Dynamic Composition and Adaptation in Adapt-Medium

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

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

Abstract. In the presence of operational context changes, many applications must use dynamic adaptations in order to meet requirements. When an application has a set of distributed objects that collaborates to offer a particular function, adaptations involving simultaneous distributed processes may affect such collaborations, planning distributed adaptations is thus a complex task for developers. This paper presents Adapt-Medium, an architecture of adaptive distributed components. In the architecture, adaptations are realized by performing dynamic compositions of distributed components. We introduce a model-based process for 1) specifying architecture variants of such distributed components and 2) automatically generating adaptation plans that are performed at runtime to switch between architecture variants.

1 Introduction

Increasingly, applications must support runtime adaptations and in response to changes in execution environment. Such adaptation require applications to change behaviors or even change their internal structures dynamically in maintaining continuous availability [1]. When an application has a set of distributed objects that collaborates to offer a particular function, adaptations involving simultaneous distributed processes may affect such collaborations. Supporting runtime adaptations of this class of applications can be challenging. In this paper, we introduce an architecture-based approach to dynamic adaptation of such applications.

Architecture-based adaptation is mainly concerned with structural changes at the level of software components [2]. In the context of applications having distributed functional collaborations, building runtime adaptations features requires some challenging tasks:

- *Specifying consistent architecture variants:* Through an adaptation, an application moves from a consistent architecture variant to another consistent architecture variant. Specifying such consistent architecture variants of the application is thus critical to ensure the correctness of the collaboration after adaptations.
- *Supporting runtime transitions:* Runtime transitions of architectures are also critical in order to preserve states and data through adaptations. Such transitions involve simultaneous distributed processes having dependencies between them. Planning adaptations is thus a complex task for developers. Moreover, an important class of applications requires continuous availability, adaptations must be transparent to users.

Addressing these issues, we propose Adapt-Medium, an architecture of adaptive distributed collaborations. From a collaboration abstraction called *medium*, we can build architecture variants and embed these variants into a platform that can dynamically select a proper running variant. Thanks to a design process, all the architecture variants are consistent. Moreover, because all the variants are embedded in the platform at design time, data of a replaced variant can be transferred to the new one during runtime variant transitions.

In our approach, we specify transition actions within the refinement process (of the collaboration abstraction into architecture variants). Then, by using model-based techniques, we can automatically generate the target adaptive distributed application with adaptation plans.

The remainder of this paper is organized as follows. Section 2 presents the original medium that was proposed previously in our project [3]. Section 3 introduces the design principles of Adapt-Medium that is based on the medium architecture. Section 4 illustrates how our refinement process can build consistent architecture variants and how we can generate adaptation plans by an example. Section 5 presents related work. Finally, Section 6 summarizes the paper and discusses future work.

2 Medium

A medium is a collaboration abstraction represented as a software entity. An application is built by interconnecting some functional components with a medium that represents the collaboration of the functional components [3].

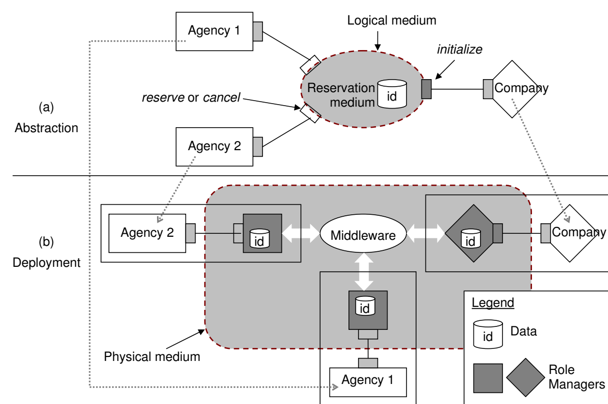

Fig. 1. Medium deployment architecture

For example, consider an airplane seats reservation application of an airline company with travel agencies located worldwide. As shown in Figure 1 (a), we can specify a *reservation medium* managing seats' identifiers (IDs) and offering *medium services* to initialize information about seats, to reserve seats and to cancel reservations. The reservation application can then be built by interconnecting the reservation medium and local functional components representing the airline company and the agencies.

Figure 1 (b) shows the deployment architecture of the reservation medium that splits into physical *role managers*. Each role manager is associated with a local functional component and implements the medium services used by this functional component. As shown in the figure, the seats' IDs set may be distributed between role managers. Depending on the data distribution architecture (e.g., distributed, centralized) or the data type chosen for data management, the medium at the deployment level may be different.

3 Adapt-Medium

3.1 Design Principle

Our approach is to generalize the refinement process in order to obtain all planned evolutionary architecture variants from a collaboration abstraction, then compose these variants

into an adaptable medium that can dynamically select a proper running variant at runtime. The architecture meta-model also allows to build and integrate new variants at runtime.

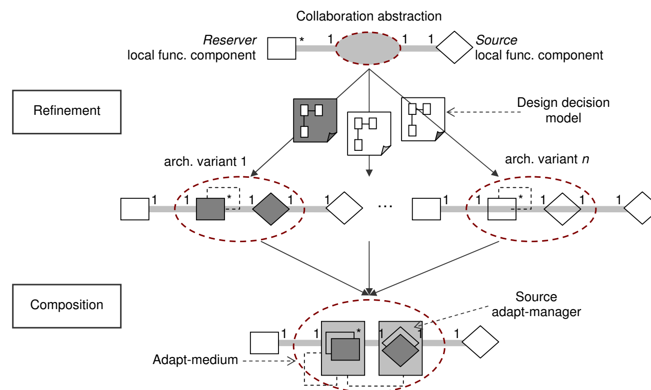

Fig. 2. Adapt-Medium design principle

Figure 2 briefly shows our development approach in building adaptable and evolvable reservation medium (called reservation adapt-medium). This adapt-medium can be used in reservation applications that can dynamically switch the data type and the data management algorithm used for managing seats' IDs when the execution context changes (e.g., the number of agencies increases, evolution of database systems).

- *Refinement.* From the collaboration abstraction, we specify design decision models. Each of these models contains a sequence of design alternatives that lead the refinement of the abstraction to an architecture variant. All the architecture variants conform to the abstraction.
- *Composition.* All the role managers corresponding to a functional component are composed into an *adapt-manager*. A generic implementation of adapt-managers is introduced in this step. Models of the architecture variants, the adapt-medium model, and design decision models are preserved in the adapt-medium.

3.2 Planning Transitions

Because the adapt-medium contains all the implementations of the medium architecture variants at runtime, the data of a replaced architecture variant can be transferred to the new architecture variant.

Transition plans can be built by analyzing the two design decision models corresponding to the current running variant and the target variant to determine which data from which managers of the current variant should be read, and then, should be write in which managers of the target variant. The result is then a plan of *Read* and *Write* actions. Each of these actions is refined into some coordinated distributed actions by top-down goal decompositions [4].

3.3 Adapt-Medium Architecture

Figure 3 shows the global view of a distributed application using an adapt-medium. This application is deployed on two sites, on each site, a functional component is associated

with an adapt-manager. The adapt-medium is then the logical aggregation of two adapt-managers, one per site. Each adapt-manager consists of a *composite manager*, an *adaptation controller* and a *medium logic* component. The composite manager contains all the *manager variants* of the corresponding role manager and an *adaptor*.

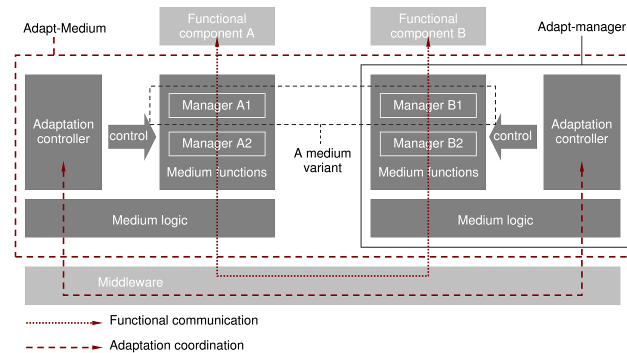

Fig. 3. Adapt-Medium architecture

The adaptor refers called medium services to the running variant by a parameter that can be changed by the executor. The adaptation controller receives context information, makes adaptation decisions, selects a proper running variant, generates adaptation plans and executes the plans. The medium logic sub-component manages medium's meta-data (information about structures of all architecture variants and design decision models) and adapt-managers' instances information. Functional collaboration between composite managers and adaptation coordination between adaptation controllers are performed through the medium logic layer. In addition, adaptation controllers use medium information managed by medium logic sub-components to schedule adaptations.

4 Example: Reservation Adapt-Medium

4.1 Building Architecture Variants

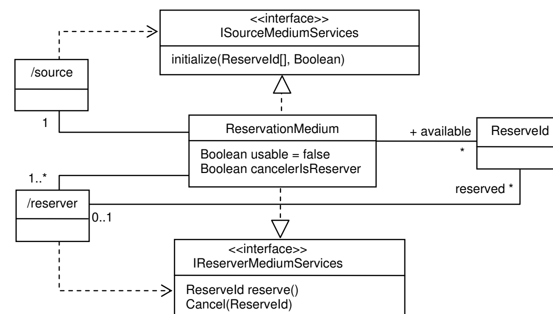

Fig. 4. Abstract specification of the reservation adapt-medium

This section illustrates the refinement process in the Adapt-Medium design principle presented in the previous section by the example of the reservation adapt-medium. The following are some class diagrams of the reservation application in some steps the refinement process.

Abstraction. Figure 4 shows the class diagram of the reservation application at the abstract level. Class `ReservationMedium` represents the logical medium that implements the `ISourceMediumServices` interface used by the `Source` class and the `IReserverMediumServices` interface used by the `Reserver` class. Class `ReserveId` represents a seat's ID. The medium manages the (`available`) set of available seats' IDs. Each instance of the `Reserver` class has a (`reserved`) set of seats' IDs reserved by the corresponding agency.

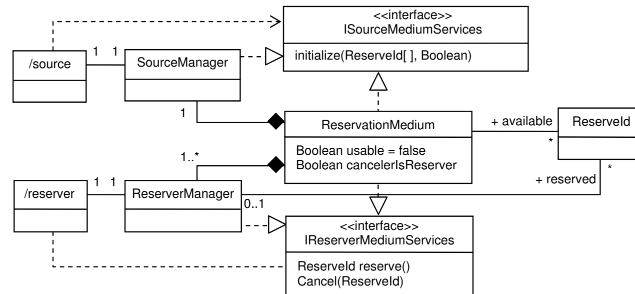

Fig. 5. Managers introduction

Introducing role managers. In order to identify and separate design alternatives, role managers are introduced, one per functional component. Figure 5 shows the class diagram of the application after the role managers introduction. The `SourceManager` and `ReserverManager` classes are introduced. These two classes implement the medium services, but the `available` data are still managed by the `ReservationMedium` class representing the logical medium. The `reserved` data of the `Reserver` class in the abstract specification is now managed by the `ReserverManager` class.

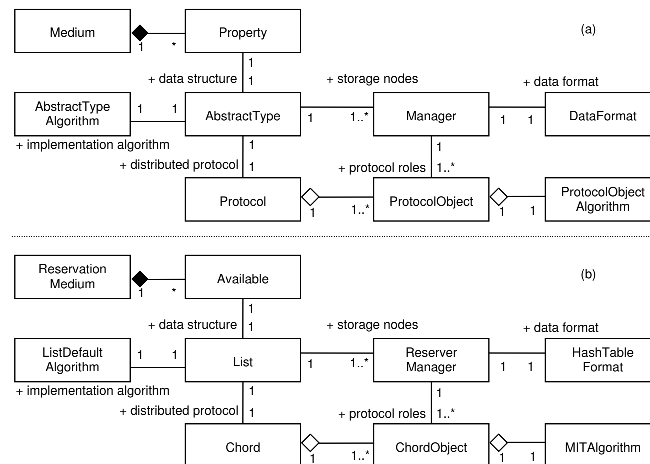

Fig. 6. (a) Generic design decision model and (b) a design decision model

Identifying and separating design alternatives. Generally, the logical medium at this level has data or services that need to be distributed on role managers. In the reservation medium, there is only the `available` seats' IDs set that needs to be distributed. In this example, we have identified seven design concerns of the seats' IDs distribution: the *abstract*

type used to represent distributed data, the *abstract type implementation*, the *data format* used to represent local data, the *data distribution topology* specifying role managers that can participate to the distribution, the *role of role managers* in the distribution (e.g., client, server, peer), the *distributed protocol* used to implement the data distributed strategy (e.g., Chord [5]), the *distributed protocol implementation algorithm* that specifies the protocol implementation (e.g., OpenChord [6] or MIT Chord implementation [7]). For each design concern, there are several design alternatives (e.g., Chord, Pastry design alternatives for the distributed protocol design concern).

From the identified design alternatives, we create design decision models. These design decision models guide a refinement that transforms the abstraction into some medium architecture variants. With each design decision model, a medium architecture variant's model is built.

Figure 6 shows the generic design decision model (a) and a design decision model for the distribution of seats' IDs (b). For example, the `available` set can be distributed on `ReserverManager` role managers by using the Chord algorithm implemented by MIT. The `available` distributed data can be accessed via proxies as `List` data. Primitives of the `List` data are implemented by `ListDefaultAlgorithm`. With this design decision model, the implementation class diagram of the corresponding architecture variant is shown in Figure 7.

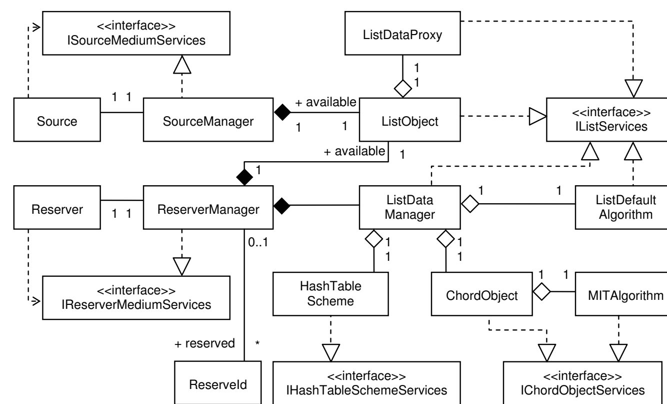

Fig. 7. An architecture variant of the reservation medium

By this refinement process with design decision models, all the medium architecture variants conform to the medium abstraction. Thereby, these medium architecture variants are consistent from the viewpoint of distributed functional collaboration and the distributed functional components of the application using the adapt-medium can correctly collaborate after switching variants.

4.2 Generating Adaptation Plans

Figure 8 shows another viewpoint of the refinement process. Design alternatives refine a collaboration abstraction (medium) into architecture variants through several internal variants. A sequence of design alternatives forms a design decision model. For example, the sequence of design alternatives corresponding to the internal variants {(1),(2),(3),(4),(5)} is the design decision model in Figure 6 (b).

In order to automatically generate adaptation plans, we aim to specify transition actions within the steps of the refinement process. Along every sequence of variants, from the abstraction to the architecture variant through internal variants, we specify 1) actions that

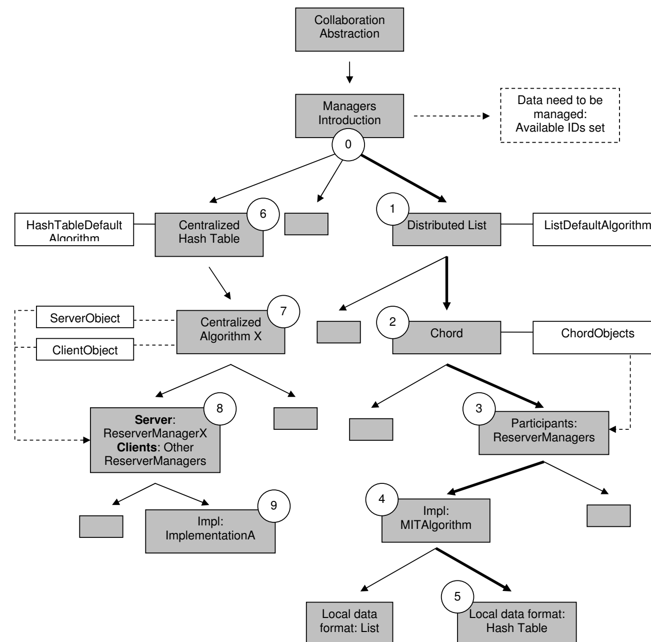

Fig. 8. Specifying architecture variants transitions

need to be executed to transfer data from a variant to the next one and 2) actions that are needed to restore data of this variant from the next variant.

For example, with the sequence {(1),(2),(3),(4),(5)}, the actions are described as follows:

```
From (0) to (1):
  for item in (0).Available
    (1).ListDefaultAlgorithm.Write(item)
Restore (0) from (1):
  (1).Available = null
  Do
    item = (1).ListDefaultAlgorithm.Read()
    if item <> null then (0).Available.add(item)
  Until item = null
From (1) to (2):
  No data to be transferred
From (1) to (3):
  (1).ListDefaultAlgorithm.Write(item) = {
    (a ReserverManager).ChordObject.Write(item)
  }
Restore (1) from (3):
  (1).ListDefaultAlgorithm.Read() = {
    (a ReserverManager).ChordObject.Read()
  }
From (3) to (4):
  No data to be transferred
From (3) to (5):
  No data to be transferred
```

Consider another design decision model corresponding an architecture variant (call variant `B`) in which the `Available` set is organized in a centralized way. This design decision

model corresponds to the sequence of internal variants: {(6),(7),(8),(9)}. With this sequence, we can specify the following transition actions:

```
From (0) to (6):
  for item in (0).Available
    (1).HashTableDefaultAlgorithm.Write(item)
Restore (0) from (6):
  (1).Available = null
  Do
    item = (1).HashTableDefaultAlgorithm.Read()
    if item <> null then (0).Available.add(item)
  Until item = null
From (6) to (7):
  No data to be transferred
From (6) to (8):
  (1).HashTableDefaultAlgorithm.Write(item) = {
     (ReserverManager X).ServerObject.Write(item)
  }
Restore (6) from (8):
  (1).HashTableDefaultAlgorithm.Read() = {
    (ReserverManager X).ServerObject.Read()
  }
From (8) to (9):
  No data to be transferred
```

From these actions, we can automatically generate adaptation plans for switching between two architecture variants corresponding two design decision models.

In our approach presented in this section, we focus only on transitions of functional data (seats' ID). In order to optimize transitions, other data can be also transferred between architecture variants. For example in the internal variant (4), routing data can be transferred to other implementations of the Chord algorithm.

5 Related Work

Many research projects have been investigating techniques to support runtime adaptation of distributed applications. But currently, to the best of our knowledge, there does not exist an approach that supports automatically planning runtime adaptations of applications having distributed functional collaboration.

In the field of robotic, some work supported automatically planning adaptation. For example, in [8], Daniel Sykes *et al* proposed a three-layer model in which adaptation plans are generated from goal models expressed in temporal logic. The plans are executed by selecting alternative components. In the context of distributed collaboration, e.g., two robots collaborate to perform a task, this work does not ensure the correctness of the collaboration between alternatives components of the robots.

A number of approaches supports adaptation mechanisms by replacing or rebinding components [9] or by customizable frameworks to developing adaptable component-based applications [10, 11]. In these approaches, the authors did not focus on the distributed functional logic of applications.

In [12], Gautier Bastide *et al* proposed an approach to create composite components from monolithic ones by restructuring the latter. A composite component consists of subcomponents that are deployed on distributed hosts in order to adapt to deployment policies (e.g., when the monolithic component cannot be deployed on a host). Compared with Adapt-Medium, the monolithic component corresponds to the abstraction and a composite

component corresponds to an architecture variant. However, because the goal of [12] is to adapt the application deployment, this approach does not support mechanism to switch composite components. Moreover, as concluded in [12], this work does not allow runtime adaptations.

A few approaches support multiple distributed adaptations. In ACEEL [13], an adaptive distributed application has some distributed coordinators that coordinate multiple distributed adaptation in order to maintain the cooperation of distributed components. The coordinators collaborate by using an *adaptation policy* provided by developers. In [14], Kurt Geihs *et al* proposed an approach to develop component-based distributed applications that includes a framework for selecting proper variants based on the current state of the execution context. In this work, the creation of the application variants is also based on some *component plans* describing the components composition defined by developers. By allowing developers define the adaptation policy [13] and the component plans [14], these approaches support a large class of applications, but the capability to maintain distributed collaboration thus depends on developers.

From the viewpoint of distributed components connection, mediums have a similarity to explicit software connectors [15] used in ArchStudio [1] to supporting runtime evolution. But they differ in many aspects: In contrary to mediums being reusable components, connectors are built by compilers that analyze interfaces specifications of distributed components that need to be connected. Moreover, mediums implement functional collaboration, but connectors implement non-functional interaction of distributed components.

6 Conclusion

In this paper, we presented Adapt-Medium, an architecture of adaptive distributed components. In the architecture, adaptations are realized by performing dynamic compositions of distributed components. We introduced a model-based process for 1) specifying architecture variants of such distributed components and 2) automatically generating adaptation plans that are performed at runtime to switch running architecture variant. The context includes applications having distributed functional collaborations. In this class of applications, adaptations involving distributed processes may affect the collaborations, planning adaptations is thus a complex task for developers.

In our approach, a distributed application is firstly specified using a collaboration abstraction called medium. Then we presented a refinement process that transforms this abstraction into many architecture variants. These architecture variants are then composed into an adapt-medium that can select a proper running variant and dynamically switch between variants in order to adapt to context changes. We proposed to specify adaptation actions within the refinement process, thus automatically generate plans for performing adaptations.

We have automated the refinement process by model transformations [16]. Our future work includes defining an action meta-model, integrating specifications of transition actions into the model-based process, thus validate our approach to automatically generating distributed adaptation plans, reducing development tasks of developers.

Our current architecture does not support continuous availability [1]. An adapt-medium enables the application using it to move from a consistent architecture to another consistent architecture at runtime without loss of data, but during the data transfer, the medium services must be stopped. Our ongoing work includes specifying local data as shared objects between manager variants by analyzing common design alternatives of the design decision models. Thus we could replace the Read and Write actions in transitions by rebinding components.

Our future work investigates an integrated tool suite that 1) enables developers to specify distributed collaborations, 2) automatically generates runtime adaptable collaborations as components, and 3) automatically generates coordination models for executing distributed adaptation processes.

References

1. Oreizy, P., Medvidovic, N., Taylor, R.N.: Architecture-based runtime software evolution. In: ICSE '98: Proceedings of the 20th international conference on Software engineering, Washington, DC, USA, IEEE Computer Society (1998) 177–186
2. Cheng, B.H.C., et al: Software Engineering for Self-Adaptive Systems: A Research Road Map. (In: Dagstuhl Seminar, http://drops.dagstuhl.de/opus/volltexte/2008/1500/)
3. 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
4. Malone, T.W., Crowston, K.: The interdisciplinary study of coordination. ACM Computing Surveys **26**(1) (1994) 87–119
5. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In: ACM SIGCOMM Conference, San Diego (2001)
6. Bamberg University, Distributed System Group: Openchord. http://www.uni-bamberg.de/projects/openchord (2007)
7. Massachusetts Institute of Technology: lsd. http://www.pdos.lcs.mit.edu/chord/ (2004)
8. Sykes, D., Heaven, W., Magee, J., Kramer, J.: From goals to components: a combined approach to self-management. In: SEAMS '08: Proceedings of the 2008 international workshop on Software engineering for adaptive and self-managing systems, ACM (2008) 1–8
9. 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 DAIS2003). Volume 2893 of Lecture Notes in Computer Science., Paris, Federated Conferences, Springer-Verlag (2003) 1–14
10. 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
11. 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
12. Bastide, G., Seriai, A., Oussalah, M.: Adaptation of Monolithic Software Components by Their Transformation into Composite Configurations Based on Refactoring. In: Proceedings of The 9th International ACM SIGSOFT Symposium on Component-Based Software Engineering. Lecture Notes in Computer Science, Springer-Verlag (2006) 368–375
13. Chefrour, 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
14. Geihs, K., Khan, M.U., Reichle, R., Solberg, A., Hallsteinsen, S., Merral, S.: Modeling of component-based adaptive distributed applications. In: Proceedings of the 2006 ACM symposium on Applied Computing (SAC'06), ACM Press (2006) 718–722
15. Shaw, M., Garlan, D.: Software architecture: perspectives on an emerging discipline. Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1996)
16. Phung-Khac, A., Beugnard, A., Gilliot, J.M., Segarra, M.T.: Model-Driven Development of Component-based Adaptive Distributed Applications. In: Proceeding of the 23rd ACM Symposium on Applied Computing (SAC'2008), track on Dependable and Adaptive Distributed Systems (DADS), Fortaleza, Ceará, Brazil, ACM Press (2008)