JOURNAL OF OBJECT TECHNOLOGY
Online at http://www.jot.fm. Published by ETH Zurich, Chair of Software Engineering. ©JOT, 2005

Vol. 4, No. 3
Special issue: 6th GPCE Young Researchers Workshop 2004

Generative Components for Hybrid Systems Tools

Jonathan Sprinkle, University of California, Berkeley, CA 94720, USA

Generative techniques, while normally associated with programming languages or code generation, may also be used to produce non-executable artifacts (e.g., configuration or toolchain artifacts). Coupled with domain-specific modeling, generative techniques provide a way to consolidate toolchains and complex domains into a relatively compact space, by providing generators that produce artifacts in the appropriate semantic domain. This paper describes the motivation and usage of one such environment, in the domain of hybrid systems, as well as discussion and goals for future research.

1 MOTIVATION

Writer's block—no written progress due to lack of ideas, or difficulty in expressing them properly—is the most infamous affliction for writers of all kinds; poets, journalists, fictionalists, and computer scientists. Faced with a blank screen in a programming IDE, even the most seasoned programmer can be overwhelmed with the design and implementation possibilities of the final system—assuming that the programmer correctly understands what the final system should do!

While providing a programmer with a general-purpose tool that can be useful when solving many different types of problems, such power and flexibility also means that the space of all possible solutions is dramatically increased. This makes design difficult for programmers without experience, since they lack the knowledge of best practices; i.e., they may know exactly what they *want to say*, but have absolutely no idea *how to say it*.

When considering the latter problem (difficulty encoding knowledge), the main issue that of the overwhelming possibilities provided by a 101-key ASCII keyboard. Domain-specific languages address this issue exactly, by providing a highly-restrictive programming language that is customized for a particular domain. By using a domain-specific modeling environment (DSME) it is possible to provide an integrated development environment for a particular domain, for rapid development and specification of systems. A major advantage of this is that it is possible to export artifacts from a system specification using one or more generative components associated with the DSME. This paper describes a DSME (and several components) for the domain of hybrid systems, and explains how this DSME is useful not only for system specification, but toolchain construction.

Cite this article as follows: Jonathan Sprinkle: "Generative Components for Hybrid Systems Tools", in *Journal of Object Technology*, vol. 4, no. 3, April 2005, Special issue: 6th GPCE Young Researchers Workshop 2004, pp. 35–39, http://www.jot.fm/issues/issue_2005_04/article5

2 HYBRID SYSTEMS, AND TOOLS

Hybrid systems are systems that may be described using discrete states in which continuous time dynamics govern the laws of behavior. This combination of discrete states and continuous dynamics means that traditional systems analysis is infeasible for large systems. As hybrid systems are becoming more commonplace, determination of the safety and reliability of such systems is an emerging research opportunity. Currently, the analysis [**?**], simulation (common to most tools), verification[**?**], validation [**?**], and code synthesis [**?**] of controllers for hybrid systems is such a large and complex problem that no one tool integrates every one of these aspects. Dedicated researchers have developed applications that evaluate (or generate) hybrid systems controllers in one (or sometimes, two) of these aspects, but since research has been ongoing for many years, the specifications for the existence of those hybrid systems is given in a proprietary format defined by each tool developer.

An interesting aspect of these proprietary formats is that they are usually derivable from some common specification: mathematics. For each well-formed mathematical specification of a hybrid system (including the differential equations that govern the continuous behavior, and the state descriptions of their discrete behavior) there exists a transformation to one of these tools.

The process is not without problems, however. The "math2tool" transformations usually exist only in the heads of the tool experts. There is no simple extraction from a tool specification back to mathematics, that preserves all aspects of the hybrid system; that is, not all tools use all parts of the mathematical specification. Also, tools may require tool-specific entries that configure the tool to have a certain behavior, but have no impact on the specification of the system. Combined with the undocumented (or ill-documented) syntax and semantics of the tool specifications, these problems indicate the difficulty of developing a cohesive toolchain.

3 A COMMON FORMAT

In order to address the need for a hybrid systems toolchain, researchers combined to create the Hybrid Systems Interchange Format (HSIF). This format, defined as part of the DARPA MoBIES project, and under consideration for OMG standardization, could then be used as a common repository (or export option) for hybrid systems models. Tools could either import HSIF models, export HSIF models, or both (if appropriate), as shown in Fig. 1.

HSIF model specification

HSIF, as a syntax, is not meant to be human readable; rather, it is designed such that it is an unambiguous specification of the hybrid models. For example, the equation $\dot{x} = \frac{1}{4}e^{0.1x} + x$ is a simple equation for expressing the derivative of some

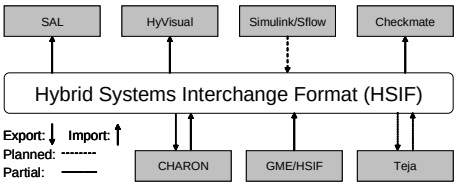


Figure 1: Layout of the HSIF toolchain implementation

variable, x, with respect to time. Given that some variable x exists, this is a valid equation; however, consider a hierarchical model where a variable x may be defined in some parent of this model. The issue of *scope*—that is, which x is used for this specification—becomes important. Different tools, as they have been developed independently of one another, have not implented this (and other, more complex) possibly ambiguous specification identically.

The goal of HSIF then is to provide the specification of the mathematical definition of these hybrid models—*even at the cost of readability*—and thus provide a model-based representation of the mathematics, suitable for input to generators that can provide tool-specific versions of the models. This obviously required a well-defined syntax for HSIF [**?**], and (less obviously, but intuitively upon further reflection) required a well-defined semantics for HSIF.

HSIF Modeling Environment (HSIF-ME)

The reader may have two questions at this point. "Why is readability is sacrificed, especially if the models need to be visualized (presumably to check correctness of entry)?" and "If readability is sacrificed, what does that say about *writability*?" The answer to each of these questions is that HSIF models are toolchain *artifacts*, which does not necessarily mean that they must be written (or read) by humans.

Rather, a convenient entry format which is as similar to the mathematical definition of a hybrid system as possible is provided through a DSME customized for the hybrid systems domain. Using the HSIF-ME, it is possible to create a hybrid system model of average size (2 hybrid automata, each with 4 discrete states and 2 variables, connecting with signals) in about 30–60 minutes (depending on familiarity with hybrid systems and computer efficiency). The DSME is implemented using the GME toolsuite [**?**], and provides three components: to generate an HSIF artifact, to directly generate an artifact for CheckMate (using the HSIF artifact as an intermediary), and to import and HSIF artifact.

The last component allows HSIF artifacts generated from other tools to be imported into the HSIF-ME for visualization. This provides a means through which to check some correctness metrics for the models, to ensure that no syntactic or static semantics errors have been made[1].

[1]The HSIF storage format is XML, which implicitly has a syntax checker in the XML parser

4 LOOKING FORWARD

HSIF—as an interchange format—suffered from the famous "design by committee" problem, with a swelling syntax and some tool-specific enhancements. In short, HSIF emerged from existing formats, not all of which were well designed (to put it politely). As such, it was acceptable as a first cut, but is more useful as a "lessons learnt" artifact than a basis for future designs. The next stage is to explore the realm of an optimal (or usefully optimal) format for the specification of hybrid systems, which will drive the formats of tools that will be developed in the future.

Research and state-of-the-art exploration is currently underway to design a framework for hybrid systems tool development. This framework will provide a visual shell, and leverage generative components to implement simulators, validators, design environments, and code generators. The framework will be something like a cross between Matlab and Eclipse. The emphasis will be on a domain-specific environment with multiple backends for processing, rather than a textual IDE with associated compilers. An early sketch of this design is presented in Fig. 2.

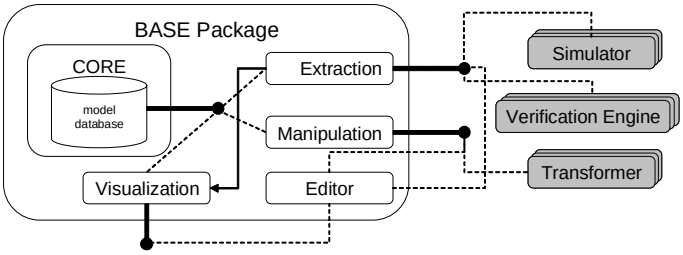


Figure 2: The future designs involve not only a core model of the hybrid system, but also a componentized view of the manipulation, editing, extraction, and visualization of those models, which may be arbitrarily extended by tool designers (components shown in dark), and improved upon in future releases while insulating the models themselves.

This research is expected to continue for years to come, with incremental releases, and the cooperation and integration of tools and algorithms as developed by hybrid systems researchers all over the world.

5 CONCLUSIONS

What makes the HSIF-ME an interesting player as it currently exists is its lightweight presence (no hybrid systems tools are necessary to create hybrid systems models)

used to read the file. However, not all XML generation engines guarantee correctness of output to the HSIF schema, nor do they guarantee the correctness of the model to certain *static semantics*, which check the models to ensure that all variables are correctly declared, no cyclical dependencies are specified, etc.

and its ability to both import and export hybrid models. HSIF-ME's proximity to the mathematical specifications of a hybrid system also makes it an excellent modeling environment, especially for students familiar with hybrid systems, but unfamiliar with tools that evaluate them.

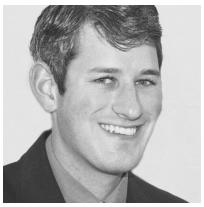
In our future work, we hope to preserve the lightweight presence by developing the tool framework as a core with associated components. Using just the basic facilities, it would be possible to design just the hybrid system, which could be shipped to more complete implementations of the framework, or to independent tools for conversion. In the meantime, we plan to maintain the concept of a domain-specific interface as the primary human input mechanism, and to continue to investigate the best ways to encode the designs for portability.

ACKNOWLEDGEMENTS

The author would like to thank Gábor Karsai (whose figure appeared as Fig. 1), and Oleg Sokolsky, for their contributions to the development of the HSIF semantics and syntax. Additional thanks to S. Shankar Sastry, Edward A. Lee, and Aaron Ames for their work and comments regarding the future work.

Portions of this research were funded by the Defense Advanced Research Projects Administration Information Exploitation Office under contract #F30602-00-1-0580 (MoBIES), the NSF ITR Center for Hybrid and Embedded Software Systems, and also under subcontract to Northrop Grumman Corporation in association with the DARPA SEC project.

ABOUT THE AUTHORS



Jonathan Sprinkle Dr. Jonathan Sprinkle is a postdoctoral researcher at the University of California, Berkeley since 2003. His research interests and experience are in systems control and engineering, through modeling and metamodeling. He may be reached at sprinkle@EECS.Berkeley.Edu, or on the web at http://www.eecs.berkeley.edu/ sprinkle/.