Runtime Models for Self-Adaptation in the Ambient Assisted Living Domain

Daniel Schneider, Martin Becker

Fraunhofer IESE, Fraunhofer Platz 1,
67663 Kaiserslautern, Germany
{Daniel.Schneider, Martin.Becker}@iese.fraunhofer.de

Abstract. Ambient Assisted Living systems (AAL) must fulfill several challenging requirements as, for instance, the ability to render their services at a quality level that is high enough to enable an independent living. This requires a sound understanding of the current situation of the users, their environment, and, the availability of required resources. Further, AAL systems need the ability to adapt and extend the system behavior, as the demands for living assistance substantially differs between different individuals (differential aging) and changes during a person's life. Both are requirements for a sound adaptation support at runtime that require adequate models.
In this paper we identify stereotypical adaptation scenarios in the AAL domain, elaborate on components and their respective models to support the adaptation scenarios, and discuss the evolution of these models.

Keywords: Self-Adaptation, Ambient Assisted Living, Context Management, Adaptation Management, Configuration Management

1 Introduction

Driven by demographical and societal changes in most industrialized countries, the development of Ambient Assisted Living (AAL) systems seems to be a promising answer to the question of how to enable people with specific needs, e.g. elderly or disabled people, to live longer independent lives in their familiar residential environments [1]. Typical services comprise assistance in daily routine and emergency treatment services. In order to succeed with this, AAL systems must meet several challenging requirements. Among these are (i) the ability to render their services at a quality level that is high enough to enable an independent living, which often requires a sound understanding of the current situation of the users and their environment and the availability of required resources, and (ii) the ability to adapt and extend the system behavior, as the demands for living assistance substantially differs between different individuals (differential aging) and changes during a person's life. Due to financial reasons, AAL systems will comprise in most cases just that hardware components and resources that are necessary to meet the current assisting demands. We call this the "just enough principle". Obviously, the situation awareness and adaptability while tackling with resource constraints are among the stereotypical characteristic of AAL systems.

In the context of the BelAmI [2] project solutions and engineering approaches for runtime-adaptable solutions have been investigated in an application-driven manner. Among the addressed issues was to identify stereotypical adaptation scenarios, as well as to provide a system architecture, respective models and components that meet these requirements.

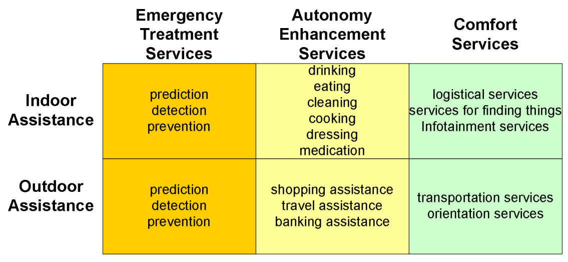
The investigation of the state-of-the-art on adaptable and adaptive systems revealed a substantial amount of approaches and solutions that already exist. However their applicability in the AAL context was for various reasons (e.g. impact on performance and resource consumption, timeliness, flexibility) not that clear. In addition, the relevance of the adaptation scenarios addressed by the approaches was often difficult to assess. This complicates the application of those approaches and solutions in the AAL domain substantially.

In order to pave the way for solutions that meet the adaptation demands of the AAL domain, the paper (i) identifies stereotypical adaptation scenarios in the AAL domain, (ii) elaborates on components and their respective models to support the adaptation scenarios, and, (iii) discusses the evolution of these models.

The paper is structured as follows: Section 2 describes the AAL domain and identifies typical adaptation scenarios therein. By means of these scenarios we deduce architectural entities and their respective runtime models in section 3, 4 and 5 and discuss the evolution of these models. Section 6 gives a brief overview on related work. We conclude in section 7 and give a short outlook on future work.

2 AAL Domain and Adaptation Scenarios

Ambient Assisted Living (AAL) [1] denotes concepts, products, and services that interlink and improve new technologies and social systems, with the aim of enhancing the quality of life for all people during all stages of their lives. AAL could therefore be translated best as “intelligent systems of assistance for a better and safer life” [3]. The potential range of services belonging to the AAL domain is huge. It encompasses any assistive service that facilitates daily life. The classification scheme in [5] structures this domain into six stereotypical subdomains with clearly separated responsibilities. As classification parameters, are used: (i) location where the assisted living service is rendered (row), and (ii) assistance types (columns):



	Emergency Treatment Services	Autonomy Enhancement Services	Comfort Services
Indoor Assistance	prediction detection prevention	drinking eating cleaning cooking dressing medication	logistical services services for finding things Infotainment services
Outdoor Assistance	prediction detection prevention	shopping assistance travel assistance banking assistance	transportation services orientation services

Figure 1 A classification scheme for the AAL services [5]

Within the BelAmI [2] project our primary interest is on the indoor emergency treatment services as they form the kernel of any AAL system that shall enable people to live an independent live alone at home.

In this paper we will use the following system fragment as running example: The system automatically senses the location and activities of the user to detect actual emergencies, e.g. sudden falls, or noticeable trends that could lead to a critical situation, e.g. the sudden increase of toileting activities or the decrease of drinking activities. In case of “suspicious” situations, the system checks with the user the real situation, and informs adequate assistance personnel if required. The system consists of a location device, a cup that senses drinking activities, a concentrator device that renders the emergency services, and a series of interaction devices, as pushbuttons, and several optional information displays. The concentrator is connected to the internet and allows a remote management of the system by a service provider.

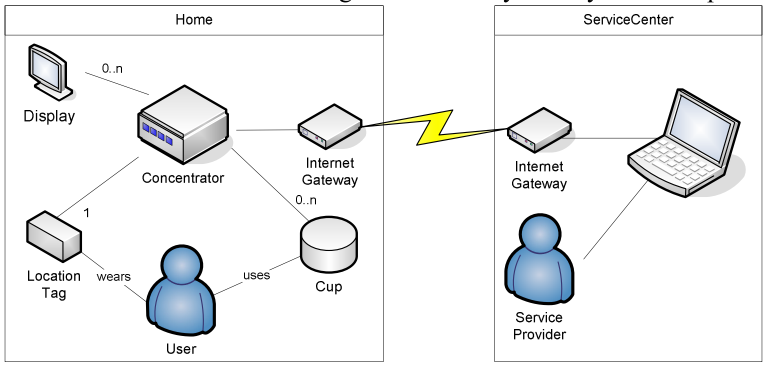


Figure 2 Example System

Within the system, we can identify the following types of adaptation scenarios

S1:Local Adaptation: Here we refer to adaptation scenarios where the adaptation decision is rendered locally, within the component that is to be adapted. Consider the following example: The localization device decides to decrease (downscale) it’s sampling rate to save energy, if its location has not changed much in the past. Conversely, it will increase (upscale) its sampling rate if its location changed. The interesting point in this scenario is that the device decides locally its adaptation, and that the adapted parameters could be continuous.

S2:Remote Adaptation: In contrast to local adaptation, this type of adaptation scenario refers to adaptation decisions made by an external (dedicated) component. Consider the following as an example scenario: A simple drinking reminder is integrated into the cup (beeper), and thus enables the cup to render a complete drinking reminding service on its own. However, if the device is brought into an environment where a better reminder service is available, the reminder on the cup is deactivated and the better one is used. This operation is to be reverted if the cup and the concentrator are disconnected, or if the information display is deactivated. The characteristic problem in this scenario is that the adjustment takes place through the interaction of several distributed devices, and that only discrete features, activation or deactivation, are affected.

S3:Conflict negotiation (local adaptation goals vs. global adaptation goals). If both, local and global adaptations, are supported there might be conflicting adaptation decisions. The following scenario exemplifies this: The location device tries to save energy by switching into a low-power mode after a certain period of inactivity. In an emergency situation, an external agent must be able to prevent this default behavior. The typical trait of this scenario is that a device has to react reliably according to conflicting commands.

S4:Set point adaptation: In the set point adaptation scenario, a default set point for triggering a certain event is adjusted in a specific context, and the machine is perceived to adapt its parameter by learning. For example, the set point for triggering an alarm if a person has an abnormally high pulse is raised if that person executed a temporary exhaustive action. The machine registers that these two events have happened together and adapts accordingly, so that it is able to infer abnormal changes when a similar event happens again in the future.

S5:Manual adjustment: In the manual adjustment scenario, maintenance personnel are able to adjust the current composition, adaptation parameters and also adaptation rules. A further aspect of this scenario is the exchange of components, i.e. adding new components and removing old components. The characteristic point in the manual adjustment scenario is the manual intervention with either maintenance or diagnostic purposes.

In the following sections we elaborate on the components and their respective models we have added to our system architecture in order to support these adaptation scenarios.

3 Configuration

We have started with the simplest adaptation scenario, the manual adjustment S5. Here the adaptation is conducted by maintenance personnel at runtime. They are supported by the system to change the configuration of the system and its components, e.g. for the cup the volume of the beeper or the mode (normal, energy_saving) can be configured. With configuration we refer to a point in the configuration space. The configuration space is spanned by the supported configuration parameters (dimensions) and can be constrained by constraints. Different component configurations can result in different "internal" behaviors as well as in different connection topologies (compositions). A configuration can comprise one or many components of the system.

To realize dynamic (re)configuration we followed the ComponentConfigurator pattern of Buschman et al [6]. The aim of this design pattern is to "*allow an application to link and unlink its component implementations at run-time without having to modify, recompile, or statically relink the application.*"

According to the pattern, an interface should be defined which allows to configure the implementer. This interface evidently must be implemented by every component that should be configurable. Furthermore, a Configurator is required to coordinate the (re)configuration of components, being especially of interest, if several components have to be configured in a coherent way. The Configurator implements a mechanism to interpret and execute scripts specifying the configuration, which is to be instantiated. We correspondingly introduced a Configurator component as an interface which is to be implemented by configurable components (cf. Figure 3).

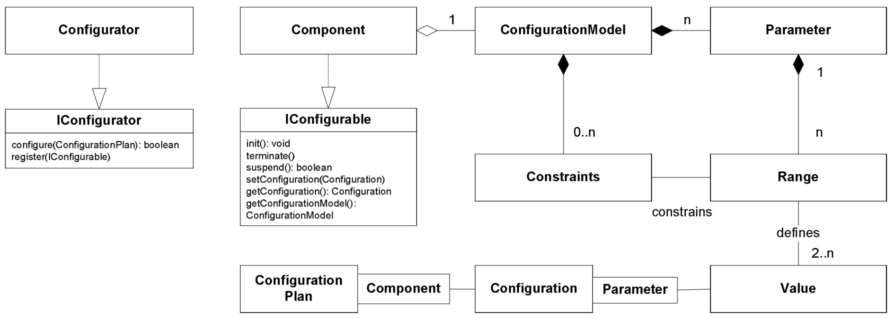


Figure 3 Configuration Support

The responsibilities of the Configurator component (cf. Figure 4) are threefold:

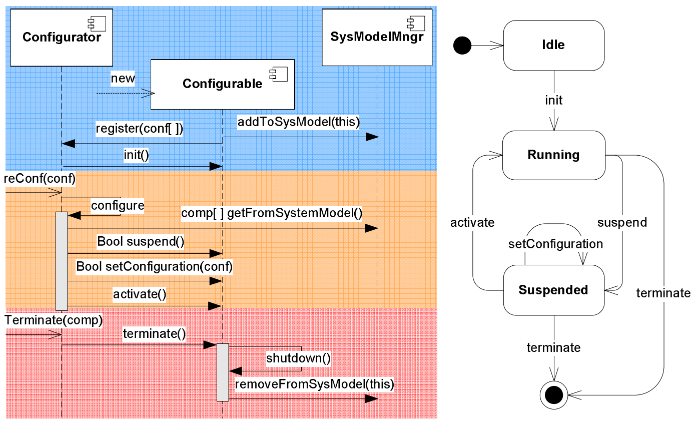


Figure 4 Configurator Responsibilities and Configurable Lifecycle

Component initialization: When a new component is loaded into the system it introduces itself to the SystemModelManager (SMM) and to the Configurator. The SMM maintains a model of the current system comprising the components and their current configurations. The model hence covers both, current component structure and parameters. After the registration phase, the Configurator configures the component into a predefined configuration.

Component (re-)configuration: The Configurator's main task is to manage the transition from a current configuration to a target configuration at runtime. In the manual adjustment scenario, such target configurations are provided by a maintenance engineer. To build a configuration, the maintenance engineer relies on information about the available components and their required and provided properties.

If other components are required to instantiate a certain configuration, the Configurator tries to get them from the SMM. For each of these components, the

Configurator then suspends the component before he applies the new configuration. If the instantiation of the new configuration was successful for all participating components the Configurator reactivates all suspended components.

Component termination: When a component is no longer needed the Configurator is responsible for terminating it in a controlled way (i.e. conduct reconfigurations of connected components when necessary).

4 Context-Awareness

Any sophisticated assistance function in the AAL domain and beyond requires a sound understanding of the current situation in order to plan and execute necessary actions. Self-adaptation falls into that class of functions, as the system adapts itself and thus relieves the system maintainers of this task. Situation awareness can be achieved by a rather simple sensor fusion, e.g. in the cup in our example system, or through a sophisticated fusion of information from different information channels that is known as context awareness in recent years and forms one of the most important ingredients for achieving the Ambient Intelligence **Fehler! Verweisquelle konnte nicht gefunden werden.** or Ubiquitous Computing paradigm [7]. Context awareness comprises three main functionalities: context sensing, context fusion (aka. context interpretation) and context management. Intuitively, many people perceive 'context' as aspects from the users' environment like location and temperature. Despite this common notion, it is hard to define context precisely. We adopt the general definition proposed in [8]: "*...Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves.*" Context-awareness denotes the use of contextual information in computer system functionality.

In order to realize all above listed scenarios that comprise self-adaptation, we added a context management system to the architecture as illustrated in Figure 5. It supports the various system components with access to context information in a device-independent manner. The system provides a push (asynchronous) as well as a pull (synchronous) modus to access information. The context information is managed by the ContextManager in the system. It encapsulates the context distribution strategy in the distributed system and provides the various components with a n:m connector for context information distribution. The managed information and its interrelations are defined by the ContextModel, which is a meta-model. To make use of the context, any application component can register itself with the ContextManager as IContextProvider or IContextSubscriber. Basic context information is provided by sensors (IContextProvider) that provide data on various parameters in the environment, e.g. location of user or objects, and the system itself, e.g. system mode. Components that want to be notified on changes in the context information can register themselves as IContextSubscriber. Components that produce new context information based on basic context information are called IContextAggregators (they are subscribers and providers as well). They can be considered as a kind of logical sensor. The ContextManager also assures the persistence of context information that should be stored for later retrieval. There are various ways to represent context information within the systems. Familiar approaches are *Key-value pairs* (uses the key to refer to the variables and the value of the variable holding the actual context data), *tagged encoding* (the context data is modelled by semi-structured tags, and attributes, e.g. in XML), *object-oriented models* (the context data is embedded in object states, and objects provide methods to access and modify the states), and, *logic-based models* (context data are expressed as facts in a rule-based system).

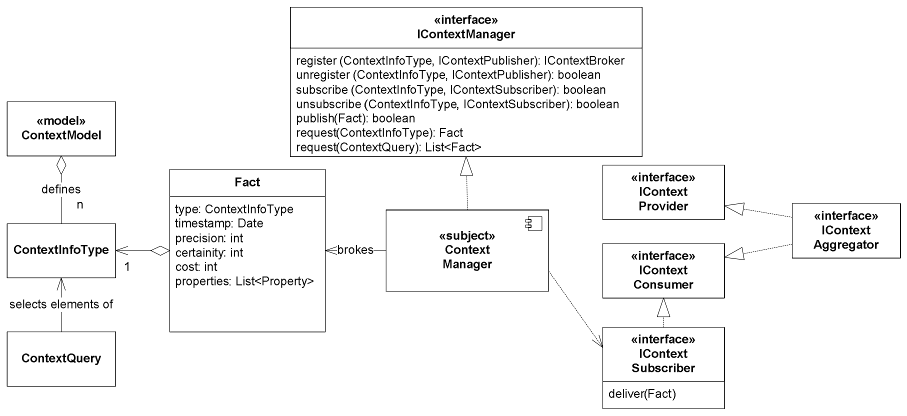

«interface» IContextManager
register (ContextInfoType, IContextPublisher): IContextBroker
unregister (ContextInfoType, IContextPublisher): boolean
subscribe (ContextInfoType, IContextSubscriber): boolean
unsubscribe (ContextInfoType, IContextSubscriber): boolean
publish(Fact): boolean
request(ContextInfoType): Fact
request(ContextQuery): List<Fact>
«model» ContextModel
defines
n
ContextInfoType
1
selects elements of
ContextQuery
Fact
type: ContextInfoType
timestamp: Date
precision: int
certainity: int
cost: int
properties: List<Property>
brokes
«subject» Context Manager
«interface» IContext Provider
«interface» IContext Consumer
«interface» IContext Subscriber
deliver(Fact)
«interface» IContext Aggregator

Figure 5 Context Management

We have followed the Tagged encoding approach, which allows us to realize a rather lightweight context management approach and provides us with sufficient classification facilities. The ContextModel defines the context information that can be queried and the quality of the context information. It consists of a list of context information topics with the respective attributes. The topics are structured hierarchically, which has been inspired by the OSGi event mechanism. This allows an efficient registration and broking of a group (subtree) of context information items via wildcards. Currently, we are considering migrating to an OWL-based [9] classification scheme, to provide more flexility in the type structure. The ContextModel is also a suitable place to specify, which context information is transient or should be stored for later retrieval. The ContextModel only needs to be evolved if new context information should be processed by the system, e.g. when a new sensing device is added or an improved ContextAggregator is added.

All context information within the system is represented as facts as depicted in Figure 5. Facts have a general, simple data structure which provides information about the type, time, precision, certainty, and cost of the fact, as well as a set of properties. Also the quality of the fact is of interest, as the quality of the context information can change over time.

Based on the ContextManager, components that adapt themselves in a situation aware way already can be realized, e.g. S1, S4. However, it was our goal to clearly separate adaptation logic from functionality in components, in order to make the adaptation logic explicit and to support global optimizations. To this end we introduced a third component: the AdaptationManager that is described in the next section.

5 Adaptation Manager

In order to address the scenarios S2 and S3 the Configurator and ContextManager are not sufficient. In these scenarios there is no human in the loop and hence the system must be able to take over the maintenance engineer's tasks. More precisely, the system must analyze the current situation and plan corresponding changes when necessary. These changes can then be executed by the Configurator as described in section 3. To this end, we introduce a further platform component: the AdaptationManager (cf. Figure 6). The AdaptationManager consumes the context in order to evaluate the current situation and to decide upon possible system changes. To this end, all adaptable components have registered their AdaptationModel with the AdaptationManager and ConfigurationModel with the Configurator. Whenever the AdaptationManager identifies a necessary change, it plans the change (ConfigurationPlan) and sends it to the Configurator. The Configurator then takes care for the execution of the configuration. As the Configurator, the AdaptationManager needs to know which system variants exist. However, the information contained in the ConfigurationModel does not suffice. Deeper knowledge is required to identify situations in which configuration should take place and to define which configuration to take in which situation. Several approaches are reasonable to this end:
Rule-based approaches are widely spread in the domain of embedded systems. One reason is that such systems heavily rely on light-weight approaches due to the inherent scarcity of resources (e.g. processing time) and must be deterministic. The rule sets are to be defined at design time and usually have a "Event-Condition-Action" (ECA) form.

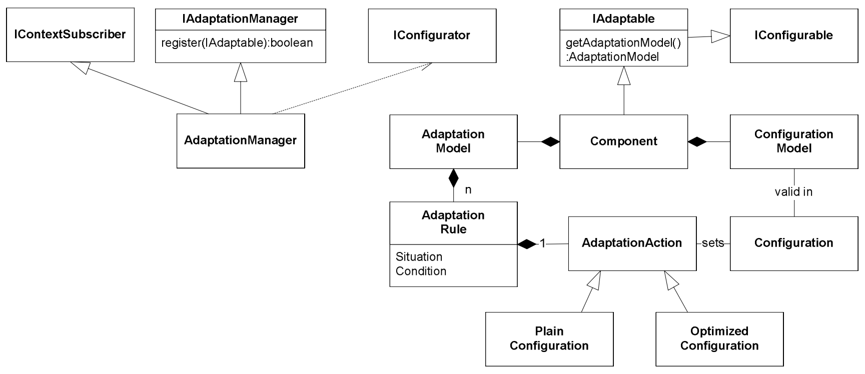


Figure 6 Adaptation Management

Goal-based approaches equip the system with goal evaluation functions. Neglecting available optimization strategies, the brute force approach would determine and evaluate all valid system configurations at runtime and choose that variant that meets best the given goals. Thus, goal based adaptation is quite flexible and it is likely that optimal configurations are identified at runtime.

We follow a hybrid approach that allows the combination of both approaches depending on the actual adaptation needs. Overall we specify the adaptation behavior with AdaptationRules. In their action part, we distinguish between PlainConfiguration, which directly sets a predefined configuration or OptimizedConfiguration, which relies on an utility function to select an appropriate configuration. An example that illustrates the application of the latter is scenario S2, where the best output mechanism is used, depending on the current location of the user and the devices. Here the utility function maps the distance between the user and device to the utility value.

6 Related Work

In recent years numerous approaches emerged in the areas of adaptive systems. Many of them have been motivated by the upcoming paradigms of ubiquitous computing [7] and Ambient Intelligence (AmI). Prominent examples are Robocop, Space4U, and Trust4All. Robocop and Space4U extend the KOALA component model [10] with respect to different component views with corresponding models [11] and dynamic binding of components. Trust4All introduces the notion of trustworthiness and provides means to preserve a certain system level of dependability and security at runtime [12]. With a particular focus on adaptivity, MADAM and the follow-up MUSIC are closely related to our work. These approaches enable runtime adaptation by exploiting architecture models and generic middleware [13][14]. MUSIC builds upon the results of MADAM and introduces different extensions and optimizations [15].

Apart from the largely middleware centric work in the AmI domain there exist recent results in the area of model driven engineering of adaptive systems. Cheng et al. introduced a method for constructing and verifying adaptation models using Petri nets [16]. An interesting approach using software product lines and model driven techniques [17] as well as corresponding tool support [18] to develop adaptive systems is proposed by Bencomo et al. Their work will be of particular interest for our future work.

7 Conclusion and Outlook

Situation awareness and adaptability while tackling with resource constraints are among the stereotypical characteristic of AAL systems. Over the last years, a substantial amount of approaches and solutions emerged in this context. However, their applicability in the AAL context was for various reasons not that clear. In addition, the relevance of the adaptation scenarios addressed by the approaches was often unclear. Therefore we have elaborated on components and their respective models to support typical adaptation scenarios in the AAL domain and have raised some issues with regard to the evolution of these models.

Currently we are implementing our concepts in our ambient assisted living lab. The configurator, adaptation manager and context manager components have been realized as OSGi service bundles. As of now, the different adaptation scenarios can be supported. However, we still need to do some consolidation and hence corresponding refinement and validation of the presented mechanisms is our next step.
The main goal of our future work is to come up with a solution for variability management that can be continuously applied in the lifecycle of a software intensive product (family), ranging from development time to runtime.

Acknowledgements
Part of this work has been funded by the Federal Ministry of Education and Research (BMBF) and the Ministry of Science, Continuing Education, Research and Culture of the state of Rhineland-Palatinate in the context of the BelAmI [2] and AmbiComp projects [3].

References

[1] Ambient Assisted Living Joint Program, http://www.aal-europe.eu, last visited 19.09.2008.
[2] BelAmI, Bilateral German-Hungarian Collaboration Project on Ambient Intelligence, http://www.belami-project.org, last visited 19.09.2008.
[3] http://www.ambicomp.org/ (german site), last visited 19.09.2008.
[4] mst news: "Ambient Assisted Living", mstnews 06/07, http://www.mstnews.de/past-issues, last visited 03.03.2008.
[5] J. Nehmer, A. Karshmer, M. Becker, and R. Lamm, “Living Assistance Systems – An Ambient Intelligence Approach”, International Conference on Software Engineering (ICSE), 2006.
[6] D. Schmidt, M. Stal, H. Rohnert and F. Buschmann, “Pattern-Oriented Software Architecture.”, Wiley, 2000.
[7] M. Weiser, “The Computer for the Twenty-First Century”, Scientific American, p. 94-104, September 1991.
[8] A. K Dey, D.Salber, G. D. Abowd, “A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications”, Human-Computer Interaction 16, pp. 97-166, 2001.
[9] OWL Report, http://www.w3.org/TR/owl-features/
[10] R. van Ommering, F. van der Linden, J. Kramer, J. Magee, “The Koala component model for consumer electronics software”, IEEE Computer, vol.33, no.3, pp.78-85, Mar 2000.
[11] J. Muskens, M. Chaudron, “Integrity Management in Component Based Systems”, Proc. of 30th EUROMICRO Conference (EUROMICRO'04), pp. 611-619, 2004.
[12] G. Lenzini, A. Tokmakoff, and J. Muskens, “Managing Trustworthiness in Component-based Embedded Systems”, Electron. Notes Theor. Comput. Sci. 179, 143-155, Jul. 2007.
[13] Jacqueline Floch, Svein Hallsteinsen, Erlend Stav, Frank Eliassen, Ketil Lund, Eli Gjorven, "Using Architecture Models for Runtime Adaptability," IEEE Software, vol. 23, no. 2, pp. 62-70, 2006.
[14] S. Hallsteinsen, E. Stav, A. Solberg, J. Floch, "Using product line techniques to build adaptive systems", 10th International Software Product Line Conference, pp. 21-24, 2006.
[15] R. Rouvoy, F. Eliassen, J. Floch, S. Hallsteinsen, and E Stav, “Composing Components and Services using a Planning-based Adaptation Middleware”, Proc. of 7th Intl. Symp. on Software Composition (SC'08), Springer LNCS, pp. 16, Budapest, Hungary, 2008.
[16] J. Zhang, and B. H. Cheng, “Model-based development of dynamically adaptive software”, 28th International Conference on Software Engineering, Shanghai, China, 2006.
[17] N. Bencomo, P. Sawyer, G. Blair, and P. Grace, "Dynamically Adaptive Systems are Product Lines too: Using Model-Driven Techniques to Capture Dynamic Variability of Adaptive Systems", 2nd International Workshop on Dynamic Software Product Lines, Limerick, Ireland, 2008.
[18] N. Bencomo, P. Grace, C. Flores, D. Hughes, and G. Blair, "Genie: Supporting the Model Driven Development of Reflective, Component-based Adaptive Systems", 30th International Conference on Software Engineering, Formal demos at ICSE 2008, Leipzig, Germany, 2008.