Modeling eHome Systems

Ulrich Norbisrath
Department of Computer Science 3
RWTH Aachen University
Ahornstr. 55, 52074 Aachen, Germany
uno@i3.informatik.rwth-aachen.de

Ibrahim Armac
Department of Computer Science 3
RWTH Aachen University
Ahornstr. 55, 52074 Aachen, Germany
armac@i3.informatik.rwth-aachen.de

Daniel Retkowitz
Department of Computer Science 3
RWTH Aachen University
Ahornstr. 55, 52074 Aachen, Germany
retkowitz@i3.informatik.rwth-aachen.de

Priit Salumaa
Playtech Estonia OÜ
Emajõe arikeskus, Soola 8, 51013 Tartu, Estonia
priit.salumaa@playtech.com

ABSTRACT

New developments and decreasing costs of electronic appliances enable the realization of pervasive systems in our daily environment. In our work, we focus on eHome systems. The cost-intensive repetitive development process for every new eHome environment is one of the major problems preventing their widespread use. So, we transformed the repetitive development process to a single one, followed by a repetitive configuration process. To support this configuration process, we introduce a model capable of storing all the parameters relevant for this specific process. To enable semi-automatic configuration based on the model, a specification is required beforehand. In this paper, we will show how the necessary specification is covered by the introduced model, and how the model supports the eHome system configuration and context inferring at runtime.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Management—*Software configuration management*; D.2.1 [**Software Engineering**]: Requirements/Specifications—*Methodologies*; D.2.12 [**Software Engineering**]: Software Architectures—*Domain-specific architectures*; J.7 [**Computer Applications**]: Computers in other systems—*Consumer products*

General Terms

Design

Keywords

Smart home, model, object oriented, pervasive computing, context-aware

MPAC '06, November 27-December 1, 2006 Melbourne, Australia

1. INTRODUCTION

The market already offers a wide range of appliances usable in home automation environments. Affordable examples are: X10 appliances, standard USB devices, or Connex appliances. From this point of view, the hardware for the realization of smart home environments is already available at reasonable costs. Various setups of such smart home environments prove the applicability in practice. We call those environments *eHomes* or *eHome systems* [**?**, **?**, **?**]. All these implementations are either research or hobby projects. The question arising is: What is missing to make eHomes more widespread? We believe that the main obstacle preventing eHomes from becoming more widely used is the relatively high price of the software driving the eHome, since it needs to be developed or adapted for each particular eHome. A complete software development process per case is not affordable for everyone [**?**].

In our process, the development of eHome services has to be done only *once* for all regarded environments. The *repetitive* portion is reduced to the level of mere specification of the target environment, as well as interactive configuration of the given service components for the eHome system, with no coding overhead. In our approach, the eHome services are implemented as OSGi [**?**] bundles using the object-oriented programming language Java [**?**]. OSGi allows for developing service-oriented software components, respective OSGi frameworks provide an execution environment for OSGi bundles and also manage the lifecycle of the software components. Our goal is the support of composition and configuration of services for eHome systems. In this context, we introduce a *Specification, Configuration, and Deployment process*. We will refer to this process as the eHome SCD-process.

The reason for developing the SCD-process for eHome systems is to establish a low cost process to introduce eHomes to a wider constituency. To support the SCD-process, we developed an eHome model, which is the main focus of this paper. The eHome model specifies a graph-based data structure which is used in the SCD-process. As the name of the SCD-process indicates, the process consists of three phases. All three phases are supported by the eHome model.

In the *specification phase*, the eHome environment and the required services are specified. During this phase, the architectural information about the eHome is captured. The given appliances and their location in the home environment are described. Along with the eHome environment, the services used later in this environment, plus the required devices and functionalities, need to be

defined and specified once beforehand.

In the *configuration phase*, the services selected in the specification phase are automatically adapted to the specified environment. This means that necessary appliances are added to the configuration. Likewise, required sub-services that are missing are selected to match the functional requirements of the selected services. For example, if the lighting service needs at least one lamp per room and one switch to control the lamp, these devices will be added to the configuration. Furthermore, the corresponding driver component services for the lamp and switch controllers are added to the configuration.

In the *deployment phase*, the service configuration is deployed, i.e., the software components specified and configured during the first two phases are deployed automatically onto the service gateway residing in the eHome. The software components are also initialized properly and launched automatically.

The details of the three phases mentioned above would go beyond the scope of this paper, as we want to focus on the eHome model here. An in-depth presentation of the SCD-process can be found in [**?**]. The eHome model will be described in detail in the following section.

2. THE EHOME MODEL

The whole SCD-process is strongly related to the *eHome model*. The eHome model comprises all the information necessary for supporting the SCD-process and execution of the eHome services at runtime. It contains several contexts to model the different aspects of eHome systems. These are the *functionality context*, *device definition context*, *environment context*, *service context*, *service instance context*, and *person context*.

The instanced model is actively involved during the runtime of the eHome system as it provides information about the home environment for eHome services. This enables the development of *context-aware* eHome services which need the real-time information about the inhabitant's location, and the states of other services, or appliances. In other words, the eHome model instance is also a *communication medium* for eHome services (a similar communication approach is also followed by [**?**]).

The eHome model is being developed by using the Fujaba tool suite [**?**]. Fujaba enables the development of an object-oriented model and the generation of fully executable Java code for the model. The model has its statics and dynamics. The static structure is designed using a UML class diagram describing the classes of the model and their relations to each other. During runtime, the objects of the model classes are created according to the structure of the class diagram. For more details see [**?**].

In the following subsections we will describe the details of the different contexts mentioned before. All contexts are interrelated and form together the eHome model. However, they will be presented separately to emphasize the context-specific details.

2.1 Functionality Context

The smallest but one of the most important contexts of the eHome model is the part describing *functionalities* of the services, as well as the appliances (see Figure **??**). The functionalities are described by means of the `Function` class which has a reflexive relation describing that one functionality can be refined by another one. For example, a detection functionality can be refined by the functionalities: movement detection, smoke detection, glass breakage detection, etc. This reflexive relation of the `Function` class results in a tree structure of objects from this class. Functionalities are defined by their names and the refinement relation of the functionalities should be used in such a way that the most general functionality is the root of the tree and the most specific functionalities are the leaves of the tree.

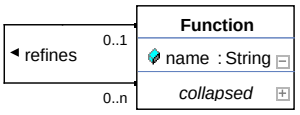

Figure 1: The functionality context

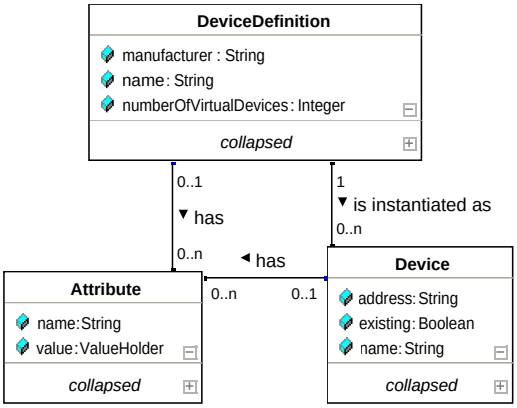

Figure 2: The device definition context

2.2 Device Definition Context

The devices used in the environment specification phase of the SCD-process are predefined. Therefore, before a device can be added to the eHome, it must be specified in terms of a *device definition*, which includes the manufacturer information, the name of the device, and further device-specific attributes. Attributes can have an arbitrary structure suitable to store the necessary information. Therefore, the `Attribute` class contains an abstract `ValueHolder` that can be extended to encapsulate arbitrary data types. A concrete device is specified in the environment as an instance of the device definition, identified using specific attributes, like addresses (IP address, house code for X10, USB address, etc.), and recognizable names. In Figure **??**, the device definition is represented by the `DeviceDefinition` class. The concrete device, as the instance of its definition, is represented by the `Device` class. The attributes are represented by the `Attribute` class; it will also be used in further contexts, as we will see later.

The functionalities provided by the devices are important for the SCD-process. Nevertheless, they are not specified in the device definition but in specification of the corresponding device driver services. This is due to the fact that the hardware itself, as well as the physical connections, have no significant role in the software configuration step during the SCD-process. The configuration is done using the corresponding software components only.

2.3 Environment Context

The *environment* context models the location information according to the floor plan of the home and the given appliances. It can provide a set of different environments which are connected via locations or location elements for the eHome. For example, a house can be connected with an external garage, by a hall or just a door. There is also a sub-location concept implemented. A floor

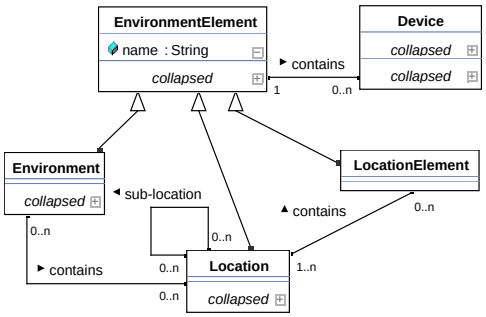

Figure 3: The environment context

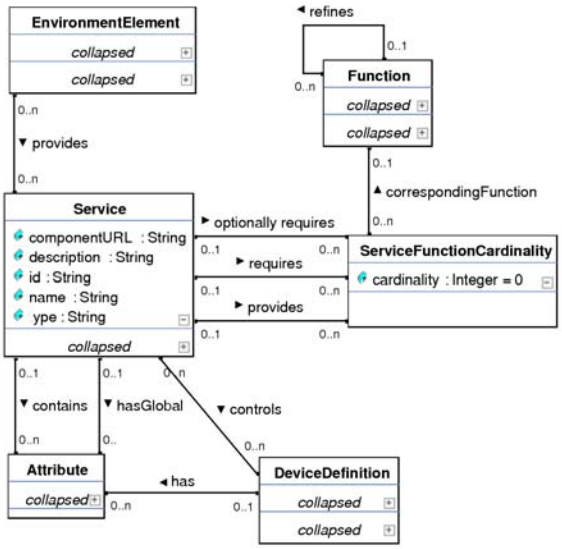

Figure 4: The service context

can have its rooms as sub-locations or a room can have different service-related areas near the windows, doors or a TV set. This kind of modeling freedom and generality gives us a mechanism that is powerful enough to express any kind of architectural designs.

The environment context has the `EnvironmentElement` class as a super-class for any other class describing location information of the home (see Figure **??**). `EnvironmentElement` aggregates the common features of the classes describing the home environment. Since the classes `Environment`, `Location`, and `LocationElement` inherit from it, they have a relation to the `Device` class. This means that every element in the environment context can have appliances related to them. For example, in the living room, there can be a lamp, a media set with speakers and an LCD screen, controlling switches, etc.; the door in the room can have a movement detector attached to it; and the window can have a glass breakage detector attached to it.

Environments, locations, and connecting location elements can form a complex graph that represents the logical connections in an architectural design. The connections describe the relations between the entities in the architectural design, thus modeling also the three dimensional relations in the building. An example of an environment is depicted in Figure **??**.

2.4 Service Context

The *service* context of the eHome model represents eHome service descriptions, or more accurately, the definitions of eHome services. It is crucial for the SCD-process that the eHome service software components configured during the automatic configuration phase are modeled beforehand, since the automatic configuration relies on the abstract description of the eHome service software. The service context does not include the runtime configuration of the selected services, which has to be deployed onto the service gateway in the eHome during the deployment phase of the SCD-process.

Figure **??** outlines the service context. The service is modeled by the `Service` class, which not only contains information such as the id, name, type, and description of the service, but also the information on the resource URI of the corresponding software component installed during the deployment phase. The `Service` class is an abstract description of the corresponding software component executed during the runtime of the eHome system.

The essential part of the service description is specified by functionalities. The `Service` class has three indirect relations to the `Function` class over the `ServiceFunctionCardinality` class. A service is described by the functionalities it provides, requires, and optionally requires. As mentioned before, the functionalities of the devices are considered to be part of the device driver services. The functionalities allow for a dynamic composition and dependency resolution during the automatic configuration step, forming an abstraction layer for service composition.

The `ServiceFunctionCardinality` class is also used by the service requirements. The cardinality at the required service indicates the quantity of the functionality required. Figure **??** shows the `Switch` service controlling an on/off switch that can be used by a two level heating service. This service would require the `switching` functionality with cardinality “2”, because the first switch is used to turn on and off the radiators on the walls, and the second switch is to turn on and off the floor heating. If the heating service is installed in a location, there will be two `Switch` service instances added into the configuration, as well as two toggle switch devices at the corresponding location. If some other `Switch` service offered the `switching` functionality with a cardinality greater than or equal to “2”, the heating control service would only need one instance of this kind of `Switch` service. If the `Switch` service offers the `switching` functionality with a cardinality greater than “2”, the same instance could be used by other services requiring the `switching` functionality.

The `Service` class has two relations to the `Attribute` class. These two relations model the global and the specific attributes. Global attributes are valid for the complete eHome environment, while specific attributes may differ from location to location.

Services are related to the environment information. The relation between an `EnvironmentElement` and a `Service` class implies that the environment element offers a service for the eHome inhabitants. This relation is typically used for locations. The links between locations and services are created when the given service is selected during the specification phase of the SCD-process.

The `Service` class and `DeviceDefinition` class are related, as the devices are controlled by the software. This is the case for services which in fact are the driver components of devices. The device driver component provides the other services or the end-user with functionalities attributive to the controlled device. However, devices as such have no great significance for the automatic configuration phase of the SCD-process. An example of a device driver service is presented in Figure **??**.

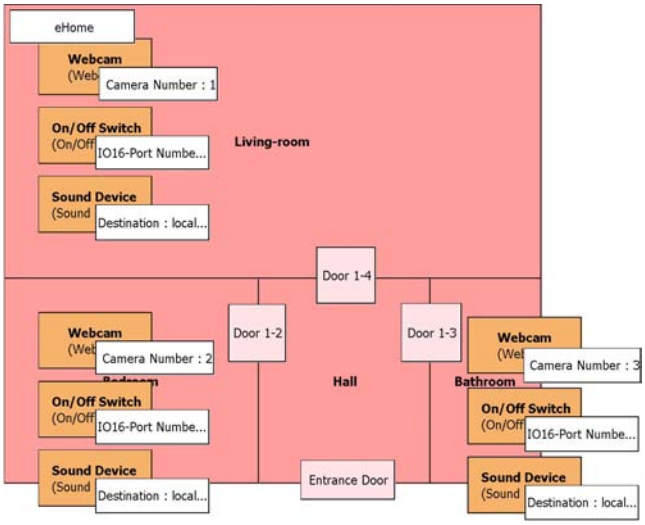

Figure 5: The necessary devices in the eHome environment supporting the Music Follows Person service

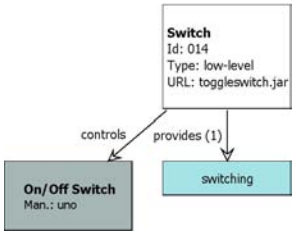

Figure 6: The specification of the Switching service to control a toggle switch

2.5 Service Instance Context

While the service context models a service by its functionality dependencies, the *service instance* context models the runtime configuration of services in the eHome system. During the configuration step of the SCD-process, the parts of the eHome model instance corresponding to this context are built automatically. To give a better overview, the service instance context is depicted on two figures **??** and **??**.

According to Figure **??**, the `ServiceObject` class models the service instance and has a relation to the `Service` class, indicating which service is instantiated. The idea of the service instantiation is to assign a `ServiceObject` with its specific configuration to every `EnvironmentElement` which provides the selected `Service`. Thus, Figure **??** shows a relation between the `ServiceObject` and the `EnvironmentElement` class. The `is in` relation corresponds to the relation `offers` in Figure **??**.

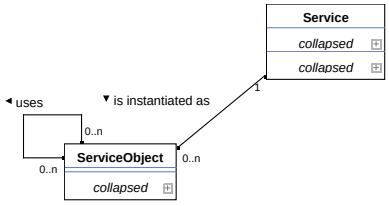

Figure 7: The relation between service and service instance

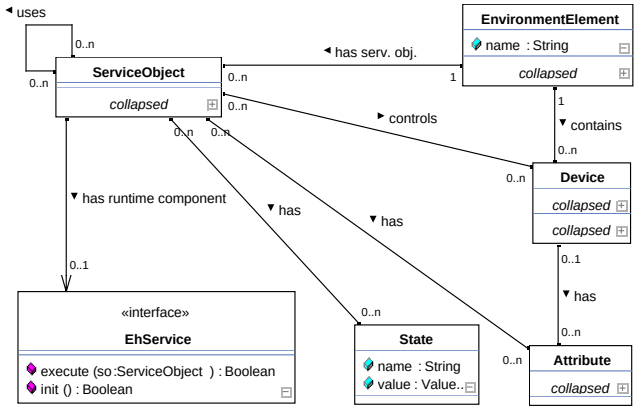

Figure 8: The service instance context

The most important relation in the service instance context for the configuration step in the SCD-process is the self-relation `uses` of the `ServiceObject` class. This relation expresses the usage relation between the services during runtime. In the service context, there is no explicit relation between the services. The reason for that is to use the functionality abstraction around the services for the service composition to enable automatic configuration of the services during the SCD-process.

The service instance context is filled with objects and data during the configuration step of the SCD-process. The tools supporting the automatic configuration take into account the selected services and their functionality requirements, and then compose a suitable set of services to meet the requirements of the services selected by the user. The usage relations between the services are expressed explicitly. The composition uses the indirect `provides`, `requires`, and `optionally requires` relations between the `Service` class and the `Function` class (see Figure **??**). These relations are used to construct a usage and dependency graph of the service instances, expressing the runtime structure and the configuration of the eHome services in the eHome system.

2.6 Person Context

In the *person* context, the person-related information in eHomes is modeled. Information about the person plays an essential role during the runtime of the eHome system, for example, when communicating information about the inhabitant's location to the services. Person related information is also important for other eHome processes, such as business processes [**?**] and possible migration of services between different eHome environments. For those reasons, the person related information is modeled in the eHome model.

Figure **??** presents the class `Person`, modeling the most important properties of the person under consideration. The relation `is in` between the classes `Person` and `Location` represents the location information of the inhabitant describing where the person is currently located. This information is needed for the services which require information about the location of the people in the eHome. The implementation of person-related eHome services is supported by the `uses` relation between `Person` and `Service`. Furthermore, persons can also have attributes to store personal profiles.

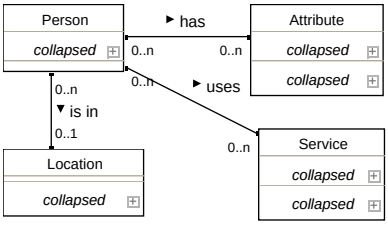

Figure 9: The person context

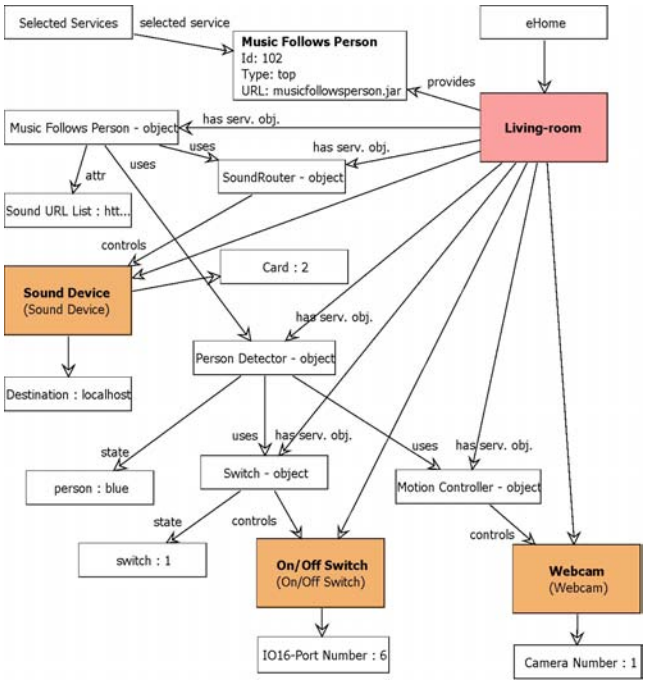

Figure 10: A section of the eHome model instance

2.7 Example: Music Follows Person

As aforementioned, the eHome model supports all phases of the SCD-process. We developed a tool called *eHomeConfigurator* [**?**, **?**] that is based on the eHome model and supports users carrying out tasks for the whole SCD-process. First, a service provider can use the *eHomeConfigurator* to *specify* the services he offers and the device types he supports. A customer can use the same tool to *specify* his home environment with the concrete devices he owns. After selecting the desired services, he can initiate their *configuration*. Based on the provided and required service functionalities, the services are composed and the attributes are assigned to concrete values by the *eHomeConfigurator*. Now, there is an instance of the eHome model that contains all necessary information for the automatic *deployment*, that in turn generates the service instance related information.

As an example of the SCD-process, we consider a small apartment, consisting of a hall, a bathroom, a living-room, and a bedroom. The living-room has a small kitchen corner for cooking (see Figure **??**). We assume that a student couple is living in this apartment. These students want to listen to their favorite music everywhere in the apartment, for instance while taking a bath. Nearly all of their music collection is on the PC. They would like to have the following feature: if a particular person listens to the music, the music played at the time will follow the person from room to room. Yet, we use a simple strategy to resolve conflicts occurring when two persons meet in the same room: The music of the person entering last will be played.

Figure **??** illustrates a section of the fully configured eHome model instance during runtime for our example described above. Depicted is just the part relevant for the living room, providing the `Music Follows Person` service, and the corresponding relations between the objects and the environment element `Living-room`. We see the dependencies of the service objects: `Music Follows Person - object` uses the `Person Detector - object`, which is using `Switch - object` and `Motion Controller - object`. This composition has been automatically generated using the *provided* and *required* functionalities of the corresponding services. Attributes, such as the `IO16-Port Number` of a switch, are connected to services and devices. Furthermore, services may also have *states*, which are similar to attributes, but are changeable runtime properties. An example for a state is the `person` state that indicates which person has been detected in the living room.

3. RELATED WORK

Arroyo et al. propose a task-driven design model for ambient intelligent systems [**?**]. It allows the representation of concepts such as tasks and laws, and physical objects such as devices needed for tasks, as well as computers or humans carrying tasks out. The communication is realized on top of a blackboard, storing the information of the whole system. This allows for a very flexible way of communication, as every part is just responsible for the part of the blackboard handled by itself. The modeling of abstract tasks, physical objects, and relations is very similar to our model. But the model proposed in that paper is up to now not embedded in a development and configuration process, and so will not solve the effort arising from the creation of service components, resolving of dependencies, and finding necessary parameters for the context. In terms of contracts for components, their model offers a higher expressiveness which could be used as a base for our future work.

For our model, we can rely on results from the conceptual building design approach [**?**, **?**], which represents a link from software engineering to civil engineering. When designing a building, an experienced architect implicitly applies his aggregated knowledge to the new sketch. Constructive elements, such as walls, windows, or doors are used with their *conceptual meaning*, namely to form organizational areas or rooms, to guarantee e.g. light and ventilation, or to ensure accessibility. These conceptual elements form a functional view on the future building. At this early design stage, called *conceptual design*, most architects work with pencil and paper, as there is no support for this stage by current CAD system. The conceptual information he/she had in mind gets lost. This information would form a very valuable input for our automation approach. We are going the reverse way, since our SCD tools allow an a posteriori specification of the details lost from the conceptual design phase.

The Java Context Awareness Framework, or in short JCAF [**?**], is a Java framework which offers a set of interfaces to support the development of context-aware applications. JCAF's main features can be summarized as follows: It is a framework which was designed specifically for event-based applications. This also includes services which usually react to events triggered by sensor-devices in the eHome environment. It supports distributed and cooperating services, i.e. it provides methods of inter-process communication. JCAF also offers a set of security features to restrict access to sensor data and actuator control. It provides methods to verify the

origin of incoming events. The last main feature concerns starting, modifying, and stopping software without the need to restart the whole framework.

4. CONCLUSION & OUTLOOK

This paper gave an overview of the SCD-process for eHome systems and discussed thoroughly the eHome model and its structure. The eHome model is used to support the SCD-process by covering all the necessary contexts relevant for inferring information during configuration and runtime of eHome systems. The introduced contexts are the functionality, device definition, environment, service, service instance, and person contexts. The music follows person example illustrated the application of the proposed eHome model. This model is actively used in our group for automatic configuration and deployment of different demonstration environments. These have been demonstrated for example in [**?**, **?**].

The idea of automatic configuration is simple because of the functional abstraction layer it uses. But it still lacks the rigorous mechanism for composition verification on the software component level. The automatic configuration does not check formally if the combined components actually work together and will give a reasonable result during runtime. This issue and QoS aspects could be addressed e.g. using parametric contracts [**?**]. This kind of addition would help verifying the composition verification and composing the components produced by different software manufacturers.

Further future work includes the extension of the SCD-process and of the eHome model for connecting multiple environments and, thus, supporting mobility of users and services between different environments, like home, car, work place etc. This includes especially the person identification and the location awareness of eHome services. Another extension of the model will cover spatial aspects between environment elements enhancing the connection information. Also dynamics of appliances appearing and disappearing in the home will be addressed in future work. Currently, we are working on device discovery and the automatic integration of the identified devices into the configuration. Additionally, we will work on the definition of a services concept for eHomes, like how services can be classified: person-related, location-related, informational etc. Last but not least, we are working on the integration of a security and privacy concept for our eHome prototype.

5. REFERENCES

[1] Ibrahim Armac and Michael Kirchhof. Process Support in Ehome Systems: Empowering Providers to Handle a Future Mass Market. In Thibaud Latour and Michaël Petit, editors, *Proceedings of the CAiSE'06 Workshops and Doctoral Consortium*, pages 219–228. Presses Universitaires de Namur, 2006. Workshop: UMICS'06.

[2] R. F. Arroyo, M. Gea, J. L., and P. A. Haya. A Task-Driven Design Model for Collaborative AmI Systems. In Thibaud Latour and Michaël Petit, editors, *Proceedings of the CAiSE'06 Workshops and Doctoral Consortium*, pages 969–983. Presses Universitaires de Namur, 2006. Workshop: UMICS'06.

[3] Jakob Eyvind Bardram. The java context awareness framework (JCAF) – A service infrastructure and programming framework for context-aware applications. In Hans Gellersen, Roy Want, and Albrecht Schmidt, editors, *Proceedings of the 3rd International Conference on Pervasive Computing (Pervasive 2005)*, lncs, Munich, Germany, May 2005. Springer Verlag.

[4] Beisheim Holding GmbH. FutureLife – Das Haus der Zukunft. `http://www.futurelife.ch/`.

[5] David Flanagan. *Java in a Nutshell*. O'Reilly, 5th edition edition, March 2005.

[6] inHaus Duisburg. Innovationszentrum Intelligentes Haus Duisburg. `http://www.inhaus-duisburg.de`.

[7] Michael Kirchhof, Ulrich Norbisrath, and Christof Skrzypczyk. Towards Automatic Deployment in eHome Systems: Description Language and Tool Support. In Robert Meersman and Zahir Tari, editors, *On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE: OTM Confederated International Conferences, Proceedings, Part I*, number 3290 in LNCS, pages 460–476. Springer, 2004.

[8] Bodo Kraft, Oliver Meyer, and Manfred Nagl. Graph technology support for conceptual design in civil engineering. In M. Schellenbach-Held and H. Denk, editors, *Int. Workshop of the European Group for Intelligent Computing in Engineering (EG-ICE 2002)*, pages 1–49. VDI Fortschritt Berichte, 2002.

[9] Bodo Kraft and Manfred Nagl. Semantic tool support for conceptual design. In I. Flood, editor, *Proceedings of the 4th Int. Symposium on Information Technology in Civil Engineering*, pages 1–12, 2003.

[10] Ulrich Norbisrath and Christof Mosler. Functionality Configuration for eHome Systems. In *CASCON '06: Proceedings of the 2006 conference of the Centre for Advanced Studies on Collaborative research*. IBM Press, 2006. accepted for publication.

[11] Ulrich Norbisrath, Priit Salumaa, and Adam Malik. eHomeConfigurator. `http://sourceforge.net/projects/ehomeconfig`.

[12] Ulrich Norbisrath, Priit Salumaa, Erhard Schultchen, and Bodo Kraft. Fujaba based tool development for eHome systems. In *Proceedings of the International Workshop on Graph-Based Tools (GraBaTs 2004)*, volume 127 of *Electronic Notes in Theoretical Computer Science*, pages 89–99. Elsevier, 2005.

[13] Ralf H. Reussner, Steffen Becker, and Viktoria Firus. Component composition with parametric contracts. In *Proceedings of the Net.ObjectDays*, pages 155–169, 2004.

[14] Tampere University of Technology, Personal Electronics Group. Smart Home Project. `http://www.ele.tut.fi/research/personalelectronics/projects/smart_home.htm`.

[15] The OSGi Alliance. OSGi Service Platform Core Specification. `http://www.osgi.org/osgi_technology/download_specs.asp#Release4`, August 2005. Release 4.

[16] Ulrich Norbisrath, Priit Salumaa, Adam Malik. Specification, Configuration, and Deployment in eHome Systems. `http://math.ut.ee/~peeter_l/seminar/eelmised/05k/ehome.html`.

[17] Ulrich Norbisrath, Priit Salumaa, Adam Malik. eHome Specification, Configuration, and Deployment. `http://ubicomp.org/ubicomp2005/programs/demos.shtml`, 2005. Demonstration Paper D15 on UbiComp2005.

[18] Albert Zündorf. FUJABA (From UML to Java and Back Again). `http://www.fujaba.de`.