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 planed 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

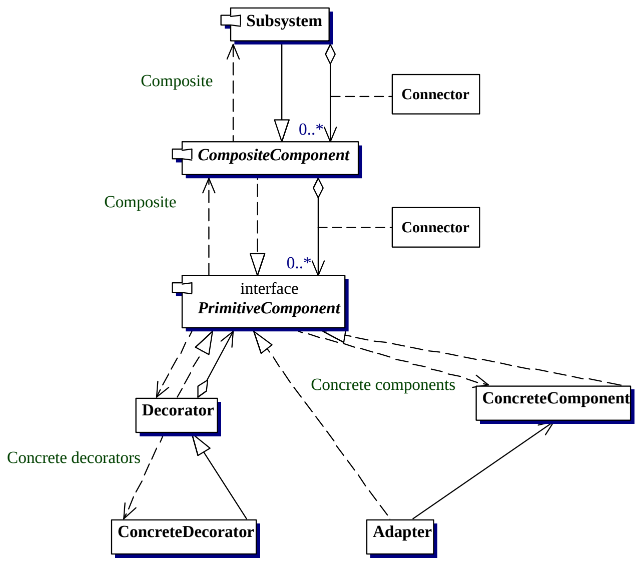

Figure 1. System Configuration

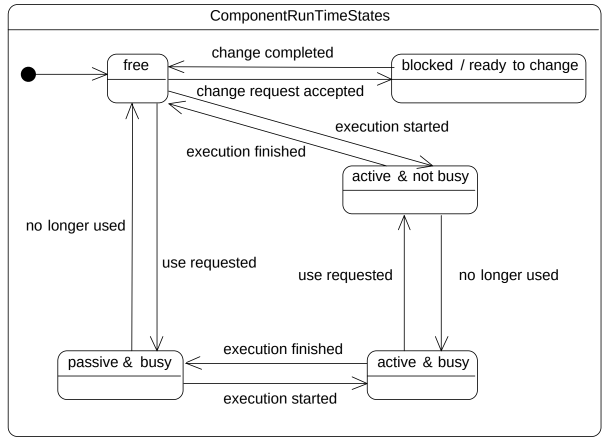

Figure 2. Component Run Time States

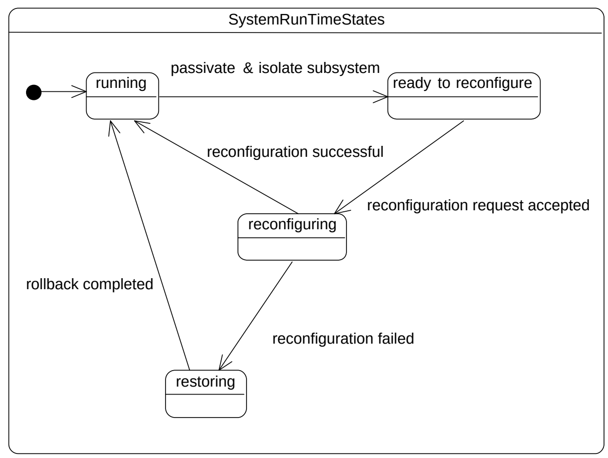

Figure 3. System Run Time States

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

[1] A. Carzaniga, A. Fuggetta, R. S. Hall, A. van der Hoek, D. Heimbigner, and A. L. Wolf. A characterization framework for software deployment technologies. Technical Report 857-98, Department of Computer Science, University of Colorado, Apr. 1998.

[2] M. Fowler. *Refactoring: improving the design of existing code*. Addison-Wesley Longman Publishing Co., Inc., 1999.

[3] Gamma, Helm, Johnson, and Vlissides. *Design Patterns Elements of Reusable Object-Oriented Software*. Object-Oriented Technology. Addison-Wesley, Massachusetts, 1995.

[4] J. Kramer and J. Magee. The evolving philosophers problem: Dynamic change management. *IEEE Transactions on Software Engineering*, 16(11):1293–1306, Nov. 1990.

[5] M. Larsson. *Applying Configuration Management Techniques to Component-Based Systems*. PhD thesis, Uppsala University, Sweden, Dec. 2001.

[6] M. Larsson and I. Crnkovic. Configuration management for component-based systems. In *Proceedins of the Tenth International Workshop on Software Configuration Management*, Toronto, Canada, May 2001.

[7] N. Medvidovic and R. N. Taylor. A classification and comparison framework for software architecture description languages. *IEEE Transactions on Software Engineering*, 26(1):70–93, 2000.

[8] R. H. Reussner. *Parametrisierte Verträge zur Protokolladaption bei Software-Komponenten*. PhD thesis, Universität (T. H.) Karlsruhe, 2001.

[9] M. J. Rutherford, K. Anderson, A. Carzaniga, D. Heimbigner, and A. L. Wolf. Reconfiguration in the Enterprise JavaBean component model. In J. Bishop, editor, *Proceedings of IFIP/ACM Working Conference on Component Deployment*, pages 67–81, Berlin, Germany, June 2002. Springer-Verlag Berlin Heidelberg.

[10] Sun Microsystems. *Java 2 Platform, Enterprise Edition Specification, Version 1.3*, 2002.

[11] R. N. Taylor, N. Medvidovic, K. M. Anderson, E. J. W. Jr., J. E. Robbins, K. A. Nies, P. Oreizy, and D. L. Dubrow. A component- and message-based architectural style for GUI software. *IEEE Transactions on Software Engineering*, 22(6):390–406, 1996.

[12] A. van der Hoek, R. S. Hall, A. Carzaniga, D. Heimbigner, and A. L. Wolf. Software deployment: Extending configuration management support into the field. *CrossTalk The Journal of Defense Software Engineering*, 11(2):9–13, Feb. 1998.

[13] A. van der Hoek, D. Heimbigner, and A. L. Wolf. Software architecture, configuration management, and configurable distributed systems: A ménage a trois. Technical Report 948-98, University of Colorado, Department of Computer Science, Software Engineering Research Laboratory, Colorado, 1998.