UNANTICIPATED DYNAMIC ADAPTATION OF CONTEXT-AWARE SERVICES

Marcel CREMENE* Michel RIVEILL** Christian MARTEL***
**Universitatea Tehnica din Cluj-Napoca, Romania, cremene@com.utcluj.ro*
*** Universite de Nice, Sophia-Antipolis, France, riveill@unice.fr*
****Universite de Savoie, Chambery, France, christian.martel@univ-savoie.fr*

Abstract: Usually, for building context-aware services, the developer must anticipate all the possible contexts elements and their states: user preferences, physical resources, location, etc. This anticipation is required in order to specify service and context-specific adaptation rules, which are necessary for the adaptation management. Because of this, the adaptation rules are not reusable and the service cannot react to unanticipated context changes. In this paper, we propose a platform for dynamic and unanticipated adaptation of component-based services. Our solution is based on a knowledge representation describing the service and the context as a whole system. Based on this service-context representation, the adaptive platform is able to check the adequacy between the service and the context and to search for solutions if necessary. The difference between the classical approach and our approach is that we use only general rules. We have tested our architecture using a prototype that shows that a forum service may be adapted to the user language without using service and context-specific rules.

Key words: *Context-aware services, Components, Knowledge representation, Adaptation rules*

I. INTRODUCTION

In order to offer an intuitive understanding of the problem, we start with a simple example. Suppose we have a distributed service as a forum. This service is integrated in a university portal. Any student can use this service if he/she has a PC connected to the Internet, with an ordinary HTML browser installed. The forum architecture contains the forum server and several forum clients. The client is a standard HTML browser. The forum language is English.

Suppose that a foreign student arrives in the university campus and wants to use this forum. Unfortunately, his English is not very good. In this case, we can affirm that the forum service is not adequate to the context; the student cannot use this forum. If another user, a blind person, wants to use the same forum but having a mobile phone as terminal, he will not be able to use the forum neither. A service that was not designed for a certain context usually does not work in a new context.

Obviously, all these situations described here may be predicted by the service developer. The service may be conceived differently, by adding, at the construction time, additional components such as: a set of translator components, a set of text to speech and voice recognition components for all possible languages. But who can imagine *all* the possible situations? What if other context elements change? Even if the developer takes into account a very large number of different situations, the number of possible combinations will lead to a large number of adaptation rules and additional components.

In this paper we are interested in solving the service adaptation problem assuming that the context is not anticipated at the design time. We also want to avoid inserting into the service a lot of additional components for all possible contexts.

According to the component-based and service-oriented approach, we consider that a service is composed by a sum of reusable components and we have a repository containing numerous reusable components produced by different component developers.

This paper is organized as it follows: the section two presents our approach inspired from the AI (Artificial Intelligence) domain and closed-loop control principle. The same section contains the general architecture of the adaptive platform. Section three introduces the *profile* concept. The profile function is to offer an additional semantic description about the service and the context. In section four we validate our proposal using a forum prototype that is dynamically adapted to the user language. Section five contains the related work and the last section contains the conclusions.

II. APPROACH AND ARCHITECTURE

In the AI field, the adaptation problem may be seen as a search problem in a solution space given by the service and context structure. The AI field has three important aspects: a) the *knowledge representation* is the problem description; b) the *search* problem is solved by different algorithms that analyze the knowledge representation space; c) the *learning* is present in some AI systems that are able to update their knowledge.

According to the closed-loop control principle, we can consider an adaptive system as composed by three main

Manuscript received February 29, 2008; revised April 17, 2008

parts:

a) The *Reconfigurable elements* are the system elements that may be changed, reconfigured at runtime. In our case the possible modifications in a component-oriented service are: adding new components, removing components, replacing components, migrating components and modifying the component parameters.

b) The *Monitors* are specialized components used for context observation. The observation result is used in order to decide if the service is adequate to the context.

c) The *Controller* contains the adaptation "intelligence"; it decides what reconfigurable element must be changed in order to adapt the system.

The main idea of the proposed solution is to observe the service, the context and decide if the service is adequate to the context. If the service is not adequate, the adaptation platform search for possible solutions and apply one of these solutions. The user may also be involved in this process: for instance, in the forum example, the user may decide between two possible translation components, one that is more accurate but slower and another one that is less accurate but faster.

The adaptive platform general architecture is depicted in the figure 1.

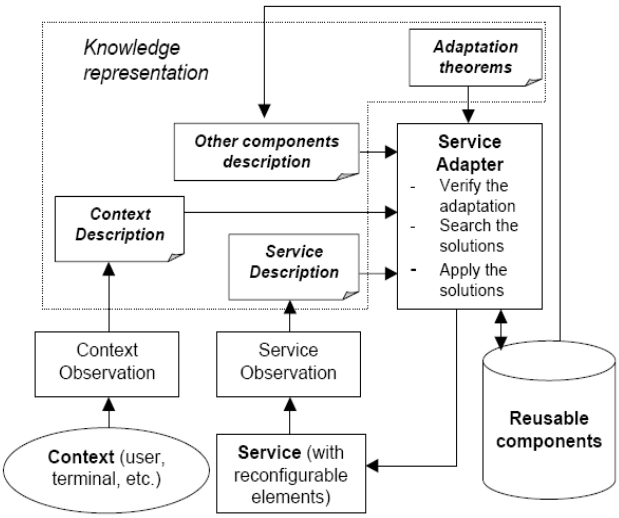


Figure 1. Adaptive platform architecture.

The most important modules of the proposed architecture are described bellow.

Service definition, observation and description. The service is seen as a composite (a component composed by other components) and distributed component. This type of component may be described, for instance, using Fractal component model [1]. The service architecture may be described using an ADL (Architecture Description Language), for instance Fractal ADL [1].

Middleware solutions as CORBA or SOAP represent solutions that may solve the component distribution problem. The service observation may be done using a reflective platform [3, 8]. Reflective adaptation techniques give us the possibility to dynamically modify the service, for instance by using ISL interaction patterns [4] or other adaptation techniques [5].

Context definition, observation and description. The user profile, the available physical resources, the social context, the location, etc. represent the context. The context is observed using special components called *monitors.* For instance, in the forum example a language detection component, a monitor, may be used in order to determine the user language. Other monitors can observe the terminal available memory or bandwidth and may be provided by the terminal operating system. In order to describe the context, it is preferable to use standard languages for the context description: for instance CC/PP [6] from W3C. MPEG 21 contains also some elements describing the terminal multimedia capabilities.

Reusable components and component database. A large number of reusable components exist and they may be described by using different technologies such as: CCM, DCOM, Java Beans or even Web Services (described with WSDL). We suppose to have centralized lookup service that allows us to search services based on different criteria. Such a lookup service may be implemented using CORBA Trader [2] or UDDI.

Service Adapter. The most important part of the proposed architecture is the *Service Adapter*, which takes the decisions on how the service must be modified in order to become an adapted service. If the service S1 in not adapted to the context C, the Service Adapter will apply a transformation to S1 and will generate S2. In order to decide which transformation is necessary, the *Service Adapter* needs to "understand" if the service S is adequate to the context C. In order to do that, the Service Adapter needs a set of adaptation theorems, which are basically service independent rules. The service is considered adequate to the context if these rules are all respected.

III. SERVICE AND CONTEXT KNOWLEDGE REPRESENTATION

The knowledge representation about the service and the context is a description that gives us the possibility to decide if a service S is adapted to a certain context C. In order to obtain this, the main idea is to use the same parameters while we describe the context and the service, implicitly the components. In the paper [7] we are proposing the *profile* concept, which is related to components, service and also to context. The profile is the key element for the service-context description because the service profile and the context profile become comparable using the same profile parameters. The profiles are necessary because the existent service and context representation languages do not offer enough semantic information. For instance, using just a service ADL architecture description and a CC/PP context description it is not possible to say if the service is adapted to this context or not.

Context profile. The context may be described as a parameter list. This list may be organized hierarchically if the context is more complex. Some parameter examples are: user language, visual accuracy**,** terminal memory, display, software platform.

The idea is not to have from the beginning an exhaustive profile, but rather to be able to add new parameters to the context profile, to have an evolving context profile. The parameter values will be offered by the available monitor components, in fact the context knowledge depends of the available context monitors. The monitor components may be included in the existent software platform (OS, Virtual Machine), for instance the J2ME (Java 2 Micro Edition)

Runtime.freeMemory() or may be found in the component database (a language detection component).

Component profile. A component is described by a profile that contains the same parameters as the context profile but in association with other aspects. A component profile is related to the component syntactic description (for instance the CCM [2] model) and it is composed by:

- A *parameter list* (like that for the context, using the same names for the parameters),
- For each parameter from this list, there is a *list of association points*. An association point may be:
  - The component itself,
  - A component interface,
  - An object that is passed through a component interface (argument or method return)
- For each association point there is a list of conditions on the parameter. The association point is either an *input* or an *output* and the conditions are preconditions or post conditions according to the type of association point.

Figure 2 shows the component profile structure described using an XML based language.

```
<profile component_name = "ComponentName">
    <param name = "Param_1">
        <point id = "pt_in_1", type = "input">
            <linkto>(component, interface, method,
        object)
            </linkto>
            <condition>
                expr(Param_1)
            <\condition>
        </point>
        …
        <point id = "pt_out_j", type = "output">
            …
        </point>
    </param>
    …
    <param name = "Param_k">
    …
    </param>
</profile>
```

Figure 2. An XML based representation of the component profile.

Service profile. A service is seen as a composite component, and, consequently, the service profile has the same structure as the component profile. As a composite component may be described hierarchically by its internal components, the service profile results in a composition of the internal component profiles.

In order to be able to evaluate the service behavior after a modification, it is important to compute the service profile starting from its internal component profiles. The service profile results as follows:

- The parameter list is obtained as the reunion of the internal profile parameters list
- For each parameter, the association points are: the whole service, the service interfaces, the passed objects that have a connection to the association points of the internal profiles
- The conditions for the association points are the same as the conditions of the internal profiles.

We say that the parameter and the conditions are *propagated* through two types of connections. In order to have a better understanding of this concept we present the profile composition for the forum service, figure 4. Before explaining this concept we present the forum service, S, in figure 3. It is seen as a composite component (CCM model notations) that includes three other components:

- The forum server, F, having two facet-type interfaces: "toSend" having a method "post" for posting messages, and "toView" with the method "get" for retrieving messages
- A GUI component for viewing the forum content, V, having a receptacle with F and a IHM with the user
- A GUI component for composing new messages, C, having a receptacle with the F and IHM with the user

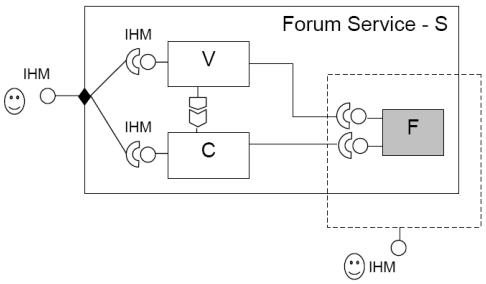


Figure 3. The forum service

A particular aspect in our composition model is that the IHM is considered a facet (CCM model element) with the user. The service user interface, IHM, is composed by the internal components IHM. This was necessary because some parameters, the language for instance, are transferred through the IHM. Another particularity is the sharing of the component F between several client component chains. The component sharing is a concept treated in Fractal [1].

Figure 4 depicts what we call the *propagation graph.* For each parameter we have one propagation graph, for instance for the language the graph is shown bellow.

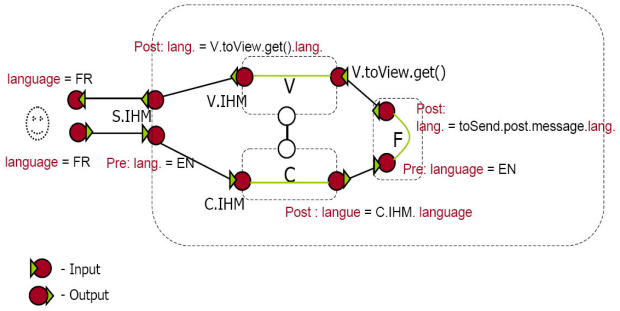


Figure 4. Profile composition

This graph has, as nodes, the association points (components, interfaces, objects passed) and as arcs the connections between two components but also the "internal" connections. For instance, the component C has an internal dependence: the language of the composed message is the same with the language used by the user at the IHM level this is an internal dependence between two association points, in this case interfaces: C.IHM and the C's receptacle with the F.

This graph shows that, because F has a precondition at its

input interface: langue = EN, and because C and V do not change the value of the parameter "language", then the whole service, S, will have the same precondition at the association point S.IHM, the input. This means that if a user does not write his message in English the F precondition is not respected and we have a service non-adaptation for that user. The F has an internal dependence too: the language at the F output is the same with the language at the F input.
In figure 4 we have a non-adaptation situation because a French user tries to send a message on the English-only forum. The non-adaptation is detected at the S.IHM point because this represents the connection between the service and the context (in this case the user profile). For this example a monitor component is necessary: a language detection component.

Any condition associated with the service has a root inside the service. If several roots exist, the final condition will be an AND between all the root conditions.

Profiles compatibility. Suppose we have 2 interconnected components, each having a profile. If all the conditions at the correspondent association points are respected the profiles are considered compatible. In the previous example, the service and the context (user) are not compatible because the user imposes language = FR and the service imposes language = EN at the IHM level which is impossible to respect. We must anyway notice that the equivalence is not the only possible operator while verifying the compatibility. If the parameter is memory then compatibility means to have the required memory inferior to the available memory.

Profile extensibility. The possibility to extend the profiles is very important because the idea is not to establish a priori exhaustive profiles but rather to add new parameters to these profiles. We observed that different parameters might have different composition rules and also different compatibility rules, for instance the language and the memory. For each parameter we need to specify:

- *Definition domain*. For instance, the parameter "language" has as definition domain {EN, FR, DE, …, *, ?} where "*" is any and "?" is unknown.
- *Composition operator*. If two language association points, one input and one output, are connected, the language value will be the same for each point. The input will take the same value for the language as the output. In this case the composition operator is ":=".
- *Compatibility operator*. The compatibility operator helps us to decide if the service is adapted or not (for a specific parameter). For the language case, if the user knows EN, FR and the service accepts EN then the service is adapted because $\{EN, FR\} \cap \{EN\} \neq \emptyset$, at least one language is common.

Suppose we want to add a new parameter to the profile, for instance "IHMsupport". It is necessary to specify the three aspects presented before:

- Definition domain: {HTML, VoiceXML, WML, Java-Swing, J2ME-LCDui, ….}
- Composition operator: a service composed by two components each having two different IHMsupport parameters, requires an IHM support given by the reunion of the two IHM supports. In this case, it may be useful to use operators like AND, OR in order to express that a component requires a HTML support OR a VoiceXML support for instance.
- Compatibility operator: a service is compatible with the terminal if the required IHM support is satisfied by the terminal software platform. For instance, if the service requires Java-Swing AND Java-Speech support and the terminal has only Java-Swing, the service in not adapted; but if there is OR instead of AND the service is adapted.

If we want to be able to adapt the service to a parameter P, the Service Adapter must have the knowledge about this parameter P; this knowledge is given by the three aspects previously presented: definition domain, composition operator, compatibility operator. In figure 5 we have depicted a table with several parameters that may be included in the component profiles.

Parameter name	Domain	Context Association	Association point (component)	Composition operator	Compatibility operator	Observations
Language	{EN, FR, DE, … *, ?}	User	IHM, passed objects	:= Assignment OR ⇔, Equivalence	=, Equals OR $\cap \neq \emptyset$	
Memory	(0…M_max)	Terminal, Machines	Whole component, passed object	+, Sum	M_req < M_disp	Logical components/ physical machines
IHM Support	{HTML, WML, Swing, …}	Terminal API, OS	IHM	∪, Reunion	$\cap \neq \emptyset$	
GUI size	{(m1xn1, (m2xn2), ….)}	Terminal display	IHM	+, Sum OR **Max**, maximum	S_req > S_disp	

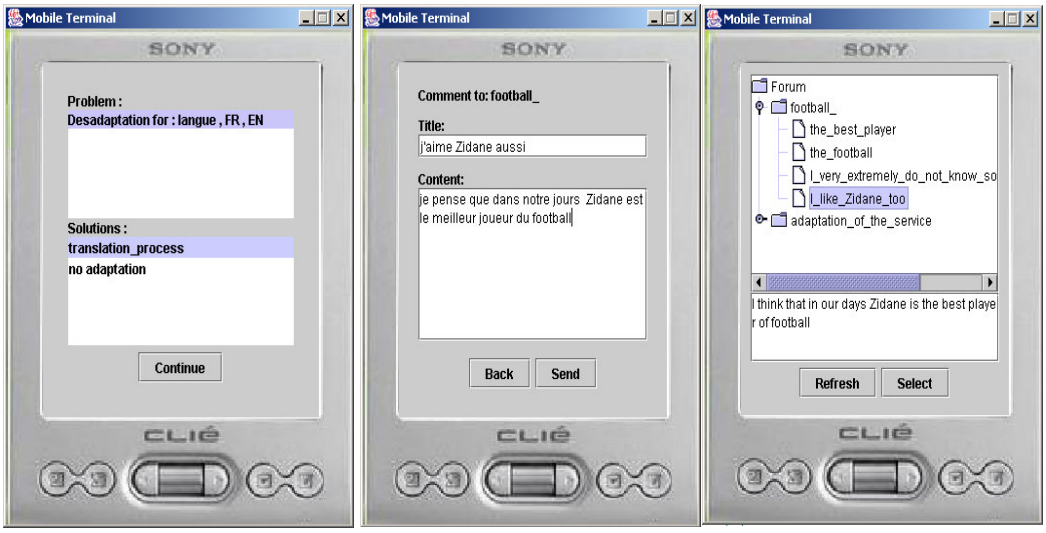
Figure 5. Profile composition

For some parameters additional information is required at the level of service description, for instance, the "memory" parameter requires to know how the software components are distributed over the physical machines.

III. IMPLEMENTATION

In order to verify our model we have implemented a prototype of the forum service. The F, forum server, is a service included in the educational portal of the University of Savoie, France. The forum server component, F, is accessible by SOAP (or by HTML using a browser). The V, forum view, and C, message composer, GUI components are considered here to be installed on the mobile device, a PDA for instance. The PDA does not have a HTML browser so the components V and C use SOAP to connect to F.

In figure 6 we can see the V and C IHMs. A message is written in French, the Service Adapter detects the language problem and asks the user if he prefers to use a translator or not. The user chooses to use the translator and the service changes by including a new component, a translation Web Service accessible by SOAP. The Service Adapter inserts this new translation component between F and C for the syntactic compatibility reasons at the interface level. A regular graph search algorithm is used by the Service Adapter in order to propagate the parameter values, conditions and to verify the parameters compatibility.



a **b** **c**

Figure 6. The forum prototype GUI

Technologies. The most important technologies used in order to build our prototype were:

- The java components C and V are described using the CCM component model. We have created an XML based language describing the component interfaces and also the service architecture.
- The language detection and translation components are Web Services accessible by SOAP. We use java proxy components for these two components.
- In order to be able to modify the service at runtime, we have used dynamical interaction patterns presented in [4, 5], the Noah platform [4] and the ISL (Interaction Specification Language) [4]. The Service Adapter uses interaction pattern to dynamically add the translation component. Java reflection was also used.

IV. RELATED WORK

Service adaptation issue was broadly studied and many researchers have proposed different solutions. Ubiquitous computing gives us a lot of situations where service adaptation is needed. The majority of the proposed solutions use an anticipated approach: the service has, from its creation state, all the mechanisms necessary to adapt themselves to several distinct contexts.

Some papers like [9] propose a semantic component description using semantic graph and ontology but they are using the semantic component description in order to automatically deploy services from a user natural language based description.

According to [10], in a completely unanticipated system, the answers to questions like *when, where, what* and *how* to 'unanticipated' reconfigure are known only at runtime. But even the "Chisel" [10] system that claims to be 'completely requires a human operator for modifying the rules/strategies. This fact means that a human operator and not the system, establish when the service is adapted and what the system must do in each situation.

Another limitation of this proposal is that only the rules are reconfigurable, not the context event types that are pre existent, and the strategies/actions names (and definitions) also. In the "Madam" platform [11], the idea is to use one abstract service architecture description that may be automatically projected towards several possible concrete implementations, function on the particular context. The implementation choices are based on some 'utility functions' which are defined using rules. The utility functions are service-specific and they must by specified by the developer and this is a not very simple task. In this case, we have the same situation that is not suitable for autonomic computing: the use of service-specific rules/functions defined by the service developer. Another limitation is that the choice is limited to the pre-existent component implementations and a new, dynamical component insertion in the abstract architecture is not taken into account.

V. CONCLUSIONS AND FUTURE WORK

In this paper we presented a platform for dynamical and unanticipated service adaptation, having as central point the *Service Adapter* module, which is basically an expert system performing the following: verifies if a service is adapted to a specific context, searches the solutions if the service is not adapted and applies the solutions. A service is seen here as a component-based architecture and it may be adapted by: adding, removing, replacing or modifying the components. The *Service Adapter* needs a knowledge representation about the: service, context and components in general. The existent component models do not give us enough semantic

information in this direction so a new concept is introduced: the *profile.*

An important aspect of our proposal is the profile composition. The composition is achieved by using a graph. The graph's nodes are the points that are associated with a profile parameter and the arcs are the component connections and internal dependences.

Another important issue is the profile extension, the platform is able to use a new profile parameter if we are specifying for this parameter: a name, a definition domain, a composition operator and a compatibility operator. The extension algorithm assures a *generalization* of the proposed model: the same solution search algorithm may be used for different attributes it the attribute is specified as depicted in the table from figure 5.

Hence, the Service Adapter may use a generic algorithm for the parameter composition and compatibility verification it will use the operator that we have specified. We are looking forward to use some AI algorithms such as logical inference engines and genetic algorithms and add also a learning function to *Service Adapter*.

Our model was tested with a prototype, which adapts a forum service to the user language by searching and inserting a new translation component into the original forum service. This insertion may be done at runtime due to the reflective platform that is used, based on dynamical interaction patterns. The languages used were English, French and German but the model should enable any other languages like Bulgarian and Chinese. In fact we are no creating these components but we only suppose their existence and if the required component exists we use it.

At this moment, we have tested only the component insertion strategy. As future development, we intend to create a strategy selection algorithm that will be able to combine different operations such as: insertion, replacement, parameterization and migration of components. Another future issue is to enhance the profiles model by using semantic graph and build a test platform that will allow us to test more complex adaptation scenarios.

REFERENCES

[1] Bruneton E., Coupaye T.,Stefani J. B, "Recursive and Dynamic Software Composition with Sharing", *Proceedings of ECOOP Workshop on Component-Oriented Programming*, Malaga, Spain, 2002.

[2] OMG Specification, CORBA Components, version 3.0, June 2002, http://www.omg.org/technology/documents/formal/components.htm

[3] Capra L., Emmerich W., Mascolo C., "Reflective Middleware Solutions for Context-Aware Applications", *Proceedings of the Third International Conference on Metalevel Architectures and Separation of Crosscutting Concerns*, Springer-Verlag London, UK, 2001, p. 126-133.

[4] Blay-Fornarino M., Ensellem, D., Occello A., Pinna-Dery A-M., Riveill M., Fierstone J., Nano O., and Chabert G., "Un service d'interactions : principes et implementation", *Actes des Journées composants*, INRIA, Grenoble, France, 17-18 octobre 2002, p. 175-186.

[5] Aksit M., Choukair Z., Dynamic, "Adaptive and Reconfigurable Systems Overview and Prospective Vision", *Proceedings of ICDCSW'03*, Providence, Rhode Island, USA, May 19-22, 2003., p. 84-92.

[6] W3C Recommendation 15 January 2004, Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies 1.0, http://www.w3.org/TR/2004/REC-CCPP-struct-vocab-20040115/

[7] Cremene M., Riveill M., Martel C., Loghin C., "Adaptation de services nomades par assemblage dynamique de composants", *Proceedings of DECOR'04,* Grenoble, France, 2004. pag. 53-65. ISBN-2726112765

[8] Kon, F., Costa, F., Blair, G.S., Campbell, R., "The Case for Reflective Middleware: Building middleware that is flexible, reconfigurable, and yet simple to use", *ACM Communications*, Vol 45, No 6, 2002.

[9] Fujii K., Suda T., "Component Service Model with Semantics (CoSMoS) : A New Component Model for Dynamic Service Composition", *Proceedings of the International Symposium on Applications and the Internet Workshops (SAINTW'04),* Tokyo, Japan, 2004, p. 348-355

[10] J., K., *Completely Unanticipated Dynamic Adaptation of Software*. PhD thesis, University of Dublin, Trinity College, Distributed Systems Group, October 2004.

[11] Floch, J., Hallsteinsen, S., Stav, E., Eliassen, F., Lund, K., Gjorven, E. "Using architecture models for runtime adaptability", *IEEE Software,* 23(2), 2006, p. 62-70.