Proceedings 2011 Design Automation and Test Europe (DATE), March, 2011.

Component-Based Design for the Future

Edward A. Lee[¶] and Alberto L. Sangiovanni-Vincentelli[‖]
[¶]University of California at Berkeley [‖]University of California at Berkeley and University of Trento

Motivation. The specific root causes of the design problems that are haunting system companies such as automotive and avionics companies are complex and relate to a number of issues ranging from design processes and relationships with different departments of the same company and with suppliers[1] to incomplete requirement specification and testing.[2]

Further, there is a widespread consensus in the industry that there is much to gain by optimizing the implementation phase that today is only considering a very small subset of the design space. Some attempts at a more efficient design space exploration have been afoot but there is a need to formalize the problem better and to involve in major ways the different players of the supply chain. Information about the capabilities of the subsystems in terms of timing, power consumed, size, weight and other physical aspects transmitted to the system assemblers during design time would go a long way in providing a better opportunity to design space exploration. In this landscape, a wrong turn in a system design project could cause so much economic, social and organizational upheaval that it may imperil the life of an entire company. No wonder that there is much interest in risk management approaches to assess risks associated to design errors, delays, recalls and liabilities. Finding appropriate countermeasures to lower risks and to develop contingency plans is then a mainstay of the way large projects are managed today. The overarching issue is the need of a substantive evolution of the design methodology in use today in system companies. The issue to address is the understanding of the principles of system design, the necessary change to design methodologies, and the dynamics of the supply chain. Developing this understanding is necessary to define a sound approach to the needs of the system companies as they try to serve their customers better, to develop their products faster and with higher quality.

An important approach to tackle in part these issues is component-based design.

Component-based Design. Whereas layered designs decompose complexity of systems "vertically", by splitting the design into multiple design layers, component-based approaches reduce complexity "horizontally" whereby designs are obtained by assembling strongly encapsulated design entities called "components" equipped with concise and rigorous interface specifications. Re-use can be maximized by finding the weakest assumptions on the environment sufficient to establish the guarantees on a given component implementation.

One challenge, then, for component-based design of embedded systems, is to provide interface specifications that are rich enough to cover all phases of the design cycle. This calls for including non-functional characteristics as part of the component interface specifications. Current component interface models, in contrast, are typically restricted to purely functional characterization of components, and thus cannot capitalize on the benefits of contract-based virtual integration testing.

The second challenge is related to product line design, which allows for the joint design of a family of variants of a product. The aim is to balance the contradicting goals of striving for generality versus achieving efficient component implementations.

Platform-based design [52] has been formulated to help to achieve this balance and to support component-based design. To do so, the design is seen as a meet-in-the-middle approach where the bottom-up part is related to the characterization and re-use of components. The 'rules' for composing components are an essential part of the methodology.

In all cases, a mathematically rigorous form (language) for expressing heterogeneous components and their interfaces together with their non functional characteristics is necessary.

Present State of Design Methodologies and Languages. Despite considerable progress in languages, notations, and tools for model-based design [57] and model-driven development [56], major problems persist. In practice, system integration, adaptation of existing designs, and inter-operation of heterogeneous subsystems remain major stumbling blocks that cause project failures. We believe that model-based design, as widely practiced today, largely fails to benefit from the principles of component-based design and platform-based design [52] as a consequence of its lack of attention to the semantics of heterogeneous subsystem composition.

Consider for example a widely-used software system description language such as UML 2 [4], [5], or more directly its derivative SysML [47]. The internal block diagram notation of SysML, which is based on the UML 2 composite structure diagrams, particularly with the use of flow ports, has severe weaknesses to address system design problems. The SysML standard defines the syntax of these diagrams, not their semantics. Although the SysML standard asserts that "flow ports are intended to be used for asynchronous, broadcast, or

This work was funded in part by the European STREP-COMBEST project number 215543, by the Artist Design NoC and by the MuSyC MARCO FCRP Center.

[1]Toyota sticky accelerator problem came in part from components provided by two contractors whose interaction was not verified appropriately, Airbus delay problems were in part due to contractors who had different versions of the CAD software

[2]Boeing stated that a structural problem was discovered late in the design process

send-and-forget interactions" [47], the standard fails to define the semantics of such interactions. Implementers of tools are free to interpret this intent, resulting in a modeling language whose semantics is defined by the tools rather than by the language itself. There are many semantic alternatives [30], consequently the same SysML diagram may be interpreted very differently by different observers. MARTE (Modeling and Analysis of Real-Time and Embedded systems) [46] also specifically avoids "constraining" (or even defining) the execution semantics of models. Instead it focuses on providing alternative ways of expressing today's ad-hoc, non-composable design practices such as concurrency based on threads [28]. Standardizing *notation* is not sufficient to achieve effective analysis methods and unambiguous communication among designers. More importantly, without semantics, the modeling framework fails to provide a *platform* for design. Further, the very flexibility of these modeling notations may account for some of their success, because designers can become "standards compliant" without changing their existing practice. They merely have to adapt their notation. Moreover, the notations can be freely reinterpreted by defining a "profile," greatly weakening the value of the notation as an effective communication vehicle and design tool. We believe that *constraints that lead to well-defined and inter-operable models have potentially far greater value*. More importantly, such constraints are essential for these modeling frameworks to become a central part of a platform-based engineering practice [53].

The inclusion by OMG of Statecharts [20] in the UML standard has helped to narrow the variability, but in many cases, the exact semantics are determined by the implementation details of the supporting tools rather than by an agreed-upon standard semantics. In fact, Statecharts also suffers from inadequate standardization. Despite their common origin, variants have proliferated [59]. Even the most widely used implementations of Statecharts that claim to be standards-compliant have subtle semantic differences big enough "that a model written in one formalism could be ill-formed in another formalism" [12]. In many implementations, including the widely used RHAPSODY tool from IBM, the semantics is (probably inadvertently) nondeterminate [55].

Over the last 20 years, we at Berkeley have been developing model-based design techniques with sufficiently well-defined semantics to provide an effective basis for platform-based design and engineering. Components can be designed to operate with a model, and when deployed, will operate in predictable ways with the deployed system. The rigorous foundations of the models [34] provide a solid basis for integration across design domains, design adaptation and evolution, and analysis and verification. Our work has been demonstrated in the open-source software frameworks Ptolemy Classic [8], Ptolemy II [16], Polis [1], Metropolis [2] and MetroII [13]. Many of the techniques that we developed have been deployed in a wide range of domain-specific applications, including hardware and FPGA synthesis, signal processing, automotive system design, computer architecture design and evaluation, instrumentation, wireless system design, mixed signal circuit design, network simulation and design, building system simulation and design, financial engineering, and scientific workflows. We believe these approaches can be successfully applied to defense system design. The goal of this extended abstract is to highlight the key applicable ideas.

Models of Computation. At Berkeley, we have established that models should be built using well-defined models of computation (MoCs) [14]. An MoC gives semantics to concurrency in the model, defining for example whether components in the model execute simultaneously, whether they share a notion of time, and whether and how they share state. An MoC also defines the communication semantics, specifying for example whether data is exchanged for example using publish-and-subscribe protocols, synchronous or asynchronous message transmission, or time-stamped events. We have provided a formal framework within which the semantics of a variety of models of computation can be understood and compared, and within which heterogeneous interactions across models of computation can be defined [34]. This formal foundation has been elaborated and applied to multi-clock (latency insensitive) systems [10], globally asynchronous, locally synchronous (GALS) designs [3], and to timed models of computation capable of reflecting real-time dynamics [41].

Abstract Semantics. In many situations, using a single general MoC for an entire design requires giving up any possibility of property checking except by extensive simulation. More restricted (less expressive) MoCs yield better to analysis, enabling systematic exploration of properties of the design, often including formal verification. But less expressive MoCs cannot capture the richness and diversity of complex designs. The solution is heterogeneous mixtures of MoCs. Indeed, the heterogeneous nature of most defense systems makes multiple MoCs a necessity.

In addition, during the design process, the abstraction level, detail, and specificity in different parts of the design vary. The skill sets and design styles that engineers use on the project are likely to differ. The net result is that, during the design process, many different specification and modeling techniques will be used. The challenge is how to combine heterogeneous MoCs and determine what the composition's behavior is. Unfortunately, the semantics of different MoCs are typically incompatible.

A way to solve this problem is to embed the detailed models into a framework that can understand the models being composed. A theoretical approach to this view, which is well beyond the scope of this article, can be found in [9] who used an abstract algebra approach to define the interactions among incompatible models. In some sense, we are looking at an abstraction of the MoC concept that can be refined into any of the MoCs of interest. We call this abstraction an *abstract semantics*, first introduced in [33], [26]. The inspiration on how to define the abstract semantics comes from the consideration that MoCs are built by combining three largely orthogonal aspects: sequential behavior, concurrency,

and communication. Similar to the way that a MoC abstracts a class of behavior, abstract semantics abstract the semantics The concept is called a "semantics meta-model" in [54], but since the term "meta-model" is more widely used in software engineering to refer instead to models of the structure of models (see [45] and http://www.omg.org/mof/), we prefer to use the term "abstract semantics" here. The concept of abstract semantics is leveraged in Ptolemy II [16], Metropolis [2], and Ptolemy Classic [8] to achieve heterogeneous mixtures of MoCs with well-defined interactions.

Hybrid Systems. Cyber-Physical Systems (CPS) integrate computation, networking, and physical dynamics. As a consequence, modeling techniques that address only the concerns of software are inadequate [27], [29]. Integrations of continuous physical dynamics expressed with ordinary differential equations with the discrete behaviors expressed using finite automata are known as *hybrid systems* [43]. At Berkeley, we have previously done a detailed study and comparison of tools supporting hybrid systems modeling and simulation [11]. Moreover, we have developed a rigorous MoC that provides determinate semantics to such hybrid systems [39], [42], [36]. This work has influenced development of commercial tools such a Simulink and LabVIEW and has been realized in the open-source tool HyVisual [6]. Moreover, we have leveraged the notation of abstract semantics to integrate such hybrid systems with other MoCs such as synchronous/reactive and discrete-event models [37]. This integration enables heterogeneous models that capture the interactions of software and networks with continuous physical processes.

Heterogeneity. Integrating multiple MoCs such that they can inter-operate, which is far from trivial, has been called "multimodeling" [44], [18], [7]. Many previous efforts have focused on tool integration, where tools from multiple vendors are made to inter-operate [40], [19], [22]. This approach is challenging, however, resulting in fragile tool chains. Many tools do not have adequate published extension points, and maintaining such integration requires considerable effort. At Berkeley, our approach has been to focus on the interfaces between MoCs. We have built a variety of modeling, analysis, and simulation tools based on different MoCs [14], and have shown how such interfaces can facilitate more robust inter-operability. These include discrete-event [25] (useful for modeling networks, hardware architecture, and real-time systems), synchronous-reactive [15] (useful for modeling and designing safety-critical concurrent software), dataflow [32] (useful for signal processing), process networks [48] (useful for asynchronous distributed systems), and continuous-time models [37] (useful for physical dynamics). Influenced in part by our work, SystemC, a widely used language in hardware design, is capable of realizing multiple MoCs [50], [21], although less attention in that community has been given to inter-operability.

For nearly all of these MoCs, the emphasis in our design has been on providing determinate behavior (where the same inputs always result in the same outputs, and introducing nondeterminacy only where it is needed by the application (for example to model faults). The result is a family of far better concurrency models than the widely used thread-based models that dominate software engineering [28].

Modularity. Key to effective design of complex systems is modular design, where modules have well-defined interfaces, and composition of modules can be checked for compatibility. We have shown that object-oriented concepts such as classes, inheritance, and polymorphism can be adapted to concurrent, actor-oriented components [31] (see also [23]). We have developed advanced type systems for such component compositions, enabling type inference and type checking across large models with polymorphic components [60]. We have adapted such type systems to capture domain-specific ontology information, checking for correct usage and correct interpretation of shared data [38]. And we have shown how to check for compatibility of protocols in compositions [35] and to synthesize interface adapters for separately defined components [49].

Linking Behavior to Implementation: Quantity Managers. To support evaluation of design choices, modeling frameworks need to enable weaving together a multiplicity of models that cover different aspects of a system. For example, a choice of networking fabric will affect temporal behavior, power usage, and vulnerability to faults. The Metropolis project [2], [13] introduced the notion of "quantity manager," a component of a model that functions as a gateway to another model. For example, a purely functional model that describes only idealized behavioral properties of a flight control system could be endowed with a quantity manager that binds that functional model to a model of a distributed hardware architecture using a particular network fabric. By binding these two models, designers can evaluate how properties of the hardware implementation affect the functional behavior of the system. For example, how does a time-triggered bus protocol affect timing in a distributed control system, and how do the timing effects change the dynamics of the system? Similarly, a functional model could be endowed with a quantity manager that measures power usage and identifies potential overloads that may result from unexpectedly synchronized interactions across a distributed system.

The notion of quantity managers brings to model-based design a capability analogous to aspect-oriented programming in software engineering [24]. Separately designed models can be woven together using quantity managers in a manner similar to the weaving of separately designed classes in aspect-oriented design.

Semantics-Preserving Transformation and Implementation. Effective use of models requires well-defined relationships between the models and systems being modeled. In many cases, models can be used as specifications, and implementations can be synthesized from these specifications. The key challenge is that such synthesis must preserve the semantics of the implementation. We have many years of experience with semantics-preserving code generation [51], [61], [58] and model transformation [17].

Conclusion. Tools and techniques for model-based design are evolving rapidly and show considerable promise for delivering robust, adaptable, platform-based design techniques that include formal ways of dealing with components, abstractions and non functional characteristics.

The range of application of these methods is also evolving to include on one side large, complex distributed systems of systems such as traffic management systems, water management systems and smart grids and on the other biological systems including synthetic biology and hybrid systems that involve real neurons controlling a mechatronic system. The future is bright!

REFERENCES

[1] F. Balarin, M. Chiodo, P. Giusto, H. Hsieh, A. Jurecska, L. Lavagno, C. Passerone, A. Sangiovanni-Vincentelli, E. Sentovich, K. Suzuki, and B. Tabbara. *Hardware-Software Co-Design of Embedded Systems–The Polis Approach*. Kluwer, 1997.

[2] F. Balarin, H. Hsieh, L. Lavagno, C. Passerone, A. L. Sangiovanni-Vincentelli, and Y. Watanabe. Metropolis: an integrated electronic system design environment. *Computer*, 36(4), 2003.

[3] Albert Benveniste, Benoï¿½t Caillaud, Luca P. Carloni, and Alberto Sangiovanni-Vincentelli. Tag machines. In *EMSOFT*, Jersey City, New Jersey, USA, 2005. ACM.

[4] Conrad Bock. SysML and UML 2 support for activity modeling. *Systems Engineering*, 9(2):160 –185, 2006.

[5] G. Booch, I. Jacobson, and J. Rumbaugh. *The Unified Modeling Language User Guide*. Addison-Wesley, 1998.

[6] C. Brooks, A. Cataldo, C. Hylands, E. A. Lee, J. Liu, X. Liu, S. Neuendorffer, and H. Zheng. HyVisual: A hybrid system visual modeler. Technical Report UCB/ERL M05/24, University of California, Berkeley, July 15 2005. Available from: http://ptolemy.eecs.berkeley.edu/publications/papers/05/hyvisual/index.htm.

[7] Christopher Brooks, Chihhong Cheng, Thomas Huining Feng, Edward A. Lee, and Reinhard von Hanxleden. Model engineering using multimodeling. In *International Workshop on Model Co-Evolution and Consistency Management (MCCM)*, Toulouse, France, 2008.

[8] Joseph T. Buck, Soonhoi Ha, Edward A. Lee, and David G. Messerschmitt. Ptolemy: A framework for simulating and prototyping heterogeneous systems. *Int. Journal of Computer Simulation, special issue on "Simulation Software Development"*, 4:155–182, 1994. Available from: http://ptolemy.eecs.berkeley.edu/publications/papers/94/JEurSim/.

[9] J. R. Burch, R. Passerone, and A. L. Sangiovanni-Vincentelli. Overcoming heterophobia: Modeling concurrency in heterogeneous systems. In *International Conference on Application of Concurrency to System Design*, page 13, 2001.

[10] L.P. Carloni, K.L. McMillan, and A.L. Sangiovanni-Vincentelli. The theory of latency insensitive design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 20(3), 2001.

[11] Luca P. Carloni, Roberto Passerone, Alessandro Pinto, and Alberto Sangiovanni-Vincentelli. Languages and tools for hybrid systems design. *Foundations and Trends in Electronic Design Automation*, 1(1/2), 2006. Available from: http://dx.doi.org/10.1561/1000000001.

[12] Michelle L. Crane and Juergen Dingel. Uml vs. classical vs. rhapsody statecharts: Not all models are created equal. In *International Conference on Model Driven Engineering Languages and Systems (MODELS)*, volume LNCS 3713, pages 97–112, Montego Bay, Jamaica, 2005. Springer.

[13] Abhijit Davare, Douglas Densmore, Trevor Meyerowitz, Alessandro Pinto, Alberto Sangiovanni-Vincentelli, Guang Yang, and Qi Zhu. A next-generation design framework for platform-based design. In *Design Verification Conference (DVCon)*, San Jose', California, 2007.

[14] S. Edwards, L. Lavagno, E.A. Lee, and A. Sangiovanni-Vincentelli. Design of embedded systems: formal models, validation, and synthesis. *Proceedings of the IEEE*, 85(3):366–390, 1997.

[15] Stephen A. Edwards and Edward A. Lee. The semantics and execution of a synchronous block-diagram language. *Science of Computer Programming*, 48(1):21–42, 2003. Available from: http://dx.doi.org/10.1016/S0167-6423(02)00096-5.

[16] Johan Eker, Jï¿½rn W. Janneck, Edward A. Lee, Jie Liu, Xiaojun Liu, Jozsef Ludvig, Stephen Neuendorffer, Sonia Sachs, and Yuhong Xiong. Taming heterogeneity—the Ptolemy approach. *Proceedings of the IEEE*, 91(2):127–144, 2003. Available from: http://www.ptolemy.eecs.berkeley.edu/publications/papers/03/TamingHeterogeneity/.

[17] Thomas Huining Feng and Edward A. Lee. Scalable models using model transformation. In *Workshop on Model Based Architecting and Construction of Embedded Systems (ACES-MB)*, 2008. Available from: http://chess.eecs.berkeley.edu/pubs/487.html.

[18] Paul A. Fishwick and Bernard P. Zeigler. A multimodel methodology for qualitative model engineering. *ACM Transactions on Modeling and Computer Simulation*, 2(1):52–81, 1992.

[19] Zonghua Gu, Shige Wang, S Kodase, and K. G. Shin. An end-to-end tool chain for multi-view modeling and analysis of avionics mission computing software. In *Real-Time Systems Symposium (RTSS)*, pages 78 – 81, 2003.

[20] David Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, 1987.

[21] Fernando Herrera and Eugenio Villar. A framework for embedded system specification under different models of computation in SystemC. In *Design Automation Conference (DAC)*, San Francisco, 2006. ACM.

[22] Gabor Karsai, Andras Lang, and Sandeep Neema. Design patterns for open tool integration. *Software and Systems Modeling*, 4(2):157–170, 2005. Available from: http://dx.doi.org/10.1007/s10270-004-0073-y.

[23] Gï¿½bor Karsai, Miklos Maroti, ï¿½kos Lï¿½deczi, Jeff Gray, and Janos Sztipanovits. Type hierarchies and composition in modeling and metamodeling languages. *IEEE Transactions on Control System Technology*, to appear, 2003.

[24] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-oriented programming. In *ECOOP, European Conference in Object-Oriented Programming*, volume LNCS 1241, Finland, 1997. Springer-Verlag.

[25] Edward A. Lee. Modeling concurrent real-time processes using discrete events. *Annals of Software Engineering*, 7:25–45, 1999. Available from: http://dx.doi.org/10.1023/A:1018998524196.

[26] Edward A. Lee. Overview of the Ptolemy project. Technical Report UCB/ERL M03/25, University of California, Berkeley, July 2 2003. Available from: http://ptolemy.eecs.berkeley.edu/publications/papers/03/overview/.

[27] Edward A. Lee. Cyber-physical systems - are computing foundations adequate? In *NSF Workshop On Cyber-Physical Systems: Research Motivation, Techniques and Roadmap*, Austin, TX, 2006. Available from: http://ptolemy.eecs.berkeley.edu/publications/papers/06/CPSPositionPaper/.

[28] Edward A. Lee. The problem with threads. *Computer*, 39(5):33–42, 2006. Available from: http://dx.doi.org/10.1109/MC.2006.180.

[29] Edward A. Lee. Cyber physical systems: Design challenges. In *International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC)*,, pages 363 – 369, Orlando, Florida, 2008. IEEE. Available from: http://dx.doi.org/10.1109/ISORC.2008.25.

[30] Edward A. Lee. Disciplined heterogeneous modeling. In D. C. Petriu, N. Rouquette, and O. Haugen, editors, *Model Driven Engineering, Languages, and Systems (MODELS)*, pages 273–287. IEEE, 2010. Available from: http://chess.eecs.berkeley.edu/pubs/679.html.

[31] Edward A. Lee, Xiaojun Liu, and Stephen Neuendorffer. Classes and inheritance in actor-oriented design. *ACM Transactions on Embedded Computing Systems (TECS)*, 8(4):29:1–29:26, 2009. Available from: http://ptolemy.eecs.berkeley.edu/publications/papers/07/classesandInheritance/index.htm.

[32] Edward A. Lee and Eleftherios Matsikoudis. The semantics of dataflow with firing. In Gï¿½rard Huet, Gordon Plotkin, Jean-Jacques Lévy, and Yves Bertot, editors, *From Semantics to Computer Science: Essays in memory of Gilles Kahn*. Cambridge University Press, 2009. Available from: http://ptolemy.eecs.berkeley.edu/publications/papers/08/DataflowWithFiring/.

[33] Edward A. Lee, Stephen Neuendorffer, and Michael J. Wirthlin. Actor-oriented design of embedded hardware and software systems. *Journal of Circuits, Systems, and Computers*, 12(3):231–260, 2003.

[34] Edward A. Lee and Alberto Sangiovanni-Vincentelli. A framework for comparing models of computation. *IEEE Transactions on Computer-Aided Design of Circuits and Systems*, 17(12):1217–1229, 1998. Available from: http://ptolemy.eecs.berkeley.edu/publications/papers/98/framework/.

[35] Edward A. Lee and Yuhong Xiong. A behavioral type system and its application in Ptolemy II. *Formal Aspects of Computing Journal*, 16(3):210 – 237, 2004.

[36] Edward A. Lee and Haiyang Zheng. Operational semantics of hybrid systems. In Manfred Morari and Lothar Thiele, editors, *Hybrid Systems: Computation and Control (HSCC)*, volume LNCS 3414, pages 25–53, Zurich, Switzerland, 2005. Springer-Verlag. doi:10.1007/978-3-540-31954-2_2.

[37] Edward A. Lee and Haiyang Zheng. Leveraging synchronous language principles for heterogeneous modeling and design of embedded systems. In *EMSOFT*, pages 114 – 123, Salzburg, Austria, 2007. ACM. doi:10.1145/1289927.1289949.

[38] Man-Kit Leung, Thomas Mandl, Edward A. Lee, Elizabeth Latronico, Charles Shelton, Stavros Tripakis, and Ben Lickly. Scalable semantic annotation using lattice-based ontologies. In *International Conference on Model Driven Engineering Languages and Systems (MODELS)*, Denver, CO, USA, 2009. ACM/IEEE.

[39] Jie Liu. Responsible frameworks for heterogeneous modeling and design of embedded systems. Ph.D. Thesis Technical Memorandum UCB/ERL M01/41, University of California, Berkeley, December 20 2001. Available from: http://ptolemy.eecs.berkeley.edu/publications/papers/01/responsibleFrameworks/.

[40] Jie Liu, Bicheng Wu, Xiaojun Liu, and Edward A. Lee. Interoperation of heterogeneous cad tools in Ptolemy II. In *Symposium on Design, Test, and Microfabrication of MEMS/MOEMS*, Paris, France, 1999.

[41] Xiaojun Liu and Edward A. Lee. CPO semantics of timed interactive actor networks. *Theoretical Computer Science*, 409(1):110–125, 2008. doi:10.1016/j.tcs.2008.08.044.

[42] Xiaojun Liu, Jie Liu, Johan Eker, and Edward A. Lee. Heterogeneous modeling and design of control systems. In Tariq Samad and Gary Balas, editors, *Software-Enabled Control: Information Technology for Dynamical Systems*. Wiley-IEEE Press, 2003.

[43] Oded Maler, Zohar Manna, and Amir Pnueli. From timed to hybrid systems. In *Real-Time: Theory and Practice, REX Workshop*, pages 447–484. Springer-Verlag, 1992.

[44] Pieter J. Mosterman and Hans Vangheluwe. Computer automated multiparadigm modeling: An introduction. *Simulation: Transactions of the Society for Modeling and Simulation International Journal of High Performance Computing Applications*, 80(9):433ï¿½450, 2004.

[45] G. Nordstrom, Janos Sztipanovits, G. Karsai, and A. Ledeczi. Metamodeling - rapid design and evolution of domain-specific modeling environments. In *Proc. of Conf. on Engineering of Computer Based Systems (ECBS)*, pages 68–74, Nashville, Tennessee, 1999.

[46] Object Management Group (OMG), . A UML profile for MARTE, beta 2. OMG Adopted Specification ptc/08-06-09, OMG, August 2008. Available from: http://www.omg.org/omgmarte/.

[47] Object Management Group (OMG), . System modeling language specification v1.2. Standard specification, OMG, June 2010. Available from: http://www.sysmlforum.com.

[48] Thomas M. Parks and David Roberts. Distributed process networks in Java. In *International Parallel and Distributed Processing Symposium*, Nice, France, 2003.

[49] Roberto Passerone, Luca de Alfaro, Thomas A. Henzinger, and Alberto Sangiovanni-Vincentelli. Convertibility verification and converter synthesis: Two faces of the same coin. In *Proceedings of International Conference on Computer Aided Design*, San Jose, CA., 2002.

[50] H. D. Patel and S. K. Shukla. *SystemC Kernel Extensions for Heterogeneous System Modelling*. Kluwer, 2004.

[51] Josï¿½ L. Pino, Soonhoi Ha, Edward A. Lee, and Joseph T. Buck. Software synthesis for DSP using Ptolemy. *Journal on VLSI Signal Processing*, 9(1):7–21, 1995.

[52] Alberto Sangiovanni-Vincentelli. Defining platform-based design. *EEDesign of EETimes*, 2002.

[53] Alberto Sangiovanni-Vincentelli. Quo vadis, sld? reasoning about the trends and challenges of system level design. *Proceedings of the IEEE*, 95(3):467–506, 2007.

[54] Alberto Sangiovanni-Vincentelli, Guang Yang, Sandeep Kumar Shukla, Deepak A. Mathaikutty, and Janos Sztipanovits. Metamodeling: An emerging representation paradigm for system-level design. *IEEE Design and Test of Computers*, 2009.

[55] Wladimir Schamai, Uwe Pohlmann, Peter Fritzson, Christiaan J.J. Paredis, Philipp Helle, and Carsten Strobel. Execution of umlstate machines using modelica. In *3rd International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools (EOOLT)*, volume 47, pages 1–10, Oslo, Norway, 2010. Linköping University Electronic Press, Linköping University. Available from: http://www.ep.liu.se/ecp/047/.

[56] B. Selic. The pragmatics of model-driven development. *IEEE Software*, 20(5):19–25, 2003.

[57] Janos Sztipanovits and Gabor Karsai. Model-integrated computing. *IEEE Computer*, page 110ï¿½112, 1997.

[58] Stavros Tripakis, Dai Bui, Marc Geilen, Bert Rodiers, and Edward A. Lee. Compositionality in synchronous data flow: Modular code generation from hierarchical sdf graphs. Technical Report UCB/EECS-2010-52,, EECS Department, University of California, Berkeley, May 7 2010.

[59] Michael von der Beeck. A comparison of Statecharts variants. In H. Langmaack, W. P. de Roever, and J. Vytopil, editors, *Third International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 863 of *Lecture Notes in Computer Science*, pages 128–148, Lübeck, Germany, 1994. Springer-Verlag.

[60] Y. Xiong, E. A. Lee, X. Liu, Y. Zhao, and L. C. Zhong. The design and application of structured types in Ptolemy II. In *IEEE International Conference on Granular Computing (GrC)*, Beijing, China, 2005.

[61] Gang Zhou, Man-Kit Leung, and Edward A. Lee. A code generation framework for actor-oriented models with partial evaluation. In Y.-H. Lee et al., editor, *Internation Conference on Embedded Software and Systems (ICESS)*, volume LNCS 4523, page 786ï¿½799, Daegu, Korea, 2007. Springer-Verlag.