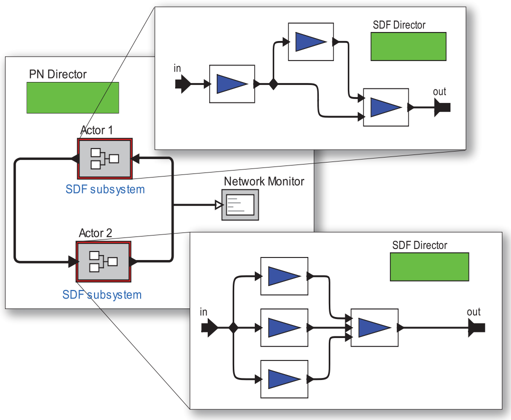
An Extensible Software Synthesis Framework for Heterogeneous Actor Models

Man-Kit Leung, Edward A. Lee

EECS
UC Berkeley
Berkeley, USA

A model of computation (MoC or domain) is a modeling abstraction of how components interact. Researchers have proposed a variety of MoCs to describe different kinds of interaction semantics. For example, some MoCs may use deterministic data streams to express communication between components while some MoCs may govern the interaction through real-time or some abstraction of time. A specialized MoC imposes stricter constraints on the model and, thus, provides more analyzability. However, specialized MoCs sometimes may not be expressive enough for complex interaction, so one has to revert to using a more generic (and complex) MoC, which comes at the cost of analyzability. *Heterogeneous modeling* is an innovative technique that proposes composition of multiple MoCs. It attempts to maximize both analyzability and expressiveness by providing finer granularity in the use of MoCs. The Globally-Asynchronous, Locally-Synchronous (GALS) efforts are one example of mixing MoCs; interacting finite state machines (FSM) is another class of heterogeneous models; they combine FSMs with other concurrent MoCs.



Let's take the model on the right as an example. The top-level is modeled as a Kahn Process Network (PN) [4] that contains two hierarchical components (Actor1 and Actor2) which in turn contain refinement models. The PN MoC specifies Actor1 and Actor2 to execute in separate threads, while the refinement models are directed by the Synchronous Dataflow (SDF) MoC. The heterogeneity of MoCs is accomplished through hierarchical composition. By composing PN and SDF, we effectively control the degree of con-

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: `www.elsevier.nl/locate/entcs`

currency while retaining SDF's determinacy and understandability in the subcomponents. In order to program efficiently using heterogeneous modeling, we build an extensible code generation framework which contributes a new interface for synthesizing and composing domain-specific code.

Actor-oriented design[1] is good for its decoupling of model composition and actor definition. It uses polymorphic elements to maximize component reuse. However, code generation requires extra effort to specialize these elements at compile time. We specifically define three forms of polymorphism that require compile-time specializations: type, actor, and domain. Polymorphism traditionally refers to *type polymorphism* which allows multiple data types to be compatible under one common type; *actor polymorphism* refers to the parameterizations that cause an actor to perform different functions; *domain polymorphism* is the compatibility of an actor under different MoCs. Each form of polymorphism presents an orthogonal dimension of specialization.

Given the complexity of software synthesis for actor models, extensibility is the big challenge in supporting code generation for heterogeneous models. While previous code generation research[2,5] mainly focused on a particular domain, we extend the software synthesis infrastructure to support multiple domains, and to support the use of multiple domains in the same model. We have developed a new interface for synthesizing and composing domain-specific code, keeping it distinctly separate from the domain-independent infrastructure. We then map each MoC to this code generation interface. We build on earlier techniques that demonstrated heterogeneous code generation for combinations of SDF with FSM MoCs [6].

The use of an interface systematically minimizes the size of the code generator by maximizing the domain-independent infrastructure. Moreover, a uniform interface allows us to easily compose the code generation for multiple MoCs. It supports multiple implementations, enabling us to incrementally add MoCs. We can isolate complexity and, at the same time, enjoy the reusability of the common infrastructure. We have implemented code generation for the Synchronous Dataflow (SDF), Heterochronous Dataflow (HDF), Finite State Machine (FSM), and Process Network (PN) domains. The code generator supports any hierarchical composition of these MoCs, though some combinations have little utility [3].

References

[1] E. A. Lee, S. Neuendorffer, and M. J. Wirthlin. Actor-oriented design of embedded hardware and software systems. Journal of Circuits, Systems, and Computers, scheduled for publication June 2003.

[2] S. Neuendorffer. Automatic Specialization of Actor-Oriented Models in Ptolemy II, Master's Report, Technical Memorandum UCB/ERL M02/41, University of California, Berkeley, CA 94720, December 25, 2002.

[3] A. Goderis, C. Brooks, I. Altintas, E. A. Lee, and C. Goble. Heterogeneous composition of models of computation. Technical Report UCB/EECS-2007-139, EECS Department, University of California, Berkeley, November 2007.

[4] G. Kahn. The semantics of a simple language for parallel programming. In J. L. Rosenfeld, editor, *Information processing*, pages 471–475, Stockholm, Sweden, Aug 1974. North Holland, Amsterdam.

[5] J. Tsay, C. Hylands, and E. A. Lee. A code generation framework for java component-based designs. In *CASES '00, November 17-19, 2000, San Jose, CA*, November 2000.

[6] G. Zhou, M. K. Leung, and E. A. Lee. A code generation framework for actor-oriented models with partial evaluation. In Y.-H. L. et al., editor, *Proceedings of International Conference on Embedded Software and Systems 2007, LNCS 4523*, pages 786–799, May 2007.