Software Architecture Description supporting Component Deployment and System Runtime Reconfiguration

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

Abstract

Architecture description languages (ADLs) can be used for describing architectures of component-based software systems. Typical ADLs provide explicit support for specifying components, connectors and configurations as well as for building hierarchical system configurations. All of them allow to specify structural dependencies among components, thus describing static configurations. This may be sufficient for an initial system composition, but does not provide enough information for post-deployment and runtime reconfiguration. Only a few ADLs provide some support for dynamics, usually without a clear differentiation between a possible behaviour of component descriptions and a runtime behaviour of component instances. Even XML-based ADLs such as xADL 2.0, which clearly distinguishes between the design-time and run-time, only defines structural instance schemata.

In our approach, we observe the "use" dependencies among instances of components (called "live components") of an already deployed and running system. The life components are constrained by specified structural dependencies (defined in "component descriptions"). Live components are hosted in containers. The "Service-Connector-Container" view of our model provides a way to describe the runtime behaviour of a system. Thus, it supports dynamic reconfiguration of component-based software systems. We use "service effect automata" for runtime behaviour specification and intend to extend them using the timing and liveness concepts of "live sequence charts".

Topics from CFP dynamic composition of component-based systems, dynamic architectures, architecture description languages suitable to guide COP, system design for hot-swappable components, addressing variability requirements in component-based solutions

1 Introduction

Operative software systems have to evolve to meet their ever changing requirements. Reengineering approaches address this problem by means of techniques for re-designing the systems. Requirements engineering approaches aim at improving the definition of evolving requirements and, this way, support system evolution. Variability management approaches concentrate on design of systems which include variation points for post-deployment system adaptation. All of them support system evolution, but do not help with maintaining high availability of software systems during the process of applying the changes.

Runtime reconfiguration aims at providing high availability of software systems. Main issues are maintaining the consistency of systems during reconfiguration and minimising the down-time caused by the reconfiguration.

Consequently, techniques are required which identify the components affected during reconfiguration, and, accordingly, those components which can continue execution during reconfiguration. In order to identify the minimal set of affected components, we need a system description which provides information of its runtime behaviour, particularly concerning the use dependencies among instances of components. Furthermore, we should be able to recompose the system during its runtime.

In an already deployed and running system we determine time-constrained use dependencies among instances of components (live components), which are constrained by the structural dependencies. A container provides the runtime environment for the live components [Obj03, Sun03]. The *Service-Connector-Container* view of our Model provides an appropriate description of the system runtime behaviour to enable dynamic reconfiguration and recomposition of component-based software systems [MMHR03].

This paper is organised as follows. First, we determine the requirements on an ADL to support deployment and runtime reconfiguration in Section 2 and then give a short summary of ADLs according to those requirements (Section 3). Next, we present our Meta-Model in Section 4. In Section 5 we show a way to perform a runtime reconfiguration using our model. Finally, we conclude and indicate further work in Section 6.

2 Requirements on an ADL to support component deployment and system runtime reconfiguration

Currently, the term ”system architecture” is equated with a structural decomposition of a system into subsystems and their connections. Although this may already form a base for communicating on the system, a mere structural description of a system fails short in (a) supporting further system implementation, (b) allowing system analysis and (c) enabling system reconfiguration. Therefore, we take the position, that an ADL should support the following kinds of views:

1. Structural View: The classical boxes and lines diagrams. The boxes represent components (in a loose sense of the term component) and the lines connect components. This view is used to decompose a system.

2. Dynamic View: This view describes the dynamic behaviour of a system at runtime. The term *dynamic* does not necessarily refer to dynamic changes of the structural view (*dynamism* as defined in [MT00]). Muchmore, we think a dynamic view must be concerned with the path of the control-flow(s) through the system's structure. Of course, this can result in dynamic changes of the structure, e.g., if components are created or destroyed dynamically, but this has not necessarily to be the case.

3. Resource Mapping View: The components of the structural view have to be mapped (a) to hardware units (such as processors, network connections, or specific devices, like printers, etc) and (b) to organisational entities, such as developers, teams or third-party vendors. This is important as an extension of the pure software reconfiguration for considering quality of service (QoS) properties of the system.

These views are not fully supported by the three viewtypes defined in [CBB$^+$02]. The module viewtype and the component-connector viewtype are concerned with structural views, while the allocation viewtype matches with the resource mapping views. Besides that, the above list is orthogonal to the four views taken by [HNS99]. We consider Hofmeister's conceptual view and module view as structural view, Hofmeister's code view and execution view partially model information of our dynamic view (or are at least strongly influenced by that information) and Hofmeister's execution view corresponds to our resource mapping view. However, the above list of view kinds emphasises the fact that an architecture is not just defined by structural information. This is somewhat different to Medvidovic and Taylor, who emphasise the existence of connectors as first-class entities as a defining property of ADLs [MT00].

From our point of view, referring to our approach, the most essential requirement on an ADL concerning component deployment and system runtime reconfiguration is the support of (1) dynamic configurations, which alone is not sufficient. To check and ensure consistency of the system at runtime after performing the reconfiguration, ADLs have to provide (2) a dynamic component behaviour description and (3) timing constraints. They are also

	(1) dynamic configurations	(2) dynamic behaviour	(3) timing constraints	(4) runtime dependencies	(5) subsystem composition
ACME	-	-	-	-	+
Aesop	-	-	-	-	-
C2	+	-	-	-	+
Darwin	+	+ -	-	-	+
MetaH	-	-	-	-	-
Rapide	+	+	+	+ -	-
SADL	-	-	-	-	-
UniCon	-	-	-	-	+
Weaves	+	-	-	-	+
Wright	+	+	-	+ -	+
xADL 2.0	+	-	-	-	+

Figure 1. ADL support for component deployment and system runtime reconfiguration

needed for determining a convenient point in time for the reconfiguration. Finally, in order to isolate the minimal set of affected components and to rebuild the system, we need an appropriate description of (4) runtime dependencies among components and (5) composition properties of subsystems.

3 State of the Art of ADLs

The majority of the ADLs support only a structural view of the system. Even if offering any techniques for describing behaviour of the system, they only model its possible behaviour and thus can check its consistency only statically (e.g. correctness of proposed configuration, type checking, pre- or post-conditions, protocol). A few of them (C2, Darwin, Rapide, Weaves and Wright) support dynamic configurations [MT00]. C2 [TMA$^+$96] specifies only pre- and post-conditions, Darwin [MDEK95] expresses component semantics in terms of $\pi - calculus$, Weaves [GR91] defines partial ordering of data-flow over input and output objects, but only Rapide [LV95] and Wright [All97] specify dynamic component behaviour. Wright focuses on specifying communication protocols among components and uses a variant of CSP [Hoa85] to describe architectural behaviour. It treats both components and connectors as processes, which synchronise over suitably renamed alphabets. But, it implies a component interface extension in case of permitted reconfiguration and checks only if a connector protocol is deadlock-free as a consistency check [AGD97]. Finally, Rapide is the only ADL that provides timing constraints. It focuses on event-based communication (POSET - partially ordered set of events) [LVB$^+$93] where dependencies and time relations between events are specified, but it supports only instances of the architecture of the whole system and no subsystem composition properties. It includes different sublanguages and a separate executable language for implementing the architecture [Tea97].

The XML-based ADL xADL 2.0 [DvdHT01], clearly distinguishes between the design time and runtime, but defines basically a structural instance schema, describing the topological organisation of component an connector instances and no specification of their runtime behaviour.

A summary of our evaluation is shown in Figure 1.

4 Our Meta Model

There are a lot of definitions of the term *component* depending mainly on the abstraction level (visibility) of their implementation. We consider, from a static point of view, a *component description* in terms of [Szy02]:

> ”A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties. For the purposes of independent deployment, a component needs to be a binary unit. Technically, a component is a set of primitive components, each of which is a module plus resources.”

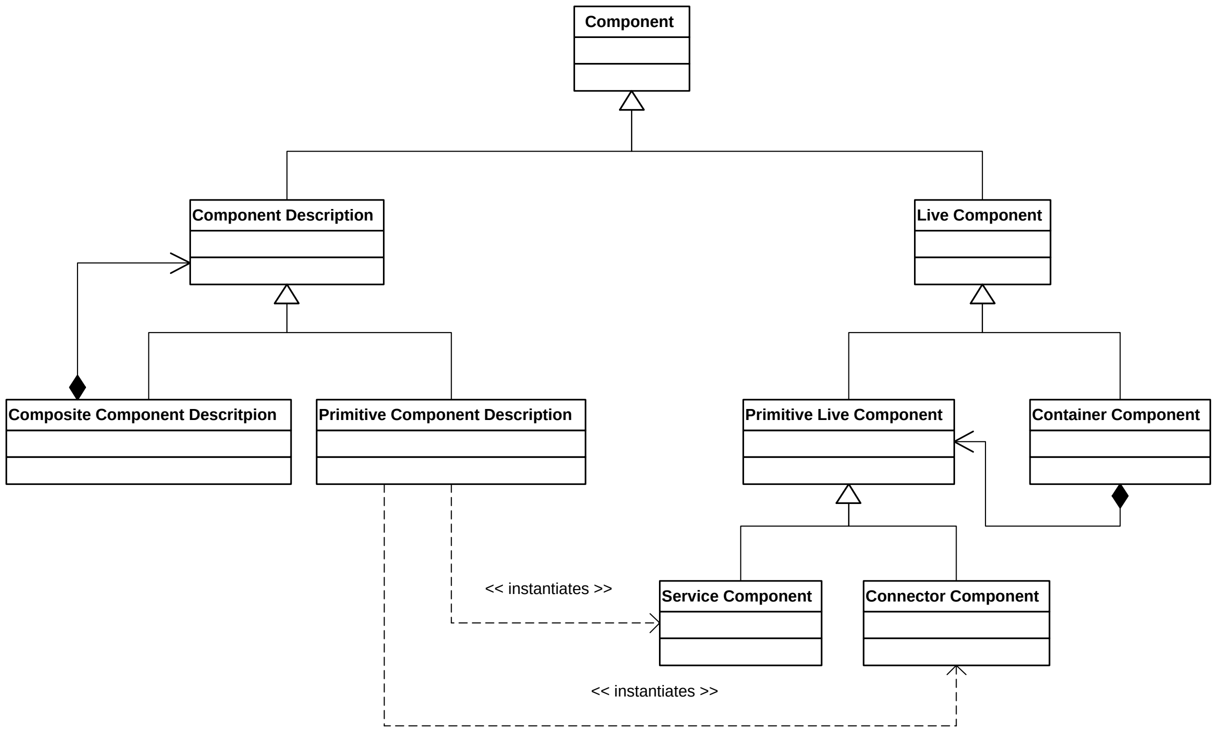

Figure 2. Our Meta Model

Because we consider the system at deployment time and runtime, it is important to distinguish between components and their instances. Such component instances are also components and we call them *live components* to stress the dynamics. We define a *component* as a first-class entity and derive *(1) component description* and *(2) live component* (Figure 2).

1. Structural View: A *component description* including *(1) primitive* and *(2) composite components* enables building a hierarchical system architecture based on a GoF-Pattern *Composite* [GHJV95] and thus fulfilling our fifth requirement (subsystem composition).

2. Dynamic View: A *live component* has two specialisations *(1) primitive live component* and *(2) container component* and presents the runtime view of the system. The primitive component description instantiates both specialisations of the primitive live component: *(1) service component* or *(2) connector component*. The connector component is used to describe both uses and interaction dependencies among live components. Container components are an essential extension to establish modelling of deployment and runtime properties of the system. They host primitive live components without considering hierarchical aspects. To describe the runtime behaviour of the system we use *service effect automata* (as specified in [Reu01]). For determining the reconfiguration point in time we propose to use a special extension of message sequence charts called *live sequence charts* [DH01] because they can additionally express liveness and timing constraints. This should provide a basis for fulfilling all other requirements as specified in section 2.

3. Resource Mapping View: As a part of consideration of non-functional reconfigurations [MMHR03] concerning QoS properties (e.g. performance aspects), we intend to add a resource mapping view to our model as future work.

5 Platform Independent Reconfiguration Manager - PIRMA

In this section we present our work in progress implementing the concepts of [MMH03] using our model. Our reconfiguration manager (Figure 3) also presented in [MMHR03] consists of the following four top-level compo-

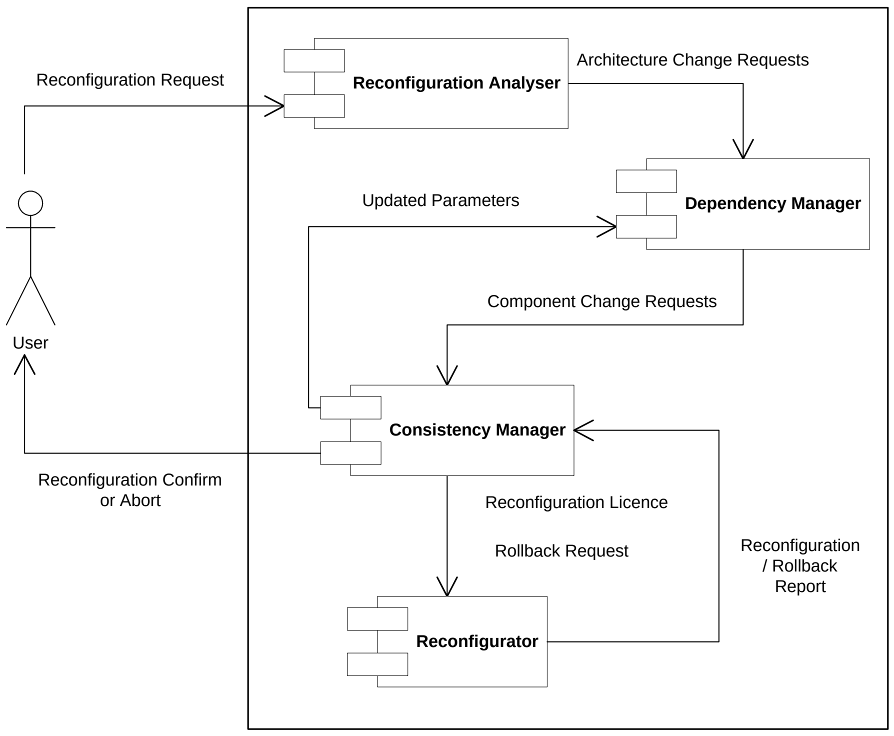

Figure 3. Architecture of our Reconfiguration Manager PIRMA

nents: *Reconfiguration Analyser*, *Dependency Manager*, *Consistency Manager* and *Reconfigurator*. It plays the role of a *Component Configurator* [SSRB00, Kon00]. Currently we follow an approach of sequential reconfiguration request processing which presumes a hierarchical system decomposition (Fig. 4). Therefore our container component contains only primitive live components. In principle this is not necessary and is not always a fastest way to reconfigure the system (eg. coexistent insertion or change of several independent components), but it provides a solid basis for checking and maintaining the consistency of the system at runtime after performing the reconfiguration.

Finally, applying required changes to an already deployed and running system usually triggers changes in a system configuration and implies its reconstruction and redeployment to obtain a consistent system after reconfiguration. Our approach concerning the deployment called *controlled runtime redeployment* presents an extension of the concepts of *hot deployment* and *dynamic reloading* as supported by the WebSphere Application Server [IBM03]. We additionally allow structural changes of the running system [MMHR03] and manage consistency problems in contrast to both other concepts, which only allow a simple swap of an application or a single component at runtime. Redeployment of software components in the J2EE (Java 2 Enterprise Edition) platform was first introduced by the J2EE product vendors as a convenience for component developers who continuously need to perform tests in a running environment. As a consequence to this dedication, redeployment was and actually is an operation that potentially invalidates existing user sessions. The recent published J2EE Deployment API Specification [Sea03] introduced as a part of the new J2EE 1.4 Specification [Sun03] takes the concept of redeployment one step further by demanding redeployment to be transparent to users. However, no J2EE application server includes a fully functional implementation of the Deployment API Specification yet. In the student project ”J2EE Deployment API Implementation” [Oll04] the technical basis for an API implementation for the JBoss Application Server [JBo] was developed.

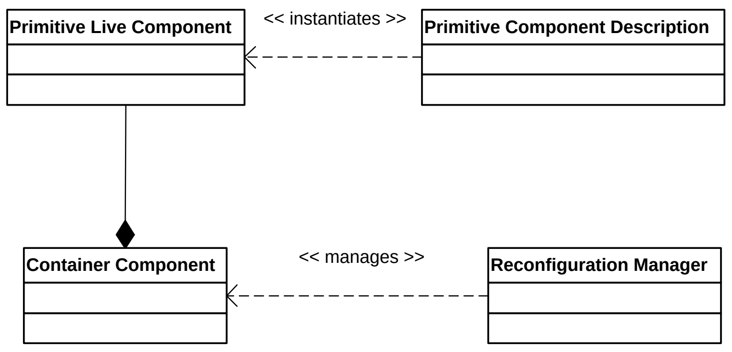

Figure 4. Context of our Reconfiguration Manager PIRMA

6 Conclusions and Future work

Requirements on an ADL to support component deployment and system runtime reconfiguration are discussed and according to them an evaluation of current ADLs is presented. A meta model for software architecture description supporting component deployment and system runtime reconfiguration was suggested and a way of performing runtime reconfiguration using the proposed model was presented.

Currently, we work on including timing and liveness constraints to the service effect automata using live sequence charts and on the implementation of our system.

Future work includes adding a resource mapping view to our model to enable quality-oriented reconfigurations (e.g., concerning performance aspects). We also investigate the benefits of simulation methods for predicting the optimal point in time for performing a particular reconfiguration request.

References

[AGD97] Robert Allen, David Garlan, and Remi Douence. Specifying dynamism in software architectures. In *Proceedings of the Workshop on Foundations of Component-Based Software Engineering*, Zurich, Switzerland, September 1997.

[All97] Robert Allen. *A Formal Approach to Software Architecture*. PhD thesis, Carnegie Mellon, School of Computer Science, January 1997. Issued as CMU Technical Report CMU-CS-97-144.

[CBB$^+$02] Paul Clements, Felix Bachmann, Len Bass, David Garlan, James Ivers, Reed Little, Robert Nord, and Judith Stafford. *Documenting Software Architectures: Views and Beyond*. Addison-Wesley, 2002.

[DH01] Werner Damm and David Harel. LSCs: Breathing life into message sequence charts. *Formal Methods in System Design*, 19(1):45–80, 2001.

[DvdHT01] Eric M. Dashofy, Andre van der Hoek, and Richard N. Taylor. A Highly-Extensible, XML-Based Architecture Description Language. In Phillipe Kruchten, Chris Verhoef, Rick Kazman, and Hans van Vliet, editors, *Proceedings of The Working IEEE/IFIP Conference on Software Architecture*, pages 103 – 112. IEEE Computer Society, 2001.

[GHJV95] Gamma, Helm, Johnson, and Vlissides. *Design Patterns Elements of Reusable Object-Oriented Software*. Object-Oriented Technology. Addison-Wesley, Massachusetts, 1995.

[GR91] Michael M. Gorlick and Rami R. Razouk. Using Weaves for software construction and analysis. In *Proceedings of the 13th international conference on Software engineering*, pages 23–34. IEEE Computer Society Press, 1991.

[HNS99] Christine Hofmeister, Robert Nord, and Dilip Soni. *Applied Software Architecture*. Addison-Wesley, 1999.

[Hoa85] Tony Hoare. *Communicating Sequential Processes*. Prentice Hall International, 1985.

[IBM03] IBM, http://www-3.ibm.com/software/webservers/appserv/doc/v40. *WebSphere Application Server Documentation*, June 2003.

[JBo] *JBoss Application Server Documentation*. http://www.jboss.org/docs/index. Retrieved 2004-03-30.

[Kon00] Fabio Kon. *Automatic Configuration of Component-Based Distributed Systems*. PhD thesis, University of Illinois at Urbana-Champaign, 2000.

[LV95] David C. Luckham and James Vera. An event-based architecture definition language. *IEEE Transactions on Software Engineering*, 21(9):717–734, September 1995.

[LVB$^+$93] David C. Luckham, James Vera, Doug Bryan, Larry Augustin, and Frank Belz. Partial orderings of event sets and their application to prototyping concurrent, timed systems. *Journal of Systems and Software*, 21(3):253–265, June 1993.

[MDEK95] Jeff Magee, Naranker Dulay, Susan Eisenbach, and Jeff Kramer. Specifying distributed software architectures. In *Proc. of 5th European Software Engineering Conference (ESEC '95), Sitges*, pages 137–153. Springer-Verlag, September 1995.

[MMH03] Jasminka Matevska-Meyer and Wilhelm Hasselbring. Enabling reconfiguration of component-based systems at runtime. In J. Bosch J. van Gurp, editor, *Proceedings of Workshop on Software Variability Management*, pages 123–125, Groningen, The Netherlands, February 2003. University of Groningen.

[MMHR03] Jasminka Matevska-Meyer, Wilhelm Hasselbring, and Ralf Reussner. Exploiting protocol information for speeding up runtime reconfiguration of component-based systems. In *Proceedings of Workshop on Component-Oriented Programming WCOP 2003*, Darmstadt, Germany, July 2003. Technical University of Darmstadt.

[MT00] Nenad Medvidovic and Richard N. Taylor. A classification and comparison framework for software architecture description languages. *IEEE Transactions on Software Engineering*, 26(1):70–93, 2000.

[Obj03] Object Management Group OMG. *CORBA Component Model, V3.0*, 2003. http://www.omg.org/technology/documents/formal/components.htm.

[Oll04] Sascha Olliges. J2EE Deployment API Implementation. Student Project, supevised by Jasminka Matevska-Meyer and Wilhelm Hasselbring, University of Oldenburg, Germany, Department of Computing Science, Software Engineering Group, Jan 2004.

[Reu01] Ralf H. Reussner. *Parametrisierte Verträge zur Protokolladaption bei Software-Komponenten*. PhD thesis, University (T. H.) of Karlsruhe, Germany, 2001.

[Sea03] Rebecca Searls. *J2EE Deployment API Specification, Version 1.1*. Sun Microsystems, http://java.sun.com/j2ee/tools/deployment/, November 2003. Retrieved 2004-03-30.

[SSRB00] D. Schmidt, M. Stal, H. Rohnert, and F. Buschmann. *Pattern-Oriented Software Architecture*, volume Vol. 2: Patterns for Concurrent and Networked Objects. John-Wiley & Sons, 2000.

[Sun03] Sun Microsystems, http://java.sun.com/j2ee/. *Java 2 Platform, Enterprise Edition Specification, Version 1.4*, 2003.

[Szy02] Clemens Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley Longman Publishing Co., Inc., 2002.

[Tea97] Rapide Design Team. *Guide to the Rapide 1.0, Language Reference Manuals*. Program Analysis and Verification Group, Sandford University, July 1997.

[TMA$^+$96] Richard N. Taylor, Nenad Medvidovic, Kenneth M. Anderson, E. James Whitehead Jr., Jason E. Robbins, Kari A. Nies, Peyman Oreizy, and Deborah L. Dubrow. A component- and message-based architectural style for GUI software. *IEEE Transactions on Software Engineering*, 22(6):390–406, 1996.