clock frequency
- 128 MB to 256 MB working memory
- 512 MB storage
- power consumption between 5 W and 10 W

On the base of this requirements, a research was done with the result that the SheevaPlug (Figure 2 and the Alix.1D (Figure 3) are possible platforms.

Fig. 2. SheevaPlug

Fig. 3. Alix.1D

The selection of the software platform is a result of the choice of the programming language and the used frameworks for the UPnP- and the web service interface. Therefore ,a second detailed research was done to determine for example the usability and the license conditions of the frameworks. As a result, the CyberLink for Java framework[2] was selected for UPnP and Jersey[3] as a REST-framework for the web service interface. With this, Java becomes the programming language for Gateway and Controller.

4. GATEWAY

The main task of the Gateway is the connection of the various sensors and actuators from different manufacturers to the system for building automation. It provides the access to the devices over a uniform interface and hides the specific access technologies of the manufacturer from the rest of the system. UPnP was selected as the technology for this uniform interface. Section 4.1 gives the reasons for this decision and describes the interface specification. Furthermore, the Gateway should support the dynamic adding and removing from sensors and actuators. To reach this hot plug support, the implementation of the Gateway is based on OSGi [4]. Section 4.2 describes the internal structure of the Gateway.

4.1. The UPnP interface

The Controller must be able to communicate with the sensors and actuators. For this communication, UPnP was selected in the run-up to this work because of the following reasons:

Locating devices
It should be possible to connect new sensors and actuators to the system during the runtime. Therefore, the Controller must be able to find new devices automatically or the devices must announce their presence to the Controller. The *Discovery* contains different procedures for these scenarios.

Learning the properties of new devices
It is not enough to know, which devices are present in the network. The Controller must know how to connect to the devices and which information they provide. There can be a large number of various devices. If the information for each device must be configured manually, the administration effort of the system would be to large if a high number of devices is connected. The system itself must learn the properties of the devices. For this task, the devices interchange description documents in XML format. This is defined in the *Description*-phase. These documents contain all information about the properties and capabilities of a device. So the Controller gets all needed information about the connected devices without a manual intervention.

Interaction with the devices
To request information or execute actions, there must be a way for the Controller to communicate with the devices. To facilitate this communi-

cation between devices from different manufacturers, a standardized protocol is needed. UPnP uses in the *Control*-phase SOAP as technology for the access to the devices.

Efficient notification
It is important for the usability of the system that the Controller is informed about changes of the sensors and actuators as fast as possible. If the Controller had to ask each device periodically to notice a changed value, the network traffic and the resource consumption would increase and the delay between the change of a value and the recognition of the Controller gets longer. UPnP solves this problem in the *Eventing*-phase by an asynchronous notification of the Controller. The devices report changed values automatically directly after the event.

Because of these capabilities, UPnP was selected for the communication technology between the sensors and actuators on one side and the Controller on the other side.

4.1.1. Classification of the Gateway

Like mentioned above, the sensors and actuators communicate via the UPnP interface with the Controller. Currently, there are no devices that support our UPnP interface specification because it is designed in this project.

To use already existing sensors and actuators from different manufacturers in our system, the Gateway must take over the communication via UPnP substitutionally for the devices. From the Controllers point of view it looks like a direct communication with the connected devices. This is important to integrate devices later that implement the UPnP interface themselves without adaption on the Controller or other parts of the system. For the Controller the Gateway is transparent. It does not need to know anything about the existence of the Gateway. For the Controller the communication looks like in Figure 4. The devices are logically connected to it via the UPnP interface. It does not notice that the devices are simulated by the Gateway. This has high significance for the specification of the UPnP interface because it must be designed for the communication between Controller and the sensors and actuators and not for the communication between Controller and Gateway.

The real communication looks like in Figure 5. The UPnP connection exists between Controller and Gateway which simulates the communication endpoints for the connected devices. The connection to the single devices takes place via the manufacturer-specific technologies and protocols. Thus, the two main tasks of the Gateway are the connection of non-UPnP devices

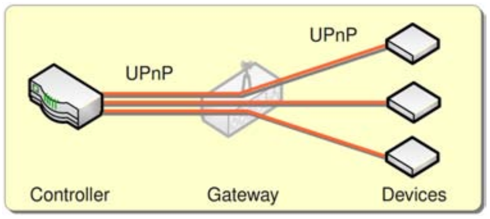

Fig. 4. Logical Communication between Controller and Devices

by simultaneously hiding of the manufacturer-specific technologies.

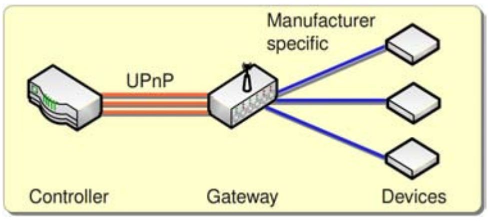

Fig. 5. Real communication between Controller and Devices

4.1.2. Requirements to the UPnP interface

Many of them are partially complied with the features of UPnP. They can be summarized as followed:

- The Controller must have the possibility to search for a single device or a group of devices. The needed procedures are defined in the `Discovery` step of UPnP. Only the device type must be defined.
- If new devices enter the network they must register themselves to the Controller. This point is complied by UPnP as well.
- The Controller must be able to request the current value or state of a device.
- Actuators must provide the possibility to change their state.
- The devices must provide the possibility to the Controller to register for automatic notifications if a value of the device changed. If this happens, the device has to inform the Controller. These requirements are nearly completely complied by UPnP.

4.1.3. Specification of the UPnP interface

For the specification, the requirements in the previous section had to be mapped to the different elements of UPnP[5]. The first step was to decide whether there are two different device types for sensors and actuators or to combine them in one. The authors decided to specify only one type for both device classes. The main reason for this is that each actuator is a sensor too because the current state could be seen as a sensor value. The difference between the device types is mapped to the different actions that are provided by the devices.

Because there is no device type defined by the UPnP-Forum for devices for building automation, an own device had to be defined. The same concerns to the provided services. All requirements are mapped to the belonging parts of UPnP like the device and the service description documents. The authors tried to keep the specification as general as possible to facilitate mapping of as much as possible devices over the interface. So it is possible to use the device type for a temperature sensor as well as for a motion detector or a garage opener. The detailed specification could be requested from the authors.

4.1.4. Address conflicts

The concept of the whole system includes no limitation for the number of Gateways. So it is possible that a wireless device is in the reach of two Gateways. Because the ID of the UPnP device is based on properties of the real device it can happen that a device with the same ID is provided by two Gateways at the same time. This facilitates a redundant connection to the device so that a failure of one Gateway could be absorbed. But this advantage could not be used with the selected UPnP framework because an evaluation of the different URLs for devices with the same ID is not possible. The different URLs for the same device ID cause an internal conflict in the framework.

A possible solution is to integrate a Gateway identification to the ID of the device so that a device has a different ID on each Gateway. In the described scenario, that one device would appear under two different IDs. But for the user and the system it would not be identifiable that it is the same device.

The implementation of an inter-gateway communication to observe the connected devices and to recognize a Gateway failure is another possible solution for this problem. Currently, no solution is implemented for this problem.

4.2. The generic interface

A central requirement to the Gateway was the possibility to add and remove devices during the runtime without restarting the Gateway or stopping already active devices. The specific code to interact with the devices shall be placed in separate driver modules. So the rest of the Gateway implementation is independent from the supported devices. Figure 6 shows the general structure of the Gateway. To put this requirement into practice, different conditions must be fulfilled. Because the implementation of the Gateway is based on Java, first a way had to be found to integrate new Java code into a running virtual machine. But this is not enough. To implement an interworking between the Gateway and the driver modules a generic interface had to be defined. This had to be designed in a way that as little as possible limitations to the functions of the connected device exist and that all requirements of the UPnP interface can be mapped.

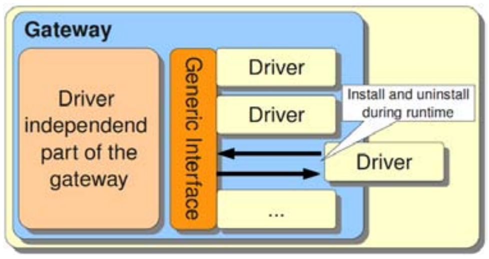
Gateway
Driver independend part of the gateway
Generic Interface
Driver
Driver
Install and uninstall during runtime
Driver
...

Fig. 6. General structure of the Gateway

Because the devices can only provide features that are included in the UPnP interface specification, a major part of the requirements to the generic interface could be copied from the UPnP interface. The specification of the generic interface consisted of two steps. In the first step, a technology was selected that allows to load driver modules into the Gateway during the running time. After that the interface was specified based on the selected technology.

4.2.1. Implementation of the Gateway

OSGi was found as the ideal technology for the Gateway. It was developed for embedded systems so the resource consumption is very low. OSGi provides a system for Java to implement modular applications with the possibility to upgrade them with other components dynamically during the runtime of the application. The generic interface consists of a number of classes and interfaces that must be implemented by a driver developer to implement a driver that could be installed into the Gateway.

All parts of the Gateway are separated in different bundles. A bundle is the smallest structure that can be installed dynamically in the OSGi framework. The Gateway implementation is divided into four bundles. All classes and interfaces that belong to the generic interface are collected in one bundle. The main component with the Gateway functionality is implemented in the second one. New installed devices will be found and the representing UPnP device is generated automatically. Another bundle is responsible for the management of the Gateway. It provides a command line interface to the user to control and configure the Gateway. The fourth bundle implements a logging infrastructure that can be used by all installed bundles including the drivers. These four bundles form the implementation of the Gateway. Each driver is packaged in its own bundle.

5. CONTROLLER

The Controller is one of the central components of the building automation system. It is the link between Management Server, HCI, mobile devices and the sensors and actuators. Its tasks contain the management of the sensors and actuators as well as the providing of information to the sensors and actuators which are connected. Beside the direct control of the sensors and actuators it should also be possible to realize complex use cases, for example an automatical temperature control or an automatical control of shutters on the base of the incidence of light. To realize these tasks, the Controller has two different communication interfaces as shown in Figure 1. The Web Service interface serves for the communication between the Controller and the Management Server, the HCI or the mobile devices. For the control of the sensors and actuators the Controller uses the UPnP interface — see section 4.1. With the UPnP interface one or more Gateways can be connected to the Controller.

5.1. Web Service interface

The Web Service interface is the Controller's communication interface to the possible clients for the configuration of the Controller and the use of the devices. These clients could be the HCI, a Management Server or a mobile client like smartphone. The choice for the use of a Web Service for this communication interface follows from the requirements and the basic concept of the whole system. This decision was determined by the advantage of the loose coupling and therewith the independence from the used technologies for the implementation and the use of the communication interface. Because of this choice it is possible to facilitate a huge number of different clients the use of the communication interface and accordingly the interaction with the Controller.

5.1.1. Requirements

The Web Service interface facilitates the communication in fact with the main component of the building automation system. Hence all interactions between the clients and the devices — the sensors and actuators — have to be realized by the Web Service interface. This basic condition and the enhanced tasks for the Controller result in a set of use cases which have to be mapped by this interface. The enhanced tasks of the Controller are:

- The configuration of groups for logical structuring of the devices. These groups can be real existing structures like a group "living-room" as well as logical structures like "temperature control devices".
- The assignment of devices to configured groups and corresponding the restructuring of the assignments.
- Creating and deleting of programs for the configuration of automatical sequences.
- The registration on the Controller for notifications on events. These events can be triggered by created programs or by the connected sensors and actuators.

5.1.2. Implementation

Based on the described requirements, there is large number of use cases which have to be realized by the Web Service interface, for example the reading of a sensors value or the creating of new groups. The implementation of the Web Service interface is realized with the REST framework Jersey. The basic elements of a REST Web Service are the resources. Thus, the modeling of the use cases is realized by the specification of the resources and the allowed HTTP methods.

The base resource (base URL) propagates the basic information of the Controller and the REST Web Service to the client. The necessary resources are divided into three groups:

1. Resources for the control of the sensors and actuators,
2. resources for the configuration of groups and
3. resources for the configuration of programs.

Based on the hypermedia principle and the knowledge of the base URL, a client should get an access point to the whole Web Service by reaching all other resources by links. At present, the exchange of information on the Web Service interface is XML based. This could be extended by other formats like JSON or HTML in one of the next possible steps of development.

The device resources are a logical representation of the real existing devices which are connected to the building automation system. The information which sensors and actuators are connected the Controller gets from the devices themselves via the UPnP interface. As a result, it is not possible to create new sensors and actuators or delete them by means of the Web Service interface.

In contrast to the devices, it is possible to create and structure groups because they are only logical resources. Thereby it is possible to assign one group to another group as a subgroup. A possible structure of groups is shown in Figure 7. As you can see, each group is maximal assigned to one other group. It is not supposed that one group is a subgroup of several other groups.

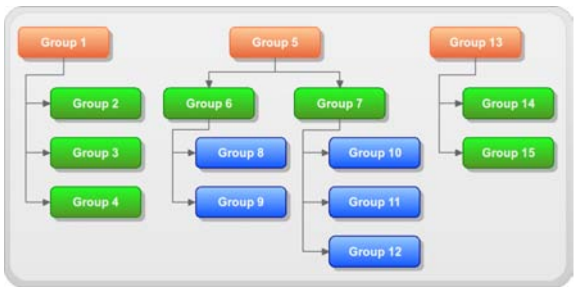

Fig. 7. A possible structure of groups

The reason why these groups have formed is as a result of applying these devices to them. This makes an easy visualization possible for a client. So it is imaginable to emulate the topology of a one-family house by several groups. This facilitates a spatially structured visualization of the one-family house for the user by a client.

The requesting of the sensors' values and the adjusting of the actuators can be done with the Web Service Interface but it is not possible to configure automatical sequences with the device and group resources. This case is realized with program resources. These resources allow clients to save, delete, deactivate and reactivate programs on the Controller.

5.2. Programs and automations

Simple processes, for example, to switch on the light or shut down the shutters, should be possible by a direct access via the Web Service interface of the Controller. In contrast, one of the most important use cases is the automation of the home control, for example, the configuration of an alarm system by means of motion sensors, door sensors and window sensors. Another scenario is the configuration of a heating control system, for instance to get a room temperature of 23 °C from Monday to Friday in the time slots from 6 to 9 am and from 5 to 11 pm as well as on Saturday and Sunday in the time slot from 8 am to 12 pm. These and similar scenarios require an automatical control of the sensors and actuators by the mean of configured programs. In general, a program stands for automatical configuration sequences for the whole system. In the case of the heating control system, the corresponding program means that a heating actuator will be turned on if the sensor falls below a specified temperature. Therewith several use cases can be realized or the programs can simply be used to automatize processes.

The basic concept of the whole system includes the execution and storage of such programs by the Controller. The reason therefore is to make the whole system independent from the Web Service interface and above all independent from the reliability of the connection to the Internet. In addition, it is important to make the programs centrally available because it should be possible to configure the Controller by several clients. As a consequence, each client can access to each existing program independently from the fact whether the client has created the program or not.

5.2.1. Requirements

The language of the programs has to be based on a very abstract level because the Controller in general only knows sensors and actuators and no specific forms like a temperature sensors, electric meters or light switches. That means the language has to include basic operations like the requesting of the sensors' values and the adjusting of the actuators.

Beside these basic operations, also some logical conditions (if-then, if-then-else), comparisons (equal, not equal, greater equal, greater, lower, lower equal) and combinations (and, or, xor) are required. Furthermore, details for the time of the execution are necessary, i. e. on which weekday and at which time or in which time interval the program should be active. Another element of the language has to be an operation for generating notifications. This is necessary, for example, to realize an alarm system on which a security service will be informed if the alarm is released.

Loops or similar elements are not required because the execution of the programs is realized in a proactive way. That implies that a program will be executed when a value of a sensor or an actuator which is included in the program had changed its state.

5.2.2. Implementation

Due to the specific requirements which are described in section 5.2.1 we decided to design an own XML based language. The decisive step was the use of the JAXB framework [6]. The definition of the particular language elements and the generation of the XML scheme was realized with this annotation based framework. The use of this framework additionally facilitates the simple implementation of an interpreter for the programs.

The JAXB framework generates an object tree after the parsing of programs XML representation. The root element is the representation of the whole program. It facilitates the execution of the program by calling the methods for the execution on each element of the program. On this scheme the execution is cascaded to the elements of the deepest nesting – the leafs of the object tree.

Regardless of the elements which are defined at the moment, the use of the JAXB framework allows an easy extension with new elements and functionalities for future development and adaptations.

5.3. Persistent storage of the configuration

Another important fact is the persistent storage of the Controller's configuration to restore the former state of the Controller after a restart. This is necessary, for example, on a power blackout. All configurations like the created groups, the assignment of devices to groups, the configured programs or the registrations for notifications on events have to be restored in such a case. It would be not acceptable that the Controller discards the configuration after a shut down — controlled or uncontrolled – and the user had to enter all the configurations once again after the reboot.

At this point the JPA specification [7] with the Hibernate implementation [8] is involved. The important information of the configuration are saved in a local database based on annotations in the data model of the Controller.

The database exists only as a backup, i. e. all interactions with the data model are based on the objects of the data model which are stored in the main memory of the Controller. Thus, not all information which are stored in the data model of the Controller are saved in the database. Only the important information which are required after a reboot are persisted. The non-persistent information are, for example, the current value of the sensors and the state of the actuators because the Controller can ask the devices for these information after the reboot.

5.4. Concurrency and deadlocks

The concept of the communication interface facilitates the interaction between several clients and the Controller and thereby the changing of the configuration. The simultaneous access of several clients can cause race conditions and concurrency. This has to be managed in a sensible way to avoid an inconsistent state on the Controller's data model.

5.4.1. Synchronized access to the resources

The first idea to solve this problem was to synchronize the accesses to the particular resources of the Web Service interface. However, after a short time it became obviously that this solution is to inefficient and not sufficient enough. Inefficient because the synchronization is based on the methods for the processing of the HTTP requests and these methods are not existing for each resource but for each resource group, for example, the devices of a group. Not sufficient because the accesses to different resources influence each other, for example, the adding of a group to another group as a subgroup and the simultaneous deleting of this group.

5.4.2. Security of the data model

The next idea was to secure the data model with lock objects from the Concurrent library [9] for race conditions. With this solution it was not possible to avoid race conditions during the saving of the changes on the Controllers configuration into the database. The changes would be consistent in the data model but inconsistent states could happen while storing the changes in the database. As a consequence, this idea was not the right solution of the problem.

5.4.3. Locks for each object of the data model

The implemented solution for the problem of the concurrent accesses and the race conditions is the implementation of an own strategy for the allocation of locks for each object of the data model by a central instance — the Lock-Manager. It is responsible for the allocation, the management and the release of locks for the objects of the data model.

Because of the manifold relations between the particular objects of the data model it is possible that deadlock situations can occur. Therefore, a way for the detection and the handling of such situations had to be found. For this reason, the request of a lock for an object is not implemented as a blocking operation but the Lock-Manager replies with `false` if the lock for the requested object can not be granted. If the Lock-Manager did not grant the requested locks for several objects on consecutive attempts, we can originate that we have a deadlock situation. The corresponding instance, i.e. the method of the Web Service interface, can react in such a situation with an error message to the client.

Beside the detection of deadlock situations, a second mechanism is implemented to resolve such situations: The request for a lock is randomly repeated three to six times. If the Lock-Manager could not grant the locks after these repetitions, the requesting instance releases all locks which it had requested till now. Afterwards the requesting instance starts a new try to get the locks for the required objects. If this process was not successful after ten repetitions, the instance will abort the request with an error message to the client.

Because of these mechanisms it is possible to detect deadlock situations and react on them in an adequate way without durably blocking the Controller.

6. CONCLUSION

The presented system facilitates a flexible use in the different use cases in the scope of the home control for the ambient assisted living. The use of UPnP in connection with the concept of the Gateway allows the usage of devices from arbitrary manufacturers. This reduces the costs because it is not necessary to develop the devices for the home control. Already existing solutions for the home control can be integrated and therewith a huge number of components are still available.

Due to the modular structure and the use of open standards, it is easy to adapt and extend the system. Feasibility studies for the implementation of a rich-client based Home Control Interface and the use of the Controllers Web Service interface with mobile devices on the basis of the Android platform already exist.

7. REFERENCES

[1] UPnP Forum, "Project homepage," Internet: URL `http://www.upnp.org/`.

[2] Satoshi Konno, "Cyberlink for java," Internet: URL `http://www.cybergarage.org/cgi-bin/twiki/view/Main/CyberLinkForJava`.

[3] Jersey, "Project homepage," Internet: URL `https://jersey.dev.java.net/`.

[4] OSGi Alliance, "Project homepage," Internet: URL `http://www.osgi.org/`.

[5] UPnP Forum, "Upnp device architecture 1.0," Internet: URL `http://www.upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.0.pdf`.

[6] Ed Ort and Bhakti Mehta, "Java architecture for xml binding (jaxb)," Internet: URL `http://java.sun.com/developer/technicalArticle1s/WebServices/jaxb/`.

[7] Sun Micrsystems, "Java persistence api," Internet: URL `http://java.sun.com/javaee/technologies/persistence.jsp`.

[8] Hibernate, "Project homepage," Internet: URL `http://www.hibernate.org/`.

[9] Sun Micrsystems, "Package java.util.concurrent – javadoc," Internet: URL `http://java.sun.com/javase/6/docs/api/java/util/concurrent/package-summary.html`.