Enabling Reconfiguration of Component-Based Systems at Runtime

Jasminka Matevska-Meyer and Wilhelm Hasselbring
Department of Computing Science, Software Engineering Group
University of Oldenburg, Germany
{matevska-meyer, hasselbring}@informatik.uni-oldenburg.de

1 Introduction

The development of software systems iterates over analysis, design, implementation, and deployment. Subsequent iterations require refactoring [2] of design and reconfiguration of deployed systems. At least three software engineering disciplines are involved when dealing with runtime reconfiguration of component-based software systems:

- software architecture,
- software configuration management, and
- software component deployment

These disciplines contribute in various ways. Software architectures play a central role at design, describing a system model and specifying it in a formal way using some architecture description language [7]. Configuration management focuses on implementation, defining a configuration from various component versions and building a system from this configuration [6]. Component deployment addresses the deployment phase, managing all dependencies among the involved components and eventually producing a running system [1, 12].

Although these three activities may evolve independently and provide their own models of the system, they are all involved when reconfiguration is required (roundtrip engineering). Applying planned changes to a deployed system usually triggers changes in all those system models to obtain a consistent system after reconfiguration. A major problem to be solved here is managing (run-time) dependencies among the components. Therefore, we need a formal system model, which covers components, their interconnection, communication, and run-time behavior, integrating all the system models of software architecture, configuration management and component deployment [13].

2 An Approach to Enabling Reconfiguration of Component-Based Systems at Runtime

We aim at Reconfiguration of Component-Based Systems at Runtime. Our proposed approach employs:

- Parameterised Contracts [8] as a method for formal component specification, adding a formal run-time component description technique,
- using graphs [5] to describe dependencies among components and considering run-time concerns,
- extending C2-ADL [11] with a concept of containers to establish modelling of a deployment and runtime properties of a system,

This combination shall be the way to provide a foundation for achieving our goals. Figure 1 displays our suggested system configuration. A system configuration is designed as a hierarchy using three GoF design patterns [3]: composite, decorator and adapter.

- Composite is required to build a system configuration,
- The Decorator pattern allows functional changes to components,
- The Adapter pattern (wrapper) allows changes of their interfaces

Furthermore the concept of containers allows us to manage the process of run-time reconfiguration as run-time re-deployment of components.

Our Reconfiguration Manager (a special type of connector) is activated on every reconfiguration request. It consists of:

- Reconfiguration Analyzer
- Dependency Manager
- Consistency Manager
- Reconfigurator

The Reconfiguration Analyzer takes a reconfiguration request, analyzes and classifies the requested change. Our Dependency Manager monitors the run-time dependencies among components, determines a minimal set of change-affected components and sends a change request for each
involved component to the reconfigurator. The Consistency Manager controls the system. We divide its activities into:

- Pre-Reconfiguration: checking the static consistency of the intended system configuration and moving a consistent system into a ready-to-configure-state, or refusing the reconfiguration request on failure.
- Post-Reconfiguration: checking the (run-time) consistency of a changed system and, on success, confirming a reconfiguration, or sending a rollback request to the reconfigurator.

The Reconfigurator realizes the reconfiguration as a dependent change transaction [4]. It starts a transaction, transfers all affected components into a blocked state, isolates an affected subsystem, applies the changes, and sends a consistency-check-request to the consistency manager. On success it commits the transaction, on failure it initiates a rollback and transfers the changed or unchanged system into a running state.

Figure 2 displays all states a component can take at system runtime. Just after it has been deployed we assume that it is free. We distinguish between the states busy, which means is in use and active, which means is executed. Therefore, a component can’t directly move into a state active & busy. Only free components can be transferred into a blocked state and be changed afterwards. This means, our reconfiguration takes place while the system is running, we are not trying to achieve an ad-hoc component change.

We assume that a (sub)system can take only four states at runtime: running, ready to configure, reconfiguring and restoring (Figure 3). For each state a corresponding part of the reconfiguration manager initiates and controls possible changes from one state into another.

3 Summary

We present an approach to enabling reconfiguration of component-based systems at runtime. This approach combines the disciplines software architecture, configuration management and component deployment.

As an implementation platform we are using J2EE-Technology [10]. We are intending to extend its Specification of the deployment process with a subprocess of reconfiguration [9]. Currently, we are investigating the possibilities to control or manipulate the deployment process at different application servers and develop a methodology for determining and formally specifying dependencies among already deployed components.
References


