

A Model of Self-Adaptive Distributed Components

An Phung-Khac, Antoine Beugnard, Jean-Marie Gilliot, and
Maria-Teresa Segarra

Department of Computer Science, ENST Bretagne
Technopôle Brest-Iroise - CS 83818 - 29238 Brest Cedex 3 - France
{an.phungkhac, antoine.beugnard, jm.gilliot, mt.segarra}@enst-bretagne.fr

Abstract. Computer software must dynamically adapt its behavior in response to changes in variable environments. In the context of distributed systems, where adaptations are performed in many sites, coordinating adaptations across sites is critical to ensure the correctness of applications during and after adaptations. Developing self-adaptive applications is thus more difficult. Addressing this issue, we introduce a model of self-adaptive distributed components. The model enables applications to coordinate distributed adaptations by specifying adaptations and communications separately. The model also enables self-adaptive applications to adapt at runtime without loss of information.

1 Introduction

In the presence of environment variability, computer applications must dynamically modify their behavior to adapt to changing context conditions. Developing self-adaptive applications can thus be a challenging software development problem.

Many research efforts have proposed solutions for building self-adaptive applications. Some efforts focus on adaptation techniques such as modeling applications with connections, disconnections and reconnections of components [1, 2]; or proposing adaptation models or customizable frameworks [3]. More formally, some other approaches focus on adaptation specifications and/or processes for building self-adaptive applications [4–6]. Few efforts have resolved the problem of coordinating multiple distributed adaptations [7, 8], especially in the context of component-based applications which is now widely used in software development.

Addressing this issue, we present a model of self-adaptive distributed components. Compared to existing approaches, our approach enables applications to coordinate distributed adaptations dynamically by specifying adaptations and communications separately. Moreover, our model renders applications able to adapt their behavior *at runtime without loss of information*.

Further, as an open issue, out of the scope of the article, we propose to build a design process for specifying and implementing our model; our process can be implemented by model transformation programs.

The next section introduces the *communication component* - an architecture to specify and implement distributed collaborations, proposed in our laboratory by E. Cariou *et al* [9]. Believing that adaptations responding to distributed environment changes must be managed at the communication level of applications, we reuse and develop the architecture of *communication component* for constructing a self-adaptive distributed components model that will be described in Section 3. Section 4 discusses several open issues and concludes the paper.

2 Communication component

A *communication component* is a communication abstraction under the form of a software component. An application is built by interconnecting *functional components* with *communication components* that manage their communication. In order to differentiate a *communication component* from other functional one, the former is called *medium* [9].

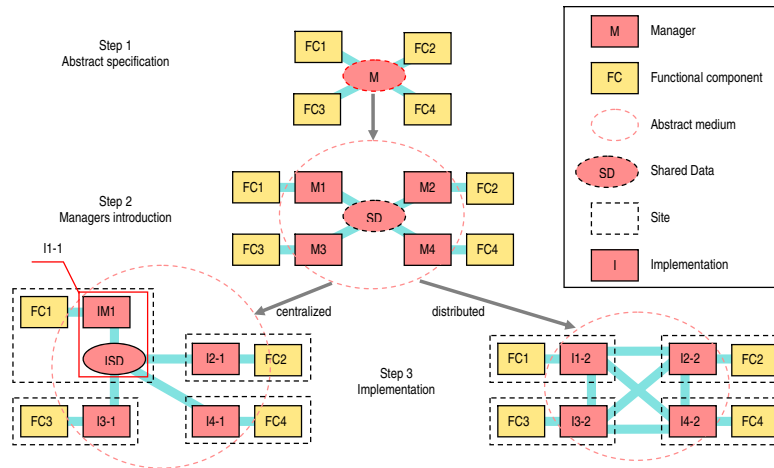


Fig. 1. Three steps of the medium refinement process

To reify a communication abstraction into an implementable software component, E. Cariou *et al* propose a refinement process that transforms a *medium* specification from an abstract level to an implementation level. Figure 1 shows three steps of such a process for an application deployed over 4 sites. In the first step, *functional components* are interconnected via an *abstract medium* which is refined in the second step by introducing *managers*, one per functional component. These *managers* give the distributed nature to the *medium*. From the viewpoint of each *functional component*, the associated *manager* is a proxy. The *medium* is thus composed of several *managers* and the *shared data* of the

medium. The purpose of the third step is to make the *shared data* disappear by distributing it on *managers*. Therefore, the *medium* is completely transformed in the implementation level. Corresponding to many distribution strategies, this step can generate several *medium* implementation variants (For example, in Figure 1, a *medium* specification at Step 1 is transformed into 2 variants at Step 3)

In [10], E. Kaboré *et al* have extended the *medium* refinement process in order to inject distribution algorithms into a *medium*. This process transforms an *abstract medium* into more than 10 *medium variants* at the implementation level using design variants such as data placement, data representation or data replication.

3 A model of self-adaptive distributed components

Generally, in order to perform an adaptation session, the self-adaptive application must observe environment changes, make an adaptation decision, plan adaptation strategies, choose and execute the most suitable one.

In the context of distributed systems, making adaptation decisions and planning adaptation strategies are more difficult because of complex constraints between distributed adaptations that need to be performed in a same adaptation session. To facilitate them, we propose a model of self-adaptive distributed components that enables applications to coordinate distributed adaptations.

In this section, we will present the first version of our model. The model is built from the need of a distributed application to change its distribution algorithm dynamically.

3.1 Constructing the model

Figure 2 shows such a model for a distributed application deployed over 4 sites (corresponding to the application in Figure 1). This model is built by embedding all *medium variants* at the last step of the *medium* refinement process into only one *composite medium*. As shown in the figure, all implementation variants of a *manager* are embedded into a *composite manager* as sub-components. At runtime, in each *composite manager*, one *manager variant* is activated by connecting it to an *adaptation manager*. A rule is implemented in every *adaptation manager* to ensure that the set of activated *manager variant* correctly corresponds to a *medium variant*. For example in the case of the application in Figure 2, only 2 sets corresponding 2 *medium variants* can be activated: {I1-1; I2-1; I3-1; I4-1} and {I1-2; I2-2; I3-2; I4-2}.

Thanks to this architecture, application's adaptations responding to distributed environment changes can be carried out by the *composite medium*.

3.2 Coordinating adaptations

When an adaptation session is performed, each *composite manager* changes its active *manager variant* thanks to its *adaptation manager*. This *adaptation man-*

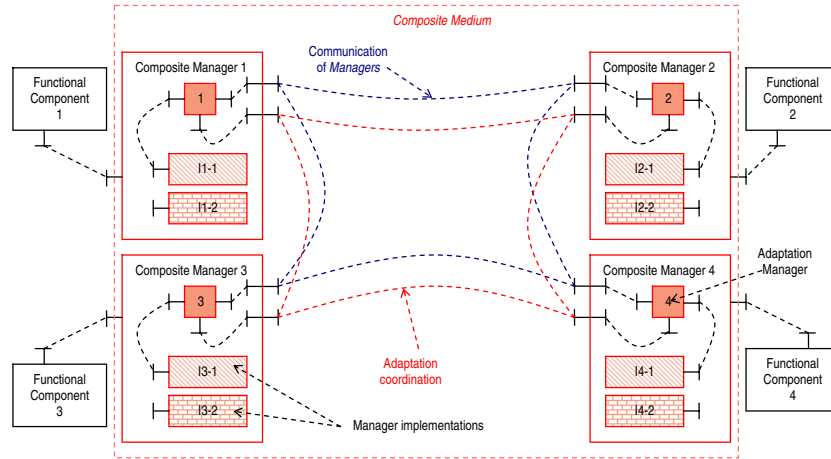


Fig. 2. Model of self-adaptive distributed components

ager can collaborate with other ones by using their *coordination protocols*. The application thus changes its global state. In fact, changes of an active *manager variant* is very complex because it requires coherent changes of many properties or data of *composite managers*. We argue that our model can resolve this problem.

Since all variants are derived from a single abstract model, the application can coordinate distributed adaptations dynamically and can adapt at runtime without loss of information or data. The current implementation state can be abstracted in order to be redistributed in the new implementation state.

Thanks to *coordination protocols*, the application can also add/remove a site into/from the global system. In this case, *adaptation managers* can recognize the role type of the connected or disconnected site to re-arrange the communication of system.

3.3 Building medium variants

Our work focuses on the research of a design process for the presented model. This process is based on the *medium* refinement process, with an adaptation specification as input. At the beginning of this process, the communication and the adaptation of an application are abstracted in an *abstract medium* that is step by step refined. In each step, some specified adaptation aspects are introduced with many configuration choices and then several *medium* variants are generated.

In other words, a *medium* can be considered as an abstraction with a state that is concretized by a refinement process into many implementation variants of the concrete state.

In the terminology of the Model-Driven Architecture (MDA) approach, this process enables transformations of a PIM (Platform Independent Model) specification into one or several PSM (Platform Specific Model) specifications [11]. These transformations can be implemented by model transformation programs that can automatically generate the model of self-adaptive distributed components.

4 Discussion and conclusion

We have presented in this paper a model of self-adaptive distributed components addressing the problem of distributed adaptations coordination, in the context of self-adaptive component-based software development.

In fact, adaptation systems are more complex. For building complete self-adaptive applications, this model still lacks an *observer* to observe environment changes, a *decider* to start an adaptation session, an *optimizer* to choice the most suitable variant. Many existing research efforts have addressed this issue [12, 7]. In this paper, we are only interested in the coordination of distributed adaptations.

To validate and improve this model, some issues require more investigation:

- *Implementing coordination protocols*: In fact, the proposed *coordination protocol* has not been enough detailed for effective coordinations. We would like to improve it for the change of active variant and the connection or disconnection of sites, thus give self-adaptive applications an open and seamless adaptive feature.
- *Evaluating medium variants*: In order to build a complete model of self-adaptive distributed components, we propose to introduce into each *composite manager* an *optimizer*. In coordinating together, *optimizers* can evaluate variants qualities in each environment state, thus enables applications to activate the most suitable one.
- *Implementing the model*: To achieve executable self-adaptive systems, we intend to implement our model in the Fractal component framework [13]. This framework renders us able to implement *adaptation managers* easily.
- *Specifying adaptations*: As presented in Section 3.3, in order to generate our model automatically, adaptations and application states must be specified as input of the refinement process. Many existing works address this problem, such as specifying adaptation behaviors in [4] or specifying adaptation goals in [6]. From these successes, we believe that the adaptation modeling is possible.

In [10], the design process for building variants of *communication components* has already been successfully automatized thanks to model transformations. We expect to use this process to embed many variants of a *medium* in a component in order to achieve the self-adaptiveness of distributed components.

References

1. David, P.C., Ledoux, T.: Towards a Framework for Self-Adaptive Component-Based Applications. In Stefani, J.B., Demeure, I., Hagimont, D., eds.: Proceedings of Distributed Applications and Interoperable Systems 2003, the 4th IFIP WG6.1 International Conference, DAIS 2003. Volume 2893 of Lecture Notes in Computer Science., Paris, Federated Conferences, Springer-Verlag (2003) 1–14
2. Ben-Shaul, I., Holder, O., Lavva, B.: Dynamic Adaptation and Deployment of Distributed Components In Hadas. *IEEE Trans. Softw. Eng.* **27**(9) (2001) 769–787
3. Segarra, M.T., André, F.: A Framework for Dynamic Adaptation in Wireless Environments. In: TOOLS '00: Proceedings of the Technology of Object-Oriented Languages and Systems (TOOLS 33), Washington, DC, USA, IEEE Computer Society (2000) 336
4. Zhang, J., Cheng, B.H.C.: Model-Based Development of Dynamically Adaptive Software. In: IEEE International Conference on Software Engineering (ICSE06), Shanghai, China, IEEE (2006)
5. Occello, A., Dery-Pinna, A.M.: Capitalizing Adaptation Safety: a Service oriented Approach. In: Proceedings of The ECOOP Third International Workshop on Coordination and Adaptation Techniques for Software Entities (WCAT'06). (2006)
6. Brown, G., Cheng, B.H., Zhang, J.: Goal-oriented Specification of Adaptation Requirements Engineering in Adaptive Systems. In: Proceedings of IEEE ICSE Workshop of Software Engineering of Adaptive and Self-Managing Systems (SEAMS06). (2006)
7. Chefrou, D.: Developing component based adaptive applications in mobile environments. In: SAC '05: Proceedings of the 2005 ACM symposium on Applied computing, New York, NY, USA, ACM Press (2005) 1146–1150
8. Ensink, B., Adve, V.S.: Coordinating Adaptations in Distributed Systems. In: Proceedings of The 24th International Conference on Distributed Computing Systems (ICDCS 2004), Hachioji, Tokyo, Japan, IEEE Computer Society (2004) 446–455
9. Cariou, E., Beugnard, A., Jézéquel, J.M.: An Architecture and a Process for Implementing Distributed Collaborations. In: Proceedings of the 6th IEEE International Enterprise Distributed Object (EDOC 2002), Lausanne, Switzerland, IEEE Computer Society (2002) 132–143
10. Kaboré, E., Beugnard, A.: On the benefits using model transformations to describe components design process. In: The ECOOP Twelfth International Workshop on Component-Oriented Programming (WCOP 2007). (2007)
11. Object Management Group: MDA Guide Version 1.0.1. <http://www.omg.org/docs/omg/03-06-01.pdf> (2003)
12. Buisson, J., André, F., Pazat, J.L.: A framework for dynamic adaptation of parallel components. In: ParCo 2005. (2005)
13. Bruneton, E., Coupaye, T., Stefani, J.: The Fractal Component Model. <http://fractal.objectweb.org/specification/fractal-specification.pdf> (2004)