

# A Serviceware Framework for Designing Ambient Services

Benjamin Hirsch, Thomas Konnerth, Axel Heßler, Sahin Albayrak  
DAI-Labor

Technische Universität Berlin

Email: {benjamin.hirsch|thomas.konnerth|axel.hessler|sahin.albayrak}@dai-labor.de

**Abstract**—This paper describes a serviceware framework for the creation of context-aware ambient services. Different communication standards can be found today through a multitude of devices are wrapped by a service execution engine which unifies access and service provision across different service domains. The framework provides basic building blocks for added security, multi-modal interaction, management, and comes with tools and a methodology.

## I. INTRODUCTION

During the last few decades, computers went through a remarkable evolution. In the beginning, they were huge machines which needed experts to use and interact with them. As recently as 10 years ago, computers were found in many households but still needed some expertise to be used. Nowadays, computing ability is being embedded in all sorts of devices, more often than not invisible to the user. At the same time, the interconnectedness of those machines has risen accordingly. We have reached a point where it soon will be difficult to buy a product that does not have some sort of computing ability and interface to some host machine. As heterogeneous as the devices are the possibilities of connecting them. Increasingly, this connection is based on ip-network and wireless technologies.

However, while devices are becoming smarter and get the ability to interface, they are still designed as single machines. Interaction between them is either impossible or very limited. Not only are different standards (such as UPnP [8], OSGi [3], webservices [21]) used, but devices generally have neither a notion of context nor the ability to interact autonomously, or with each other. As a result, users are being forced to interact with a growing number of devices and entities, many of which have overlapping functionalities [20].

For a number of years now the concept of ubiquitous computing has been a research area in its own right (e.g. [22]). However, while seamless connectivity is indeed an important aspect, we should not lose sight of the fact that it is not the solution to containing the complexity of our lives, and in fact adds to it. What is needed are new ways of containing the chaos and complexity without losing the advantages that modern technology gives us [13].

The vision of ambient intelligence builds on the ubiquity of computing devices, but while the focus of ubiquitous computing lies in the embedding and constant availability of computation, ambient intelligence aims at making use of those

entities in order to provide the user with an environment which offers services when and if needed by the user [2].

To this end, we propose a serviceware framework that allows to wrap the services being offered to us by new technologies, and create an environment which, while providing the all functionalities, does so in a human-centric way, combining and choosing between devices and services depending on the user, and the context of the user. Central to this is the concept of service, as opposed to device. In the remainder of this paper we will first go into a little more detail on the requirements for ambient intelligent services, and then proceed to describe our serviceware framework, which we believe is a (first step of a) solution to providing ambient intelligence to users today.

## II. REQUIREMENTS FOR AMBIENT SERVICES

In order to develop ambient intelligent services, several preconditions need to be met. In the current state of affairs, home and office environments provide a host of devices, many of which have some IP interface. The range goes from mobile phones and PDA's to IP-enabled cooking devices and light switches to media centres to TV's and computers. We can classify these devices as multi-purpose and usually computable devices (like computers, phones, and PDA's), and "simple" devices who usually provide some sort of monitoring and control interface, but whose purpose is well defined.

Currently, a number of different protocols are employed, such as UPnP, OSGI, but also webservices and SOAP [11]. Common to them is that they have some notion of *service* that is provided to the network. However, while structures related to services are mostly implemented in those protocols, such as some form of service directory, or service call, it is instantiated in different ways. For example, an OSGi service essentially is some Java object with defined interfaces, while UPnP equates service with some device that provides it and the protocols to find and use it. Common to them however is that the service description

- is narrowly defined to cover only elements relevant to the task at hand,
- is semantically weak, i.e. has little or no semantic information, and
- is designed for machines.

In order to create an environment in which devices become less important and instead the services which they are providing

All those devices provide services to other devices, but also to the human users.

There are some aspects to ambient services that we will, in this paper, not focus on, but which are necessary to implement ambient services. The most important aspect is the one of user interface. As we have mentioned before, the goal of ambient services is to abstract away from device and towards context-aware service provisioning. Here, it is crucial that services can interact with the user using a number of different modalities, depending on the location, context, and availability of devices. For example, the address book service will present information with picture, eMail, and addresses when accessed by the user from a notebook computer, just the number and name when accessed from the user via a mobile phone with a small screen, and via a XML-message if accessed by another service as a step in a service-chain. Information about our approach to multi-modal service interaction can be found in e.g [17].

The availability of data is another aspect that needs to be addressed. In our scenario, as well as in the conception of the framework, we essentially assume that all services and devices are more or less constantly available. However, although this is a reasonable working assumption, provisions should be made for the case when connectivity fails. For example, it should be possible to use the address book in the home GAP-phone even if the server keeping the address book data is down for maintenance. This can of course be achieved by making use of a built-in address book service of the phone, but ensuring that the "main" address book service is always synchronised with the phone is not always straight forward.

Security and authentication issues are of extreme importance. If users cannot trust that the services they use in an ambient environment are private and secure, they will reject them, and ambience will never come to pass. On the other hand, secure and simple ways of communicating with and between services is a necessary element of ambient services. Authentication, single sign-on, public-key infrastructures, and encryption of content are all issues that need to be addressed, but would lead too far in the context of this paper. Let it be said that the framework we propose has components that address security and AAA (accounting, authentication, and authorisation).

### III. THE SERVICEWARE FRAMEWORK

Any framework for intelligent ambient services will have to provide solutions for many of the issues that derive from the domain. We will try to sketch the major challenges and propose a solution that addresses these issues.

#### A. Incorporating multiple domains

Taking a look at the current service landscape one quickly realises that today's services are powered by different technologies and server architectures like OSGi, JEE [16], .NET [5] and webservice technologies. Given the fact that providers and operators have already invested much into these systems and that existing software is unlikely to be adapted, we propose a solution that is able to incorporate those existing

technologies. At the same time, our architecture will be open to future service technologies.

The general design of the framework is sketched in Figure 1. It consists of a service engine which interprets and executes our service description language, as well as a number of adapters that connect the engine to actual service providing entities and devices that are available in today's home and business environment. Each adaptor, as shown on the left of Figure 1, consists of domain specific invocation and control mechanisms, as well as a domain specific service directory which will be synched with the rich service directory of the service engine. The rich service directory, together with the engine, allows for example to expose UPnP-driven services to web-services. This way, a simple rule allows to output life weather data on a UPnP enabled television set. Note also that while we use JIAC IV [9] as a current implementation of the serviceware framework, we aim to extend the system, and view JIAC IV as one of the service providing entities. We give more detail on the current state of the implementation in Section IV.

#### B. Heterogenous Networks and physical Distribution

From a users point of view, having to deal with different and incompatible access networks as well as physical distance is quite inconvenient. End users do not want to know about networking technologies, nor do they want to see them. Therefore a framework must not only be able to channel communication through different networks, it will also have to provide an abstraction of communication that enables the developer to implement the communication independent of network technologies. The adaption to the actual network, processes like address-translation and failure handling, and of course the selection of the optimal network in case of multiple networks being available, should be handled by the architecture, thus allowing the user as well as the developer to forget about network technologies.

#### C. Active Service Composition

Our central concept for the realisation of an intelligent and dynamic framework for ambient services is a Service Description Language (SDL) that describes not only the structure of a service but also its functionality in semantical terms. We are well aware, that there are already a number of (web-)service co-ordination and orchestration languages available on the market, such as BPEL4WS [6] or BPML [12] (to name but a few). However, we feel that these languages either merely aggregate function-calls without including the proper semantics that is needed for automatic service composition, or they focus too much on the theoretical aspects of service composition, thus being not very useful in actual development.

#### D. Service Description Language

We will now try to give a brief overview over the different aspects and elements that should be covered by the SDL. Following that, we will sketch the possible development-process with such a language, and afterwards give some examples of how the framework will modify and complete the designed system to adapt to the user-context.

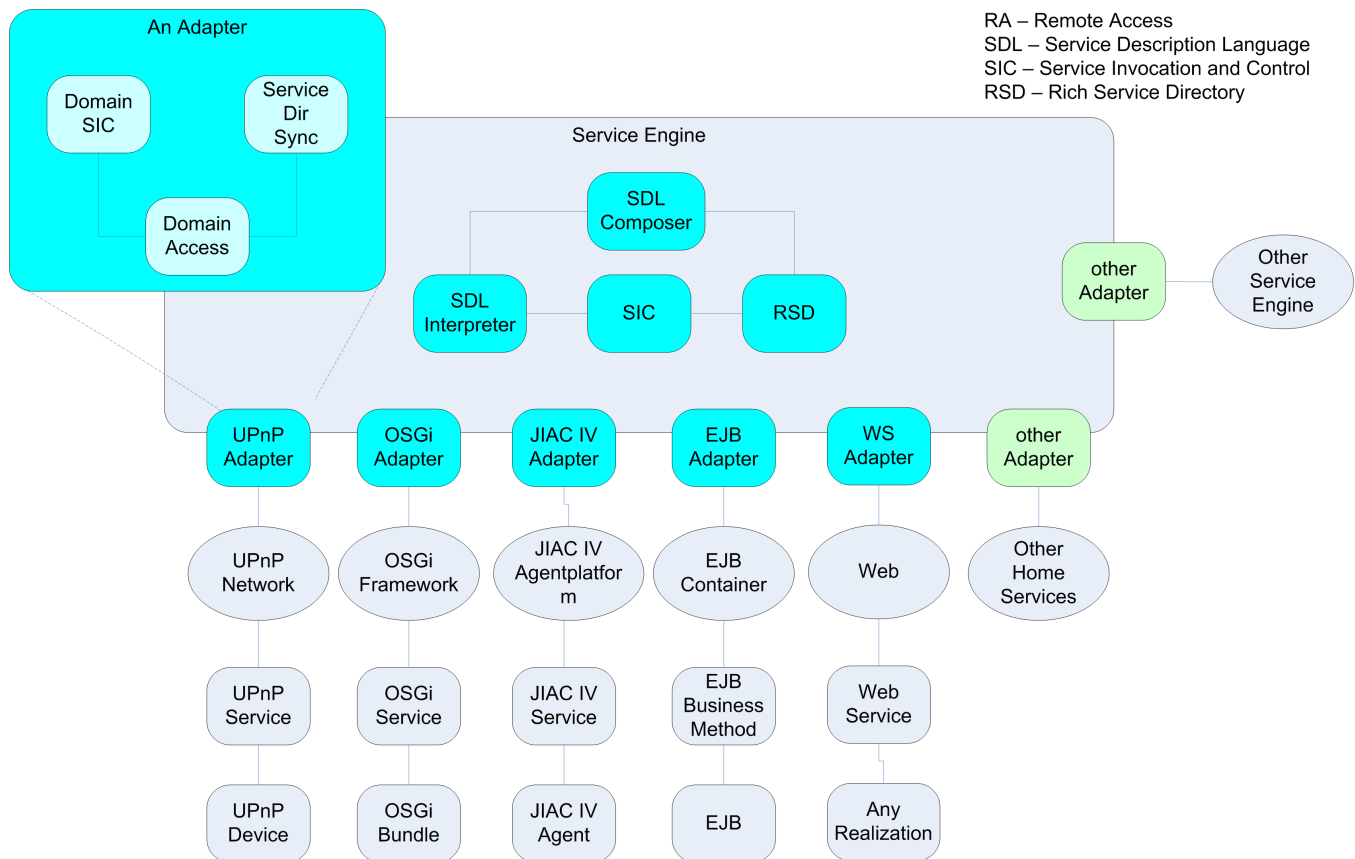


Fig. 1. The general design of the serviceware framework.

1) *Ontologies*: Ontologies are possibly the most important, and most difficult aspect — not so much because of their complexity, but because of the difficulty of describing the world. From the ideas of interoperability and openness follows the requirement that provider and consumer (to stay in the service domain) speak the same language, and interpret utterances the same way. While it may seem simple to create a common data structure used by both sides, the idea of openness requires this data structure to be publicly available and generally accepted. Therefore, current standardisation-efforts as well as the already available repositories of the ontology-community are well suited for our needs. Furthermore, if we want to apply some sort of semantical matching for service requests, ontologies are the best means to provide semantical information about the request, which can be used during the matching process.

2) *Declarative Expressions*: In order to provide context-aware services, it must be possible to describe goals, or states-to-be, rather than actions, or things-to-do. Therefore, any service description language must include a (again semantically enhanced) pre- and post condition, i.e. the state changes that executing the service will bring about. BPEL3WS for example does not allow to describe desired states, and instead forces the user to describe a sequence of service calls. This way of describing components and states has two advantages that are vital to our approach. On the one hand, a developer may

state what functionalities he needs to complete his service and the tools or a repository may find all services that can possibly help the developer at once. Compared to searching the repository simply for signatures (as it is common today), this semantical approach will return fewer and better matches.

3) *Procedural Description*: In order to create complex control- and data-flows, a description language has to support the necessary constructs to express those. This includes things like sequential or parallel execution, as well as case differentiations and loops. This is common knowledge today — however, it is vital that any Service Description Language includes such constructs.

4) *Composition of Services*: An important part of any service architecture is the composition of services. We think, that while the developer should design his application from start to end by combining and composing services, it is also important that the designed control-flow is loosely coupled. By using semantical descriptions for the parts of an application, rather than tightly linked calls, it will be possible to exchange and replace functionalities at runtime. This will be of advantage especially in open and dynamic environments, where the availability of services and providers may not be foreseen.

#### E. Designing complex Services

Given the descriptions of the last section, we will try to sketch a scenario of how a SDL may be used in development

of context aware services. Let us assume, our developer wants to create a service that looks up cooking recipes from the web, filters the available recipes according to the users preferences, and afterwards displays the recipes on a device that is close to the user.

So the list of required sub-services would look like this:

- *Fetch\_Recipes\_Service*
- *Fetch\_UserPreferences\_Service*
- *Filter\_Recipes\_Service*
- *Find\_nextScreenToUser\_Service*
- *Present\_Recipes\_OnScreen\_Service*
- *UserSelect\_Recipe\_Service*
- *Present\_Recipes\_ViaVoice\_Service*

Now let us assume that the *Fetch\_UserPreferences\_Service* already exists from a different Scenario and that there is a generic Service for localisation (*Find\_nextScreenToUser\_Service*), as well as services for handling of data (*Present\_Data\_OnScreen\_Service*, *UserSelect\_Data\_Service*).

The development process for creating a recipe service would look as follows:

First, the developer creates the core functionality of his application, i.e. he creates a Service that looks up the recipes, as well as filtering mechanisms for the retrieved recipes according to a user profile. He can do this without knowledge of the other services, as all data handled by the framework is represented by ontologies that are publicly available.

During the development of the filtering service, the developer realises that the profile-data needs to be retrieved in order to be available for filtering. Thus he queries the framework for existing services that return user profiles. The framework (in this case the development environment) responds with the *Fetch\_UserPreferences\_Service*, and the developer only has to include this service into his process model.

This first scenario describes a process that is omnipresent in software-development. The developer needs a functionality with a certain signature, looks it up and includes it into his application. While it is nothing special to look up services by their signature, we are trying to enhance this look-up process by introducing semantical descriptions of the services, and this improve lookup speed and results. To illustrate this, let us continue our example:

Once the developer is sure to have a selection of appropriate recipes, he needs to have the user select a recipe. Therefore the recipes need to be presented to the user. Accordingly, the developer looks for a service that is able to present recipes. Let us assume now that no such service exists yet — however, there is a generic service for the presentation of data on a screen. Even though the developer was looking for services that take recipes as input, the framework is able to recommend the *Present\_Data\_OnScreen\_Service*, as recipe is a subcategory of *Data*, and the semantic matching engine is able to make this generalisation.

Furthermore, when the developer tries to use the Service *Present\_Data\_OnScreen\_Service*, he is informed, that this service needs a display-device as input. He looks up

services that return screens in a house, others finds the *Find\_nextScreenToUser\_Service*, which not only returns an available screen but also the one that is closest to the user.

To complete our service, the developer now only needs to include the service for selection of recipes (*UserSelect\_Recipe\_Service*) which is a user-interaction service, and thus available in the same way as the data-presentation service. To keep things simple, our developer decides to re-use the same display service for the final recipe that he already used for the list of available recipes. Thus the application is finished.

So far, our scenario describes, how it is possible to design complex applications in a rather simple and direct way. The developer can concentrate on the control- and data-flow, and does not have to worry about the internals or the data-types of individual services. However, we can expand this scenario even further.

Once the service is deployed, a user may start it, select a recipe and afterwards move from the living room to the kitchen, as this is the place where he would actually need the service. Now lets say that there is no display in the kitchen, but only a speaker system. As the developer stated, that the recipe should be displayed on the nearest screen to the user, this would normally lead to either displaying the recipe in the wrong room, or, if the display service explicitly states that the user should be in the same room as the screen, a failure of the service. However, given a semantical description of the action, i.e. that the data should be presented to the user, the runtime engine may now decide to use a different display method rather than letting the service fail. Now the *Present\_Recipes\_ViaVoice\_Service* may be semantically equivalent to the *Present\_Data\_OnScreen\_Service*, and thus be selected by the system. Thus the recipe is read to the user, rather than displayed on a screen and is still successful. [17] describes a multi-modal user-interface that would be able to achieve the scenario described.

## F. Managing Services

From an operators point of view, maintenance of a system is something that definitely has to be addressed. While the deployment of a service on a running infrastructure is only the beginning, monitoring and control of active services also play an important role. This is especially important for intelligent and autonomous services, as these may show all kinds of unexpected behaviours that are hard to track down. Today, the management-concepts of networks and telecommunication-systems are quite advanced. We propose to use and adapt these techniques for high level services. A part of this infrastructure has also been implemented JIAC.

## G. User Access to Services

An important aspect of services is their accessibility. As the name *ambient services* suggests, we will not only try to make services available to the user always and anywhere, but we will also try to make them adapt to the situation and the location of the user. To achieve this, it will be necessary to enable the framework to provide access to all kinds of devices.

Furthermore, it has to be able to choose an appropriate user interface as well as a reasonable amount of information for that user interface. As an example, for a voice based email-service, it is certainly better to just report the subjects of the received emails, rather than read the whole email to the user. We refer to [17] for an example of multi-modal user interaction.

User data and profile-information are another aspect that should be considered by a framework. While in the modern world, users are already used to storing much of their data in computer systems, the accessibility of this data is still an issue. If you take a simple address book as an example, many people have different address books for their phone, for work and for their mobile phone. While it may be quite sensible to separate this data, synchronisation and accessibility is definitely an issue, e.g. if you want to keep the entries of your telephone at home synchronised with the entries of your mobile phone. On the other hand, if you can manage to wrap these address books with services, that allow you to access the information, and at the same time find ways to make these services accessible anywhere, synchronisation is not much of an issue any more. As this is likely to hold true for other data as well, our framework will have to provide means to access and propagate data from anywhere.

#### IV. CURRENT STATUS

The last few sections described work in progress. However, we do not start from scratch, and base our ambient framework on research and work that has been done already. Specifically, we use our agent-based serviceware framework JIAC [9] as a starting point.

##### A. The JIAC Framework

JIAC IV [9] is a FIPA-compliant [7] agent framework JIAC (Java-based Intelligent Agent Componentware). In the following we will describe the characteristics of this agent framework. The framework has been used to implement a number of different projects, such as a personal information agent [1] and entertainment solutions [23].

JIAC consists of a run-time environment, a methodology, tools that support the creation of agents, as well as numerous extensions, such as web-service-connectivity, device independent interaction [17], an owl-to-Jadl translator, a OSGI-connector and more. An agent consists of a set of components, rules, plan elements, and ontologies. Strong migration is supported.

JIAC's component model allows to exchange, add, and remove components during run-time. The components interact among each other by agent internal messages. Standard components (which themselves can be exchanged as well) include a fact-base component, execution-component, rule-component, and more [19]. These components provide individual messages to manage the appropriate actions, e.g. the *MonitorGoalMessage* executed by the *GoalBean* allows to subscribe for changes on the goal stack. A JIAC agent is defined within a property file which describes all elements that the agent consists of.

JIAC is the only agent-framework that has been awarded a common criteria EAL3 certificate, an internationally accepted and renowned security certificate [10].

Agents are programmed using the language *Jadl* [15]. It is based on three-valued predicate logic [14], thereby providing an open world semantics, and implements an BDI [4] approach. It comprises four main elements: *plans elements*, *rules*, *ontologies*, and *services*.

Communication is based on services. An agent, in order to achieve a goal, first tries to find a plan element whose effect matches the goal. If this plan element has pre-conditions, a planning component creates a plan which has to be executed in order to achieve the goal. Plan elements can be atomic actions, complex actions (written in *Jadl*, which in turn can trigger goals), and services. This means that agent communication happens transparent to the planning component. A service invocation consists of several steps, and is based on a meta-protocol, an extension of the FIPA request protocol. First, the agent contacts the Directory Facilitator (DF) in order to receive a list of possible services that could fulfil the current goal. After selecting a service and informing the DF, the agent receives a list of agents providing that service. Now an optional negotiation protocol can be executed with which the actual service provider is chosen. Only now the actual service protocol is executed. The meta protocol handles service-independent elements like security, accounting, communication failures, and other management-related issues.

##### B. Methodologies and Tools

JIAC provides a set of tools which simplify the creation of agents. This includes (Eclipse-based) textual and graphical tools for the design and creation of agents and their elements, such as ontologies, plan elements, services, and goals. Furthermore, there exist tools that support various extensions of JIAC like a security infrastructure.

Our methodology represents a process guiding the service developer from defining early requirements up to deploying an solution. We use the concept of agents which provide services to guide the developer. This way, large dynamic systems can be modularised and its complexity can be controlled.

Following this methodology, the service developer is encouraged to develop first those parts that are not captured by our domain library. He is supported by a life-cycle model, enabling him to identify urgent and important parts of his service application, to implement the whole application iteratively and incrementally and to obtain customer's feedback constantly.

In more detail, the methodology starts with defining the domain and interaction vocabulary (i.e. the ontology) and use case modelling as a means of requirements formalisation. It continues with a role modelling process where it allows to form encapsulated functionality and services as interfaces with feedback loops to adjust the ontology. Subsequently a developer has to bundle his roles and have agents instantiate the roles in order to form an ambient environment. During implementation and deployment the methodology supports

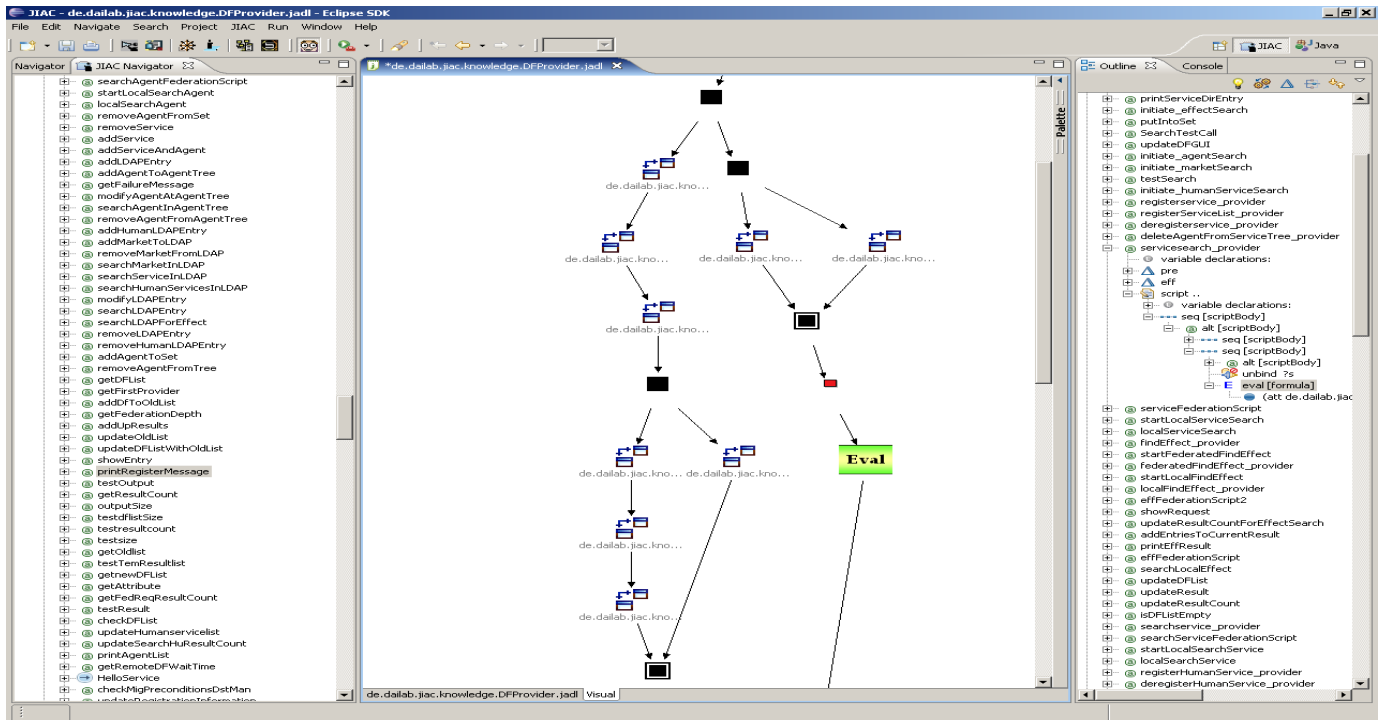


Fig. 2. Our Service Development Environment

agent unit testing, a generic debugging concept, and service component monitoring functionality.

The methodology is role-based. A role consists of several components with the capability to be adapted to a concrete environment and simply be plugged into a component. The main focus herein lies on reusability. The methodology relies on a repository allowing a developer to reuse his components, their reasoning, interaction and security capabilities, infrastructure, and management functionality as well as his domain specific capabilities.

A set of tools that are built into an integrated development environment (see Figure 2) guides the user through the steps defined by the methodology.

A repository can be directly referenced by the IDE during the development and deployment phase. A service developer can freely design the flow of control using existing service components. Data flow will be interfered from the ontological dependencies.

## V. CONCLUSION AND OUTLOOK

In this paper, we have described a serviceware framework which we believe can be used as a basis for the creation of provision of ambient services. We have identified a number of requirements that need to be met by the framework. As today's devices use a number of different service concepts, the framework needs to not only be able to talk to the devices, it needs to have a service description language that encompasses the service concepts of the device-specific protocols. This way, the services provided by the devices can be used in arbitrary

order and connection. Furthermore, some basic requirements such as security, need to be supported by the framework.

The described framework is currently under development — as we base our design and parts of the implementation on the JIAC IV framework that has been developed by us, we have already solutions for many of the issues that a serviceware framework for ambient services will need to address, such as multi-modality, security, and more. The integration of current home and enterprise-technologies in order is one of our priorities now. As described in this paper, we develop a service description that encompasses different service domains and allows to address services (of any of the domains) in a semantical fashion. While we acknowledge that there are many unsolved issues, and elements that we do ignore that the moment, we believe that our system will be yet another step in the direction of providing ambient intelligence to the user.

## REFERENCES

- [1] S. Albayrak and M. Dragan. Generic intelligent personal information agent. In *International Conference on Advances in Internet, Processing, Systems, and Interdisciplinary Research*, 2004.
- [2] M. Alcañiz and B. Rey. New technologies for ambient intelligence. In Riva et al. [18], chapter 1, pages 3–15.
- [3] O. Alliance. Osgi service platform core specification, release 4, August 2005.
- [4] M. E. Bratman. *Intentions, Plans, and Practical Reason*. Harvard University Press, Cambridge, MA, 1987.
- [5] M. Corporation. .net. <http://www.microsoft.com/net/>.
- [6] I. developerWorks. Business process execution language for web services. <http://www-128.ibm.com/developerworks/library/specification/ws-bpel/>.
- [7] FIPA. FIPA ACL Message Structure Specification. FIPA Specification SC00061G, Foundation for Intelligent Physical Agents, Dec. 2002.

- [8] U. Forum. Device security and security console standardized dcps. <http://www.upnp.org/standardizeddcps/security.asp>.
- [9] S. Fricke, K. Bsufka, J. Keiser, T. Schmidt, R. Sessler, and S. Albayrak. Agent-based Telematic Services and Telecom Applications. *Communications of the ACM*, 44(4):43–48, Apr. 2001.
- [10] T. Geissler and O. Kroll-Peters. Applying Security Standards to Multiagent Systems. In M. Barley, F. Massacci, H. Mouratidis, and P. Scerri, editors, *First International Workshop on Safety and Security in Multiagent Systems*, pages 5–16, July 2004.
- [11] W. X. P. W. Group. Soap version 1.2. <http://www.w3.org/TR/2003/REC-soap12-part0-20030624/>, 2003.
- [12] B. P. M. Initiative. Business process modeling language specification. <http://xml.coverpages.org/BPML-2002.pdf>.
- [13] IST Advisory Group. Ambient intelligence: From vision to reality. In Riva et al. [18], pages 45–68.
- [14] S. C. Kleene. *Introduction to Metamathematics*. Wolters-Noordhoff Publishing and North-Holland Publishing Company, 1971. Written in 1953.
- [15] T. Konnerth, B. Hirsch, and S. Albayrak. Jadl - an agent description language for smart agents. *Proceedings of DALTO6*, 2006.
- [16] S. D. Network. Java platform, enterprise edition. <http://java.sun.com/javace/index.jsp>.
- [17] A. Rieger, R. Cissé, S. Feuerstack, J. Wohltorf, and S. Albayrak. An Agent-Based Architecture for Ubiquitous Multimodal User Interfaces. In *The 2005 International Conference on Active Media Technology*, 2005.
- [18] G. Riva, F. Vatalaro, F. Davide, and M. Alcaniz, editors. *Ambient Intelligence: The Evolution Of Technology, Communication And Cognition Towards The Future Of Human-Computer Interaction*. O C S L Press, March 2005.
- [19] R. Sessler. *Eine modulare Architektur für dienstbasierte Interaktion zwischen Agenten*. Doctocal thesis, Technische Universität Berlin, 2002.
- [20] W. Weber and J.M. Rabaey and E. Aarts, editor. *Ambient Intelligence*. Springer Verlag, 2005.
- [21] W3C. Web services activities. <http://www.w3.org/2002/ws/>.
- [22] M. Weiser. Hot topics: Ubiquitous computing. *IEEE Computer*, 1993.
- [23] J. Wohltorf, R. Cissé, and A. Rieger. BerlinTainment: An agent-based context-aware entertainment planning system. *IEEE Communications Magazine*, 43(6):102–109, June 2005.

## VI. ABOUT THE AUTHORS



Prof. Dr. Sahin Albayrak holds a professorship chair for Agent Technologies in Business Applications and Telecommunication (AOT) at the Technische Universität Berlin. He is also head of the DAI-Labor, a distributed artificial intelligence laboratory, currently employing about 100 researchers, graduate students, postdocs and support staff. Sahin Albayrak obtained his Dr.-Ing. degree in 1992 and a habilitation (German postdoctoral lecture qualification) in 2002, both from the TU Berlin.

Prof. Albayrak is member of the Institute of Electrical and Electronics Engineers (IEEE) and the Association for Computing Machinery (ACM), as well as of Gesellschaft für Informatik (German Computer Science Society, GI), and the American Association for Artificial Intelligence (AAAI). He is also member of the steering board of Deutsche Telekom Laboratories (T-Labs) and a member of the research initiative “Vernetzte Welten” of the incentive circle initiated by the German Government for cooperation between industry and academia “Partner für Innovation”.



Dipl.-Inform. Axel Heßler is a researcher at the DAI-Labor. He has worked on a number of projects within the laboratory, including the JIAC IV agent-platform. Currently he is finalising his PhD thesis about agent-oriented methodologies in the context of complex projects.



Dr. Benjamin Hirsch studied Artificial Intelligence at the University of Amsterdam, and obtained his PhD at the University of Liverpool (with distinction, and currently pending for the “distinguished dissertation award 2006” of the BCS). He is the Director of the Competence Center Agent Core Technologies at the DAI Labor of the Technische Universität Berlin. Under his supervision, the competence center staffs several projects related to serviceware frameworks and service engineering. He is also managing director of the project offices of the DAI Labor. Furthermore, he is actively involved in research on agent technologies and has authored a number of papers, as well as reviewed in numerous agent related conferences.



Dipl.-Inform. Thomas Konnerth studied computer science at the Technische Universität Berlin. Since finishing his Diploma, he has been working as a Researcher at the DAI-Labor on several projects including the JIAC IV agent-platform. Currently he is working on his PhD thesis which is focused on the subject of integrating agent-oriented techniques with service oriented approaches.