Architectural Design of Component Composition Using First Class Connectors

Asuman Sünbül
Graduate School of Distributed Information Systems
Computation and Information Structures (CIS),
Technical University of Berlin
asu@cs.tu-berlin.de

April 9, 2000

Abstract

Today's software systems can be seen as "systems of systems". The dependencies between these parts are too complex for being fully understood by a single persons or a single program. The component based software development seems to be a convincing concept in order to manage these complexities, therefore component-oriented software development is becoming more and more important for the production of large-scale software applications.

The central problem for this task is that the existing components are often not designed to be re-used or interfere with other components, meaning that they do not provide clean interface and functionality specifications. Provided *component interfaces* or informal descriptions are not enough for designing reliable software systems. Therefore we need precise description means for formulating component behavior without uncovering implementation details of the component.

The approach presented in my work therefore tries to combine these two aspects of software development by presenting a so called *Service Layer Model* for the evolutionary design of component-based software systems. This model defines a system of abstraction layers for the composition of software components and therefore provides the possibility to actually design the composition of components rather than just implementing the glue code. Further, the design aspects of *connectors* among components play a central role.

The formal semantics of the component model is expressed using the Abstract State Machines (ASM) method. The basic entities of the CM (like component, services, etc.) are represented using ASM universes and functions. ASM rules are used to express the formal semantics in terms of operationally specified consistency check algorithms. The formalization is used to directly derive an executable CM-checker. The component model abstracts from concrete notations for expressing the functionality of the components and its behavior. The component model fixes just what can be specified but not how this is done. The component model

introduces the following specification categories for components and its services: *Constraints* regulate the component usage; *Specifications* specify provided/required properties of services, and *Import Structures* represents information how imported services are used in the component body. Because the notation for specifications are not fixed, corresponding consistency check predicates are also treated as abstract entities. These external functions are needed to formulate the consistency rules of the component model. For concrete instantiations of the component model these predicates must be given as external functionality (e.g. theorem prover).

A so called "Component Model Input Language" CMIL is used to enter component model descriptions of the second and third layer of the SLM model. CMIL is defined as a context-free grammar; its top level entities are

- "components"
- "connector"
- "types"
- "views"

The consistency rules are expressed in an operational semantics style using ASM rules. The rules specify a generic framework for component models, they can be seen as the controller for the overall consistency check algorithm. The rules are expressed in terms of the external consistency predicates mentioned above.

Implementation: The rules are directly taken as input to an ASM language that transforms the rules to executable code (Xasm). A CMIL grammar definition is used (in the same language) to automatically generated a parser. During parsing, the initial component model contained in the input file is generated.

The SLM concept serves as a basis for the stepwise development of component-based software systems. The concept of "connector refinement" has been introduced as basic development technique. The Component Model Input Language (CMIL) has been defined in order to serve as user interface to the SLM. From the semantics descriptions of the component model and the CMIL language definition, an executable interpreter has been automatically generated using Xasm support environment [Anl00].

References

[AKPS00] M. Anlauff, P. Kutter, A. Pierantonio, and A. Sünbül. Using domain-specific languages for the realization of component composition. In Tom Maibaum, editor, *Fundamentel Approaches to Software Engineering, Proceedings of the FASE 2000 Conference, ETAPS*, volume 1783 of *lncs*, pages 112–126. Springer, March 2000.

[Anl00] M. Anlauff. XASM - An Extensible, Component-Based Abstract State Machine Language. In Y. Gurevich, P.W. Kutter, M. Odersky, and L. Thiele, editors, *Proceedings of the International Workshop on Abstract State Machines (ASM 2000)*, LNCS, pages 1–22. Springer, 2000. to appear.

[C.S98] C.Szyperski. *Component Software - Beyond Object Oriented Programming.* Addison Wesley, 1998.

[NSL96] Oscar Nierstrasz, Jean-Guy Schneider, and Markus Lumpe. Formalizing composable software systems – A research agenda. In *Proceedings 1st IFIP Workshop on Formal Methods for Open Object-based Distributed Systems FMOODS'96*, pages 271–282. Chapmann and Hall, 1996.

[SA99a] A. Sünbül and M. Anlauff. Domain-specific languages in software architecture. In *Integrated Design & Process Technology.* Society for Design and Process Science, IEEE International Conference on Systems Integration, June 1999.

[SA99b] A. Sünbül and M. Anlauff. Software architecture based composition of components. Gesellschaft fuer Informatik, Sicherheit und Zuverlaessigkeit software-basierter Systeme, May 1999. Bad Honnef, Germany.

[Sün99] A. Sünbül. Abstract state machines for the composition of architectural styles. In *Third International Conference Perspectives of System Informatics*, Novosibirsk, Akademgorodok, Russia, July 1999. Lecture Notes in Computer Science (LNCS), Springer Verlag.

[WS98] J. Bosch W.Weck and C. Szyperski, editors. *Third International Workshop on Component-Oriented Programming*, volume TUCS, October 1998.