Specification and Verification of Component-Based Systems Workshop

Gary T. Leavens
Computer Science, Iowa State Univ.
229 Atanasoff Hall
Ames, IA 50011-1040 USA
+1-515-294-1580
leavens@cs.iastate.edu

Dimitra Giannakopoulou
NASA Ames Research Center
M/S 269-2
Moffett Field, CA 94035-1000 USA
+1-650-604-0504
dimitra@ptolemy.arc.nasa.gov

Murali Sitaraman
Computer Science, Clemson Univ.
419 Edwards Hall
Clemson, SC 29634 USA
+1-864-656-6738
murali@cs.clemson.edu

ABSTRACT
The associated poster summarizes results from the OOPSLA 2001 workshop on Specification and Verification of Component-Based Systems. The workshop's goal is to explore foundations for applying formal methods to component-based systems. The results and future work sections of the poster will be prepared during the workshop.

Keywords
Specification, verification, component-based systems.

1. INTRODUCTION
The goal of this workshop is to explore how formal (i.e., mathematical) techniques can be or should be used to establish a suitable foundation for specification and verification of component-based systems. Component-based systems [1][4][7] are a growing concern for the object-oriented community. Specification and reasoning techniques are urgently needed to permit composition of systems from components, for which source code is unavailable. The workshop will consider formalization of both functional and performance behavior.

The workshop hopes to bring together both researchers and practitioners in the areas of component-based software and formal methods, with the aim of addressing specification and verification problems. We are also interested in bridging the gap between principles and practice. The idea is to focus more of the effort in formal methods on component-based systems. Besides issues important in object-oriented specification and verification, other issues are also important in the practice of component-based systems, such as concurrency, mechanization and scalability, performance (time and space), reusability, and understandability. The participants will brainstorm about these and related topics to understand both the problems involved and how formal techniques may be useful in solving them. The main expected result of the meeting would be an outline of collaborative research topics and a list of areas for further exploration.

1.1 Why do Components Need Formal Methods?
To be able to use and reuse components reliably, one must be able to predict what they will do when used together in a particular context. Furthermore, one must have some degree of confidence in their correctness and performance. Fundamentally, one must have some kind of specification for the components in order to even predict their behavior and even to ask whether they are correct. The formality of the specification is not an end in itself, rather it prevents ambiguity and also allows tools to use the specification without having to: (a) understand natural language and (b) understand enough domain knowledge to resolve ambiguities (which will still not be possible in all cases).

Formal specifications lend themselves to processing by a variety of tools. For example, type checkers can check for basic kinds of consistency. Static analysis tools and model checkers can check for various kinds of deeper semantic errors. Run-time checking of specifications can help isolate errors. Specifications can also be used to help generate test inputs. At the extreme, formal verification systems can attempt to prove correctness of implementations for all legal inputs. The kinds of tools applied depend on how critical the software is; for example, software for airplane control or for controlling nuclear reactors may undergo more scrutiny than that for a video game.

Formal verification, i.e., proving the correctness of components and their compositions can be expensive. For this reason, the term in this context will be used in a broader sense, to include all the different kinds of analysis approaches and tools described above (such as type checkers or model checking). Formal verification, in this sense, is an interesting topic of discussion for the workshop for the following reasons. First, the capability to perform analysis on a system (in particular with the help of automated tools) leverages significantly the effort of formalizing specifications. Second, formal verification serves as a surrogate for understanding how a person or a tool might reason about components and their compositions. That is, one can use formal verification as a way of testing the soundness of various techniques for reasoning, even if those techniques are most often applied informally as “rules of thumb.”

The importance of using formal methods to gain understanding, even if they are not applied directly in practice can be seen by analogy to engineering disciplines. Bridge building and other structural engineering projects were successfully carried out for thousands of years before formal methods, such as Newtonian mechanics and calculus, were developed to the point of useful applicability. However, as bridge projects became more and more complex and ambitious, spectacular failures became more common; this was particularly true in the 19th and early 20th centuries [5]. Formal models developed in materials science and mechanical engineering played important roles in moving the

engineering of large structures out of the intuitive, "rule of thumb" world and into a more rigorous realm that made more ambitious projects possible. This analogy suggests that component-based software engineering should also develop formal methods, not necessarily for daily use, but in building tools that can be used daily. This seems especially true in the areas of specification, standardization, and reliability [6], all areas where solid theoretical foundations are important contributors to success.

Given the rationale above, why have even tools based on formal methods found at best a limited application in practical component-based technologies? Non-technical reasons contribute to this situation, including industrial reluctance to use formal notations, the need for training, and associated costs [2]. But other reasons are technological. Part of the problem is the lack of adequate tools, or tools that are powerful enough to justify their cost. Here, lack of formality may be part of the problem; for example, the lack of a precise formal semantics for some parts of the UML hinders tools that try to use UML specifications. Another part of the problem is the lack of expressiveness and maturity of formal specification languages. Furthermore, many analysis techniques are computationally expensive and may not be able to deal with systems of realistic size. A topic for discussion at the workshop is ways of increasing the scalability of specification and verification techniques by making them more compositional.

1.2 What's Different about Components?

Components pose some problems for formal methods that are different, at least in degree, from object-oriented software. In particular, one must have a compositional (i.e., modular) way of reasoning about system compositions, because the source code for components is not available [7]. *Compositional reasoning* is reasoning about the (functionality and performance) behavior of a system using the (functionality and performance) specifications of the components of the system, without a need to examine or otherwise analyze the implementations of those components. Of course, compositional reasoning is desirable for object-oriented software in general, but it is a necessity for component-based systems.

Components also tend to make more demands on a formal specification and verification system. One reason for this is that they are often generic; instead of a dictionary that works with keys that are a particular kind of string, a component often needs to work with any type having a "hash" method. Components often involve callbacks, which contribute to difficulties in both specification and verification [7].

Compositional reasoning is also a problem for performance (e.g., time and space). Again, this is because in building a system from components one does not have the ability to change the underlying components. Hence, to have good performance for the composed system, it becomes essential to have a way to reason about the performance used by the components. To be compositional, such reasoning must be based on the specifications of the components, something that is traditionally ignored in functional specifications.

Similarly, reasoning about concurrency properties, such as absence of deadlock, becomes more difficult with components, since aspects of the concurrent behavior of the component must be specified in order to allow compositional reasoning.

2. RELATED WORK

There have been several other workshops on component-based systems, both at OOPSLA and ECOOP. However, these workshops tend not to address formal methods. Leavens and Sitaraman previously organized a workshop at ESEC/FSE 1997 [3], which led to the publication of an edited volume [4]. Many of the papers in that volume address the concerns of the workshop. There was also a workshop addressing related concerns at ICSE this year [1]. However, much remains to be done in this area.

3. GETTING MORE INFORMATION

The workshop's web site is as follows.

http://www.cs.iastate.edu/~leavens/SAVCBS/index.html

This web site will be maintained after the workshop and can be visited for links to the participants and their papers. We hope to organize a special issue of a journal that will invite revised and expanded versions of the workshop papers.

4. ACKNOWLEDGMENTS

Leavens was supported in part by NSF grant CCR-009790. Sitaraman was supported in part by DARPA project DAAH04-96-1-0419, monitored by the U. S. Army Research Office. Leavens and Sitaraman were also supported by NSF grant CCR-0113181.

5. REFERENCES

[1] Crnkovic, I., H. Schmidt, J. Staffor, and K. Wallnau. *Proceedings of the 4th ICSE Workshop on Component-Based Software Engineering: Component Certification and System Prediction.* IEEE Comp. Soc., 2001.

[2] Hinchey, M. G., and J. P. Bowen. Applications of Formal Methods FAQ. In M. G. Hinchey and J. P. Bowen (eds.), *Application of Formal Methods*, chapter 1, pp. 1-15. Prentice-Hall, 1995.

[3] Leavens, G. T., and M. Sitaraman (eds.). *Foundations of Component-Based Systems Workshop, Proceedings*, 1997. http://www.cs.iastate.edu/~leavens/FoCBS/index.html

[4] Leavens, G. T., and M. Sitaraman (eds.). *Foundations of Component-Based Systems*. Cambridge U. Press, 2000.

[5] Petroski, H. *Design Paradigms: Case Histories of Error and Judgment in Engineering*, Cambridge U. Press, 1994.

[6] Spector, A., and D. Gifford. A Computer Science Perspective on Bridge Design, *CACM,* **29**:268-283, 1986.

[7] Szyperski, C. *Component Software: Beyond Object-Oriented Programming*. ACM Press and Addison-Wesley, NY, 1998.