HAL open science

Models at Runtime: Service for Device Composition and Adaptation

Nicolas Ferry, Vincent Hourdin, Stéphane Lavirotte, Gaëtan Rey, Jean-Yves Tigli, Michel Riveill

▶ **To cite this version:**

Nicolas Ferry, Vincent Hourdin, Stéphane Lavirotte, Gaëtan Rey, Jean-Yves Tigli, et al.. Models at Runtime: Service for Device Composition and Adaptation. 4th International Workshop Models@run.time, IEEE Computer Society; ACM, Oct 2009, Denver, United States. hal-00481778

HAL Id: hal-00481778

https://hal.science/hal-00481778v1

Submitted on 7 May 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution - NonCommercial - NoDerivatives 4.0 International License

Models at Runtime: Service for Device Composition and Adaptation

Nicolas Ferry[1,2], Vincent Hourdin[1,3], Stéphane Lavirotte[1], Gaëtan Rey[1], Jean-Yves Tigli[1,†], Michel Riveill[1] ⋆
{ferry, hourdin, lavirott, rey, tigli, riveill}@polytech.unice.fr

[1] Laboratoire I3S (Université de Nice - Sophia Antipolis / CNRS) 930 route des Colles - B.P. 145 06903 Sophia-Antipolis Cedex - France
[2] CSTB, 290, route des Lucioles, B.P. 209 06904 Sophia-Antipolis Cedex - France
[3] MobileGov, 2000, route des Lucioles - 06901 Sophia Antipolis - France
[†] currently delegated as INRIA researcher in the team PULSAR

Abstract. Our works on software architectures for highly dynamic environments, such as ubiquitous computing, led us to consider models at runtime. Indeed, the biggest challenge of these environments is adaptation, and its reactivity is a key concern. In this paper, we describe models of our dynamic service composition and adaptation approach, and the benefits of the use of metamodels. We explore how metamodels can be used at runtime, to enforce conformity of transformations results, when adaptation is seen as model transformation.

1 Introduction

Ubiquitous computing, as described by Mark Weiser [1], relies on computers present everywhere, at any times and in any things. Indeed with recent years advance in mobile communication technologies and the miniaturization of computer hardware, processing units are becoming invisible and a part of the environment. Software infrastructure appears dynamically populated by functionalities of those devices. The topology of this infrastructure is also dynamic, due to arbitrary node mobility. So that the software infrastructure of an ubiquitous application is not known *a priori*.

One of the biggest challenges of highly dynamic environments, like in ubiquitous or pervasive computing, is to handle the modifications of the infrastructure at runtime. Indeed, these systems have to adapt continuously to their environment [2]. An important constraint is that the adaptation process is driven by the environment and not by the application. Reactivity is a major concern, compared to other systems: the adaptation process must quickly be launched, in reaction to changes in the environment, and must finish before new changes happen.

However, before we can create an "universal application" able to alter its behavior and functionalities in reaction to changes of the environment, a first challenge is to maintain its predefined functionalities despite those variations,

⋆ This work is part of the Continuum Project (French research) ANR-08-VERS-005

without knowing what kind of device are going to be discovered. A continuity of service has to be ensured to mobile users in environments with variable dynamics and heterogeneous resources.

Such adaptation is equivalent to a program transformation which is a kind of model transformation [3]. But in the field of ubiquitous computing it must be a highly reactive transformation, forcing model-checking to happen at runtime, as we will study in the following section (Sect. 2).

In Sect. 3, we present our runtime approach as well as involved models and metamodels. Furthermore, we introduce our adaptation mechanism as a model transformation from an application to another. Then, in Sect. 4, we explain how adaptation is decomposed into three transformations between different models of the approach. Finally, (Sect. 5), we explain how metamodels could help us to check the validity of the adaptation process and the executing application.

2 Reactivity

Reactivity is a key concern of ubiquitous computing, both for adaptation triggering and for adaptation time. We consider that adaptive applications are always in one of the three states presented in Fig. 1. States (1) and (3) are normal execution states of the application, where it is consistent with its environment. It means that the application's behavior is based on what is relevant in its environment and this is the expected behavior for a particular situation. During the transitional state (2) the application is in its adaptation (transformation) phase and unavailable. It is considered in an inconsistent state because the application is not in line with its environment. Moreover, the time spent in this state, related to the speed of the adaptation process, has to be consistent with the dynamic of the changing environment. In other words, it is essential that:

- the system does not stay in the previous state (1) too much time before reacting to environment changes,
- the system is not unavailable for too long while adapting,
- adaptation is fast in order to obtain an application consistent with the environment. Otherwise, the system could become unstable and may never reach a normal execution state before new evolutions occurs in its environment.

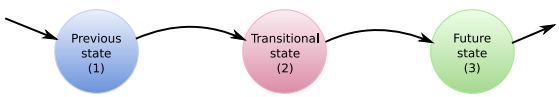

Fig. 1. The three states of an adaptive application

The complexity of the adaptation mechanism must be as low as possible and based on mechanisms that do not try to consider the whole environment (which

can be assimilated to the world) but only what is relevant to the application. The surroundings of the application should be sensed to a sufficient degree to achieve the necessary adaptation.

In some cases, adaptive systems must offer bounded response times. To get an application from one state to another, adaptation must also be consistent with the inherent dynamic of the application. Various studies have suggested response times supposed adequate to users on various systems. In the field of HCI, Berard [4] proposed that the latency of a device must be at least two times less than the user latency which is 100ms. In the area of domotics we consider that an acceptable lag is about 1s. For robotics or ubiquitous systems, latency depends on a cycle based on three steps: perception, processing, action. For the cycle, we consider that an acceptable lag is about 100ms. Thus, adaptation has to be triggered briefly, as much as it has to be effective quickly.

In the next section, we propose an approach for self-adaptive application in the field of ubiquitous computing based on models, in which adaptation is a reactive mechanism.

3 Our runtime approach

Our vision of ubiquitous systems rely on three layers: the environment, accessible through the software infrastructure, the application, as a composition of service and the adaptation mechanism. In this section, we present these layers' models, with their associated meta-models.

3.1 Services for devices: a software infrastructure

For many years, service oriented architectures (SOA) have been used in home automation, mobile and ubiquitous computing to represent as services the sets of functionalities offered by devices. They offer lots of features discussed in [5] such as encapsulation, dynamicity, discoverability and interoperability. They evolved from standard SOA to SOA for device (SOAD) by adding two main features: *decentralized reactive discovery* and *asynchronous communications*.

Decentralized reactive discovery has been popularized by projects such as SLP [1] or Jini. It suppresses the need of a service registry tracking active services in a network domain. Services advertise their presence and clients create search requests using multicasted or broadcasted messages. Asynchronous communications used by SOAD like Jini are events notifications, providing reactivity to devices often interacting with humans or the environment.

In addition, when Web technologies are used to implement SOAD, interoperability between all entities is enabled, whether they are heterogeneous devices or simple software services. Only two implementations of Web services for devices currently exist: UPnP[2] and DPWS[3]. Figure 2 represents a model of a UPnP

[1] The Service Location Protocol.
[2] Universal Plug and Play Forum: http://www.upnp.org/
[3] Device Profile for Web Services. http://www.ws4d.org/

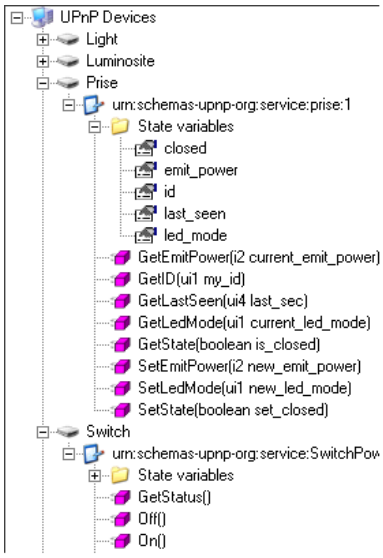

Fig. 2. An infrastructure model

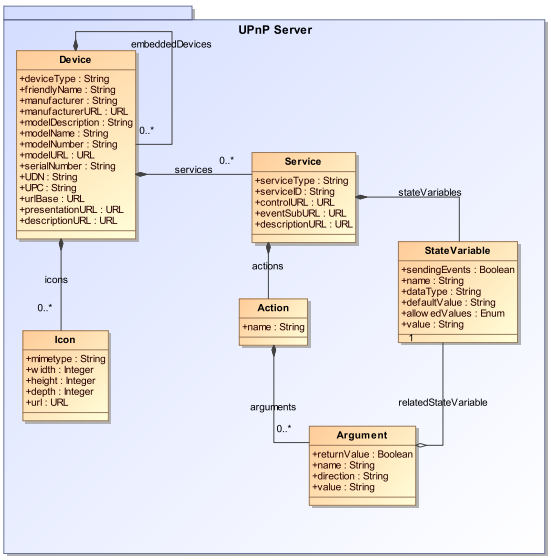

Fig. 3. Infrastructure Metamodel

architecture (what we use as software infrastructure), with four devices, and some details about the services they offer. Next to it, Fig. 3 represents the metamodel of the UPnP the infrastructure. All environmental quantity that can be used in the application is obtained through the use of services present in the infrastructure and conform to this metamodel.

Evolutions of WSOAD allow to create reactive dynamic distributed applications, suitable for ubiquitous computing environments. Using services for devices, any modification occurring in the software infrastructure can be integrated at runtime into the model of the software infrastructure. Thus, the application can react quickly to those unpredictive variations.

3.2 Dynamic service composition

To create applications from this infrastructure of services for devices, we use the Service Lightweight Component Architecture (SLCA) [6]. It allows to dynamically orchestrate and compose services for devices using lightweight component assemblies executing in containers. The container provides minimal technical services, also known as non-functional concerns helpers. Obviously, we created external tools that can generate client components from Web services for devices descriptions. We call them proxy components.

Containers manage assemblies of components fully dynamically and we use models such as Fig. 4 to manage them at runtime. Component types can be loaded and unloaded, component instances and bindings between them can be added or removed at runtime. Proxy components are generated, loaded and instantiated dynamically and automatically, following the presence of services of

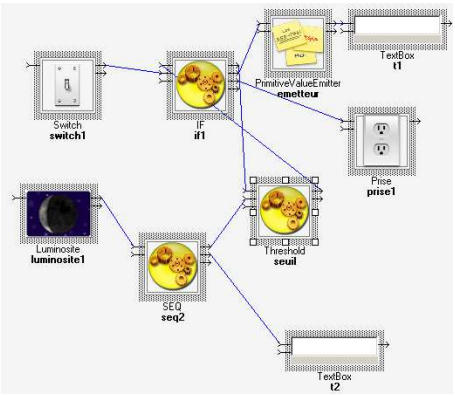

Fig. 4. A SLCA model

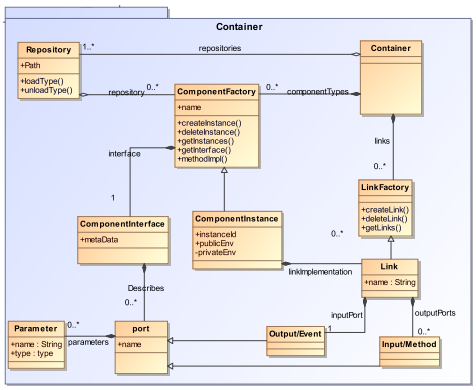

Fig. 5. SLCA Metamodel

the infrastructure. There is a direct mapping between the infrastructure metamodel and the application (SLCA) metamodel. Moreover, this mapping is highly reactive.

Applications or new functionalities are created from existing services on the infrastructure by managing an assembly of components. Proxy components are combined together or with purely functional components to transform information. SLCA components and services for devices communicate mostly using event-based communication patterns, which, more than decoupling entities and increasing dynamicity, enables the reaction to infrastructure changes efficiently.

Figure 5 represents the metamodel of this composition architecture, for a container and its internal dynamic assembly of components.

3.3 Auto-adaptation

Now that applications are created from the dynamic infrastructure, we need to adapt their behavior to changes of their environment in a reactive way. We created a paradigm called Aspect of Assembly [5] that allows us to adapt composite services according to specified rules. Aspects of assembly (AA) are pieces of information describing how an assembly of components will be structurally modified, thus keeping black-box property of components. Modifications include adding components and bindings between them. Since the mapping between the infrastructure and the application is done dynamically, an expert user, in order to build an application, has to write and select a set of AAs. Aspects of Assembly consist of two parts, like regular aspects found in Aspect-Oriented Programming (AOP) [7]: pointcut and advice. Pointcuts describe to which components the modifications described by advices have to be *weaved* (applied).

If some of the required components expressed in a pointcut are not available, the advice won't be weaved until they become all available. Since service discovery is a reactive process and that containers notifications are events too,

AAs can be weaved in response to the appearance of a service on the infrastructure. Moreover, AA composition provides associativity, commutativity and idempotence properties when several aspects are enabled to be weaved at the same time [5].

Pointcut are defined as sets of filters on base assembly meta data — for example component ID or types (see Fig. 7). Those filters construct lists of parameters, called the joinpoints, satisfying the list of variables of the associated advice. They are the set of components on which the advice will be weaved. For each generated list, the advice is duplicated, and the variables are syntactically replaced in the advice to match the base assembly joinpoints. For our experiments, we choose for convenience to express filters in using some simple pattern matching on component instances names.

Advice in AA is not a piece of code which will be weaved into components. It defines a set of component instances and links that will be weaved inside a targeted assembly of components. Advices are specified in a DSL using interaction specification defined in [5]. A link or connector between two components consists of an input event and an output method from components. More than links, rules can create assemblies thanks to some predefined operators which are components with a well-known semantic like a condition (`if`) or indeterminism (`parallel`) (see Fig. 7). Finally all those specifications are translated into a set of elementary modifications—add or remove components and links—with respect to blackbox properties of COTS components.

As we have seen, AAs are written as a pointcut and an advice. They can be written and added to the system at design-time or at runtime. Checking that an AA is conform to its metamodel (Fig. 7) is easy. However, at runtime, new informations are available, like joinpoints and the status of the application, making the checking more complicated and more relevant for a situation.

In the following section we will present how AAs are weaved and how this can be compared to a highly reactive model transformation.

4 Aspects of Assembly weaving process

The weaving process can be seen as an horizontal endogenous model-to-model transformation [8]. It re-factors an SLCA application into another.

With this approach, the transformation is done at runtime. While the adaptation is being done, the application is in transitional state, but can still be used. Only the adaptation mechanism is busy during this lapse of time. This is a major concern for service continuity and for application's reactivity in ubiquitous computing. Moreover, in potentially very large systems, using the AOP approach allows designers to write adaptation rules focusing only on a part of the system, reducing the complexity of the adaptation process. Since aspects can

```
V  Pointcut  V
aSK_RDR417:=/aSK_RDR417/
rfid2Diploma:=/rfid2Diploma/
screenDevice1:=/screenDevice1/
primitiveValueEmitter:=/primitiveValueEmitter/
screenDevice2:=/screenDevice2/
service:=/service/

schema Sl(aSK_RDR417,rfid2Diploma,screenDevice1,primitiveValueEmitter,
screenDevice2,service):

provider : 'WComp.Beans.Provider' ;

singleToDoubleString1 : 'WComp.InterfaceTranslator.SingleToDoubleString1' ;

aSK_RDR417.^CurrentRFID -> (
provider.set_CurrentRFID
)

provider.^set_CurrentRFID_event -> (
rfid2Diploma.GetDiplomaIdent
)

rfid2Diploma.^GetDiplomaIdent -> (
provider.set_DiplomaIdent
)

provider.^Start_event -> (
(aSK_RDR417.CurrentRFID+primitiveValueEmitter.InfoNumber)
)
^  Advice  ^
```

Fig. 6. Aspects of Assembly model

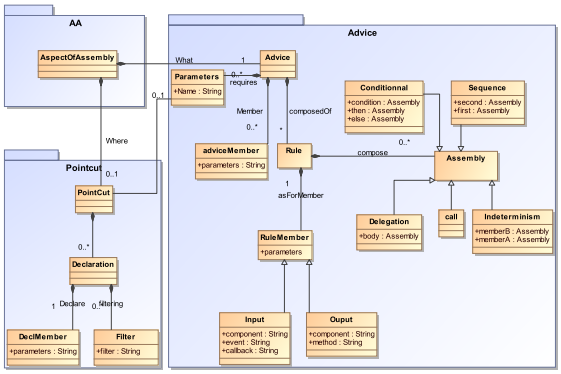

Fig. 7. Aspects of Assembly Metamodel

be duplicated in several places, the number of aspects to consider may be lower than in other approach. The complexity of the adaptation mechanism is reduced to provide a better reactivity.

Other works have explored the use of models at runtime for aspect-based adaptation [9,10]. They need the dynamicity to validate new adaptation rules at runtime, before making them automatic scripts. But because our mechanism is build on less steps (especially model transformations) we seem more able to be reactive. In fact there are only few works on highly reactive mechanism for adaptation in ubiquitous systems. OpenORB [11] proposes a way to adapt a system at runtime in a very reactive way but it does not consider the whole classical mechanism for context-awareness. CORTEX [12] also addresses the concern of reactivity but as OpenORB, it does not manage potential conflicts between several adaptations.

4.1 The weaving process

In this section we study more precisely the weaving process that can be decomposed into three steps (Fig. 8). First, pointcut matching is a function that has a set of components from the initial assembly and pointcuts from a set of selected AA as input. Its goal is to find the components, called *joinpoints*, on which advices will be woven. Based on pointcut matching results, an advice can be weaved several times in the same weaving process.

The second step is called the advice factory. It generates instances of advices, replacing variable components in advices of selected aspects by joinpoints given by the first step. Instances of advices describe modifications to be weaved in an actual assembly of components.

Finally, the composition engine merges all instances of advices with the initial assembly in order to generate a single instance of advice that will be weaved as the final assembly. This merging mechanism is able to resolve conflicts between

various instance of advices [5]. Thus we can build applications by composing several aspects of assembly at the same time, at runtime.

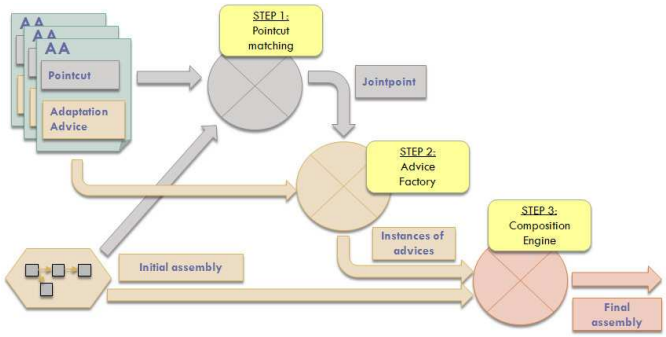

Fig. 8. Detailed weaving process

In order to be more reactive, the weaver can be triggered in two ways. The first is user-driven and changes the set of AAs given as input to the weaver, by selecting/deselecting or adding/removing aspects of assembly at runtime. When the set of AAs is modified, the weaver is triggered, leading to adaptation if an added AA can be applied or if a removed AA was. The second way of triggering adaptation is driven by infrastructure changes. When a new device appears or disappears in the environment, its proxy component is dynamically instantiated in or removed from the assembly. The adaptation process is triggered and only AAs that are able to apply according to newly available components are applied.

An important point for the reactivity of such a mechanism is that the system doesn't requires any information about the state of the software infrastructure. With a dynamic of its own, the infrastructure imposes his pace. The inclusion of this type of triggering mechanism allows us to build opportunistically from an infrastructure services and devices, applications with a "bottom-up" approach. Applications are built in line with their infrastructure in a reactive way.

4.2 Synthesis

Our approach allows us to build applications able to adapt at runtime to their software infrastructure. Moreover, coupling the event paradigm and AOP allows us to transform our applications and our models in a very reactive way. Our approach is decentralized and distributed since each weaver is associated to a composite service and deals with AAs in order to consider only relevant part of the environment. This caused a sharp reduction of adaptation's complexity.

We validated our approach with some experiments on the cost in time of a weaving process on randomly generated component assemblies. Some of those results can be found in [13]. As result of these experiments, we found that with two AAs conflicting we are able to weave about 40 components (e.g to merge

20 instance of advice) in 100ms. Without conflicts we are able to compose more than 100 components (e.g. to compose 50 instance of advice) in 100ms.

5 The contribution of metamodels

More than a reactive model-to-model endogenous transformation [8] at runtime, our adaptation process can be seen as a parametrized transformation. All selected and applicable AAs are parameters that define the transformation taking place. Without these parameters, the transformation does not involve any modification of the initial assembly. In fact, this transformation is decomposed into three low cost transformations (Fig. 9) to be done at runtime.

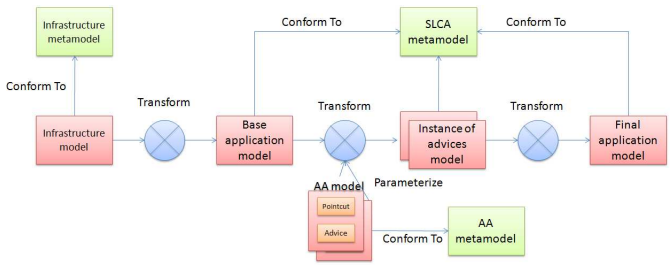

Fig. 9. The adaptation process as three transformations

1. The first transforms the software infrastructure model, conform to the infrastructure metamodel, into a component assembly conform to the SLCA metamodel. This is an horizontal model-to-model and one-to-one exogenous transformation.
2. The second transforms the application obtained in 1. into instances of advices which are the sub-assemblies to weave in the final application. These instances are also conforming to the SLCA metamodel. This transformation is parametrized by a set of selected AAs conform to the AA metamodel. This is an horizontal one-to-many parametrized endogenous transformation.
3. Finally, the last transformation merges all the instances of advices into a single application also conform to the SLCA metamodel. This is a many-to-one horizontal endogenous model transformation.

As we have seen, our mechanism for adaptation in ubiquitous environment is a set of highly reactives models transformation at runtime. The introduction of metamodels defining the three levels of our approach has allowed us in the first place to abstract the work already undertaken and to ease its reusability. Secondly, the ANR RNTL FAROS project has enabled us to validate our metamodels and to check models conformity at design-time. We are now working on setting up mechanisms to check the conformity at runtime of our applications through those metamodels.

6 Conclusion

We presented our approach for dynamic service composition and adaptation based on aspects of assembly, that is equivalent to a highly reactive model transformation at runtime. Models allow us to check the conformity of applications, infrastructure, and adaptation parameters at design-time. Incidentally, what would be the impact of such checks at runtime on the reactivity of our approach? Moreover, how can we benefit from our three-step transformation to check partial conformity of models during the whole adaptation process? Finally, can we compare the model transformation to runtime aspect-oriented modelling?

References

1. Weiser, M.: The computer for the twenty-first century. Scientific American **265**(3) (Sep 1991) 94–104
2. Berry, D., Cheng, B., Zhang, J.: The four levels of requirements engineering for and in dynamic adaptive systems. In: 11th International Workshop on Requirements Engineering Foundation for Software Quality (REFSQ). (2005)
3. Czarnecki, K., Helsen, S.: Feature-based survey of model transformation approaches. IBM Syst. J. **45**(3) (2006) 621–645
4. Crowley, J., Coutaz, J., Bérard, F.: Perceptual user interfaces: things that see. Communications of the ACM **43**(3) (2000)
5. Tigli, J.Y., Lavirotte, S., Rey, G., Hourdin, V., Cheung-Foo-Wo, D., Callegari, E., Riveill, M.: WComp Middleware for Ubiquitous Computing: Aspects and Composite Event-based Web Services. Annals of Telecommunications (AoT) **64**(3–4) (Apr 2009) 197–214
6. Hourdin, V., Tigli, J.Y., Lavirotte, S., Rey, G., Riveill, M.: SLCA, composite services for ubiquitous computing. In: Proceedings of the 5th International Mobility Conference, Singapore Chapter of ACM (2008)
7. Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., marc Loingtier, J., Irwin, J.: Aspect-oriented programming. In: ECOOP, SpringerVerlag (1997)
8. Mens, T., Czarnecki, K., Gorp, P.V.: A taxonomy of model transformations, Germany, Internationales Begegnungs und Forschungszentrum für Informatik (2005)
9. Fleurey, F., Dehlen, V., Bencomo, N., Morin, B., Jézéquel, J.: Modeling and Validating Dynamic Adaptation. In: 3rd International Workshop on Models@ Runtime (MODELS.08), France, Springer (2008)
10. Morin, B., Fleurey, F., Bencomo, N., Jezequel, J., Solberg, A., Dehlen, V., Blair, G.: An aspect-oriented and model-driven approach for managing dynamic variability. In: 11th International Conference on Model Driven Engineering Languages and Systems (MODELS), Springer (2008)
11. Grace, P., Coulson, G., Blair, G., Porter, B.: A distributed architecture meta-model for self-managed middleware. In: Proceedings of the 5th workshop on Adaptive and reflective middleware (ARM'06), ACM (2006)
12. Verissimo, P., Cahill, V., Casimiro, A., Cheverst, K., Friday, A., Kaiser, J.: Cortex: Towards supporting autonomous and cooperating sentient entities. In: Proceedings of European Wireless. (2002) 595–601
13. Ferry, N., Lavirotte, S., Tigli, J.Y., Rey, G., Riveill, M.: Context Adaptative Systems based on Horizontal Architecture for Ubiquitous Computing. In: International Mobility Conference, France, ACM (September 2009)