Rigorous System Design

Rigorous System Design

Joseph Sifakis

EPFL
Switzerland
Joseph.Sifakis@epfl.ch

the essence of knowledge

Boston – Delft

Foundations and Trends® in Electronic Design Automation

Published, sold and distributed by:
now Publishers Inc.
PO Box 1024
Hanover, MA 02339
USA
Tel. +1-781-985-4510
www.nowpublishers.com
sales@nowpublishers.com

Outside North America:
now Publishers Inc.
PO Box 179
2600 AD Delft
The Netherlands
Tel. +31-6-51115274

The preferred citation for this publication is J. Sifakis, Rigorous System Design, Foundations and Trends® in Electronic Design Automation, vol 6, no 4, pp 293–362, 2012

ISBN: 978-1-60198-660-3

Foundations and Trends® in Electronic Design Automation

Volume 6 Issue 4, 2012

Editorial Scope

Foundations and Trends® in Electronic Design Automation will publish survey and tutorial articles in the following topics:

- System Level Design
- Behavioral Synthesis
- Logic Design
- Verification
- Test
- Physical Design
- Circuit Level Design
- Reconfigurable Systems
- Analog Design

Information for Librarians
Foundations and Trends® in Electronic Design Automation, 2012, Volume 6, 4 issues. ISSN paper version 1551-3939. ISSN online version 1551-3947. Also available as a combined paper and online subscription.

Foundations and Trends® in
Electronic Design Automation
Vol. 6, No. 4 (2012) 293–362
© 2013 J. Sifakis
DOI: 10.1561/1000000034

now
the essence of knowledge

Rigorous System Design

Joseph Sifakis

RiSD Laboratory, EPFL, Lausanne, Switzerland, Joseph.Sifakis@epfl.ch

Abstract

The monograph advocates rigorous system design as a coherent and accountable model-based process leading from requirements to correct implementations. It presents the current state of the art in system design, discusses its limitations, and identifies possible avenues for overcoming them.

A rigorous system design flow is defined as a formal accountable and iterative process composed of steps, and based on four principles: (1) separation of concerns; (2) component-based construction; (3) semantic coherency; and (4) correctness-by-construction. The combined application of these principles allows the definition of a methodology clearly identifying where human intervention and ingenuity are needed to resolve design choices, as well as activities that can be supported by tools to automate tedious and error-prone tasks. An implementable system model is progressively derived by source-to-source automated transformations in a single host component-based language rooted in well-defined semantics. Using a single modeling language throughout the design flow enforces semantic coherency. Correct-by-construction techniques allow well-known limitations of a posteriori verification to be overcome and ensure accountability. It is possible to explain, at each design step, which among the requirements are satisfied and which may not be satisfied.

The presented view for rigorous system design has been amply implemented in the BIP (Behavior, Interaction, Priority) component framework and substantiated by numerous experimental results showing both its relevance and feasibility.

The monograph concludes with a discussion advocating a system-centric vision for computing, identifying possible links with other disciplines, and emphasizing centrality of system design.

Contents

1 Introduction

1.1 About Design

Design is the process that leads to an artifact meeting given requirements. These comprise functional requirements describing the functionality provided by the system and extra-functional requirements dealing with the way in which resources are used for implementation and throughout the artifact's lifecycle.

Design is a universal concept, a par excellence intellectual activity linking the immaterial world of concepts to the physical world. It is an essential area of human experience, expertise, and knowledge which deals with our ability to mold our environment so as to satisfy material and spiritual needs. The built world is the result of the accumulation of artifacts designed by humans.

Design has at least two different connotations in different fields and contexts. It may be simply a plan or a pattern for assembling objects in order to build a given artifact. It also may refer to the creative process for devising plans or patterns. In this monograph we adopt the latter denotation with a focus on the formalization and analysis of the process.

Design can be decomposed into two phases. The first is *proceduralization*, leading from requirements to a procedure (executable description) prescribing how the anticipated functionality can be realized by executing sequences of elementary functions. The second is *materialization* leading from a procedure to an artifact meeting the requirements (Figure 1.1). A main concern is how to meet extra-functional requirements by using available resources cost-effectively.

Design is an essential component of any engineering activity. It covers multiple disciplines including electrical, mechanical, thermal, civil, architectural, and computing systems engineering. Design processes should meet two often antagonistic demands: (1) *productivity* to ensure cost-effectiveness; (2) *correctness* which is essential for acceptance of the designed artifacts, especially when they involve public safety and security.

Design is a "problem-solving process". As a rule, requirements are declarative. They are usually expressed in natural languages. For some application areas, they can be formalized by using logics. When requirements are expressed by logical specifications, proceduralization can be considered as a synthesis problem: procedures are executable models meeting the specifications. Model synthesis from logical requirements often runs into serious technical limitations such as non-computability or intrinsically high complexity. For all these reasons, in many areas of engineering, design remains to a large extent an empirical activity relying on the experience and expertise of engineering teams. New complex products are seldom designed from scratch. Their designs follow principles and reuse solutions that have proven their worth. Even if some segments of the design process are fully automated by using tools (e.g., CAD tools), there exist gaps that can be bridged only by creative thinking and insightful analysis.

Design formalization raises a multitude of deep theoretical problems related to the conceptualization of needs in a given area and their effective transformation into correct artifacts. So far, it has attracted little attention from scientific communities and is often relegated to second-class status. This can be explained by several reasons. One is the predilection of the academic world for simple and elegant theories. Another is that design is by nature multidisciplinary. Its formalization

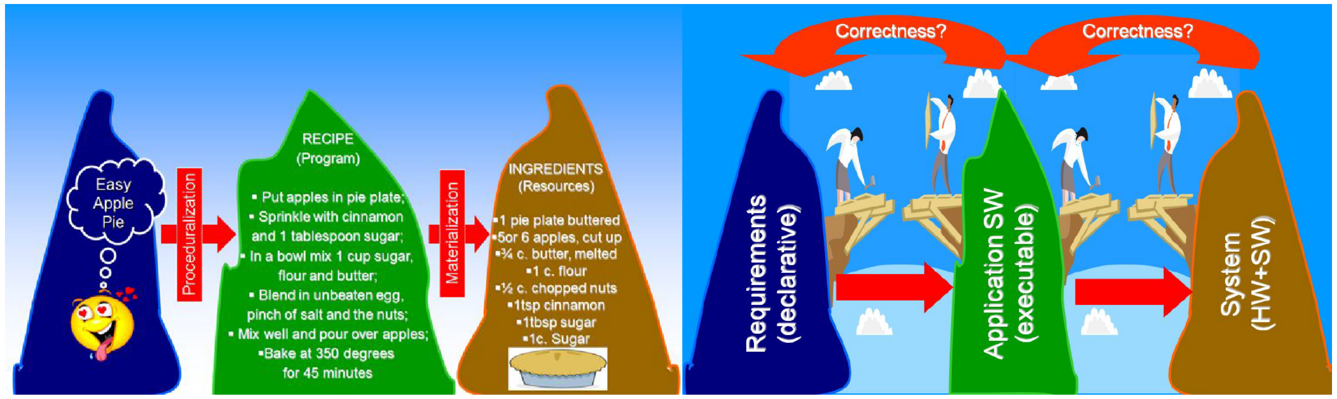
Easy Apple Pie
Proceduralization
RECIPE (Program)
▪ Put apples in pie plate;
▪ Sprinkle with cinnamon and 1 tablespoon sugar;
▪ In a bowl mix 1 cup sugar, flour and butter;
▪ Blend in unbeaten egg, pinch of salt and the nuts;
▪ Mix well and pour over apples;
▪Bake at 350 degrees for 45 minutes
Materialization
INGREDIENTS (Resources)
▪1 pie plate buttered
▪5or 6 apples, cut up
▪¾ c. butter, melted
▪1 c. flour
▪½ c. chopped nuts
▪1tsp cinnamon
▪1tbsp sugar
▪1c. Sugar
Correctness?
Correctness?
Requirements (declarative)
Application SW (executable)
System (HW+SW)

Fig. 1.1 Design is a universal concept applicable from cooking to computing systems.

requires consistent integration of heterogeneous system models supporting different levels of abstraction including logics, algorithms and programs as well as physical system models.

1.2 System Design

The monograph deals with the formalization of the design of mixed hardware/software systems. As a rule, these are interactive systems continuously interacting with an external environment. Their behavior is driven by stimuli from the environment, which, in turn, is affected by their outputs. They drastically differ from function systems which compute an action on an input, producing an output some time later, and stopping. Interaction systems can receive new inputs and produce new outputs while they are already in operation. They are expected to operate continuously.

Interactive systems are inherently complex and hard to design due to unpredictable and subtle interactions with the environment, emergent behaviors, and occasional catastrophic cascading failures, rather than to complex data and algorithms. Compared to function software, their complexity is aggravated by additional factors such as concurrent execution, uncertainty resulting from interaction with unpredictable environments, heterogeneity of interaction between hardware and software, and non-robustness (small variations in a certain part of the system can have large effects on overall system behavior). Henceforth, the term "system" stands for interactive system.

In system design, proceduralization leads to an application software meeting the functional requirements. Materialization consists in building an implementation from application software and models of its execution platforms. As program synthesis is intractable, writing trustworthy application software requires a good deal of creativity and skills. Materialization also requires a deep understanding of how the application software interacts with the underlying hardware and, in particular, how dynamic properties of its execution are determined by the available physical resources.

The monograph advocates rigorous system design as a coherent and accountable process aimed at building systems of guaranteed quality cost-effectively. We need to move away from empirical approaches to a well-founded discipline. System design should be studied as a formal systematic process supported by a methodology. The latter should be based on divide-and-conquer strategies consisting of a set of steps leading from requirements to an implementation. At each step, a particular humanly tractable problem must be solved by addressing specific classes of requirements. The methodology should clearly identify segments of the design process that can be supported by tools to automate tedious and error-prone tasks. It should also clearly distinguish points where human intervention and ingenuity are needed to resolve design choices through requirements analysis and confrontation with experimental results. Identifying adequate design parameters and channeling the designers' creativity are essential for achieving design goals.

The design methodology should take into consideration theoretical obstacles as well as the limitations of the present state of the art. It should propose strategies for overcoming as many of the obstacles as possible. The identified theoretical obstacles are the following:

Requirements formalization: Despite progress in formalizing requirements over the past decades (e.g., by using temporal logics), we still lack theoretical tools for the disciplined specification of extra-functional requirements.

For instance, security and privacy requirements should take into account human behavior which is mostly unpredictable and hardly amenable to formalization. Exhaustive and precise specification of system security threats depends on our ability to figure out all possible attack strategies of intruders. Similarly, for privacy violation we need theory for predicting how global personal data can be inferred by combining and interpreting partial data.

Another difficulty is linking user-defined requirements to concrete properties satisfied by the system. This is essential for checking system correctness. The simple requirement that "when an elevator cabin is moving all doors should be closed" may be implied by a mutual exclusion property at system level. To prove formally such an implication, requirements should be analyzed to relate system states to stimuli provided by user interfaces.

Intractability of synthesis/verification: Designers need automated techniques either to synthesize programs from abstract specifications or to verify derived models against requirements. Both problems do not admit exact algorithmic solutions for infinite state systems.

Hardware–Software interaction: We currently have no theory for predicting precisely the behavior of some given software running on a hardware platform with known characteristics. This difficulty lies in the fundamental difference between hardware and software. Software is immaterial. Software models ignore physical time and resources. Hardware is subject to laws of physics. Its behavior is bound to timing constraints, its resources are limited by their physical characteristics. Program execution dynamics inherit hardware-dynamic properties. These properties cannot be precisely characterized or estimated owing to inherent uncertainty and the resulting unpredictability.

Despite these obstacles and limitations, it is important to study design as a systematic process. As absolute correctness is not achievable, we advocate *accountability*, that is, the possibility to assert which among the requirements are satisfied and which may not be satisfied. Accountability can be enhanced by using property-preservation results: if some essential property holds at some design step then it should hold in all subsequent steps. We present rigorous design as a process rooted in four principles.

Separation of concerns: The separation between proceduralization and materialization is crucial for taming complexity. It allows separation of *what* functionality is provided by the system by focusing only on functional requirements, from *how* this functionality is implemented by using resources. Rigorous system design is a formally defined process decomposed into steps. At each step the designer develops a model of the system to be designed at some abstraction level. Within each step, abstraction is progressively reduced by replacing conceptual constructs and primitives by more concrete ones. The final model is a blueprint for building the physical implementation.

Component-based construction: Components are essential for enhanced productivity and correctness through reuse and architectures. In contrast to many other engineering disciplines, computing systems engineering lacks a component taxonomy and theory for component composition. Electrical and mechanical engineering are based on the use of a few component types. Electrical engineers build circuits from elements of predictable behavior such as resistances, capacitances, and inductances. System designers deal with a large variety of heterogeneous components with different characteristics and unrelated coordination principles: synchronous or asynchronous, object-based or actor-based, and event-based or data-based. This seriously limits our ability to ensure component interoperability in complex systems.

Semantic coherency: The lack of a framework for disciplined component-based construction is reflected in the existence of a large variety of languages used by designers. Application software may be written in Domain-Specific Languages (DSL) or general purpose programming languages. Specific languages may be used for modeling, simulation, or performance analysis. These languages often lack well-founded semantics and this is a main obstacle to establishing semantic coherency of the overall design process. Frequently, validation and performance analyses are carried out on models that cannot be rigorously related to system development formalisms. This introduces gaps in the design process which seriously lessen productivity and limit our ability for ensuring correctness. To overcome these limitations, designers should use languages rooted in well-founded semantics defined in a common host language. This language should be expressive enough to establish source-to-source translations between the hosted languages, in particular for enhanced traceability of analysis results at different abstraction levels.

Correctness-by-construction: Correctness-by-checking suffers from well-known limitations. An alternative approach is achieving correctness-by-construction. System designers extensively use algorithms, architectures, patterns, and other principles for structuring interaction between components so as to ensure given properties. These can be described and proven correct in well-founded languages and made available to system designers. A key issue is how to combine existing solutions to partial problems and their properties in order to solve design problems. For this we need theory and rules for building complex designs meeting a given requirement by composing properties of simpler designs.

The monograph proposes a view for rigorous system design and identifies the main obstacles and associated scientific challenges. This view summarizes key ideas and principles of a research program pursued for more than 10 years at Verimag. It has been amply implemented in the BIP (Behavior, Interaction, Priority) component framework [30] and substantiated by numerous experimental results showing both its relevance and feasibility.

BIP consists of a language for component-based construction and an associated suite of system design tools. The language allows the modeling of composite, hierarchically structured systems from atomic components characterized by their behavior and their interface. Components are coordinated by layered application of interactions and of priorities. Interactions express synchronization constraints between actions of the composed components, while priorities are used to filter amongst possible interactions and to steer system evolution so as to meet performance requirements, e.g., to express scheduling policies. Interactions are described in BIP as the combination of two types of protocols: rendezvous, to express strong symmetric synchronization and broadcast, to express triggered asymmetric synchronization. The combination of interactions and priorities confers BIP expressiveness not matched by any other existing formalism. It defines a clean and abstract concept of architecture separate from behavior. Architecture in BIP is a first-class concept with well-defined semantics that can be analyzed and transformed. BIP relies on rigorous operational semantics that has been implemented by specific run-time systems for centralized, distributed, and real-time execution.

The monograph is structured as follows.

Section 2 presents significant differences between programs and systems. Section 3 discusses the concept of correctness characterized by two types of hardly reconcilable requirements: trustworthiness and optimization. Trustworthiness requirements capture qualitative correctness while optimization requirements are constraints on resources. Their interplay determines levels of criticality in system design. Section 4 presents existing approaches for system design and their limitations.

We discuss how existing rigorous design paradigms can be transposed to system design. Section 5 discusses the four principles for rigorous system design and their application in the BIP framework. Section 6 presents a system-centric vision for computing, discusses possible links with other disciplines and emphasizes on centrality of system design.

References

[1] T. Abdellatif, J. Combaz, and J. Sifakis, "Model-based implementation of real-time applications," *EMSOFT*, pp. 229–238, 2010.

[2] K. J. Astrom and B. Wittenmark, *Adaptive Control.* Boston, MA, USA: Addison-Wesley Longman Publishing Co., 2nd ed., 1994.

[3] Beck, Kent; et al., *Manifesto for Agile Software Development.* Agile Alliance, 2001.

[4] S. Bensalem, M. Bozga, A. Legay, T.-H. Nguyen, J. Sifakis, and R. Yan, "Incremental component-based construction and verification using invariants," in *FMCAD*, pp. 257–256, Lugano, Switzerland, October 20–23 2010.

[5] S. Bliudze and J. Sifakis, "A Notion of Glue Expressiveness for Component-Based Systems," *Lecturer Notes in Computer Science*, vol. 5201, pp. 508–522, 2008.

[6] P. Bogdan and R. Marculescu, "Towards a science of cyber-physical systems design," *Proceeding ICCPS '11 Proceedings of the 2011 IEEE/ACM Second International Conference on Cyber-Physical Systems*, pp. 99–108.

[7] B. Bonakdarpour, M. Bozga, M. Jaber, J. Quilbeuf, and J. Sifakis, "From high-level component-based models to distributed implementations," *EMSOFT*, pp. 209–218, 2010.

[8] P. Bourgos, A. Basu, M. Bozga, S. Bensalem, J. Sifakis, and K. Huang, "Rigorous system level modeling and analysis of mixed HW/SW systems," *MEMOCODE*, pp. 11–20, 2011.

[9] M. Butler, M. Leuschel, S. L. Presti, and P. Turner, "The use of formal methods in the analysis of trust (Position Paper)," *Lecture Notes in Computer Science*, vol. 2995/2004, pp. 333–339, 2004.

[10] G. Buttazzo, *Hard Real-Time Computing Systems, Predictable Scheduling Algorithms and Applications*. Real-Time Systems Series, Springer, vol. 24, 2001.

[11] E. M. Clarke, E. A. Emerson, and J. Sifakis, "Model checking: Algorithmic verification and debugging," *CACM*, vol. 52, no. 11, November 2009.

[12] J. M. Cobleigh, G. S. Avrunin, and L. A. Clarke, "Breaking up is hard to do: An evaluation of automated assume-guarantee reasoning," *ACM Transactions on Software Engineering and Methodology*, vol. 17, no. 2, 2008.

[13] P. Derler, E. A. Lee, and A. Sangiovanni-Vincentelli, "Modeling cyber-physical systems," *Proceedings of the IEEE* (special issue on CPS), vol. 100, no. 1, pp. 13–28, January 2012.

[14] D. Garlan, R. Monroe, and D. Wile, "Acme: An architecture description interchange language," in *Proceedings of the 1997 Conference of the Centre for Advanced Studies on Collaborative Research (CASCON 97)*, pp. 169–183, IBM Press, 1997.

[15] D. Q. Goldin, S. A. Smolka, P. C. Attie, and E. L. Sonderegger, "Turing machines, transition systems, and interaction," *Information and Computation*, vol. 194, no. 2, pp. 101–128, November 2004.

[16] N. Halbwachs, *Synchronous Programming of Reactive Systems.* Kluwer Academic Pub., 1993.

[17] T. Hannay, "The controlled experiment," in *This Will Make You Smarter*, (J. Brockman, ed.), Happer Perennial.

[18] D. Harel, A. Marron, and G. Weiss, "Behavioral programming," *Communications of the ACM*, vol. 55, no. 7, July 2012.

[19] T. A. Henzinger and J. Sifakis, "The discipline of embedded systems design," *COMPUTER*, vol. 40, pp. 36–44, 2007.

[20] H. Hoos, "Programming by optimization," *Communications of the ACM*, vol. 55, no. 2, February 2012.

[21] International Council on Systems Engineering (INCOSE), *Systems Engineering Handbook* Version 3.1. August 2007.

[22] H. Kopetz, "The rationale for time-triggered ethernet," *Proceedings of the 29th IEEE Real-Time Systems Symposium.*

[23] H. Kopetz, R. Obermaisser, C. E. Salloum, and B. Huber, "Automotive software development for a multi-core system-on-a-chip," *Fourth International Workshop on Software Engineering for Automotive Systems (SEAS'07)*, 2007.

[24] E. A. Lee and T. M. Parks, "Dataflow process networks," *Proceedings of the IEEE*, vol. 83, no. 5, pp. 773–801, May 1995.

[25] J. Magee and J. Kramer, "Dynamic structure in software architectures," in *Proceedings of the 4th ACM SIGSOFT Symposium on Foundations of Software Engineering (SIGSOFT 96)*, pp. 3–14, ACM Press, 1996.

[26] S. Maoz, D. Harel, and A. Kleinbort, "A compiler for multimodal scenarios: Transforming LSCs into AspectJ, September 2011," *Transactions on Software Engineering and Methodology (TOSEM)*, vol. 20, no. 4.

[27] D. H. Mcknight and N. L. Chervany, "The meanings of trust," *Trust in Cyber-Societies-LNAI*, pp. 27–54, 2001.

[28] R. A. D. Millo, R. J. Lipton, and A. J. Perlis, "Social Processes and Proofs of Theorems and Programs," *CACM*, vol. 22, no. 5, May 1979.

[29] D. K. Mulligany and F. B. Schneider, "Doctrine for Cybersecurity," Technical Report, Cornell University, May 2011.

[30] Rigorous Design of Component-Based Systems — The BIP Component Framework: http://www-verimag.imag.fr/Rigorous-Design-of-Component-Based.html.

[31] J. Reineke, B. Wachter, S. Thesing, R. Wilhelm, I. Polian, J. Eisinger, and B. Becker, "A definition and classification of timing anomalies," in *Sixth International Workshop on Worst-Case Execution Time (WCET) Analysis*, Dresden, Germany, July 4 2006.

[32] J. Sifakis, "A framework for component-based construction," in *IEEE International Conference on Software Engineering and Formal Methods (SEFM05)*, pp. 293–300, Koblenz, September 7–9 2005.

[33] SOFTWARE 2015: A national software strategy to ensure U.S. security and competitiveness report of the 2nd national software summit, April 29, 2005.

[34] R. J. van Glabbeek, "Ursula Goltz: Refinement of actions and equivalence notions for concurrent systems," *Acta Information*, vol. 37, no. 4/5, pp. 229–327, 2001.

[35] J. van Leeuwen and J. Wiedermann, "The turing machine paradigm in contemporary computing," in *Mathematics Unlimited — 2001 and Beyond*, (B. Enquist and W. Schmidt, eds.), LNCS, Springer-Verlag, 2000.

[36] D. A. Watt, B. A. Wichmann, and W. Findlay, "Ada: Language and Methodology," 1987.