# Composition Languages

*James Adam Cataldo*
*Edward A. Lee*

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2006-24
http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-24.html

March 17, 2006

Acknowledgement

This work was supported in part by the Center for Hybrid and Embedded Software Systems (CHESS) at UC Berkeley, which receives support from the National Science Foundation (NSF award #CCR-0225610), the State of California Micro Program, and the following companies: Agilent, DGIST, General Motors, Hewlett Packard, Infineon, Microsoft, and Toyota.

# Composition Languages

Adam Cataldo and Edward Lee

March 15, 2006

*Actor-oriented design* [1] is a rapidly evolving paradigm in the design of complex systems, where concurrent components (called *actors*) communicate with one another through ports. This paradigm is being developed in the form of programming languages, such as architecture description languages [2] and coordination languages[1] [3], as well in the form of software tools such as Simulink [4] and Ptolemy II [5]. To support large systems, actor-oriented design encourages *hierarchy*; a network of components may be bundled together to form a new component. The resulting systems are typically easier to reason about than those programmed with threads [6], since concurrent communication is much more explicit. We thus believe that actor-oriented design will become increasingly relevant as large system design becomes increasingly common.

For large actor-oriented systems, it may be appropriate to use different models of computation for component interaction at different levels of a model's hierarchy. Such systems are called *heterogeneous* or *multi-paradigm* [7]. For these, *metamodeling* [8] tools, such as GME [9], are making it easier to create domain specific modeling environments, while model-based design tools, such as Ptolemy II, are making it easier to construct models with heterogeneous behavior.

These approaches make it simpler to design systems with complex *semantics*. As actor-oriented systems become larger and larger, however, new techniques for simple *syntactic* descriptions of systems, whether visual or textual, will become equally important. As an example, imagine designing a distributed system with $10,000$ components. After the system is built, a new customer wants the same system, but this time with $20,000$ components. It seems unreasonable to have to respecify all the connections in the new system. If the hierarchy in the original model is relatively flat, this could be particularly challenging, as it could require copying and pasting thousands of components and connections.

We thus propose *composition languages* as a way to specify actor-oriented models. The key to composition languages is the ability to succinctly specify *higher-order models*. As an example, a higher-order model may be a distributed sort model. The model may be parameterized by a *divide* component (or model), a *conquer* component, and the respective numbers of divide and conquer components. A programmer will specify this higher-order model once and can then use it for an arbitrary number of components with arbitrary divide and conquer components. This particular example is similar to the MapReduce programming environment used by Google for distributed computation [10]. We seek to take this a step further by providing a generic language for describing higher-order models. We believe composition languages will become increasingly important in actor-oriented design, since they will enable rapid development of large systems.

[1] Coordination languages may be either agent based (data driven) or rendezvous based (control driven).

## References


[1] E. A. Lee, "Model-driven development - from object-oriented design to actor-oriented design," in *Workshop on Software Engineering for Embedded Systems: From Requirements to Implementation (a.k.a. The Monterey Workshop)*, (Chicago), 2003.

[2] N. Medvidovic and R. N. Taylor, "A classification and comparison framework for software architecture description languages," *IEEE Transactions on Software Engineering*, vol. 26, pp. 70–93, January 2000.

[3] G. A. Papadopoulos and F. Arbab, *Advances in Computers*, vol. 46, ch. Configuration Models and Languages, pp. 329–400. Academic Press, 1998.

[4] J. B. Dabney and T. L. Harman, *Mastering SIMULINK*. Prentice Hall Professional Technical Reference, 2003.

[5] C. Brooks, E. A. Lee, X. Liu, S. Neuendorffer, Y. Zhao, and H. Zheng, "Heterogeneous concurrent modeling and design in Java," Tech. Rep. UCB/ERL M05/21, EECS, University of California, Berkeley, 2005.

[6] E. A. Lee, "The problem with threads," Tech. Rep. UCB/EECS-2006-1, EECS Department, University of California, Berkeley, January 10 2006.

[7] H. L. Vangheluwe, J. de Lara, and P. J. Mosterman., "An introduction to multi-paradigm modelling and simulation," in *Proceedings of the AIS 2002 Conference (AI, Simulation and Planning in High Autonomy Systems)* (F. Barros and N. Giambiasi, eds.), (Lisboa, Portugal), pp. 9–20, 2002.

[8] G. G. Nordstrom, *Metamodeling - Rapid Design and Evolution of Domain-Specific Modeling Environments*. Electrical engineering and computer science, Vanderbilt University, 1999.

[9] A. Ledeczi, M. Maroti, A. Bakay, G. Karsai, J. Garrett, C. Y. T. IV, G. G. Nordstrom, and P. Volgyesi, "The generic modeling environment," in *Workshop on Intelligent Signal Processing*, (Budapest, Hungary), May 2001.

[10] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in *Sixth Symposium on Operating System Design and Implementation (OSDI)*, (San Francisco, CA), 2004.