

# Verified Systems by Composition from Verified Components

Fei Xie and James C. Browne

Dept. of Computer Sciences, Univ. of Texas at Austin, Austin, TX 78712, USA  
Email: {feixie, browne}@cs.utexas.edu Fax: +1 (512) 471-8885

## I. INTRODUCTION

Component-based development (CBD), developing software systems through composition of components, is one of the most important technical initiatives in software engineering research. Testing is the most commonly used technique for validating software systems being built with CBD. Testing has the inherent test case coverage problem, which hinders development of highly reliable component-based software systems. Application of model checking to software is an important method for improving reliability of software systems. Model checking provides exhaustive state space coverage for the systems being checked. Model checking is particularly effective at detecting coordination errors which frequently result from component compositions and are notoriously difficult to detect by testing. However, model checking often cannot handle large-scale software systems due to state space explosion.

Model checking and CBD are synergistic. Model checking can potentially enable effective development of more reliable component-based software systems. CBD introduces compositional structures, clean component interfaces, and standard composition rules to the systems being built, which can reduce the state spaces model checkers have to handle.

## II. INTEGRATION OF MODEL CHECKING INTO CBD

This paper presents an approach to integrating model checking into the CBD of software systems, which contributes to solution of fundamental problems in both CBD and model checking:

- Development of components which can be reused with certainty that their behaviors will meet their specifications in a proper composition;
- Identifying proper components for a composition;
- Establishing that a component composed from “correct” components will meet its specifications;
- Alleviating the state space explosion problem.

This approach can be summarized as follows:

- As a software component is being built, temporal properties of the component are established, verified, and then packaged with the component.
- Selecting a component for reuse considers not only its functionality but also its temporal properties.
- Properties of a composed component are verified by reusing verified properties of its sub-components and applying compositional reasoning [1].

A general component model, which provides a framework for representing components and their properties and for composing components, is defined. In this model, a property of a component is defined with assumptions on the environment of the component. The property is verified on the component under these assumptions. When the component is reused in the composition of a larger component, the property is *enabled* if the environment assumptions made in its verification hold on other components in the composition and/or the environment of the composed component.

The general component model can be instantiated in many different computation models upon which the syntax and semantics for components, properties, and component compositions are precisely defined. We have instantiated the general component model in an Asynchronous Interleaving Message-passing (AIM) model. In this instantiation, executable representations of components are specified in an executable object-oriented modeling language whose semantics conform to the AIM computation model, such as xUML [3]. This instantiation is of interest due to that although many component-based software systems are not developed on the AIM model, most of them can be readily transformed to systems based on the AIM model.

Components are categorized as *primitive* components (components built from “scratch” and not composed from other components) and *composed* components. Properties of primitive components are verified by directly model checking the object-oriented design models of the components using methods established in our previous work [4, 6, 7]. Properties of a composed component, instead of being model checked on its executable design model, are checked on its *abstraction*. The abstraction is composed

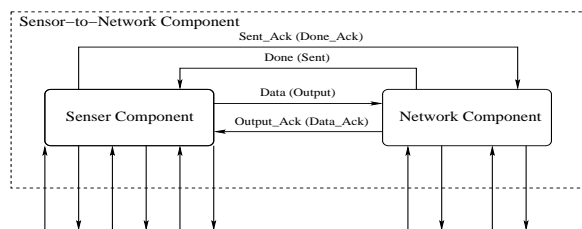
of simple automata corresponding to environment assumptions of the composed component and verified properties of its sub-components. A sub-component property is included in the abstraction if it is enabled in the composition and related to the properties to be checked according to cone-of-influence analysis. The abstraction is constructed based on the composition and reusing the verified properties of the sub-components. If the abstraction of the composed component is still too complex to be checked directly, compositional reasoning is applied to decompose the abstraction. If the abstraction is too abstract for verifying the desired properties, it is refined by verifying additional properties of the sub-components so that the necessary properties of the sub-components can be included into the abstraction.

This approach is founded on compositional reasoning [1]. There has been extensive research on compositional reasoning, most of which applies compositional reasoning in a top-down approach: To check properties of a large system, the system is decomposed into modules recursively in a top-down fashion. Our work combines the top-down approach with the bottom-up component composition process of CBD. Properties of components are verified as they are composed from simpler components in a bottom-up approach and verifications of these properties involve compositional reasoning.

Our approach is most suitable for application to a family of software systems that are built from a set of software components. We have identified two major application domains: Families of software systems based on a specific hardware/software architecture, such as the TinyOS [2] run-time environment, and families of distributed large-scale software systems based on component platforms such as CORBA, DCOM, and EJB.

### III. CASE STUDY

Our approach has been applied to improve reliability of run-time images of TinyOS [2], a component-based run-time environment for networked sensors. A TinyOS run-time image, the Sensor-to-Network component shown in Figure 1, is composed from the Sensor and Network compo-



**Figure 1. Sensor-to-Network Component**

nents by establishing message communications between the two components. The following two properties have been model checked on the Sensor-to-Network component.

**Property 1** *The Sensor-to-Network component transmits sensor readings on physical network repeatedly.*

**Property 2** *The Sensor-to-Network component never transmits any physical sensor reading duplicately.*

Property 1 was verified on the abstraction constructed based on the composition, the verified properties of the two sub-components, and the environment assumptions of the composed component. Verification of Property 2 on the abstraction returns false. The abstraction is then refined by introducing and verifying additional properties of the Sensor component. A bug was detected and corrected when verifying these properties on the Sensor component, which led to re-verification of all properties of the component. Detailed discussions of this case study can be found in [5].

### IV. CONCLUSIONS

Our approach to integrating model checking into CBD has demonstrated substantial potential for improving reliability of component-based software systems and reducing verification complexity of these systems. Detailed discussions and illustrations of this approach can be found in [5]. We are instantiating this approach on other computation models and applying it to improve reliability of other software component families.

### ACKNOWLEDGMENT

We gratefully acknowledge the intellectual contribution of Robert P. Kurshan to this research. This research was partially supported by NSF grant 010-3725.

### REFERENCES

- [1] W. de Rover, F. de Boer, U. Hannemann, J. Hooman, Y. Lakhnech, M. Poel, and J. Zwiers. *Concurrency Verification: Introduction to Compositional and Non-compositional Proof Methods*. Cambridge Univ. Press, 2001.
- [2] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System Architecture Directions for Networked Sensors. *Proc. of ASPLOS-IX*, 2000.
- [3] Kennedy Carter. <http://www.kc.com/MDA/xuml.html>. Kennedy Carter, 2002.
- [4] F. Xie and J. C. Browne. Integrated State Space Reduction for Model Checking Executable Object-oriented Software System Designs. *Proc. of FASE 2002*, 2002.
- [5] F. Xie and J. C. Browne. Verified Systems by Composition from Verified Components. *UTCS Technical Report TR-02-40*, 2002.
- [6] F. Xie, V. Levin, and J. C. Browne. Model Checking for an Executable Subset of UML. *Proc. of ASE 2001*, 2001.
- [7] F. Xie, V. Levin, and J. C. Browne. ObjectCheck: A Model Checking Tool for Executable Object-oriented Software System Designs. *Proc. of FASE 2002*, 2002.