

A Model-driven Architecture-based Approach to Runtime Adaptable and Evolvable Distributed Collaborations

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

Department of Computer Science, TELECOM Bretagne, France
`an.phungkhac@telecom-bretagne.eu`

Abstract. Increasingly, applications must support runtime adaptation and evolution in response to changes in execution environment and user requirements. When a set of distributed objects of an application collaborates to offer a particular function, runtime adaptation and evolution involving simultaneous distributed processes may affect such collaboration, thus make the application development challenging. This thesis expects to contribute a model-driven, architecture-based approach to developing runtime adaptable and evolvable distributed collaborations, basing on a collaboration abstraction called *medium*.

1 Motivation

Increasingly, applications must support runtime adaptation and evolution in response to changes in execution environment and user requirements. Such adaptation and evolution 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, adaptation and evolution involving simultaneous distributed processes may affect such collaborations. Supporting runtime adaptation and evolution of this class of applications can be challenging.

Many types of adaptation techniques have been developed: architecture-based adaptation, parametric-based adaptation, aspect-oriented-based adaptation, etc. [2]. With these techniques, building runtime adaptation and evolution features in applications having distributed functional collaborations requires some challenging tasks:

- *Specifying consistent variants:* Through adaptation and evolution, an application moves from a consistent variant to another consistent variant. Such application’s variant composes of distributed local variants corresponding to application’s local parts. Specifying consistent variants of the application is thus critical to ensure the correctness of the collaboration after adaptation and evolution.
- *Supporting runtime variant transitions:* Runtime transition of variants is also critical in order to preserve states and data through adaptation and

evolution. Moreover, an important class of applications requires continuous availability, adaptation and evolution must be transparent to users.

- *Coordinating variant transitions*: Adaptation and evolution evolve simultaneous distributed processes. In order to safely transfer applications’ data, these processes must be performed coordinately.

Addressing these issues, this thesis contributes a model-driven, architecture-based approach to generate runtime adaptable and evolvable applications having distributed functional collaborations.

2 Proposed Approach

In our approach, we adopt a collaboration abstraction called *medium* that was proposed previously in our project [3].

Medium - A Collaboration Abstraction. *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].*

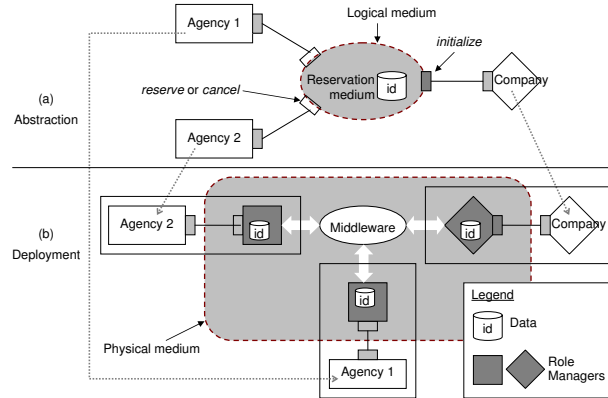


Fig. 1. Medium deployment architecture

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.

Developing Adaptable Medium. 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.

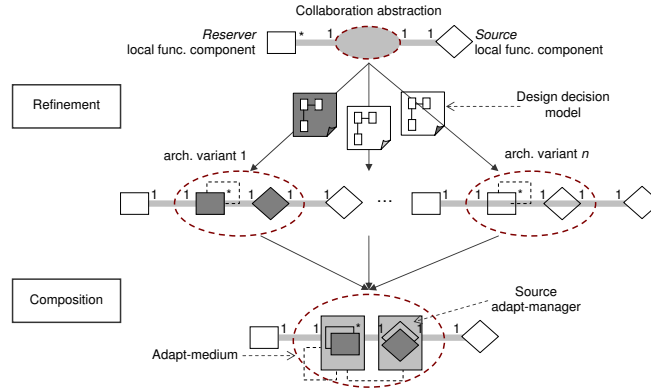


Fig. 2. Adapt-Medium design principle

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.

Because the adapt-medium contains all 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 Preliminary Results

A Model-based Design Process. We have automated our development process by using model-based techniques [5]. A models transformation program in Kermeta [6] merges a model representing the UML class diagram of the abstraction with design alternatives' models by analyzing design decisions' models, then generates architecture variants' models and composes an adapt-medium's model. All these models are represented in XMI (XML Metadata Interchange).

A Model of Adaptable Mediums. We have also implemented a component model of adapt-mediums [7] in Java using the Fractal component model [8]. The model of adapt-mediums allows to integrate managers' implementations and enables runtime switching of running variants.

From the adapt-medium's model, another transformation program in Java integrates role managers' implementations (implemented by developers), design alternatives' implementation (reused from libraries or/and implemented by developer), and adaptation controllers (implemented beforehand) into the adapt-medium implementation. The result is deployable adapt-managers Fractal components.

4 Ongoing and Future Work

Modeling Coordination. Adaptation plans can be generated by top-down goal decomposition [4]. However our implemented prototype in [7] is quite simple from the viewpoint of distributed adaptations coordination. When the number of role types increases and there are many data/services that need to be distributed, there exist dependencies of distributed adaptation actions, adaptation coordination thus needs to be modeled carefully.

We are defining a coordination meta-model that includes typing adaptation actions and defining dependencies of these actions. We intend to support specifying transition actions with the design decision models. Every design decision model is specified with the (abstract) actions that need to be performed to transfer data from the collaboration abstraction to the corresponding architecture variant, and (abstract) actions for restoring the abstraction. Thereby, we can automatically generate coordination models for performing runtime transitions.

Supporting Continuous Availability. Our implemented 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. These shared objects can be implemented as shared Fractal components [8], thus we could replace the *Read* and *Write* actions in transitions by rebinding components.

Supporting Runtime Evolutions. At runtime, an adapt-medium may need to integrate new architecture variants. Because models of existing architecture variants and design decision models are preserved in the adapt-medium, we could use runtime model transformations in order to build new architectural data of the logical adapt-medium and then integrate the new architecture variants. But ensuring the coherence between runtime models and real systems, reorganizing cross-cutting design decision models that contain new design alternatives are complex tasks. Our future work will focus on this challenge.

Moreover, when user requirements of functionality change, mediums should modify medium services in order to support runtime evolution, but our proposed model-driven process in [5] does not allow such modifications. We intend to deal with this challenge by specifying an abstraction of mediums (abstraction of collaboration abstractions) that enables to define modified functionalities, thus allow mediums to evolve dynamically.

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 runtime adaptations of applications having distributed functional collaboration.

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

A few approaches support simultaneous distributed adaptations. In ACEEL [11], 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 [12], 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 [11] and the component plans [12], these approaches support a large class of applications, but the capability to maintain distributed collaboration thus depends on developers.

In [13], Nelly Bencomo *et al* proposed an approach to modeling structural variants of component-based applications. The such variants correspond to configurations that are executed by a reflective component framework. Because this work does not particularly focus on distributed collaborations, the presented configurations are not “distributed”.

From the viewpoint of distributed components connection, mediums have a similarity to explicit software connectors [14] 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. More-

over, mediums implement functional collaboration, but connectors implement non-functional interaction of distributed components.

6 Summary

The expected contribution of this research is a model-driven, architecture-based, component-based approach to supporting runtime adaptation and evolution of applications having distributed functional collaborations. We expect to build an integrated tool suite that 1) enables developers to specify such collaborations, 2) automatically generates runtime adaptable and evolvable collaborations as components, and 3) automatically generates coordination models for executing distributed adaptation and evolution 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'02). (2002)
4. Malone, T.W., Crowston, K.: The Interdisciplinary Study of Coordination. *ACM Computing Surveys* **26**(1) (1994) 87–119
5. 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'08))
6. IRISA Triskell Team: Kermeta. (<http://www.kermeta.org/>)
7. Phung-Khac, A., Beugnard, A., Gilliot, J.M., Segarra, M.T.: A model of self-adaptive distributed components. In: Proceedings of the ECOOP 4th Workshop on Coordination and Adaptation Techniques for Software Entities (WCAT07). (2007)
8. Bruneton, E., Coupaye, T., Stefani, J.: The Fractal Component Model. <http://fractal.objectweb.org> (2004)
9. David, P.C., Ledoux, T.: Towards a Framework for Self-Adaptive Component-Based Applications. In: Proceedings of Distributed Applications and Interoperable Systems 2003 DAIS2003). (LNCS)
10. Ben-Shaul, I., Holder, O., Lavva, B.: Dynamic adaptation and deployment of distributed components in hadas. *IEEE Trans. Softw. Eng.* **27**(9) (2001)
11. Chefrou, D.: Developing Component-based Adaptive Applications in Mobile Environments. In: SAC'05: Proceedings of the 2005 ACM symposium on Applied computing. (2005)
12. 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). (2006)
13. Bencomo, N., Blair, G., Flores, C., Sawyer, P.: Reflective Component-based Technologies to Support Dynamic Variability. In: 2nd Workshop on Variability Modelling of Software-intensive Systems (VaMoS08), Germany (2008)
14. Shaw, M., Garlan, D.: Software Architecture: Perspectives on an Emerging Discipline. Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1996)