Modelling Changes and Data Transfers for Architecture-Based Runtime Evolution of Distributed Applications

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

Department of Computer Science, Télécom Bretagne
Technopôle Brest-Iroise - CS 83818 - 29238 Brest Cedex 3 - France
{an.phungkhac,jm.gilliot,mt.segarra,antoine.beugnard,eveline.kabore}
@telecom-bretagne.eu

Abstract. Architecture-based approaches for runtime evolution enable software systems to dynamically move between consistent architectural variants. Successful runtime evolution must enable the new, replacement variant to be initialized with the data of the replaced one. In distributed systems, however, the initialization is complex and may be time-consuming due to data transfers across sites. Identifying systems' components subject to change is then critical for planning evolution and reducing replacement actions, avoid unnecessary data transfers, and then, reduce downtime of system services. Addressing this issue, this paper presents an approach to runtime evolution of distributed applications. We present how a development process allows to 1) specify architectural variants of an application and 2) identify components subject to change and operations for transferring data managed by these components. Moreover, the design information is used at runtime to automatically plan evolution.

1 Introduction

An important class of software systems needs to evolve at runtime in order to adapt to changing executing environments. Moreover, during evolution, they are expected to be continuously available which require the software system to modify its own architecture at runtime [1]. Such runtime modifications include 1) replacement, addition, and removal actions to achieve the target variant, and 2) initialization of the replaced variant with data of the replacement one.

As the above tasks may disrupt collaboration among components, coordination is needed when performing modification actions. Such coordination is even more difficult when considering distributed software. Furthermore, initializing the replacement variant with data of the replaced one may be time-consuming due to data transfers across sites. Such data transfers make continuous availability difficult or even impossible to achieve. Therefore, planning evolution, including identifying components subject to change and operations for transferring data managed by these components, becomes a critical task in order to

M. Ali Babar and I. Gorton (Eds.): ECSA 2010, LNCS 6285, pp. 392–400, 2010.

avoid unnecessary replacement of components and data transfers, thus reducing downtime of system services.

In our previous work [2], we have proposed an architecture-based approach, called *adapt-medium* approach, for runtime adaptation and evolution of distributed applications. The approach is based on a model-based development process that allows to generate consistent architectural variants of a distributed application, and then, embed the variants into an adaptive distributed component. However, although an adapt-medium component is able to evolve at runtime without recompilation, the whole running variant must be replaced when performing evolution.

This paper extends our previous work by allowing identification, through the model-based development process, of the variants' components subject to change and operations for manipulating their data. Therefore, when performing evolution, instead of replacing the whole running variant, only the necessary components are replaced. Moreover, we describe how the system can exploit design information of the model-based process in order to automatically plan evolution.

The remainder of the paper is organized as follows. Section 2 briefly presents the adapt-medium approach that was presented in [2]. Section 3 describes the basics when applying this approach to develop an adaptive publish/subscribe system. Section 4 presents the main contribution of this paper, i.e., how our approach allows identifying components subject to change and operations for manipulating their data. Section 5 discusses related work and Section 6 concludes the paper.

2 Adapt-Medium Approach Overview

To cope with distribution complexity and manage evolution we adopt the *adapt-medium approach* when developing a distributed software system. This approach is mainly defined by (see Figure 1) :

- **A high-level abstraction** of the system with a set of fixed services. As defined by [3], these services define the functional properties of the system and is represented by the dotted-line oval on top of Figure 1, called *medium*.
- **A distributed architecture** for implementing the system. As the system should allow distributed collaborations among services, its internals are implemented as a set of distributed components, called managers. Managers collaborate to provide the specified services.
- **A development method** proposed in [4] which consists in a set of refinements successively applied. Each refinement considers a particular design concern and each concern has several alternative solutions. The refinement process can be described and automated by using reusable model transformations.

In [2], we reused the refinement process in order to build architectural variants of a software system. The process was extended to allow evolution and completed with a composition step enabling to embed all variants of a manager into