Service Composition in eHome Systems: A Rule-based Approach

Michael Kirchhof and Philipp Stinauer

Department of Computer Science III, Aachen University of Technology,
Ahornstr. 55, 52074 Aachen, Germany

Abstract. This paper deals with *eHome systems*. We focus on the *service composition* in order to reduce complexity and to leverage maintainability and extensibility of eHome services. Talking about services, we mean any piece of software, which is executed in a *network environment*, making the usage and administration of ubiquitous appliances easier. Current situation is, that the complete functionality is hard-coded into services without the facilities to be extended or reused. Many logical correlations (e.g., how to react if an alarm condition is raised) are made explicit in an inappropriate way. To tackle this problem, we introduce a declarative approach to specify logical correlations and to combine functionalities and services to new services, offering the required flexibility and comprehensiveness.

1 Introduction

As technology emerges to everyday life, it is brought to households, too. There is a variety of areas, in which the usage of technology (i.e., smart devices and computers) makes sense. Services of the first area target the comfort of inhabitants, e.g., remote access and automatic control of appliances. Services in the security area can be surveillance of the house or alarming the house owner if something unexpected happens. Communication services comprise enhanced electronic mail and instant messaging. Services in the health-care area targets for example instructions in the case of illnesses or diets. Infotainment stands for video on demand and similar. Consumption targets the monitoring and optimization of energy consumption. Services in all these areas can be autonomous, i.e. the execution of them only depends on the computer system in the house. A natural extension would be the usage of remote systems providing information database and services. Services of this nature are also known as *value-added services*. Bringing an enhanced lifestyle to consumers and, in turn, bring new revenues to companies seems to open a new powerful market [1].

In this paper we will take a look at systems combining automated homes, called *eHomes*, with enterprises and virtual enterprises. We call these systems *eHome systems*. We focus on the *service composition* in order to reduce complexity and to leverage maintainability and extensibility of eHome services. Talking about services, we mean any piece of software, which is executed in a *network environment*, making the usage and administration of ubiquitous appliances easier.

Kirchhof M. and Stinauer P. (2005).
Service Composition in eHome Systems: A Rule-based Approach.
In *Proceedings of the 2nd International Workshop on Ubiquitous Computing*, pages 28-38
DOI: 10.5220/0002576700280038
Copyright © SciTePress

The scenario discussed in this paper is illustrated in figure 1. The connected home (1) at the bottom of the drawing is equipped with a so called *residential gateway* (2), a hardware device, which provides access to connection infrastructure and acts as runtime environment for the *service gateway*. The service gateway manages and runs eHome services. These services realize *eHome services* (3), which are then visible to the house owner. Examples of this kind are an automated air conditioning system, or an automated energy control and optimization system. The household is equipped with several devices. These are connected with arbitrary network protocols to the residential gateway in the same manner as the services inter-communicate (4). The gateway acts as the central intelligence of the eHome. From the consumer's point of view, the residential gateway is a maintenance-free computing device. The connection (5) to the service provider is realized through a dialup connection (e.g., DSL or cable modem). The services interacting *sporadically* with the provider are connected via the Internet. User interaction (6) is realized via different kind of devices, e.g. mobile phones, PDAs, terminals, and personal computers.

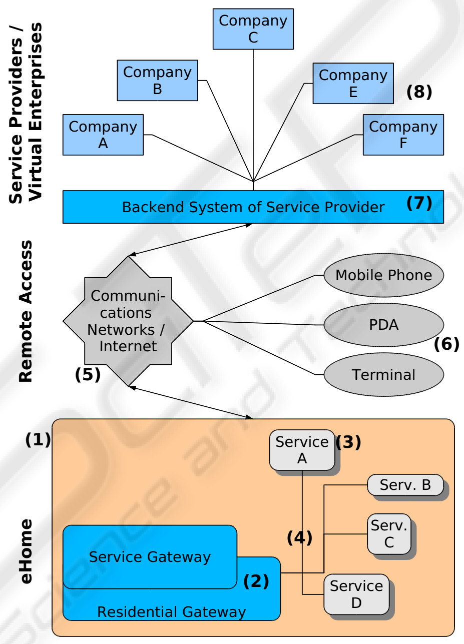

Fig. 1. Scenario

One often mentioned problem is the existence of many established home automation standards. While several papers address this topic, we use an OSGi-compliant gateway [2–4] 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. We are more interested in the mechanisms of *composite services*, rather than in the mechanisms of *local communication and service development*. To ease the development of eHome services, we rely on the Layered Approach to OSGi-based service gateways (PowerArchitecture) [5] and the Distributed Services Framework (DSF) [6].

One application of these technologies is a modular alarm system. The alarm system consists of multiple cameras, multiple sensors, an outbound connection for alarm messages (e.g., email facility, SMS), and some power switches. All components are connected to the residential gateway. The residential gateway integrates and coordinates the components. The procedure is the following: when some of the sensors, for example motion detectors, detect something worth mentioning, a predefined subset of available switches (also called actors) are activated and selected cameras should take pictures of the location and store them. The house-owner is informed about what is currently happening in his house in order to take adequate measures based on the kind of event and the pictures obtained from within his house. Possible actions would be to call the police in emergency cases or to discard the event, in situations where the cat raised the alarm accidentally. The system should be extensible by additional functionality. For example, the alarm system setup can be tweaked by the user in terms of which rooms should be monitored. Or, the way to alarm the owner should be made flexible.

Another example is a wakeup system, which leverages the comfort of the eHome users. The time to get up in the morning is usually determined -in a broad sense- by the first appointment of the next day. If the user sets up his alarm clock, many different factors are taken into account: the time needed in the bathroom, duration of having breakfast, other people sharing the bathroom, the traffic conditions, and so on. Based on the cumulative time needed for all these tasks, the wake up time is roughly estimated and is entered manually into the alarm clock. Other systems utilized in the morning like the water heating system, the alarm system, the independent vehicle heating, and similar are not been covered in this decision making. This would make the estimation too difficult and the required interaction with all the subsystems would be too time consuming. The point in time when to start these diverse systems is set to the earliest reasonable value and is triggered by a clock timer, which leads to a waste of energy by providing unnecessary capacities. In an enhanced solution, the user should still be able to enter a desired time to get up, but the application should support the user. For example, a warning can be raised if the selected time would lead to lateness at the first appointment or a time for wake up can be suggested. This suggestion can be based on the current traffic conditions, the personal preferences (e.g., breakfast, bath) and the times the bathroom is occupied by other residents. Preferably, energy consumption can be optimized by an accurate triggering of the water heating system and other systems requiring leadtime.

Current situation is, that the complete functionality is hard-coded into services without the facilities to be extended or reused. Many logical correlations (e.g., how to react if an alarm condition is raised) are made explicit in an inappropriate way. These conditionals are transferred into a procedural programming language and in turn become incomprehensible. To tackle this problem, we will introduce a declarative approach to specify logical correlations and to combine functionalities and services to new services, offering the required flexibility and comprehensiveness.

2 Rule-based Approach

Looking at an eHome from an abstract view, it consists of a set of sensors, actors, and eHome services. To offer functionality to the user, in most cases an eHome service has to monitor some sensors and control some actors. In our system, these devices are also represented by services (see section 3). Thus, the service has to compose other services to deliver the expected result.

Our approach separates service composition from services. Each composition is described declaratively by a set of rules responsible for flow control. Every rule consists of conditions and actions. For example, the alarm system introduced in section 1 consists of rules describing which conditions have to be fulfilled to trigger some action. For evaluating the conditionals, an execution environment called *rule engine service* is provided. For each composed service a rule set is installed in the execution environment. At runtime, the execution environment listens to events from other services, checks the conditionals, and executes the actions of satisfied rules.

The residential gateway can contain services in many different combinations. Thus, it is important not to have too many dependencies between services. To achieve loosely coupled services, a message-driven approach is pursued. Services can send and receive messages through a message service and do not depend on the existence of other services. In addition to the reduction of dependencies, it is also possible to replace a service with another one from a different vendor as long as the same message format is being used. Sensors send notification messages through the message service to the rule engine. The messages contain *service events* stating what has happened and where. Depending on the installed rule sets, the engine involves some action like an activation.

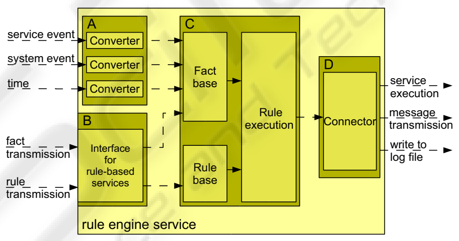

Fig. 2. Modules of the Rule Engine Service

Describing the composition declaratively by using a rule-based approach offers many advantages. It is the natural way of human thinking. So, it is easy to formulate rules for deciding which services to compose under which circumstances. Furthermore, having rules instead of a hard-coded imperative program improves variability. Rules can be replaced during runtime without having to rebuild and install a whole eHome service.

Having a closer look at the rule engine service, it consists of four main modules as depicted in figure 2. Module A is responsible for getting and converting information from the outside. Beside the introduced *service event* two other types of events exist.

System events inform about newly started or stopped services and *time events* supply the current time. The module converts every event, tags it with a timestamp, and saves it in the fact base located in module C. Rule sets can be installed and removed in the rule engine through the service interface offered by module B. They are separated from each other by different namespaces to prevent name collisions. Facts specific to a rule set are facts that belong to a specific rule set and exist only in the namespace of this rule set. Adding, modifying, and deleting of facts specific to a rule set is also possible. Module C evaluates the installed rules against the fact base in a periodic interval. If a rule matches, the appropriate action is triggered. The action can be to modify data, to execute services, or to send messages. Generating messages and calling eHome services is handled by module D. For tracking the system behavior during rule execution a logging facility is offered.

Conducting an eHome service from within the rule engine consists of *dynamic service selection and execution* at runtime. For this purpose, every service is described by some properties, e.g., name of the service, service type and location of the service. The values are partially set by the vendor and partially by the resident. When a service should be called from within the rule engine, it is selected dynamically using the service properties. Therewith, writing a music-follows-user service becomes quite simple: Having a sensor for user detection and a loudspeaker in every room, the composition consists of two rules. The first rule activates the speaker if the user enters the room. The selection of the speaker service depends on the location property transmitted by the sensor. The second rule turns the speaker off again after the user left the room and is stated analogous. As seen in this example, the advantage of dynamic service selection is that the developer can write more generic rules and thus reduce the number of rules which improves maintainability.

Two different possibilities for service execution exist. As explained in the music-follows-user example, exactly *one* service matching the properties can be called. But what if there are more loudspeakers -represented by services- in the same room? For this purpose a second method exists, which allows the developer to call *all* equal services matching the specified criteria. Using this *multiple service execution* all speakers in a room can be switched on with one statement from within the rule.

Having some variability from the rule execution environment to the outside, there is also some variability when an event coming from the outside is inserted in the fact base. One event can activate the same rule several times, because of other conditions in the rule which have more than one fact matching a condition. For example, a rule for a alarm clock consists of three conditions: (1) get current time fact, (2) get alarm time fact for a user, and (3) compare both times. If two alarm time facts exist in the fact base with same time values for different users, the rule engine will activate this rule twice and the appropriate action for each user will be executed. This makes it also possible to write more generic rules, where the action is based on some facts from the conditions. See section 3 for a extensive example from our prototype.

This approach offers the required flexibility and comprehensiveness. It is easy for the service developer to create new rule sets, because the execution environment provides all necessary data for triggering rules. Thus, the developer can concentrate on specifying the logic in a declarative language, which is natural in terms of the human way of thinking.

3 Software Architecture and Protoype

As stated, we use an OSGi-conform service gateway as a basis for our solution. This gives a quite complete component model of basic infrastructure services and makes an additional component-based development easy and naturally. Within OSGi, components are called *bundles*. Relevant for service and component development are two questions. Talking about component-based development, it is a design prerequisite to choose between (a) specialized components, which can not be used in other contexts, and (b) generic components, which can be used several times, but always with modifications necessary [7]. Applied to service development based on OSGi, the task is to strike the balance between isolation and sharing among bundles, which means that there is no reuse of basic components [4].

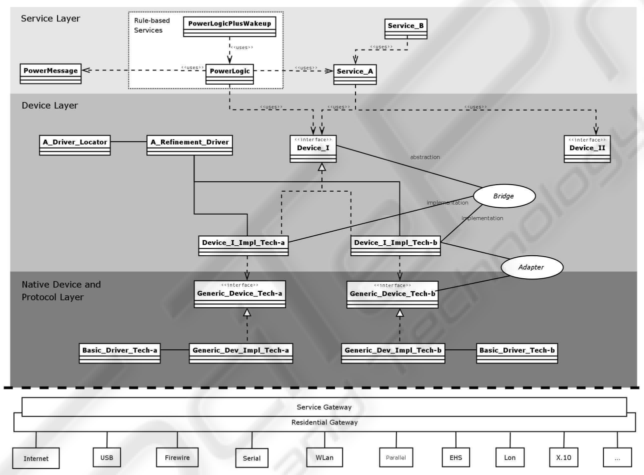

Fig. 3. PowerArchitecture with Rule-based Service

The architecture of the current prototype developed at our department is called PowerArchitecture [5] (cp. figure 3.) The lowest level comprises components mirroring functionality of native devices and network protocols. We named this layer *native device and protocol layer*. The middle layer is called the *device layer*. There the representations of device and protocols as *services* reside. These components wrap details visible at native device and protocol layer. This is also known as a structural design pattern called *facade* [8]. From an outside point of view, we have now an abstract view on the basis, the device infrastructure. OSGi's device access technology is integrated into both lower and middle layer, enabling to reflect the dynamic aspects of the eHome system. Seamless integration of device access and protocol drivers provide means to reflects the *dynamic of the system*.

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

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

PowerArchitecture describes a model for eHome services, especially for partitioning and integration of those. With this model, a structure is introduced, which separates the application logic of eHome services from infrastructure details. The so defined levels of abstraction build the basis for the rule-based approach. The application logic resides on the service layer. Thus, the composition of eHome services is reasonable for artifacts on the service layer.

The rule-based approach to service composition leads to the architecture shown in figure 3. The rule-specific parts are marked by a white box. The component PowerLogicPlusWakeup contains the rules and facts for the wakeup application. It utilizes the component PowerLogic, which realizes the rule execution engine. PowerLogic interacts with the eHome system via the functionality-driven interfaces on the Device Layer and the messaging system PowerMessage. The components required by the rule-based applications fit well into the purpose-build architecture PowerArchitecture.

The specification of rules provides an adequate way to formalize interdependencies. A rule consists of an if-part containing conditions and a then-part with desired actions to be taken if all the conditions are satisfied. An example rule from the wakeup system is shown in listing 1.1. The if-part of the rule consists of four conditions: (1) get a resident profile fact where the alarm is turned on (`alarmActive TRUE`) and save user name and wakeup time in local variables (starting with question mark) for later use, (2) get a wakeup action fact for *this* user, (3) get the time fact which contains the current time, and (4) check whether current time and wakeup time are equal. If all of these conditions are satisfied the action-part behind the `=>` is triggered. *?powerLogic* is a reference to module D of the rule engine service (see figure 2) and `runServices` the method for service selection and execution. This method selects and executes services matching the given class name, property filter, and method signature. For each matching combination of resident profile, wakeup action, and current time the action-part is executed independently.

The realization of PowerLogic is based on Jess [9], a Java rule engine. For integration into the system, we developed three different ways: First, the integration can be message-driven. PowerLogic can be triggered by messages as well as other components can be triggered by messages. Second, method invocation can be used. Third, data in the eHome system can be bidirectionally reflected within the facts of the rule engine.

Listing 1.1. Rule Cutout from PowerLogicPlusWakeup

```
(defrule wakeup-alert
  (resident_profile (alarmActive TRUE)
    (name ?name) (wakeupTime ?wTime) )
  (wakeup_action (name ?name)
    (wakeupClass ?wClass)
    (wakeupFilter ?wFilter)
    (wakeupMethod ?wMethod)
    (wakeupParameter ?wParameter) )
  (MAIN::current_time (time ?cTime) )
  (test (= ?wTime ?cTime) )
=>
  (?powerLogic runServices
    ?wClass          ; class name
    ?wFilter         ; service property filter
    ?wMethod         ; method to call
    ?wParameter      ; parameter list
  )
)
```

This enables the usage of the most appropriate realization strategy per application while maintaining the software architecture of the system (PowerArchitecture).

4 Related Work

Several approaches in the field of service composition exist. In this section, we will give an overview of these and point out the advantages of our solution. As we will show, none of the current developments offers a manageable and complete solution to the specific problems in the area of eHome systems.

eFlow [10] composes services by defining a workflow process which is modeled by a graph. The graph consists of service, decision and event nodes. A decision node allows the parallel execution of different services and synchronizes the flow afterwards. An event node receives event notifications from services. A service node can contain a service or a composed service described by another workflow graph. To execute equal services in parallel, eFlow has the concept of multiservice nodes. When creating the workflow process, it is not possible to know all combinations of services for the workflow. Therefore, eFlow offers the concept of generic service nodes, which allow place holders in the graph to define the services at runtime. A comparison between eFlow and the rule-based approach shows many similarities. Multiservice nodes are equivalent to the runServices method. Generic service nodes are similar to generic rules which are instantiated through facts. Both solutions offer dynamic service selection. Main difference is the way the flow is defined. eFlow uses a workflow graph to identify the sequence of service execution. The rule engine allows complex conditions to combine events, which is not possible in eFlow.

SELF-Serv [11] is a framework for peer-to-peer service composition in distributed systems. A statechart containing states and transitions describes the composition. The transitions can have Event-Condition-Action rules to control the flow. The states can be divided into simple and composed states representing simple and composed services where composed services are statecharts again. Every service has a wrapper responsible for life cycle management of this service and a coordinator responsible for handling the flow. A complex service configures the coordinators of the needed services to create the composition. At runtime, the coordinators communicate with each other peer-to-peer. The advantage of this solution is that there is no central unit for flow control at runtime and the system scales better than central systems. Comparing SELF-Serv with the introduced rule-based approach shows that both languages have the same expressiveness. Our eHome system can not taken advantage of the peer-to-peer concept, because within the eHome a centralized approach is pursued by relying on the idea of residential gateways. The coordinators would only lead to an unnecessary overhead in the system. The same applies for SELF-Serv's wrapper: it is not needed, because the life cycle is managed by the underlying platform.

The *Ninja* project [12] has the goal to build a scalable and robust service execution platform. The architecture consists of the four elements base, unit, proxy, and path. Services are executed in bases. A base can be a server or a cluster of servers. Units are devices used to communicate with services. Proxies convert data between services and units. A path describes the composition of units, services and proxies to let the user interact with a service through a given unit. Ninja offers automatic path creation for a given set of items. Beyond, a service can be composed of different stages. A stage is a small component which can be independently executed within the cluster. Using stages improves scalability. The composition through path creation is static. Ninja can not react to events within path execution. In contrast, our approach offers dynamic composition during runtime. Service execution is triggered by events, which offers a more suitable way for the eHome environment. Nevertheless, scalability and robustness through stages are interesting features for a composition environment and will be further researched for incorporation in our approach.

Web services [13] offer interoperability between various software applications running on various platforms. For composition of web services, several approaches exist, e.g. Business Process Execution Language (BPEL4WS) [14], Web Services Choreography Interface (WSCI) [15], WS-Coordination [16], and WS-Transaction [17]. The granularity of web services differs from the granularity of eHome components. Web services are adequate to integrate external Internet-based services. So, the composition techniques are concerned with problems of remote invocation and remote interaction, but not with the evaluation of multiple aspects within a dynamic environment.

5 Summary and Outlook

In this paper we discussed the composition of eHome services. We introduced a rule-based approach, which can overcome the problems of combining functionalities and appropriate specification of logical correlations. First and foremost, it fits very well into an OSGi-based residential gateway and second it has a clear and straight-forward declarative programming interface. This eases the introduction of the rule-based approach into OSGi-based solutions. Thus, it insures the investment in the large number of OSGi-based residential gateways in the intended mass-market and it provides means for continuous development and realization of eHome systems.

The concept has been proved with a prototype. The implemented functionality has shown, that the rule-based approach eases the realization and extension of eHome services. The separation of concerns is preserved throughout the whole system. The integration into the eHome system is enabled by three means: method invocation, data manipulation, and message-driven. Thus, the components of the eHome system are loosely coupled within the architecture.

While the rule-based approach is very promising, problems are to be expected for maintainability if the composition of eHome services are taken onto a very high level. Such problems fields cover scalability issues, security issues, context issues, and conflict situations. Also, ensuring that the system is available and failsafe is an open problem. Other areas for future work are the problems of a data layer covering the distribution aspects while addressing security issues. Furthermore, integration aspects have to be observed in the domains of configuration and deployment management and the conceptual modeling of buildings.

We do feel confident, that the proposed approach provides a flexible and extensible solution to the problems in composite eHome systems. Several eHome-services are developed according to PowerLogic to further validate the proposed approach. The described service PowerLogicPlusWakeup is currently evaluated by the project in-Haus [18] in Duisburg, Germany.

References

1. Sun Microsystems: The Connected Home. `http://www.sun.com` (2002)
2. Open Services Gateway Initiative: OSGi Service Platform Specification. (`http://www.osgi.org/osgi_technology/download_specs.asp` (2.3.2005))
3. Gong, L.: A Software Architecture for Open Service Gateways. IEEE Internet Computing **5** (2001) 64–70
4. Chen, K., Gong, L.: Programming Open Service Gateways with Java Embedded Server Technology. Addison-Wesley Professional (2001)
5. Kirchhof, M., Linz, S.: Component-based Development of Web-enabled eHome Services. In Baresi, L., Dustdar, S., Gall, H., Matera, M., eds.: Proceedings of Ubiquitous Mobile Information and Collaboration Systems Workshop 2004 (UMICS 2004). Volume 3272 of Lecture Notes in Computer Science., Springer (2004) 181–196 Revised Selected Papers.
6. 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). Volume RA221 W0411-084 of IBM Research Report., IBM (2004) 123–131 Refereed Papers.
7. Szyperski, C.: Component Software. 2 edn. Addison-Wesley/ACM Press (2002) ISBN 0-201-74572-0.
8. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley (1995)
9. Friedman-Hill, E.: Jess, The Rule Engine for the Java Platform. (2003) Version 6.1.
10. Casati, F., Ilnicki, S., Jin, L., Krishnamoorthy, V., Shan, M.C.: Adaptive and Dynamic Service Composition in eFlow. In Wangler, B., Bergman, L., eds.: Advanced Information Systems Engineering: 12th International Conference, CAiSE 2000, Proceedings. Volume 1789 of LNCS., Springer (2000) 13–31

11. Sheng, Q.Z., Benatallah, B., Dumas, M., Mak, E.O.Y.: SELF-SERV: A Platform for Rapid Composition of Web Services in a Peer-to-Peer Environment. In: Proceedings of The 28th International Conference on Very Large Data Bases (VLDB02), Morgan Kaufman (2002) 1051–1054
12. Gribble, S.D., Welsh, M., von Behren, J.R., Brewer, E.A., Culler, D.E., Borisov, N., Czerwinski, S.E., Gummadi, R., Hill, J.R., Joseph, A.D., Katz, R.H., Mao, Z.M., Ross, S., Zhao, B.Y.: The Ninja Architecture for robust Internet-scale Systems and Services. Computer Networks **35** (2001) 473–497
13. Gottschalk, K.D., Graham, S., Kreger, H., Snell, J.: Introduction to Web Services Architecture. IBM Systems Journal **41** (2002) 170–177
14. Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Satish Thatte, I.T., Weerawarana, S.: Business Process Execution Language for Web Services Version 1.1 Specification. `http://www.ibm.com/developerworks/library/ws-bpel/` (24.02.2005) (2003)
15. W3C (World Wide Web Consortium): Web Service Choreography Interface (WSCI). `http://www.w3.org/TR/wsci` (24.02.2005) (2002)
16. Cabrera, L.F., Copeland, G., Cox, W., Feingold, M., Freund, T., Johnson, J., Kaler, C., Klein, J., Nadalin, A., Orchard, D., Robinson, I., Shewchuk, J., Storey, T.: Web Services Coordination (WSCoordination) Specification. `ftp://www6.software.ibm.com/software/developer/library/ws-coordination.pdf` (09.08.2004) (2003)
17. Cabrera, F., Copeland, G., Cox, B., Freund, T., Klein, J., Storey, T., Thatte, S.: Web Services Transaction (WS-Transaction) Specification. `http://www.ibm.com/developerworks/library/ws-transpec/` (09.0ß8.2004) (2002)
18. inHaus Duisburg: Innovationszentrum Intelligentes Haus Duisburg. (`http://www.inhaus-duisburg.de` (22.6.2004))