Verification Support for Plug-and-Play Architectural Design

Shangzhu Wang, George S. Avrunin, Lori A. Clarke
Department of Computer Science
University of Massachusetts Amherst
Amherst, MA 01003
{shangzhu,avrunin,clarke}@cs.umass.edu

1. INTRODUCTION

In software architecture, connectors are intended to represent the specific semantics of how components interact with each other, capturing some of the most important yet subtle aspects of a system. In practice, choosing the appropriate interaction semantics for the connectors in a system tends to be very difficult. The typical design process often involves not only a choice from commonly used interaction mechanisms, such as remote procedure call, message passing, and publish/subscribe, but also decisions about such details as the particular type and size of a message buffer or whether a communication should be synchronous or asynchronous. Given such a large design space, it is important that designers be able to get feedback about the appropriateness of their design decisions on interaction semantics, based on the correctness of the overall system behavior. In particular, one would like to be able to propose a design, and then use design-time verification to determine whether important properties of the system are satisfied. This practice may repeat until a desired design of the system is achieved.

One major obstacle to the realization of this vision of design and design-time verification is that the semantics of the interactions are often intertwined with the semantics of the components' computations. For example, a change from an asynchronous communication to a synchronous one often requires making changes to the components so that a callback can be placed to explicitly notify the sender of the receipt of messages. Experimenting with alternative choices of interaction semantics tends to be difficult and inefficient when changes made in the interactions require nontrivial changes in the components' computations. This problem also complicates design-time verification. When using finite-state verification techniques, for instance, it is necessary to build a model of the system that represents the computation of each component and the interactions between them. With the semantics of interactions intertwined with the semantics of computations, changes made to the interactions will often result in not only the re-construction of the connector models but also the component models. When repeated changes and verification of a design are necessary, the lack of reusability of the component and connector models could increase the cost of the design-time verification significantly.

Our approach defines a small set of standard interfaces that components can use to communicate with each other through different connectors. This set of standard interfaces allows connectors to be modified or replaced without causing significant changes in the components. To support the standard interfaces, we decompose connectors into *ports* and *channels* that capture different aspects of the interaction semantics represented by connectors. Ports are mediators between components and channels and are responsible for hiding the semantic differences between connectors from the components, normally capturing such semantics as synchronous/asynchronous and blocking/nonblocking. Channels are used to represent the other aspects of interaction semantics represented by a connector such as the communication media.

The decomposition of connectors into ports and channels not only makes it possible to support the standard component interfaces, but also facilitates providing a library of reusable building blocks from which a wide variety of connectors can be constructed. With our approach, constructing a connector with specific semantics is a matter of combining a subset of the building blocks from the library. Changes can be made to a connector by selecting a new subset of building blocks for the connector. With the standard component interfaces, such changes in the connectors often require no or few changes in the components.

Our approach uses finite-state verification to provide designers with feedback about the correctness of the overall system design while they experiment with alternative design choices. This plug-and-play style of design facilitates verification in a number of ways. First, since changes in the connectors usually do not require changes in the components, component models often do not have to be reconstructed when verification needs to be re-applied because of the changes. In addition, pre-defined formal models can be constructed for the library of building blocks. These models can be reused in the model of any system that uses these building blocks. Therefore, our approach can create significant savings in model-construction time during design-time verification.

2. PLUG-AND-PLAY MESSAGE PASSING

To give a basic idea of our approach, we briefly describe how it can be realized for message passing, one of the most commonly used interaction mechanisms in distributed sys-

ROSATEA'06, July 17, 2006, Portland, Maine, USA.

tems. Based on a study of widely used message passing semantics (e.g., MPI), we have defined a library of reusable building blocks for the construction of a variety of message passing connectors. Such building blocks include different kinds of *send ports*, *receive ports*, and *channels*. Channels are essentially message buffers and ports are mediators between components and channels that capture such semantics as whether a message should be sent synchronously or asynchronously, whether a component should block when the message buffer is full while sending a message, or when the desired message is not available while receiving a message.

We define the standard component interfaces for sending and receiving messages in terms of the protocols between the sending component and the send port and between the receiving component and the receive port, respectively. The interfaces are designed to work with different kinds of ports so that ports can be replaced without changing the components. For sending a message, the component first forwards the message to an appropriate send port and then waits for a *SendStatus* message back from the port. It is the send port that determines when the *SendStatus* message is sent and what the content of the status message is, resulting in different message sending semantics. For example, an asynchronous nonblocking send port would return an *OK* status message to the sending component immediately after it receives a message from the component, while a synchronous blocking send port would return the *OK* status message after the message has been delivered to a receiving component by the channel. With such protocols, changing between different message sending semantics is a simple matter of replacing the send port being used. The standard receive interface is defined in a similar way to work with different kinds of receive ports. A fuller description of our approach, including details of the message passing building blocks and the protocols between components and building blocks, is given in [1].

3. DESIGN-TIME VERIFICATION

For an initial evaluation of our approach, we use SPIN as our design-time verification tool and the overall system design is modeled in Promela, the input language of SPIN.

In our approach, a system model is a composition of all the Promela models for the building blocks and the components in the system. Specifically, models for ports and channels are pre-defined as communicating Promela processes and can be simply included in the system model at verification time. The building block models are defined in a parameterizable and reusable way so that they can be easily instantiated and used in any system that uses these building blocks. For our current approach, component models that implement the standard interfaces are manually constructed as separate Promela processes. In principle, however, we expect that such component models can be automatically extracted from their designs in some suitable language. The native Promela channels are used to model the internal communications between components and ports and between ports and channels. To allow the component models to be composed properly with the building block models, appropriate Promela channels are used to set up the connections between component processes and building block processes at the start of a Promela system.

With this approach, when the semantics of a connector are changed and the system design needs to be re-verified, formal models of the system can be modified by replacing the Promela processes of the existing building blocks of the connector with those of the new ones. By employing predefined models for the connectors and reusing the models of the components that now stay relatively stable when only interactions are changed, we reduce the cost of repeated verification in the iterative design process, and therefore make it easier and more efficient to experiment with alternative design choices for the interaction mechanisms. The full set of Promela models for our building blocks is available at `http://cs.umass.edu/~shangzhu/`.

Note that our approach is not tied to any particular model checker or modeling language. By using Promela and SPIN, we are only showing one possible way of modeling our building blocks and applying design-time verification. In fact, we have defined the same set of building blocks in the process algebra FSP and used LTSA (the Labeled Transition System Analyzer) to verify the system designs. Somewhat different strategies may, of course, be appropriate when modeling the building blocks in a different modeling language.

4. CONCLUSION AND FUTURE WORK

In this paper, we describe an approach that allows designers to easily experiment with alternative design choices of interaction semantics through the use of standard component interfaces and a library of building blocks for constructing different connectors. Our approach facilitates design-time verification by saving on repeated model construction time as design evolves and verification is re-applied.

Our ongoing work includes the implementation of a tool that supports this plug-and-play design and verification approach and the extension of the current approach to support other kinds of interaction mechanisms such as publish/subscribe and remote procedure calls. We are also conducting some nontrivial case studies to evaluate our approach. Since our current models for the library of building blocks are only intended for proof of concept and may not be the most efficient, one important direction for future work is to study what kind of techniques may be applied to simplify and optimize the models created using our plug-and-play approach so that finite-state verification can be applied efficiently.

5. ACKNOWLEDGMENTS

We are grateful to Prashant Shenoy for helpful conversations about this work. This material is based upon work supported by the National Science Foundation under awards CCF-0541035, CCF-0427071, and CCR-0205575 and by the U.S. Department of Defense/Army Research Office under award DAA-D19-01-1-0564 and award DAAD19-03-1-0133. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the National Science Foundation or the U. S. Department of Defense/Army Research Office.

6. REFERENCES

[1] S. Wang, G. S. Avrunin, and L. A. Clarke. Architectural building blocks for plug-and-play system design. In *Proc. 9th Intl. SIGSOFT Symp. on Component-Based Software Engineering*, Västerås, Sweden, June 2006. To appear.