A 3-Layer Architecture for Smart Environment Models

A model-based approach

Grzegorz Lehmann, Andreas Rieger, Marco Blumendorf, Sahin Albayrak
DAI-Labor
Technische Universität Berlin
Berlin, Germany
<first name>.<last name>@dai-labor.de

Abstract— **Enriched with more and more intelligent devices modern homes rapidly transform into smart environments. Their growing capabilities enable the implementation of a new generation of ubiquitous applications, but also raise the complexity of the development. Developers of applications for smart environments must cope with a multitude of sensors, devices, users and thus contexts. We present a model-based approach for modeling of, reasoning about and controlling smart environments. A context model provides adaptive applications with a unified access to the smart home environment and, through a unique approach of utilizing executable models, also reflects its state at runtime. The presented approach supports runtime user interface adaption and reconfiguration for seamless interaction and has been successfully utilized to build several context-adaptive applications running in our smart home testbed.** *(Abstract)*

Smart environments; context models; executable models; ambient intelligence

I. Introduction

Applications in smart environments target the assistance of the user during every day life by embedding ambient intelligence into the environment. The complexity of these environments (the availability of a variety of multimedia and interaction devices, sensors and appliances) is an essential issue in the development of context-adaptive applications. Additionally, special needs of different target groups like supportive or rehabilitative usage, non disruptiveness, invisibility, declining capabilities of users, low acceptance for technical problems and the involvement in the active everyday life [1, 3] have to be considered carefully. While personalization puts a strong focus on the user as the main actor for any kind of system, context-of-use adaptation goes one step further and comprises adaptation to user, platform and environment. Applications in smart environments are required to monitor themselves and their environment and provide appropriate reactions to monitored changes before they lead to a disruption of operation [10, 17]. The required context information must be well defined and properly modeled [13].

In this work we concentrate on modeling and processing of context information for adaptive applications in smart environments. We propose an approach of utilizing executable context models to provide a uniform access to context information at runtime. The models hide irrelevant details and provide applications, built atop of them, access to the context information on a high level of abstraction. Our approach enables to describe and configure a smart home environment at design time and empowers the applications to access contextual information, reason about it, detect situation changes and even influence the environment at runtime. This allows to adapt services and user interfaces in a smart environment and to realize applications, facilitating seamless interaction that merges interaction between different devices, modalities and objects.

In the next section we discuss the context modeling issues addressed by our approach. Section 3 presents the details of our approach. On its basis we have implemented three context-adaptive applications. Section 4 therefore contains a description of the implementation, followed by a description of the realized applications in section 5. After the discussion of the related work in section 6, section 7 eventually concludes the paper.

II. Problem Description

The main focus of applications in smart environments lies on supporting users within their daily living by sensing the environment and generating appropriate reactions to changing situations. Developers need to create applications that adapt to changing conditions dynamically. This includes users (e.g. capabilities, or current health status), device capabilities and usage situations (e.g. sitting on the couch, or preparing a meal). Within the application logic and the user interface, there is a strong need to reflect knowledge about the current context (user, device and environment) and to provide the required adaptation means.

The life cycle of context-sensitive and adaptive applications can be split in three phases: design time, configuration time and execution time.

At design time the application developer designs and implements the application. In this phase there is a need to name the context information relevant for the application and the anticipated adaptations and cast this into program code. This can basically be brought down to (1) the identification of the relevant context information and (2) the definition of situations, which define triggers for application adaptations. In this step the developer uses some notation or language to describe how her application should react to the smart environments, in which it will be executed. The bottom line here is that the developer does not know the details of these environments at design time. She can only make some assumptions and define general rules on the basis of a context metamodel or ontology. We assume that the precise information about a smart environment is not known at least until the deployment of the application, which leads to the second phase in the life cycle of the application.

At configuration time, a model of the environment and the available context information has to be made available for the applications running in the environment. This includes (1) the configuration of static information, e.g. the floor plan of a smart home and (2) the integration of available services that provide dynamic context information, e.g. a localization system delivering real-time position data of the inhabitants of a smart home. The result of this phase is a dynamic context model representing the current state of the environment.

At execution time, the situation model of the application and the configured context model of the environment have to be brought together. Executing the application requires a possibility to detect situations based on the dynamic state of the execution context and trigger application adaptations. From this perspective a shared understanding of context is required, bridging design-, configuration- and execution time.

To address these issues we present a layered architecture, which distinguishes the general runtime context information and the relevant application context situations. To the best of our knowledge separating the phase of context definition and the scope of context information at runtime have not been sufficiently discussed in the literature yet. Our approach is based on the concept of executable runtime models to describe context information. A common metamodel provides a general language and ensures interoperability between the design time and configuration time models. The models enable the monitoring of the context at runtime, while also integrating the static, design-time information.

III. Approach

To support context modeling at different development phases, we propose a layered architecture of context models and sensors, which clearly structures context information and allows orchestrating it at runtime. The hierarchy proposed in our approach represents the different stages of context information processing:

- L3 – the top layer contains the context information relevant for an application and identified by its developer at design time.
- L2 – holds context models describing the smart home environments, in which the application is executed. The L2 context models are configured during the deployment phase.
- L1 – is compromised out of all hardware sensors and actors available in the environment that sense the status of the environment and allow user interaction.

Figure 1 presents the dimensions of the architecture, while also illustrating a classification of context information for applications in smart environments. The horizontal axis represents the application lifecycle progressing with time, starting with the design, going through configuration and ending with application execution. The vertical dimension shows the scope of the modeled context information.

The lifecycle of an application begins with its design and development. A developer identifies the context information relevant for the application. She defines relevant situations, to which the service should react at runtime, and context information, which should be used for adaptation. The modeled information is application-dependent, as different applications may react to other contexts. Thus the decisions of the application developer have an application scope. Furthermore, at design time, the developer creates static models, which will yet be filled with dynamic data from the environment upon the application's execution. In our architecture this information is stored in the L3 layer.

Before an application can be executed, it must be deployed in the local environment. Every smart environment has its own context, defined by the available hardware, its users and the physical environment. An application executed in a smart home environment must be provided with means to analyze the local context and extract the relevant context information from it. In other words, before the service is executed, context models of the local environment must be provided in some form. Our layered architecture classifies such models as L2 models, with the scope of the local environment and created before the execution of applications.

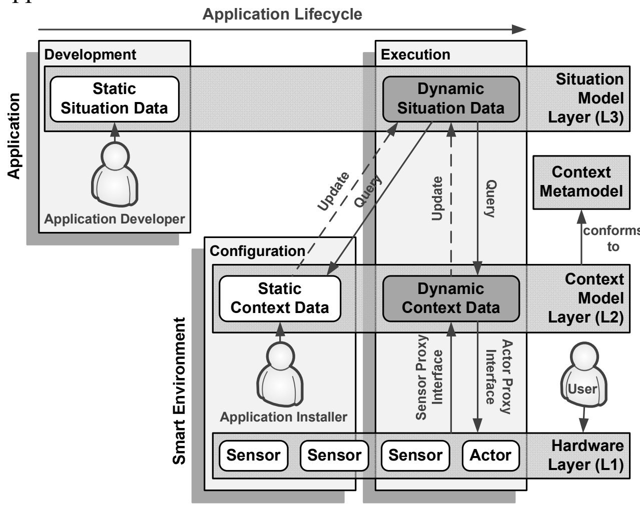

Figure 1. Three layer architecture of context modeling with the application models on top and smart environment models on the bottom.

The context information aggregated on the L2 layer is the same for all applications running in a given smart environment. Examples of such information include data about the inhabitants, a floor plan, current positions of users or devices, etc. In contrast to L2, the L3 models describe application-specific context situations and parameters (e.g. list of all currently available appliances of type T, of the current user, at most 2 meters away of her current location).

During the design phase the concrete smart environments, in which the application will be executed, are unknown. The L3 models are thus environment-independent, in the way that they do not reference concrete context data from any environment, but describe situations using abstract expressions, like *current user*, *current position of user*, etc. The abstract expressions are evaluated at runtime using the information gathered in the environment-specific L2 models. Provided with access to the L2 models, applications are empowered to extract the context information relevant for their L3 models, evaluate it with regard to the local environment and trigger appropriate adaptations.

The exchange of information between the L3 and L2 models happens on the basis of a common notation, a metamodel or ontology. It is not our goal to prescribe any notation for the L2 models, but it must be the same for all environments, in which an application is executed and clearly must be known at the design time of the application.

The utilization of the situation models in smart environments can be carried out in two directions. A continuous evaluation of the models may trigger adaptations of an application, or an intelligent self-adaptive application may evaluate the situation models and utilize the results on its own initiative. For example: continuous updates about the distance of the user to the nearest display may trigger a continuous adaptation of an application's user interface (e.g. making hardly readable text parts bigger). On the other side, an energy management application can extract data about home appliances upon user's request. Both scenarios are feasible in our approach and visualized in Figure 1 as the *Update* and *Query* arrows respectively.

From the perspective of L3 the whole L2 information varies between different environments. Thus the L2 information is dynamic from the perspective of the designed application. The static L2 information is unknown at the creation time of L3's static information. Since an application can be executed in different environments, even the static part of the local context is unforeseeable. This phenomenon is depicted by an arrow between the *Static Context Data* and *Dynamic Situation Data* in Figure 1, and makes the interconnection of both layers a complex problem. The information flow between L3 and L2 can neither be established by the developer, nor prior to the application's execution. The underlying infrastructure must thus be capable of dynamically interconnecting both layers at runtime.

As the context continuously changes at runtime, so must the models, which represent it. In order to provide applications with up-to-date context information, the data flowing from L2 models to L3 models includes both the static context data (e.g. number of rooms, age of users, devices present in the environment) as well as the dynamic information (e.g. current mood of a user, her location coordinates or heartbeat). The former is configured upon the deployment of the application in the environment (e.g. by means of manual configuration), whereas the latter can only be determined at runtime by context sensors in the environment (e.g. localization systems, temperature sensors, e-health devices). Therefore the hardware layer L1 consists of sensors and actors available in the environment. The sensors extract context data from the environment and make it available for environment-wide context models on the L2 layer. The L2 models, conforming to the common L2 notation (metamodel in the modeling nomenclature), give the proprietary data delivered by the L1 hardware semantics.

Last but not least, the L1 hardware layer is the interface between the user and the application. On the one side, the applications learn about the user and the environment by reasoning about the data delivered by the L1 hardware. On the other side, the user explicitly communicates with the application using interaction hardware available in the environment and connected to the L1 layer.

The proposed layered architecture enables a classification of context information utilized by applications in smart environments, depending on their lifecycle. It clarifies the scope of context information, its sources and flow during context processing. In the next section we describe our implementation of the proposed approach, consisting of a runtime architecture based on the notion of executable models and several assistive applications built atop.

IV. Implementation

We have implemented the proposed three layer architecture for context modeling on the basis of dynamic, executable models. Before we describe the details of our implementation in B, we first explain the concept of executable models in the next section.

A. *Executable Runtime Models*

The Model Driven Architecture (MDA) defines a model as a representation of part of the function, structure and/or behavior of a system [16]. The system is in this case also noted as the system under study and the model aims at answering questions concerning the system, without having to study it directly. In this sense dynamic and static models can be defined as models that do or do not change over time [6]. Static models provide (only) a snapshot of the system under study at a given point in time, and could thus only provide answers to "what is" kind of questions, whereas dynamic models can also describe how a system evolves and are thus also able to answer "what has been" or "what if" kind of questions [5]. Executable models are dynamic models, which reflect the state of the system at runtime. However, they also provide the static, design-time view of their system under study. In executable models, model elements describing the static structure of the model and elements responsible for storing state information are made explicit and can be differentiated at any point in time.

Finally, executable models have clearly defined operational semantics needed for their runtime interpretation. They are thus comparable to code [14]. Code receives semantics through the specification of the programming language, to which it conforms. For example, a programming language specification prescribes, how conforming programs are to be executed. Similarly, the semantics of executable models is defined by their metamodels. According to the Meta Object Facility (MOF) specification [18] a metamodel is a model that defines the language for expressing a model.

In case of executable models, the metamodel also defines the operational semantics of the conforming models.

The executable nature tightly connects executable models to their execution platform. Modifications of an executable model must lead to the modification of the code executed on the execution platform and vice versa. This issue, in the modeling community referred to as *causal connection*, is also common to context models, which gather data from heterogeneous real-world environments. To allow bridging executable models with external systems we have introduced the concept of a proxy element. Proxy elements of a model identify processes outside of the model and provide a model-internal representation of these processes. On the basis of well-defined interfaces models can communicate with their environment using proxies.

In summary, we distinguish four types of elements in dynamic, executable models:

- Static elements, defined at the design time of the model
- Dynamic elements, describing the state of an executable model at runtime
- Executable elements of the model constituting its execution logic
- Proxy elements responsible for the communication with the external world

The strong runtime focus of executable models, their dynamic and flexible nature, as well as their clear interconnection with the external world are properties also characterizing runtime context models. Motivated by the similarities of both context and executable models, we have implemented our three-layer architecture on the basis of executable models.

B. *Implementation of Executable Context Models*

The Eclipse Modeling Framework[1] (EMF) is the basis for our implementation of the executable context models described in the previous section. EMF is a meta-modeling framework based on the Ecore notation, which allows the definition of modeling languages. For each Ecore metamodel EMF is capable of generating Java class structures representing it. These can then be enriched with execution logic by a programmer, just as usual Java code can. We have utilized this feature of EMF to create Ecore-based executable models with execution logic specified in form of Java code fragments (encapsulated in Ecore's *EOperations*). Furthermore, we use Ecore's *EAnnotation* construct to annotate model element types to make the static, dynamic and proxy parts of our models distinguishable.

In our implementation every executable context model combines the static information about the environment with the dynamic data gathered and changing at runtime. Utilized at runtime the models provide access to continuously changing context information on a high level of abstraction. They do not require any reconfiguration or restructuring at runtime. By the means of the supplemented Java code, executable context models autonomously adapt to reflect the situation of the environment. This way context information is modeled with the same semantics and level of abstraction at runtime as at design time. In the process, both runtime and design time information remain clearly distinguishable. Ecore-based context metamodels (containing specifically annotated types) make clear, which parts of conforming executable context models are determined by context sensors at runtime (e.g. a person's heartbeat or location) and which must be configured by a human (room plan).

In our approach we also make use of the modifiable nature of executable models. The EMF-based approach enables us to provide means for runtime modifications of the conforming models, in form of Ecore EOperations programmed in Java. This enables the definition of context models' behavior, e.g. when new context data is delivered by sensor proxies. By the same means communication in the opposite direction can be implemented. In addition to sensor proxies, executable context models may contain actor proxies that translate the operations performed on the models to real-world activities of the connected hardware.

Making the communication between the context models and the smart environment bidirectional enables applications not only to sense their environment, but also to influence it via the models. One big advantage of this solution is that the applications access the environment on the high abstraction level of the models. An executable context model and its proxies hide the complexity of the environment and the communication with it. An example for a bidirectional communication of an application with the smart environment is an intelligent light control. An executable context model describing lamps inside a smart home not only determines the current dimming level values of the lamps via sensors but also provides means for their modification via actors. The actors provide a unified proxy interface and hide the details of the proprietary light control interface.

The adaptation of applications in smart environments heavily depends on external parameters, continuously changing at runtime and determined by heterogeneous systems in the user's environment. Our concept of model proxies provides the basis for the communication of executable context models with the environment. Proxy elements, encapsulating context sensors and actors in the environment, feed the context model with data and empower it to influence the environment through well-defined interfaces.

In our implementation of the L3 layer, the executable situation models consist of EMF-XPath[2] queries and logical expressions. This combination enables the expression of complex, application-specific context occurrences and dependencies during the design time, without referencing any concrete environment data. Evaluated during the application's execution, the executable L3 situation models extract relevant context information from L2 models by observing them and evaluating the queries.

In the next section we present three example services, implemented atop of our architecture. Utilizing the executable context models of our architecture, the

[1] http://www.eclipse.org/emf

[2] http://www.eclipticalsoftware.com/emf/xpath/

applications seamlessly integrate themselves into the home environment of the user and assist her in her daily tasks.

V. Example Applications

The SerCHo[3] smart home testbed at the Technische Universität Berlin is the reference smart environment for our work. It is a four room apartment, equipped with a multitude of networked appliances, multimedia devices and context sensors (e.g. temperature, light, localization sensors). Our goal is to develop applications that cope with the complexity of the testbed's technology and utilize its capabilities in order to optimally assist the user in her activities at home.

On the basis of the approach proposed in this work we have implemented three context-adaptive applications:

- Smart Home Energy Assistant (SHEA) – providing energy consumption measuring, analysis and optimization tools. The SHEA measures and monitors the power consumption of all home appliances and provides the user with means for their control. Additionally, the SHEA is capable of controlling the home environment depending on the context information (e.g. switching off lights in non-occupied rooms, alerting the user about unusual energy consumption patterns, controlling the heating system according to weather forecast, etc.).
- 4-Star Cooking Assistant (4SCA) – a multimodal cooking guide. The 4SCA is composed of several context-dependent sub-services: a personalized recipe finder, a shopping assistant and a cooking guide. The 4SCA makes strong use of the available context information by personalizing the recipe search and controlling the available cooking devices.
- Smart Health Assistant (SHA) – supporting the user with his everyday planning and control of diet (in cooperation with the 4SCA) and training. The SHA strongly adapts itself to the user, her health status, preferences and capabilities. Furthermore the SHA communicates with intelligent exercise and weight machines to optimize the user's training.

Each of our realized applications supports seamless interaction between multiple devices and modalities in a smart environment. They adapt to the context of use, by for example reconfiguring the user interface to the distance of the user to the screen or by choosing an appropriate modality depending on the user's situation.

In accordance with the proposed three layer context modeling architecture our L1 layer implementation consists of a number of sensors and actors, encapsulating the variety of hardware available in the SerCHo testbed. This includes sensor and actor proxies encapsulating the home appliances, multimedia and interaction devices as well as context sensors. The heterogeneous hardware available in the SerCHo testbed provides access to its functionalities through, proprietary (e.g. HTTP/XML based) and standard (e.g. UPnP) protocols. The proxy concept allowed us to separate our L2 context models from this external logic. The executable context models are thus connected to the smart home environment through well-defined interfaces of the sensor and actor proxies.

Our implementation of the L2 layer combines three sub-models within an executable context model: an environment model, a user model and a device model. Each focuses a different aspect of the context adaptation. The environment model describes the smart home as a whole, with users, rooms and current sensor data regarding the house (e.g. temperature values in the rooms or current positions of entities within the house). The user model concentrates on properties of house inhabitants, like age, sex, preferences or health-related information. The device model aggregates information about the devices available in the smart environment and provides means for interacting with them.

The L2 models are made available for the applications on a central service platform deployed in the environment. The deployment of the platform is performed in two steps. First the static elements of the provided executable M2 models are configured. In case of our currently implemented platform this includes the configuration of the static properties of the environment, the users and devices. The second step is the configuration of the L1 sensors and actors available in the local environment, and their associations with the L2 entities to assure the flow of context information at runtime. In rare cases the configuration of the L1 and L2 layers changes after the deployment, but generally we assume it to be static.

Each of the implemented applications provides its own executable situation model (L3), which extracts relevant context information from the L2 layer at runtime. Upon the deployment of an application on a platform available in the environment the L3 models of the application are connected to the L2 models of the local platform. At runtime the situations described in L3 models are continuously evaluated and the applications are made aware of changes of any relevant context information. This way, applications adapt to the context of the home environment without being directly coupled with its local technologies and hardware.

VI. Related Work

Context has been specified by Dey and Abowd [2] as "any information that can be used to characterize the situation of entities (i.e. whether a person, place or object) that are considered relevant to the interaction between a user and an application, including the user and the application themselves". Context information relevant for the adaptation of a user interface is manifold. [20] proposed a generic context model that captures information about users, the environment, devices, and applications, [8] define context of use as a combination of user, platform and environment. Independent of the considered context information or the type of adaptation, any adaptation process can be structured in the following phases (see also [7]):

- Sense the context information.
- Interpret the sensed information (detect and understand context changes).
- Select or define a suitable adaptation strategy.
- Execute/apply the adaptation strategy.

[3] The project SerCHo has been funded by BMWi

Recent work in context-aware computing showed that the concept of situations can be useful to solve some problems of context-aware service development. [19] present a state–space model, which describes context and situations as geometrical structures in a multidimensional space. In [4] situation models are also identified as a promising approach for adaptation of context aware services. Their work focuses on the usage of human feedback for refining initial situation models during runtime in order to improve the reliability of detected situations.

Similar to our approach Heckmann distinguishes in [11] between modeling time and runtime, as he states that at modeling time one cannot know which of the situational information will be considered as being context. He defines a situation model as a combination of a user model, a context model and a resource model. Going along with our point of view he sees situations as being retrieval dependent.

Huebscher and McCann present [12] the Adaptive Middleware Framework for Context-Aware Applications, which abstracts from the raw sensor information using 4-tier architecture similar to our layered approach. In contrast to our work, they focus on delivery of context information through different sensors and adaption of the middleware context service to different context providers.

In [9] WildCAT, a generic framework for context-aware applications is presented, aimed at coping with the huge number of context data sources and the difficulty of extracting meaningful information out of it. It permits the monitoring of large scale applications by allowing developers to easily organize and access sensors through a hierarchical organization backed with a powerful SQL-like language to inspect sensors data and to trigger actions upon particular conditions. However, the approach lacks an abstraction layer that encapsulates low level information.

The Mobility and Adaptation Enabling Middleware (MADAM) [15] comprises a context manager, an adaptation manager and a configurator to support the development of adaptive applications. Based on architecture models of applications the properties of each component are analyzed by the adaptation manager to identify relevant context information that it has to subscribe to at the context manager. Adaptation is supported at startup as well as at runtime (reconfiguration), by selecting the most feasible application variant according to the monitored context information.

VII. Conclusions

In this paper we have presented a three layer architecture for modeling of smart environments. The layered approach represents the life cycle of context-adaptive applications and the scope of context information at runtime. Our model-based implementation of the architecture is based on the idea of executable models. It enabled us to create several adaptive applications, deployed in our smart environment testbed.

References

[1] J. Abascal, I. Fernández De Castro, A. Lafuente, and J. M. Cia, "Adaptive interfaces for supportive ambient intelligence environments", Proc. 11th International Conference on Computers Helping People with Special Needs (ICCHP 08), Springer-Verlag, 2008, pp. 30–37.

[2] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles, "Towards a better understanding of context and context-awareness", Proc. 1st Int. Symp. on Handheld and Ubiquitous Computing (HUC 99), Springer-Verlag, 1999, pp. 304–307.

[3] M. Becker, "Software architecture trends and promising technology for ambient assisted living systems", in Assisted Living Systems - Models, Architectures and Engineering Approaches, Proc. Dagstuhl Seminar, no. 07462, 2008.

[4] O. Brdiczka, J. L. Crowley, and P. Reignier, "Learning situation models for providing context-aware services", Proc. HCI International, vol. 4555, 2007, pp. 23–32.

[5] E. Breton and J. Bézivin, "Towards an understanding of model executability", Proc. International conference on Formal Ontology in Information Systems (FOIS 01), ACM, 2001, pp. 70–80.

[6] J. Bézivin, "On the unification power of models", Software and Systems Modeling, vol. 4, no. 2, May 2005, pp. 171-188.

[7] G. Calvary, J. Coutaz, and D. Thevenin, "Supporting context changes for plastic user interfaces: A process and a mechanism", in Joint Proceedings of HCI'2001 and IHM'2001, Springer-Verlag, Sept. 2001, pp. 349–363.

[8] G. Calvary, J. Coutaz, D. Thevenin, Q. Limbourg, L. Bouillon, and J. Vanderdonckt, "A unifying reference framework for multi-target user interfaces", Interacting with Computers, vol. 15, no. 3, June 2003, pp. 289-308.

[9] P.-C. David and T. Ledoux, "Wildcat: a generic framework for context-aware applications", Proc. 3rd international workshop on Middleware for pervasive and ad-hoc computing (MPAC 05), ACM, 2005, pp. 1–7.

[10] P. L. Emiliani and C. Stephanidis, "Universal access to ambient intelligence environments: Opportunities and challenges for people with disabilities", IBM Systems, vol. 44, pp. 605-619, Issue 3, 2005.

[11] D. Heckmann, "Modeling and Retrieval of Context", in Lecture Notes in Computer Science, vol. 3946, pp. 34–47, Springer-Verlag, 2006.

[12] C. Huebscher and A. Mccann, "An adaptive middleware framework for context-aware applications", Personal Ubiquitous Comput., vol. 10, no. 1, pp. 12-20, December 2005.

[13] P. L. Emiliani, L. Burzagli and F. Gabbanini, "Ambient intelligence and multimodality", Lecture Notes in Computer Science, vol. 4555, pp. 33–42, 2007.

[14] S. J. Mellor, "Agile mda", Technical report, Project Technology, Inc., June 2004.

[15] M. Mikalsen, N. Paspallis, J. Floch, E. Stav, G. A. Papadopoulos, and A. Chimaris, "Distributed context management in a mobility and adaptation enabling middleware (madam)", Proc. ACM Symp. on Applied Computing (SAC 06) , pp. 733–734, ACM Press, 2006.

[16] J. Miller and J. Mukerji, "Model Driven Architecture (MDA)", Object Management Group, document ormsc/2001-07-01, July 2001.

[17] J. Nehmer, M. Becker, A. Karshmer, and R. Lamm, „Living assistance systems: An ambient intelligence approach", Proc. 28th International Conference on Software Engineering (ICSE 06), pp. 43–50, ACM Press, 2006.

[18] Object Management Group, "Meta Object Facility (MOF) Specification — Version 1.4", April 2002.

[19] A. Padovitz, S.W. Loke, and A. Zaslavsky, "Multiple-Agent Perspectives in Reasoning About Situations for Context-Aware Pervasive Computing Systems", IEEE Trans. on Systems, Man and Cybernetics, Part A: Systems and Humans, vol.38, no.4, pp.729-742, July 2008.

[20] T. H. Park and O. Kwon, "Ubiquitous Intelligence and Computing", Lecture Notes in Computer Science, vol. 4611, pp. 919–928, Springer Berlin / Heidelberg, 2007.