

## Preface

The idea to exploit and reuse components to build and to maintain software systems goes back to “structured programming” in the 70’ies. It was a strong argument for development of object oriented methods and languages in the 80’ies. However, it is today’s growing complexity of systems that forces us to turn this idea into practice. So far, there is no agreement on standard technologies for designing and creating components, nor on methods of composing them. Finding appropriate formal approaches for describing components, the architectures for composing them, and the methods for component-based software construction, is correspondingly challenging.

### The Theme of the Volume

The range of component technology is both wide and diverse, but some common understanding is emerging through ideas of model-based development. These include the notions of *interfaces*, *contracts*, *services*, *connectors* and *architecture*. Also key issues in application of the technology become clearer, these include: consistent integration of different views of a component, component composition, component coordination, component customization, component system reconfiguration, and component reuse. There are solutions to some of these problems such as composition, refinement and transformation for platform. However, we still know little about theories that support analysis and synthesis of component base systems, including adapting components for specific non-functional requirements.

This volume focuses on mathematical models that identifies the “core” concepts as first class modelling elements, and providing techniques for integrating and relating them. Volume contains eleven chapters by well-established researchers. The chapters are written from different perspectives. However, each chapter gives an explicit definition of components in terms of a set of key aspects and addresses some of the problems of integration and analysis of different views: *component specification*, *component composition*, *component coordination*, *refinement* and *substitution* and techniques of solving the problems. The concepts and techniques are motivated and explained with the help of examples or case studies.

## A Summary of the Chapters

The chapters are organised according the alphabetic order of of the authors. The chapters are all reviewed by experts in the community of formal methods of software system development, and sample chapters are reviewed by four referees appointed by the publisher, World Scientific Publishing. We give a brief summary of each chapter here.

### **Chapter 1. Temporal Specifications of Component Based Systems with Polymorphic Dynamic Reconfiguration, by N. Aguirre and T. Maibaum**

This chapter presents a formal characterisation of component based systems with support for polymorphic dynamic reconfiguration. *Dynamic* reconfiguration is about changes in the system architecture at run time. *Polymorphic* reconfiguration means that reconfiguration operations may concern different types of components or connections, exploiting an inheritance relationship over components, as in object orientation. On top of a first-order temporal logic, and in the form of a (rather low level) specification language, a necessary machinery is built for specifying components, connectors and amalgamations, together with inheritance and polymorphism.

### **Chapter 2. Coordinated Composition of Software Components, by F. Arbab**

This chapter leaves the realm of object oriented programming and thus Abstract Data Types (ADT) and develops a simpler model of components and their composition based on the notion of Abstract Behavior Types (ABT). Whereas the ADT model emphasizes abstract operations on data types and hide data structures, the ABT model emphasises abstract observable behavior and hides operators and data types altogether. Consequently, the ABT model supports a much looser coupling than is possible with the ADT's operational interface, and is inherently amenable to exogenous coordination. Component composition in the ABT model requires only simple operators under which the model is closed: the composition of two ABTs always yields another ABT. To demonstrate the utility of the ABT model, an exogenous coordination language, called Reo, is described for compositional construction of component connectors based on a calculus of channels.

### **Chapter 3: On the Semantics of Componentware: A Coalgebraic Perspective, by L. Barbosa, M. Sun, B.K. Aichernig and N. Rodrigues**

In this chapter software components are regarded as specifications of state-based modules, encapsulating a number of services through a public interface and providing limited access to an internal state space. Components persist and evolve in time, being able to interact with their environment during their overall computation. We adopt the standpoint of coalgebra theory to address a number of issues in the semantics, calculi and methodologies of componentware, presenting an integrated view of our current research concerns. At the specification level, the duality between

algebraic and coalgebraic structures provides a bridge between models of static and dynamic systems. At the programming level such a duality, in a canonical initial-final specialisation, captures the intuitive symmetry between data and behaviour, providing the basis for more uniform and generic approaches to systems' construction.

**Chapter 4. A Theory of Requirements Specification and Architecture Design of Multi-Functional Software Systems, by M. Broy** This Chapter extends the FOCUS model and theory of distributed concurrent interactive systems to two essentially dual concepts of systems engineering. One addresses the comprehensive functionality of multi-functional systems in terms of services and the other that of architectures formed by networks of components that are described by their interfaces. We show how these notions interact and work together in requirements engineering and systems design.

**Chapter 5. Components: From Objects to Mobile Channels, by F.S. de Boer and M.M. Bonsangue and J.V. Guillen-Scholten** This chapter introduces a formal model of components which extends object-orientation with additional structuring and abstraction mechanisms to support a modelling discipline based on interfaces. The component model formalizes the concepts of interfaces, roles, connectors, and ports. Components encapsulate their internal class structure and interact only through a certain kind of objects which are called ports. Ports are instances of classes which are represented by roles. Roles export information about the required and provided operations of these classes by means of interfaces. By means of connectors which wire roles of different components together, ports of one component can dynamically create ports of another component. As an example, it shows how to model mobile channels for the dynamic reconfiguration and exogenous coordination of components.

**Chapter 6. Formalizing the Transition from Requirements to Design, by R.G. Dromey** This chapter addresses the problem about how to construct a design out of its requirements. The author shows how a formal representation for individual functional requirements, called behavior trees makes this possible. Behavior trees of individual functional requirements may be composed, one at a time, to create an integrated design behavior tree. From this problem domain representation it is then possible to transition directly, systematically, and repeatably to a solution domain representation of the component architecture of the system and the behavior designs of the individual components that make up the system - both are emergent properties of the integrated design behavior tree.

**Chapter 7. rCOS: a Relational Calculus of Components, by Z. Liu, J. He and X. Li** This chapter defines a model for components, their composition and refinement. Components are specified for its syntactical view at the interface level, functional view at the requirement level, internal view at the design level and

how they are composed. In a component based system development, a component consists of a set of interfaces, *provided* to or *required* from the software being developed. In a component development, the component is an executable code that can be coupled with other components via its interfaces. The developer has to ensure that the specification of a component is satisfied by its design and the design is met by its implementation.

**Chapter 8. Characterising Frameworks in First-Order Predicate Logic, by S-M. Ho and K-K. Lau** This chapter provides a formal foundation to the component-based approach Catalysis. In Catalysis, a framework is a reusable artefact that can be adapted and composed into larger systems. The *signed contract* between components specifies how the required properties of one component are satisfied by the provided properties of another. The authors examine this concept in the context of framework-based development. They consider a simplified view of frameworks and their transformation into first-order logic. Theorem proving may be used to check the consistency of framework specifications. The chapter also identifies ways in which these specifications may be simplified beforehand to reduce the burden of proof.

**Chapter 9. Formalization in Component Based Development, by J.P. Holmegaard, J. Knudsen, P. Makowski, A.P. Ravn** This chapter presents a unifying conceptual framework for components, component interfaces, contracts and composition of components by focusing on the collection of properties or qualities that they must share. A specific property, such as signature, functionality behaviour or timing is an *aspect*. Each aspect may be specified in a formal language convenient for its purpose and, in principle, unrelated to languages for other aspects. Each aspect forms its own semantic domain, although a semantic domain may be parameterized by values derived from other aspects. The proposed conceptual framework is introduced by small examples, using UML as concrete syntax for various aspects, and is illustrated by one larger case study based on an industrial prototype of a complex component based system.

**Chapter 10. A Model Driven Approach for Building Business Components, by V. Kulkarni and S. Reddy** This chapter presents a methodology, emerging from and aimed at guiding the engineering practice in modern business system development. The method uses aspect-orientation and model-driven development techniques for specifying different views of interest of a system as models and transforming them in successive stages of refinement with specific aspects of interest being imparted at each stage. The chapter discusses how this approach was used to restructure a model driven development environment resulting in greater reuse and ease of its evolution.

**Chapter 11. A Formal Approach to Constructing Well-Behaved Systems using Components, by S. Moschoyiannis, J.K. Filipe and M.W. Shields**

This chapter is motivated by the fact that present-day software systems are in increasing need of modification and evolution due to changing requirements. It argues that component-based development constitutes a key methodology for creating large-scale, evolvable systems in a timely fashion as it advocates the (re)use of prefabricated replaceable software components. However, it is often the case that undesirable or unpredictable behaviour emerges when components are combined. This is partly due to lack of behavioural information about the individual components. To deal with this problem, the authors describe a formal model for component specification which can be used to support the analysis and predictability of component composition and to identify undesirable behaviour. In their approach, component behaviour is modelled by so-called behavioural presentations, a powerful model of true-concurrency. Moreover, the framework is compositional and supports the assembly of the final system from arbitrary components. Practical benefits of our framework are discussed.

### *Acknowledgements*

We would like to thank all the authors for their hard work and high quality contribution to the volume. We would also like to thank the chapter reviewers for their comments and suggestions on improvement of each chapter, and the volume reviewers for their support. We appreciate the constant support from and the collaboration with Ian Selstrup, Senior Editor of World Scientific Publishing.

Zhiming Liu, International Institute for Software Technology,  
United Nations University,  
Macao SAR, China  
He Jifeng, Software Engineering Institute  
East China Normal University, Shanghai, China

July 2006