A Formal Framework for Scalable Component-Based Systems

Chafia Bouanaka, LIRE Laboratory, University of Constantine2-Abdelhamid Mehri, Constantine, Algeria
Ahmed Amar Debza, LIRE Laboratory, University of Constantine2-Abdelhamid Mehri, Constantine, Algeria
Faiza Belala, LIRE Laboratory, University of Constantine2-Abdelhamid Mehri, Constantine, Algeria
Nadia Zeghib, LIRE Laboratory, University of Constantine 2- Abdelhamid Mehri, Constantine, Algeria

ABSTRACT

Constructing systems from components and building components for different systems require well-established methodologies and processes. This article proposes a formal framework for designing and specifying component-based systems (CBS). The two-dimensional evolutions of CBS are architectural reconfiguration and behavioral adaptation, when user requirements and/or runtime contexts change. This framework provides an incremental design methodology where component interfaces and their corresponding ports are the basic units of software construction, rather than components. Conceptually, interfaces serve to assemble simple components to obtain more complex ones. Behaviorally, they serve to propagate side effects of visible changes of a component on its neighbors. Interfaces also supply interactions and synchronization effects on the underlying sub-system. The calculation process is guided by changes on interfaces where the hierarchical structure of the underlying CBS is maintained. In this framework, CBS specification is supported by some execution tools based on Maude.

KEYWORDS
Component-Oriented Programming, Formal Methods, Maude, Meta-Programming, Software Adaptation

1. INTRODUCTION

Component-based engineering is today a widely recognized and well-established paradigm for tackling complexity of systems (Crnkovic et al., 2011), since the component-based approach consists in building complex systems by composing and reusing existing components. This confers numerous advantages (e.g., productivity, incremental construction, compositionality) allowing to deal with complexity in the construction phase (Falcone et al., 2011). Additionally, (Di Cola et al., 2015) consider that component-based and service-oriented approaches (SOA) are gaining in popularity because they offer other sought-after benefits, namely high modularity and low coupling, as well as the potential to tackle scalability and complexity. Consequently, the component-oriented paradigm is characterized by the emergence of interactions as a first-class concept. Besides, as a unit of design, components ensure a clear separation of concerns between the calculation (behavioral) dimension and the interaction or coordination one, by explicitly defining not only the provided services but also component dependencies. This promotes and facilitates components reuse. The component-oriented

DOI: 10.4018/IJISMD.2017010101

Copyright © 2017, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.
paradigm also endorses system structuring and makes easier software elements validation since each one is designed in a relatively independent manner.

The component-based approach is still an active area of research for both academic and technological sectors since many formal and semi-formal models are proposed and a variety of component models have been developed as: EJB, .Net, OSGI, SOFA and so on. It is worth mentioning that formal models maintain an architectural viewpoint where system specification is topped from the more detailed vision; lines of code, to an extreme level of abstraction; where components are no more than black boxes that expose some interfaces via the associated ports. Moreover, they generally lack execution environments. This restrains their utilization only to the specification phase of the software development process. On the contrary, although component technologies offer execution environments together with the necessary containers to deploy and execute components, they still maintain an object-oriented programming viewpoint by implementing components via objects with minor enrichments. Component Technologies propose various forms of class packaging: assemblies for .Net, bundles for OSGI and so on, with additional metadata to describe component-oriented concepts such as component provided/required services. Hence, they enforce component programming concepts to cope within object-oriented ones.

Since component technologies are still based on the method invocation mechanism to define component interactions, software system designers lack tools to clearly and explicitly specify the underlying system structure or architecture. Thus, they do not exploit the main benefits characterizing formal component models as an explicit formalization of the interactions and well-defined boundaries of a component entity.

To bridge the gap between academic and technological viewpoints, we propose a formal framework for scalable component-based systems. This framework considers that a component is a software unit whose functionality and dependencies are entirely defined by its interfaces. It offers an architectural view of distributed systems, specified as a set of black boxes corresponding to functional components; each one being defined in terms of its provided interfaces and eventual reuse constraints via its required interfaces. Consequently, system maintainability and scalability is ensured in terms of architectural evolution and updates by runtime addition/removal of component instances, interfaces and/or component types. A key originality of the proposed model is that it offers a more detailed view of software components by attaching to each provided interface the corresponding implementation module. Hence, the resulting executable formal specifications, implementing component interfaces, allow system validation and verification. Maude (Clavel et al., 2016) is adopted as the adequate formal basis since it is expressive enough to maintain the two views of a software component: the abstract view (structure) and the more detailed behavioral one (implantation).

Our main objective is to define a conceptual methodology for component-based systems where interfaces are now considered as the basic constructing block of software systems. Thus, a fine-grained view of both system structure and the corresponding behavior is obtained. From an architectural viewpoint, system adaptation is no longer restrained to component instances, but goes over to interfaces specification by adding/removing interface types. From a behavioral viewpoint, system functionalities are dynamically updated by changing interface definition. A direct consequence of our modeling methodology is that interfaces and thus components support updates of their specification while they are still running. Moreover, interface definition is easily updated by attaching a new implementation module to the underlying interface.

As a derived objective, we exploit the reflection and meta-programming characteristics of Maude to implement our conceptual methodology for scalable component-based systems in Maude. We define a new layer on top of core-Maude itself, called CpMaude (for Component-oriented Maude), providing a formal model for component-based systems that not only allows specifying software system architecture but also the corresponding formal implementation. Hence, we suggest an executable formal semantic setting for scalable component-based systems. A key originality of the proposed implementation model is that calculation is interface-centric where the rewriting process
A Methodology for Model-Based Reliability Estimation
www.igi-global.com/chapter/a-methodology-for-model-based-reliability-estimation/108631?camid=4v1a

A Canvas for Capturing Context of Agile Adoption
www.igi-global.com/chapter/a-canvas-for-capturing-context-of-agile-adoption/145033?camid=4v1a