Journal of Universal Computer Science, vol. 11, no. 10 (2005), 1577-1579
submitted: 18/10/05, accepted: 25/10/05, appeared: 28/10/05 © J.UCS

Compositional Construction and Reasoning Techniques for Software

J.UCS Special Issue

Farhad Arbab
Center for Mathematics and Computer Science (CWI), Amsterdam
and
Leiden Institute of Advanced Computer Science, Leiden University
The Netherlands
farhad@cwi.nl

Joost N. Kok
Leiden Institute of Advanced Computer Science, Leiden University
The Netherlands
joost@liacs.nl

Complex software systems are intrinsically difficult to conceive, develop, deploy, maintain, and evolve, especially in concurrent and distributed settings. Compositional techniques for construction and analysis of software, such as object-oriented and component-based approaches, have shown to be effective tools for breaking these complexities down to manageable sizes. The complexity of these systems is confounded in contexts such as real-time and embedded systems, where concurrency and distribution arise not merely due to performance concerns, but rather reflect the inherent requirements and operating constraints of application systems.

Interaction and coordination models that have been developed to tackle these problems often lack the adequate sophistication to accommodate systems where self-organization, emergent behavior, and evolution comprise their key concerns. Approaches based on compositional techniques seem to offer promising extensions to these models to meet the challenges of such increasingly important emerging areas as ubiquitous computing, self-organizing self-managing systems, service-oriented computing, and bio-informatics, where interaction, its composition, and its coordination play an explicit central role.

This special issue offers a collection of 6 contributions on both theoretical and systems aspects of compositional methods for construction of software. These contributions cover a wide range of topics: identifying objects in non-object-oriented specifications, characterization of multi-class grouping and encapsulation mechanisms in object oriented languages, dynamic composition of .net-based services, using coordination models to add behavioral descriptions to component interfaces, modular verification, and probabilstic models for compositional component connectors.

By providing encapsulation, abstraction, and inheritance, the object oriented paradigm offers many advantages for composition over plain procedural imperative programming. Object oriented extensions have been introduced not only to imperative programming languages (e.g., C/C++), but also to declarative modeling and specification languages, such as VDM/VDM++ and Z/Object-Z. A.M. Cruz, L.S. Barbosa, and J.N. Oliveira argue that because these extensions are rather recent and users are more familiar with the older formalisms, analogous to spaghetti code produced by those who write C programs in C++ or Java, often what can elegantly be modeled by state-less equations, ends up specified as assignments to instance variables and the like. The lack of abstraction and encapsulation that objects provide, makes such specification not only less comprehensible, but also less suitable for composition. In their paper "From Algebras to Objects: Generation and Composition" they consider the criteria under which a declarative specification in a language can be *objectified* in its object oriented extension. They describe the *objectification process* and present a tool that shows how a model in VDM-SL can be converted to VDM++ objects.

Software composition often involves groups of objects and/or classes that somehow interact with one another, as opposed to single objects or classes. Different object oriented programming languages offer different facilities for grouping of logically related classes, for instance, packages, modules, namespaces, etc. In their paper "Analyzing Module Diversity," A. Bergel, S. Ducasse, and O. Nierstrasz present a module calculus as a foundation for comparing and understanding the diversity of various grouping mechanisms available in different languages. Using their calculus, they can capture the semantics of Java packages, C# namespaces, etc., in the same framework. This leads to a taxonomy for classification of features of grouping mechanisms that clarifies their similarities and distinctions.

F. Cao, B.R. Bryant, R.R. Raje, A.M. Olson, M. Auguston, W. Zhao, and C.C. Burt address dynamic component composition in the context of Web Services. In their paper "A Non-Invasive Approach to Assertive and Autonomous Dynamic Component Composition in the Service-Oriented Paradigm" they describe two types of dynamic component composition and consider three case studies to show their use.

Component composition requires knowledge of their interactive behavior, above and beyond simple compatibility of the signatures in their interfaces. In their paper "Coordinating Behavioral Descriptions of Components," S. Amaro, E. Pimentel, and A.M. Roldan investigate the use of coordination models and languages as a framework for describing the behavior of components. They illustrate the feasibility of their proposal using two substantially different coordination models: Linda and Reo.

Formal methods for verification are indispensable in the development of reliable software systems. Specification of concurrent reactive systems typically uses automata or temporal logic as its foundation. Automata based approaches quickly lead to state explosion in large systems and using deductive methods to verify complex software requires formulating it as a theorem, which is often not easy to do. By abstraction and modularization, perhaps using domain-specific approaches, compositional verification breaks down the complexity of reasoning about the behavior of a complex system to manageable parts. In their earlier work, M. Sirjani et al. have introduced Rebeca as an actor-based language intended for bridging the gap between formal verification and real applications. This language comes with its own model-checker, plus a tool that translates Rebeca code into input for other existing model-checkers. In their current contribution "Modular Verification of a Component-Based Actor Language," M. Sirjani, F.S. de Boer, and A. Movaghar extend the underlying theory of their modular verification approach by introducing synchronous communication and the notion of components in Rebeca.

Reo is a language for compositional construction of component connectors and *constraint automata* have been proposed as an operational model for the semantics of Reo circuits. In her paper "Probabilistic Models for Reo Connector Circuits," C. Baier introduces a probabilistic variant of constraint automata. Probabilities and nondeterminism allow probabilistic constraint automata to serve as a model for connectors that may lose or corrupt their data, or perform randomized acts of coordination.

F. Arbab and J.N. Kok
Guest Editors
October 2005