

# Model-Driven Development of Component-based Adaptive Distributed Applications

An Phung-Khac, Antoine Beugnard,  
Jean-Marie Gilliot, and Maria-Teresa Segarra  
Département Informatique, GET/ENST Bretagne  
Technopôle Brest-Iroise, 29238 Brest Cedex 3, France  
{an.phungkhac,antoine.beugnard,jm.gilliot,mt.segarra}  
@enst-bretagne.fr

## ABSTRACT

This paper introduces an approach to develop component-based adaptive distributed applications. Our approach separates the communication and the functional aspects of a distributed application and specifies the communication part as an abstract distributed component called the *communication component*. We then introduce a model-based process for automatically building many evolutionary variants of this component at deployment level, and integrating these variants into the target adaptive application that can dynamically select the running variant in order to adapt to the changing context. Thanks to an *adaptation guide* generated by the process, the adaptive application can coordinate distributed adaptations to (1) consistently transfer data of the replaced variant to the new one and (2) maintain the architectural coherence between distributed parts of the application. Hence, the target adaptive application can *correctly adapt at runtime without loss of data*. In this paper, we present the principle of our approach, illustrate it with an example, and show how we have automated the development process by model transformations.

## Categories and Subject Descriptors

C.2.4 [Distributed Systems]: Distributed Applications; D.2.2 and D.2.12 [Software Engineering]: Design Tools and Techniques - *Object-oriented design methods*, Interoperability - *Distributed objects*; I.6.5 [Simulation and Modeling]: Model Development—*Modeling methodologies*

## General Terms

Design, Reliability.

## Keywords

Dynamic adaptation, Coordination, Distributed component, Evolution, Model-Driven Development

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SAC'08 March 16-20, 2008, Fortaleza, Ceará, Brazil

Copyright 2008 ACM 978-1-59593-753-7/08/0003 ...\$5.00.

## 1. INTRODUCTION

Computer software must dynamically adapt its behavior in response to changes in variable contexts. In distributed systems, adaptations are simultaneously performed at many sites. Therefore, coordinating adaptations across sites is critical to ensure the correctness of applications during and after adaptations. Developing adaptive distributed applications thus can be a challenge.

Addressing this issue, we propose an approach to develop component-based adaptive distributed applications. Our approach first separates the communication and the functional aspects of a distributed application, and specifies the communication part as an abstract distributed component called a communication component or *medium*. Being a component, the medium is specified in order to be reused in similar contexts. The distributed application is thus built by interconnecting several functional components with the medium that manages their communication. Then, we build the adaptive medium by a model-based process that can automatically transform the medium at the abstract specification level into many medium variants at deployment level, and automatically integrate these variants into a composite medium. Thanks to distributed adaptation coordinators using an *adaptation guide* generated by the process, the composite medium can dynamically change the running medium variant in order to adapt to the changing context. The target adaptive application is then built by interconnecting the functional components with the adaptive composite medium. Because the adaptive application contains all the generated medium variants as well as the adaptation guide describing variant transformations, distributed adaptations can be (1) coordinately performed at runtime without loss of data and (2) ensure the structural/architectural coherence between distributed parts of the medium.

We have automated our development process by a model transformation program in the KerMeta (meta) modeling framework [8]. This program can automatically transform the medium model at the abstract specification level into the adaptive composite medium model at deployment level. In our program, the medium variants in all the steps of the process are specified in UML and represented as EMF models [13]. Thereby, we can use code generation tools to generate the source code of the target adaptive application.

The remainder of this paper is organized as follows: After some related work discussed in Section 2, Section 3 introduces the principle of our approach. An example is pre-

sented in Section 4 to illustrate the approach and to show how we automate the development process by model transformations. Section 5 concludes the paper.

## 2. RELATED WORK

In the context of component-based adaptive distributed applications, some research [2, 6, 7] has proposed solutions for specifying the adaptation aspect of components, and/or providing mechanisms/frameworks for reconfiguring compositions of components. A few approaches have focused on adaptations of one distributed component that is deployed over many sites [1, 6, 10]. However, they do not support distributed components having architectural and functional constraints between their distributed parts, in order to ensure the distributed data consistency and the functional/architectural coherence of these distributed parts during and after each adaptation.

## 3. OUR DEVELOPMENT PROCESS

In this section, we introduce our model-based process for developing component-based adaptive distributed applications. As shown in Figure 1, our development process comprises five steps:

**Step 1 - Specifying the distributed application.** In this step, the communication aspect of the distributed application is separated from the functional aspects and specified as an abstract *communication component* (or *medium* in order to differentiate it from functional components). The application is thus built by interconnecting some functional components and the medium that carries out all the communications between these functional components. Being a component, the medium is specified in order to be reused in similar contexts. The medium architecture was proposed in previous work by our team [5]. The result of this step is a medium model at the abstract specification level<sup>1</sup>.

**Step 2 - Building medium implementation variants.** This step contains a refinement process for transforming the abstract specification of the medium into many implementation specification variants (also called medium variants) through several sub-steps. First, some *role managers* are introduced, one per functional component. Each role manager carries out the communication between the corresponding functional component and the abstract part<sup>2</sup> of the medium. The medium is then separated step by step into the role managers by introducing design variants (e.g., data distribution strategy variants and data type variants). For each design variant, a new medium variant is built. The result of this step is thus different (evolutionary) medium implementation variants, each of these variants being composed of some role managers interacting with each other. All the medium variants of the result are at the implementation specification level.

**Step 3 - Building medium deployment variants.** At runtime, the application may contain several instances of a role. This means that there may be several deployment variants corresponding to an implementation variant of the medium. The objective of this step is to build all the deployment variants of the medium from the implementation

<sup>1</sup>We distinguish the specification level from the instance level

<sup>2</sup>This part does not exist at the implementation specification level

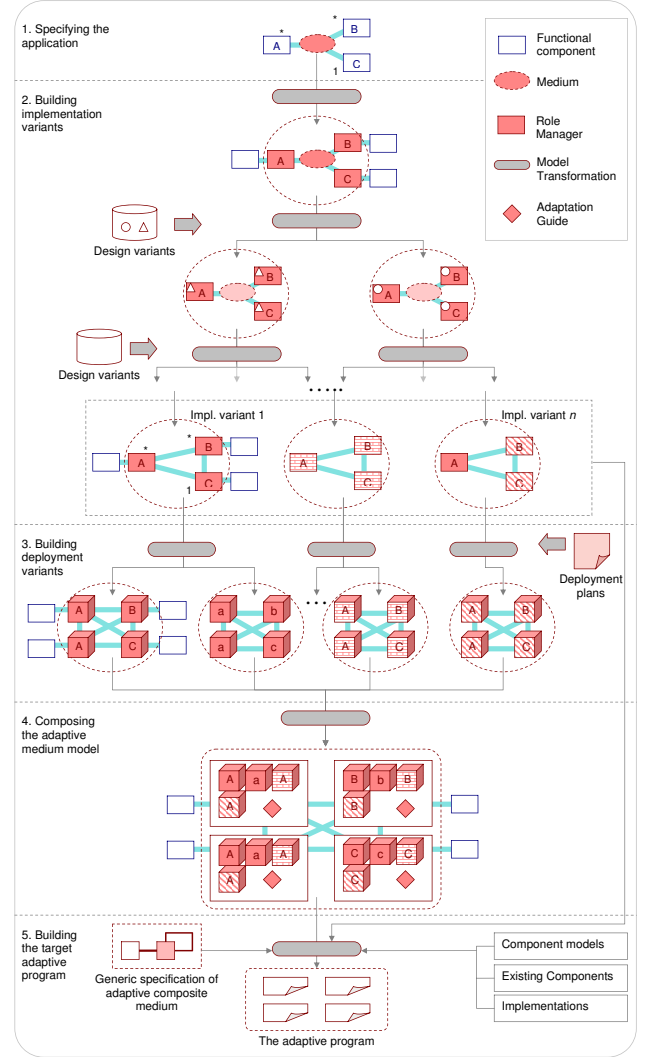


Figure 1: The five steps in the development process

variants by introducing different *deployment plans*. The result of the step is then medium deployment variants at the instance level.

**Step 4 - Composing the adaptive medium model.** To build the adaptive medium model [12], all medium deployment variants are integrated into one composite medium by embedding all their role managers corresponding to a functional component into a same *composite role manager*. In every composite manager, an *adapter*, an *adaptation coordinator* and an *adaptation manager* are implemented. Thanks to these adapters and coordinators, the composite medium can dynamically and coordinately change the running medium variant in order to adapt to context changes. The target adaptive application is then built by interconnecting the functional components with the adaptive composite medium.

At runtime, distributed adaptations are performed and coordinated by using an *adaptation guide* generated from the process. This adaptation guide is a model including the refinement process model in Step 2, design variant models in Step 2 and deployment plan models in Step 3. As shown in Step 4 of Figure 1, the result of this step is a model of the

adaptive composite medium that contains the adaptation guide model. This adaptive composite medium model is managed by adaptation managers. Thus, the model can co-exist and co-evolve with the adaptive program in order to provide adaptation and coordination plans for adapters and coordinators.

**Step 5 - Building the target adaptive program.** In this step, we propose a generic implementation specification of adaptive programs in UML. From this generic specification and the composite model of the previous step, we can use code generation tools to generate the program membrane, and implement adapters, coordinators and adaptation managers. Component models, e.g. CCM [11] or Fractal [3], and existing components can be introduced. Some other code might be implemented by developers. In this step, a module observing the context changes and mapping context-variant is integrated. This module executes adaptations by using the adaptation services of the composite medium. In the module, we use an adaptation canvas [4] for integrating *context observers* and define a constraint language for mapping context-variant in *adaptation deciders*.

We have presented the five steps of our process. This process can be automated thanks to model transformations. We have (meta)modeled the process, medium variants, design variants, deployment plans and have used some model transformations to transform medium models and to weave design variant models with medium models. More details about this automation will be given in Section 4.

## 4. EXAMPLE

This section illustrates our approach with an example. Results will be presented corresponding to each step of the development process.

### 4.1 Adaptive reservation medium

Let us imagine a simple flight booking system. An airline company offers a set of places on their flights, each place having an identifier. These places are sold by two (or more) travel agencies. A customer can book or cancel a place via any agency. When all places attributed to an agency are sold, this agency can ask the others to share their available places. On a web site of the airline company, the total number of available places is displayed

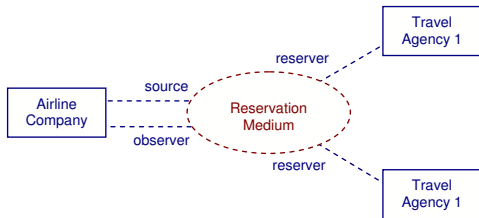


Figure 2: Reservation medium

Figure 2 shows our specification for this system. We have introduced an abstract component called *reservation medium*. The communication between the airline company and the travel agencies is then carried out by this medium. In the medium, we have specified several communication roles: the airline company plays a *source* role, its web site

plays an *observer* role and every agency plays a *reserver* role.

This reservation medium can be reused in other applications, e.g., a management application for a parking lot that has a set of places (source), cars that can come in or go out via entries (reservers), and boards displaying available places (observers). Generically, this reservation medium can be reused in every identifier's reservation system that contains one source, several reservers and several observers.

To manage the identifiers shared in the medium, data distribution algorithms can be used. These algorithms are different from each other in data storage modes (e.g., distributed, centralized, replicating, etc.), data location types (e.g., distributed hash table). Even if an algorithm is used, there may be many choices of storage points. When the context changes (e.g., network conditions, the number of travel agencies, etc.), the algorithm used should be dynamically changed in order to optimize application performance.

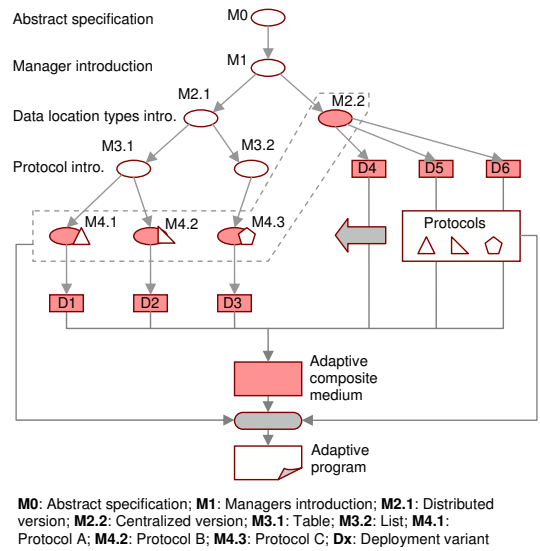


Figure 3: The development process of the adaptive reservation medium

In this section we present our model-based process for developing the adaptive reservation medium that is initially deployed over four sites: a source, an observer and two reservers. As shown in Figure 3, there are four medium implementation variants (M2.2, M4.1, M4.2, and M4.3). Medium variant M2.2 corresponds to a centralized design variant in which distributed tables and Protocol A are used for organizing identifiers. Variant M2.2 has three deployment variants (D4, D5, D6) corresponding to three storage points: two reservers and one source. From the three other medium implementation variants (M4.1, M4.2, M4.3), there are three deployment variants (D1, D2, D3). The composite medium thus contains six medium deployment variants.

We have automated this development process by a model transformation program by using the Kermeta (meta) modeling framework [8]. The aspect of process (meta) modeling [9] is not presented in this paper. In the following sub sections, we show some medium models during the steps of the development process. Models at the specification level will be represented in UML class diagrams and those at the

instance level will be represented in XML.

## 4.2 Specifying the distributed application

Figure 4 is the UML class diagram of the medium at the abstract specification level (variant M0) in which the **ReservationMedium** class represents the medium. The component playing the source role can set the **originalSet** variable (the original identifiers set) by calling the **setReserveIdSet** service of the **ISourceMediumServices** interface implemented by the medium. Similarly, the components playing the observer roles and the reserver roles can interact with the medium via the **IReserverMediumServices** interface and the **IObserverComponentServices** interface.

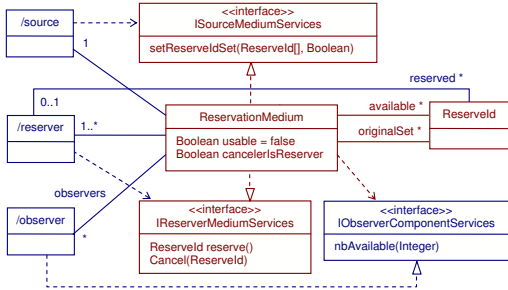


Figure 4: Variant M0: Abstract specification

## 4.3 Building medium implementation variants

First, corresponding to each role, a class called **<RoleName>Manager** is introduced into the medium. This manager is a proxy for the medium in the communication with the corresponding role. The **ReservationMedium** medium class now contains only some data variables of the medium such as **available** (list of available identifiers) or **originalSet**. These data variables are used by the managers for implementing medium services.

In the next step of the refinement process, a model of the design variants represented in XMI (XML Metadata Interchange) is introduced. Each design variant in the model indicates a data variable to be distributed (**available** in this example), some managers where the variable is distributed on, a strategy (e.g., distributed, centralized, etc.), a data location type and a distributed protocol.

```
<?xml version="1.0" encoding="ASCII"?>
<DesignModel xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="platform:/resource/adaptive_reservation_medium/design_model.ecore"
name="ReservationDesignModel">
  <element xsi:type="ManagerList" name="ManagerList">
    <manager name="SourceManager"/>
    <manager name="ObserverManager"/>
    <manager name="ReserverManager"/>
  </element>
  ...
  <element xsi:type="ProtocolLibrary" name="ProtocolLibrary">
    <protocol name="ProtocolA"/>
    <protocol name="ProtocolB"/>
    <protocol name="ProtocolC"/>
  </element>
  <element xsi:type="DesignVariant" ... />
  <element xsi:type="DesignVariant" name="M41" data="//@element.3/@data.0"
strategy="//@element.9/@strategy.0" protocol="//@element.4/@protocol.0"
dataLocationInfoType="//@element.1/@dataLocationInfoType.0"
manager="//@element.0/@manager.2" dataType="//@element.2/@dataType.0"/>
  ...
</DesignModel>
```

Figure 5 is the UML class diagram of variant M2.2. In this variant, all identifiers are managed by the **MediumManager** class that can be deployed on the same site as the source manager or a reserver manager.

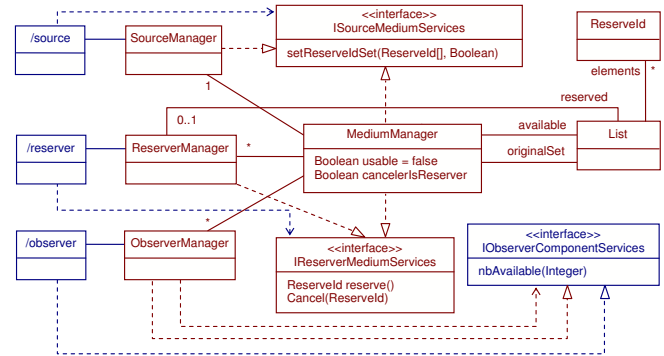


Figure 5: Variant M2.2: centralized architecture

Variant M4.1 is represented in Figure 6. In this diagram, the **available** data variable is now distributed on some variables called **localAvailable** of reserver managers and each variable is managed by a **ProtocolADataManager** class that is the proxy for **available** from the viewpoint of the corresponding reserver manager. Classes **Vector**, **DistributedTable** and **ProtocolADataManager** are introduced in three sub-steps of the refinement process in Figure 3.

In variant M2.2 and variant M4.1, the **ReservationMedium** class has disappeared. The mediums are then entirely at the implementation level.

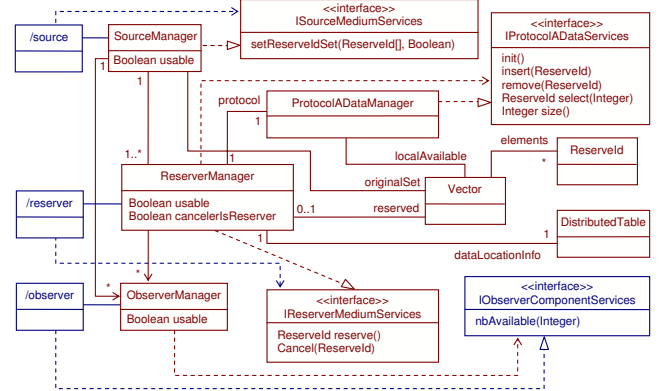


Figure 6: Variant M4.1: distributed strategy, distributed table, Protocol A

## 4.4 Building medium deployment variants

Similarly to the model of design variants, we use XMI to represent the model of deployment plans in this step. Each deployment plan in this model describes instances of each role and storage points. From each deployment strategy, a medium deployment variant model is built. In this model, each role instance is “deployed” on a **RoleDeploymentContainer**, and likewise, each role manager and is “deployed” on a **MediumDeploymentContainer**. For example, the following is the membrane of variant D1:

```

<?xml version="1.0" encoding="ASCII"?>
<Application xmlns:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="platform/resource/adaptive_reservation_medium/impl_variants.ecore" name="FlyBooking"
version="D1">
  <element xsi:type="RoleDeploymentContainer" name="AirlineCompany" ... </element>
  <element xsi:type="RoleDeploymentContainer" name="ObserverWebSite" ... </element>
  <element xsi:type="RoleDeploymentContainer" name="TravelAgency1" ... </element>
  <element xsi:type="RoleDeploymentContainer" name="TravelAgency2" ... </element>
  <element xsi:type="Medium" name="ReservationMedium" version="D1">
    <element xsi:type="MediumDeploymentContainer" name="Source" version="D1"
roleName="Source" instanceNumber="1"> ... </element>
    <element xsi:type="MediumDeploymentContainer" name="Observer" version="D1"
roleName="Observer" instanceNumber="1"> ... </element>
    <element xsi:type="MediumDeploymentContainer" name="Reserver1" version="D1"
roleName="Reserver" instanceNumber="1"> ... </element>
    <element xsi:type="MediumDeploymentContainer" name="Reserver2" version="D1"
roleName="Reserver" instanceNumber="2"> ... </element>
  </element>
</Application>

```

### 4.5 Composing the adaptive medium model

The model of the adaptive composite medium is composed of two parts: the medium variant composition and the adaptation guide. The former contains four composite managers. Each of these composite managers includes all role deployment containers corresponding to a role instance. The latter describes the refinement process in Step 2 as a “transformation tree”. Each node of this tree comprises a medium variant, a chosen design variant and all transformation actions by which this medium variant has been built. The tree root corresponds to the medium at the abstract specification level and tree leaves represent medium variants at the implementation specification level. Every leaf contains some deployment plans in Step 4.

At runtime, when a role instance comes or leaves the system, the adaptive program will modify the deployment plans, then rebuild the medium variant composition. Thereby, this model of the adaptive composite medium co-exists and co-evolves with the program.

### 4.6 Building the target adaptive program

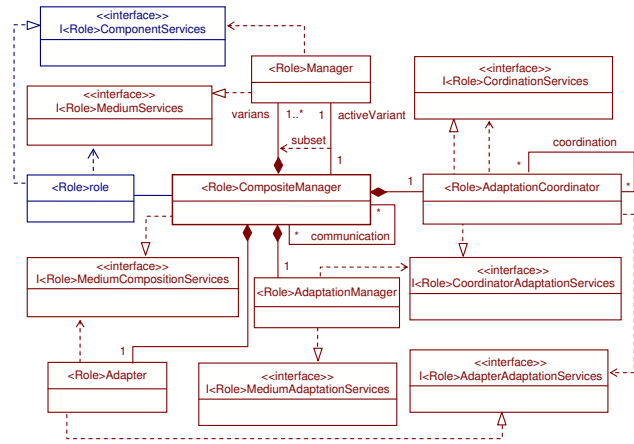


Figure 7: Generic class diagram of adaptive mediums

Figure 7 is the generic class diagram of adaptive applications using mediums. As shown in the figure, an adaptive application includes many composite managers interconnecting with each other. Each composite manager contains some manager variants, an adapter, an adaptation co-

ordinator and an adaptation manager managing the adaptation guide. When the program needs to change the running variant, an adaptation decider<sup>3</sup> calls a medium adaptation service of the adaptation manager. This adaptation manager will analyze the adaptation guide to generate a coordination plan and some local adaptation plans. These plans are then sent by the adaptation coordinator to other composite managers. On the other hand, the coordinator sends the local adaptation plan to the adapter in which local adaptation actions will be performed by reconfiguring the composite manager. In this reconfiguration, a reference of a composite manager proxy is changed to the new manager variant, and data objects of the replaced variant are transferred to the new one as shared objects. For example, to change the running variant from D1 to D2, the `DistributedTable` object can be kept and reused in the new variant as a shared object; or from D1 to D3, `DistributedTable` is reused and then its data will be transferred to a `DistributedList` object. If one (or more) role instance comes or leaves the system, the adaptation manager will rebuild the adaptation guide and send it to other composite managers. Figure 8 describes such an adaptation session.

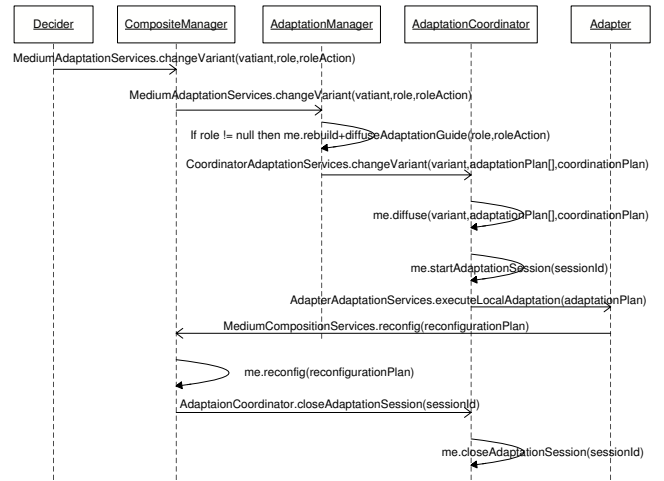


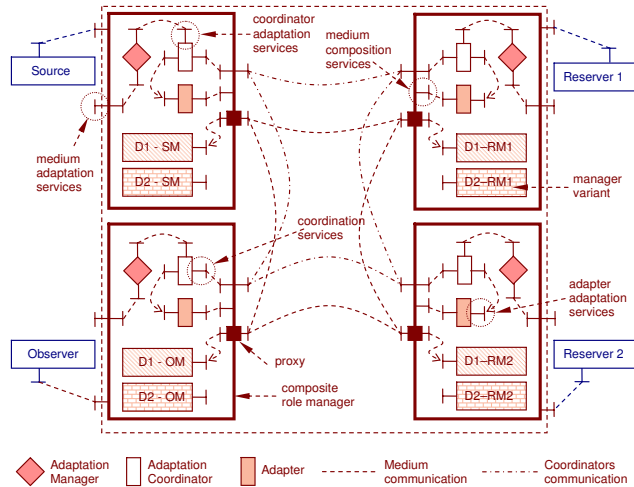
Figure 8: Description of an adaptation session

Using this generic diagram and the model of the adaptive composite medium from the previous step, an UML class diagram of the adaptive composite medium is automatically generated. The adapters, adaptation coordinators and adaptation managers are implemented in this step. Implementations of protocols, data types and data location types are also integrated thanks to pre-defined data interfaces.

Figure 9 represents a component model of the adaptive composite reservation medium that we have implemented in Java by using the Fractal component model [3] that enables us to support adaptations by connections/disconnections of sub-components as well as to reuse shared sub-components. In this model, all the *manager variants* and data used by the *adaptation managers* were generated from the development process. We have also tested the application with some

<sup>3</sup>Because this paper focuses on the structure of the adaptive distributed component, designs and implementations of *context observers* and *adaptation deciders* are not presented.

adaptation scenarios. The application was executed as expected.



**Figure 9: A component model of the adaptive reservation medium**

## 5. CONCLUSION AND FUTURE WORK

In this paper, we have presented a model-based process for developing component-based adaptive distributed applications. We have separated the communication and the functional aspects of the distributed application, specified the communication aspect as an abstract communication component (or medium), built many medium deployment variants and integrated these variants into a composite medium. In addition, we have described how to build an adaptation guide model representing the composite medium and its evolution traces. At runtime, this model co-exists and co-evolves with the target adaptive program and navigates distributed coordinators to dynamically select the running variant in order to adapt to the changing context. Moreover, we have automated our development process by model transformations. We have successfully applied our approach in order to build an adaptive distributed application that can dynamically change its data distribution algorithms.

Compared to other approaches for adaptive distributed applications [2, 6, 7, 1, 10], our approach has certain advantages. First, because application variants are embedded in the adaptive application at design time and managed by coordinators, the application can perform and coordinate distributed adaptations at runtime without loss of data. Second, through the refinement process, the architectural coherence of the distributed parts of the application is maintained from the abstract level to deployment level, thus our approach can ensure that the medium variants always satisfy the architectural requirements of the application. In other words, the approach ensures the *correctness* of adaptations. Third, because the adaptation guide, co-existing and co-evolving with the application, contains a lot of information such as design variants or deployment plans, the coordinators can optimize adaptation actions by specifying reusable objects. Hence, our approach gives the application an open, seamless and optimized adaptive feature.

Finally, being components, adaptive communication components built using our approach can be reused in many applications in similar contexts.

Our future work includes automatically integrating component models into the target program, focussing on the Fractal component model [3], in order to build an integrated environment that coordinates different tools to support our development process.

## 6. REFERENCES

- [1] D. Ayed and Y. Berbers. Dynamic adaptation of CORBA component-based applications. In *Proceedings of the 2007 ACM symposium on Applied Computing (SAC'07)*, pages 580–585. ACM Press, 2007.
- [2] I. Ben-Shaul, O. Holder, and B. Lavva. Dynamic adaptation and deployment of distributed components in hadas. *IEEE Trans. Softw. Eng.*, 27(9):769–787, 2001.
- [3] E. Bruneton, T. Coupaye, and J. Stefani. The fractal component model. <http://fractal.objectweb.org>, 2004.
- [4] J. Buisson, F. André, and J.-L. Pazat. A framework for dynamic adaptation of parallel components. In *ParCo 2005*, 2005.
- [5] E. Cariou, A. Beugnard, and J.-M. Jézéquel. An architecture and a process for implementing distributed collaborations. In *Proceedings of the 6th IEEE International Enterprise Distributed Object (EDOC 2002)*, pages 132–143, Lausanne, Switzerland, September 2002. IEEE Computer Society.
- [6] D. Chefrour. Developing component-based adaptive applications in mobile environments. In *SAC '05: Proceedings of the 2005 ACM symposium on Applied computing*, pages 1146–1150, New York, NY, USA, 2005. ACM Press.
- [7] K. Geihs, M. U. Khan, R. Reichle, A. Solberg, S. Hallsteinsen, and S. Merral. Modeling of component-based adaptive distributed applications. In *Proceedings of the 2006 ACM symposium on Applied Computing (SAC'06)*, pages 718–722. ACM Press, 2006.
- [8] IRISA Triskell Team. Kermeta. <http://www.kermeta.org/>.
- [9] E. Kaboré and A. Beugnard. On the benefits using model transformations to describe components design process. In *The ECOOP Twelfth International Workshop on Component-Oriented Programming (WCOP 2007)*, 2007.
- [10] F. Kon, J. R. Marques, T. Yamane, R. H. Campbell, and M. D. Mickunas. Design, implementation, and performance of an automatic configuration service for distributed component systems: Research articles. *Softw. Pract. Exper.*, 35(7):667–703, 2005.
- [11] Object Management Group. Corba component model specification. version 4.0. <http://www.omg.org>, 2006.
- [12] A. Phung-Khac, A. Beugnard, J.-M. Gilliot, and M.-T. Segarra. A model of self-adaptive distributed components. In *Proceedings of the ECOOP 4th Workshop on Coordination and Adaptation Techniques for Software Entities (WCAT07)*, July 2007. I.S.B.N. 13: 978-84-690-6993-6.
- [13] The Eclipse Foundation. Eclipse modeling framework (EMF). <http://www.eclipse.org/modeling/emf/>.