Int. J. of Computers, Communications & Control, ISSN 1841-9836, E-ISSN 1841-9844
Vol. VII (2012), No. 1 (March), pp. 53-60

Adaptation Mechanism based on Service-Context Distance for Ubiquitous Computing

M. Cremene, M. Riveill, A. Rarau, C. Miron, B. Iulian, V. Todica

Marcel Cremene, Anca Rarau, Costin Miron,
Benta Iulian, Valeriu Todica
Technical University of Cluj-Napoca Romania,
Cluj-Napoca, Memorandumului nr. 28, 400114
E-mail(s): cremene@com.utcluj.ro, anca.rarau@cs.utcluj.ro,
miron@bel.utcluj.ro, iulian.benta@com.utcluj.ro,
todicav23@yahoo.com

Michel Riveill
University of Nice, Sophia-Antipolis
I3S, Route de Colles, BP 145 - F-06903,
Sophia Antipolis CEDEX
E-mail: riveil@unice.fr

Abstract: Service adaptation is one of the main research subjects in Ubiquitous Computing. Dynamic service adaptation, at runtime, is necessary for services that cannot be stopped (banking, airport, etc.). The classical approaches for dynamic adaptation require predicting all service and context states in order to specify service and context-specific adaptation policies. This prediction may lead to a combinatorial explosion. The aim of this research is to create a service and context-independent adaptation mechanism. Our proposal is based on a service-context model that is causally connected with the service and context, in a *model@run.time* paradigm. A closed-loop control principle is used for the adaptation mechanism. We introduce an equivalent for the *error* that is expressed by the notion of *service-context distance*. This distance represents a measure of how *adequate* is a service to its context. This distance is computed by some generic, reusable components. The adaptation algorithm that minimizes this distance is also service and context-independent.
Keywords: Services, Dynamic Adaptation, Context, Autonomic computing, Adaptation Control

1 Introduction

Background and motivation. Service adaptation to the context (physical infrastructure/resources, user needs, and environment) is one of the main research subjects in high interest domains such as Ubiquitous/Pervasive/Mobile Computing. Adapting a service is to reconfigure that service in order to maximize its QoS (Quality of Service) and also the efficiency of the resource utilization. We use a component-oriented approach for services, thus the reconfiguration concerns the service architecture: parameterize, add/remove, connect/disconnect or migrate components.

Numerous situations require a dynamic service adaptation [1,2], at runtime, because there are an important number of services that cannot be stopped (modified and recompiled). Examples are: banking services, airport services, spatial services, corporate services, groupware services and others. Even for services that may be stopped, the human intervention means important time and costs.

Issues about dynamic adaptation control mechanism. The adaptation mechanism, that encapsulates the system "intelligence", is the most important part of an adaptive system. It main function is to decide *when* a service should be adapted and *how* to adapt it.

After studying several different approaches for dynamic service adaptation, some of them presented in [3], we observed that the large majority of these solutions are based on the prediction about all possible service and context states. This prediction is an essential condition for specifying adaptation policies. Usually these policies are service and context-specific, they are not reusable and they do not evolve as the context evolves, without human intervention. A service can be adapted only inside the limits fixed a priori by these services and context-specific polices. For instance, in the case of "Event-Condition-Action" adaptation mechanism, which is one of the most used, a service cannot be adapted if the events, the conditions and the actions have not been predicted/specified priori by a human expert. But these events, conditions and actions are service and context-specific. This means that the human expert decides a priori *when* and *how* to adapt a service.

The problem is that, the prediction of all possible service and context states, may lead to a combinatory explosion. For instance, a state machine specification MDE (Model Driven Engineering) approach leads to a extremely high number of possible artefacts, as it is shown in [2]. Another problem of the classical state machine approach is that it does not deal with adding/removing/changing dynamically new states and transitions. Thus, such an approach is impossible to be used in practice for complex services and contexts.

Objective and approach. The aim of this research is to build a service and context-independent (generic) adaptation mechanism. Our proposal is based on a *model@run.time* approach. According to this approach, we propose a service-context model describing the service and the context as a whole system. This model is causally connected with the service and the context. The service-context model is represented as a directed graph having as nodes the service components and the context elements. Each node has attributes related to the entities (services' components, context elements) and to the interactions between these entities.

A closed-loop control principle is used for the adaptation mechanism, as suggested by the *Autonomic Computing* paradigm [4]. We introduce an equivalent for the notion of *error* that is expressed in our model by the notion of *service-context distance*. This distance represents a measure of how *adequate* is a service to its context in terms of user needs satisfaction and resources utilisation. The adaptation algorithm is also service and context-independent.

Paper outline. This paper is organized as it follows: the next section is an overview about the dynamic service adaptation issues and the models@run.time approach. Section three presents the proposed solution that is based on the service-context model, service-context distance and a generic adaptation algorithm. Section four present presents the conclusions.

2 Dynamic service adaptation in a models@run.time approach

2.1 Dynamic service adaptation

For defining more precisely what *service adaptation* means for us, we consider a mixture between the categories defined by the *Autonomic Computing* [4] and the adaptation types proposed in [1]:

a. Self-healing/Corrective adaptation. The system automatically detects, diagnoses, and repairs localized software (and hardware) problems. For example, if a system component responds too slow it will be replaced by another component, with the same functions, that responds on time (or the component will be moved on another physical machine). Self-healing/corrective adaptation deals usually with failure cases, when a component must be replaced/moved.

b. Self-adaptation. The adaptation is necessary when the context state changes. We may have two cases of self-adaptation:*Adaptive adaptation* concerns the situations when the context change and the service should change also in order to keep its functions. This type of adaptation is usually transparent to the user.*Extending adaptation* concerns the situations when some new user needs are discovered a posteriori and the service should be extended in order to satisfy these new needs.

c. Self-optimization/Perfective adaptation. The system continuously searches for opportunities to improve its own performance. The performance is maximal when the QoS parameter are maximized (for instance the response time) and when the resource utilization is minimal and well balanced. Self-optimization/Perfective adaptation deals with a service that is working correctly but it may be tuned in order to increase its efficiency/quality.

The adaptation is *dynamic* when the service reconfiguration happens at runtime. In the last years, numerous researchers have concentrated on developing middleware that enables dynamic adaptation. For instance, the *WComp* middleware [5, 6] offers a support for composing and adapting services at runtime. The design and the execution phases are simultaneous. An event-based communication model is used. It offers the possibility to use services installed on various devices and also web services from Internet. WComp manages also the dynamic service apparition/extinction.

Fractal [7] proposes a middleware and a component model enabling hierarchical composition. The middleware offers support for reflection and dynamic reconfiguration. A particularity of Fractal is the possibility to share a component between two other composite components.

OpenCCM (Open CORBA Component Model) [8] is the first public available and open source implementation of CCM (CORBA Component Model).

OSGi propose a platform based on dynamic modules/components that may be assembled at runtime, it provides a service-oriented, component-based environment. A service-oriented middleware allowing spontaneous distributed service composition at runtime, based on OSGi, is proposed in [9]. Facing with a high number of existent middleware, our intention is to propose an independent solution that may be adapted to different middleware.

2.2 The *models@run.time* approach

Models@run.time [10] represents a novel approach that aim is to extend the usage of software models (MDE - Model Driven Engineering) to runtime. The need of such models is motivated by the dynamic adaptation of mission-critical software issues. This is also related with *Autonomic Computing* paradigm because the adaptation should be autonomous (without human intervention).

This approach is also strongly related with the *reflection* paradigm and has a similar principle: a *model@run.time* is an abstraction of a system that is causally connected with that system and may be used for dynamic system adaptation. The difference comparing to reflection is that in model@run.time we look for high-level models while in reflection paradigm the (meta)model is strongly related to the low-level software aspects [10]. Such a model@run.time describes aspects as: structure, behaviour and goals of the system, from a problem space perspective.

One of the advantages of the *model@run.time* approach is the possibility to treat independently two main issues:

I. Propose a model that will be used at runtime for discovering the service-context adequacy problems and for adapting the service by solving these problems. Our proposal is based on a *service-context* model, a service-context distance that reflects the service-context adequacy and an adaptation algorithm that aim is to minimize this distance (and to increase the adequacy).

II. Create a middleware that will causally connect the two parts: a) the proposed service-context model with b)the adapted system (a modification of one part will be reflected on the other part). If we want to offer a general solution, a part of this middleware will make the adaptation with a specific adaptation support-middleware (see the examples presented in the section 2.1).

The second issue is more a technical issue than a scientific one because today many middleware and context monitoring tools offers a large number of solutions that may be reused. That is why; in this paper we are concentrated only on the first issue which is the most important and difficult.

3 The service-context model, the service-context distance and the adaptation algorithm

3.1 Service-context model

We have introduced the *service-context model* concept in [3, 11] as a possible approach for autonomic computing. In this paper we re-interpret this model from a *models@run.time* perspective, we extend it with the service-context distance.

Service-context graph. The service-context model (figure 1) aim is to describe the service and the context as a whole system that may be analyzed from the service to context adequacy point of view. This model is based on a directed graph composed by the following elements:

a. The nodes correspond to service and context entities. The service′ entities (S in fig. 1) are the software components that constitute the service. In order to simplify the model, a complex component is described by decomposing it in a sum of basic components having only three types: source (1 output), sink (1 input) or filter (1 input, 1 output). The context entities are context components: users (U in fig. 1), infrastructure elements (I in fig. 1) such as devices, networks and environment elements (E in fig. 1).

b. The vertex corresponds to the relations and interactions existent between these entities. We define three types of vertex corresponding to two types of possible relations/interactions:

   - *Informational flows* interactions. The service components are communicating by exchanging information. Also, the user exchange information with the service. This information has some *attributes* that describes it.
   - *Resource utilization* interactions. The service components consume resources provided by the hardware infrastructure (memory, cpu, bandwidth, etc).
   - *Environment influence* interactions. The environment may influence the user needs (ex. location) or the infrastructure (ex. the rain effect on a radio network).

For simplifying the model, we will discuss in this paper only the most important interactions which are the first two: information flows and resource utilization.

Attributes. The graph is annotated with some attributes. Each node has attributes related to the node itself or to the node ports (input and output vertex). These attributes are making a semantic connection between the service nodes and the context nodes. For instance, the 'language' attribute means the user U language but also the service S language, the 'memory' attribute means the memory consumed by a component and the memory provided by a device.

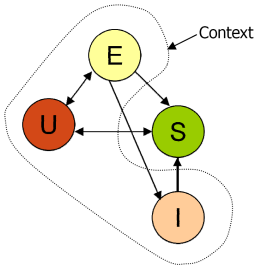


Figure 1: Service-Context general model

A common attribute vocabulary for service and context components should be respected by the component developers/providers.

For each attribute a input and output domain are specified. For filter-type components, a *transfer function* (input-output) should be specified. For instance, a translation component change the language between the input and the output, a compression component change the compression rate, etc.

The attributes, their input/output domains and the transfer functions should be specified for each component by the component developer/provider, in a form of a *profile* (i.e. an XML-based descriptor of the component). This is a necessary condition for enabling the machine to understand what a component does. Same thing for the context elements, each one has a *profile*.

Composition. One of the most important property of the service-context model is the *composition*. By composition we understand here to assimilate a graph to a node. This is a necessary condition for describing the service behaviour and properties using only the service's internal components behaviour and properties. We propose a composition mechanism based on the attributes specified by the profiles. The elementary composition cases are described in figure 2. The composition means to determine the equivalent node for a graph composed by two nodes.

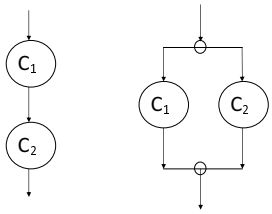


Figure 2: Serial and parallel composition

The composition operation depends on the attribute nature. The composition is different for information flows interactions and for resource utilization interactions. These operators are described in the figure 3.

3.2 Service-context distance

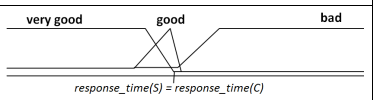
The first idea for describing the service-context adequacy, used in [11], was to use a binary approach: a service is adequate or not to its context. In order to have a more general approach, we introduce in this paper the concept of *service-context distance*. The distance is minimal when the service to context adequacy is maximal. This distance has two components: the *S-U distance* measures the distance from the information flow point of view and the *S-I distance* measures the distance from the resource utilization point of view.

Information flow composition		
Atribute name	**Serial composition**	**Parallel composition**
response_time	*Sum, +*	*Max function*
security_level	*Min function*	*Min function*
language	*language of the output component*	*Intersection*
type_hmi	*type_hmi of the output component*	*Reunion*

Resource utilization composition			
Atribute name	**Paralel composition**	**Serial composition**	**Shared composition**
memory	*Sum, +*	*Max function*	$m_1 + m_2 - m_{shared}$
cpu_time	*Sum, +*	*Max function*	$t_1 + t_2 - t_{shared}$
screen_surface	*Reunion*	*Max function*	*Reunion*

Figure 3: Composition operators for different attributes

Each distance is represented as a vector having as components the distances corresponding to each attribute. For each attribute we define a specific distance that will be computed by some dedicated components/services. In some cases the distance is expressed by a simple formula or set of rules. Some examples are described in figure 4.

Information flow distance evaluation		
Attribute name	**Domain**	**Distance**
response_time	*0..inf*	
security_level	*{none, low, medium, high, very high}*	*d = 0 if security_level(S) >= security_level(U) and ∞ else*
language	*{RO, FR, DE, ES……..}*	*d = 0 if language(S) == language(U) and ∞ else*

Resource utilization distance evaluation		
Attribute name	**Domain**	**Distance**
memory	*0…100%*	*d = memory(S) / memory(I) * 100*
cpu_time	*0..100%*	*d = cpu_time(S) / cpu_time(I) * 100*
network_capacity	*0..100%*	*d = network_capacity(S) / network_capacity(I) * 100*

Figure 4: Distance expressions for different attributes

A more complex distance is the distance between the user requests expressed in natural language and the service features. For such complex distances we cannot present a simple formula but we use dedicated component/services that implement the algorithm for computing the distance. Such an algorithm is proposed in [12].

The model may be extended by adding new attributes definitions and new distances measuring components.

3.3 Adaptation algorithm

The adaptation algorithm goal is to minimize the service-context distance. For doing that, the adaptation algorithm dynamically transforms the service-context graph from a less adequate one toward a more adequate one.

The primitive operations used by the adaptation algorithm are the following: changing a node parameter, connecting/disconnecting a node, inserting a new node, removing a node, replacing a node. In order to minimize the service-context distance, the adaptation algorithm should find a list of primitive operations that transforms the graph from the current state into the desired state (for the desired state the service-context distance is minimal). Several alternative solutions may exists. In order to make a difference between these possible solutions, we associate an *adaptation cost* to each primitive operation. For instance, the parameterization has a cost equal to 1 while the insertion cost is 5. This is because it is faster and easier to apply a parameterization than an insertion.

We have implemented an algorithm, described in [11], that uses one attribute and one adaptation strategy: the insertion of a new component. This algorithm starts from a mismatch between the desired value *V2* for an attribute *A* and the current value *V1* and search for a new component that transforms the value of the attribute from *V1* to *V2*. The place where the new component may be inserted is searched based on the syntactic interface compatibility. The user is confirmation is asked before transforming the service.

We are working in present on a general adaptation algorithm, able to apply a succession of different primitive operations and deal with several attributes simultaneously. We have done some tests with genetic algorithms which seem promising but are not very fast.

3.4 Implementation

As a proof of concept, in [11] we have implemented a simple forum service that is adapted dynamically to the user language. The components are developed in Java, using the CCM (Corba Component Model) specifications. The adaptation consists in inserting a translation component, at runtime. We have used a mechanism based on ISL (Interaction Specification Language) that is used also in WComp middleware [6].

In present we are developing a general service-context simulator, based on the open source JGraphX API that should allow us to test the adaptation at the model level by simulating various service and context states.

4 Conclusions

The issue discussed in this paper was to propose a service and context-independent adaptation mechanism for dynamic service adaptation. We have proposed a solution, based on the service-context model and service-context distance concepts, that fits into the very recent *model@run.time* approach. The generality of the proposed model is given by its following properties: the service attributes/behaviour are determined automatically by composing the attributes/behaviour of its internal components, the service-context distance depends only on the model attributes nature and are not dependent to a particular service/context and the adaptation algorithm is also service/context independent. As future work we intend to generalize the adaptation algorithm.

Acknowledgments

This work was supported by CNCSIS-UEFISCSU, PNII-IDEI, project number 1062/2007. Thanks to members of the Rainbow team, I3S laboratory, Sophia-Antipolis.

Bibliography

[1] A. Ketfi, N. Belkhatir, and P.-Y. Cunin, "Automatic adaptation of component-based software: Issues and experiences," in *PDPTA '02: Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, pp. 1365–1371, CSREA Press, 2002.

[2] B. Morin, O. Barais, J. M. Jezequel, F. Fleurey, and A. Solberg, "Models@ run.time to support dynamic adaptation," vol. 42, pp. 44–51, October 2009.

[3] M. Cremene, *Adaptation Dynamique de Services.* PhD thesis, Double coordination between University of Savoie, France and Technical University of Cluj-Napoca, Romania, 2005.

[4] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[5] D. Cheung-Foo-Wo, J.-Y. Tigli, S. Lavirotte, and M. Riveill, "Self-adaptation of event-driven component-oriented middleware using aspects of assembly," in *MPAC '07: Proceedings of the 5th international workshop on Middleware for pervasive and ad-hoc computing*, (New York, NY, USA), pp. 31–36, ACM, 2007.

[6] J.-Y. Tigli, S. Lavirotte, G. Rey, V. Hourdin, D. Cheung-Foo-Wo, E. Callegari, and M. Riveill, "WComp Middleware for Ubiquitous Computing: Aspects and Composite Event-based Web Services," *Annals of Telecommunications (AoT)*, vol. 64, Apr. 2009.

[7] E. Bruneton, T. Coupaye, M. Leclercq, V. Quéma, and J.-B. Stefani, "The fractal component model and its support in java: Experiences with auto-adaptive and reconfigurable systems," *Softw. Pract. Exper.*, vol. 36, no. 11-12, pp. 1257–1284, 2006.

[8] S. Gorappa and R. Klefstad, "Empirical evaluation of openccm for java-based distributed, real-time, and embedded systems," in *SAC '05: Proceedings of the 2005 ACM symposium on Applied computing*, (New York, NY, USA), pp. 1288–1292, ACM, 2005.

[9] A. Bottaro, A. Gerodolle, and P. Lalanda, "Pervasive service composition in the home network," in *AINA '07: Proceedings of the 21st International Conference on Advanced Networking and Applications*, (Washington, DC, USA), pp. 596–603, IEEE Computer Society, 2007.

[10] G. Blair, N. Bencomo, and R. B. France, "Models@ run.time," *Computer*, vol. 42, no. 10, pp. 22–27, 2009.

[11] M. Cremene, M. Riveill, and C. Martel, "Autonomic adaptation solution based on service-context adequacy determination," *Electron. Notes Theor. Comput. Sci.*, vol. 189, pp. 35–50, 2007.

[12] M. Cremene, J.-Y. Tigli, S. Lavirotte, F.-C. Pop, M. Riveill, and G. Rey, "Service composition based on natural language requests," in *IEEE SCC*, pp. 486–489, 2009.