



MODELS@ RUN.TIME

Gordon Blair and Nelly Bencomo, *Lancaster University*
Robert B. France, *Colorado State University*

Runtime adaptation mechanisms that leverage software models extend the applicability of model-driven engineering techniques to the runtime environment.

Contemporary mission-critical software systems are often expected to safely adapt to changes in their execution environment. Given the critical roles these systems play, it is often inconvenient to take them offline to adapt their functionality. Consequently, these systems are required, when feasible, to adapt their behavior at runtime with little or no human intervention.

Research on self-adaptive software has produced significant results, but many problems remain. A particularly challenging problem concerns the complexity that arises from the wealth of information that can be associated with runtime phenomena. A promising approach to managing complexity in runtime environments is to develop adaptation mechanisms that leverage software models, referred to as *models@run.time*. Work on *models@run.time* seeks to extend the

applicability of models produced in model-driven engineering (MDE) approaches to the runtime environment.

The aim of this special issue is to provide exposure to the excellent work that is being carried out under the banner of *models@run.time*. More specifically, the special issue seeks to

- raise awareness of the important role that software models produced during development can play at runtime;
- present a set of articles that are representative of the work emerging in this area; and
- highlight key research challenges associated with using *models@run.time* and to invite others to join the growing community of researchers tackling these challenges.

In addition to the special issue articles, we have asked three leading researchers in software engineering for their perspectives on *models@run.time*. We include contributions from Betty H.C. Cheng, Anthony Finkelstein, and Bran Selic as sidebars in this guest editors' introduction. Their insights add significantly to the value of this special issue, and we thank them for their thoughtful contributions.

WHAT IS A MODEL@RUN.TIME?

In MDE, a model is an abstraction or reduced representation of a system that is built for specific purposes. For example, technology-independent models of software describe applications using concepts that abstract over the underlying computing technologies. The models@run.time community shares this view of what constitutes a model and seeks an understanding of the roles that such models can play at runtime.

Before providing a definition of models@run.time, it is helpful to contrast this work with historic efforts in the field of reflection. The reflection community is concerned with defining representations of an underlying system that are both self-representations and causally connected.¹ That is, the models should represent the system and should be linked in such a way that they constantly mirror the system and its current state and behavior; if the system changes, the representations of the system—the models—should also change, and vice versa.

In models@run.time, we also seek appropriate self-representations. It is critical that such representations be causally connected. This is an important requirement for adaptive systems for two reasons:

- the model as interrogated should provide up-to-date and exact information about the system to drive subsequent adaptation decisions; and
- if the model is causally connected, then adaptations can be made at the model level rather than at the system level.

One conclusion that could be drawn is that models@run.time is synonymous with reflection, but that would be wrong. The work in reflection seeks models that are intrinsically related to the computation model and hence tend to be based on the solution space and often rather low level—for example, exposing the act of message passing and supporting metalevel operations like interception of such messages.

In models@run.time, we seek models at a much higher level of abstraction and, in particular, causally connected models related to the problem space. Another key distinction is that such models should be intrinsically tied to the models produced as artifacts from the MDE process and hence linked to the software engineering methodologies employed. This is an important symbiosis and one that has already paid dividends in the field of distributed systems, where there is a similarly strong link between developments in distributed objects, as in Corba, and associated object-oriented software engineering methodologies based around UML.


Models@run.time therefore builds on reflection but seeks to draw the work from the solution space up to

the problem space, mirroring the similar influence that MDE seeks more generally on software engineering methodologies.²

This leads to the following definition: a model@run.time is a causally connected self-representation of the associated system that emphasizes the structure, behavior, or goals of the system from a problem space perspective.

WHY MODELS@RUN.TIME?

Development models—models produced in an MDE process—are software models “at levels of abstraction above the code level”³ and usually convey software design meaning. Examples of such models are use cases and architectural and deployment models. These models are associated with the entities managed during the software development cycle. In contrast, runtime models provide “abstractions of runtime phenomena”³ and different stakeholders can use them in different ways.



We can envision using runtime models to fix design errors or to fold new design decisions into a running system to support controlled ongoing design.

As in the case of traditional development models, a runtime model supports reasoning. System users can also use runtime models to support dynamic state monitoring and control of systems during execution, or to dynamically observe the runtime behavior to understand a behavioral phenomenon. A runtime model can also potentially support semantic integration of heterogeneous software elements at runtime—for example, dynamically adaptable metamodels—in the domain of systems of systems.⁴ Runtime models can also assist in the automated generation of implementation artifacts that will be inserted into the system during execution and even by the system itself.⁵

In a more visionary approach, we can envision using runtime models to fix design errors or to fold new design decisions into a running system³ to support controlled ongoing design. The mechanisms used to realize this vision are expected to be significantly more complex. On the other hand, these more complex mechanisms would be able to support unanticipated modifications, a key challenge for adaptive systems generally.⁶

The potential support of models@run.time for the evolution of software design can blur the line between development models and runtime models. From this perspective, a runtime model can be seen as a live development model that enables dynamic evolution and the realization of software designs.

➤ USING MODELS@RUN.TIME TO MANAGE ULTRA-LARGE-SCALE SYSTEMS

Betty H.C. Cheng, *Michigan State University*

Society increasingly depends on cyberinfrastructure for essential functions. A robust cyberinfrastructure must be able to monitor the environment and its own behavior, adapt to changing conditions, and protect itself from component failures and security attacks. This capability is especially important as computing systems interact with the physical world through mobile ad hoc sensor networks and other embedded systems.

Emerging ultra-large-scale systems¹ have heterogeneous components and highly distributed elements on a scale several orders of magnitude greater than other software-intensive systems. A ULS system designer thus faces a challenging set of tasks: anticipating how and when the software will need to adapt in the future, codifying this behavior in decision-making components to govern the adaptation, and ensuring that system integrity is not compromised during adaptation. Current software development environments simply do not adequately address these issues.


Two ULS factors pose significant needs for the research community to address: complexity and uncertainty. Given the scale of data, heterogeneity, and decentralized control, one key need is developing the enabling technology that will help manage the complexity of the data, decision-making logic, and distributed control of ULS systems.² Another need is being able to manage the uncertainty that such systems pose. Models developed for and used at runtime offer a promising technology to address these challenges.

Modeling has long been recognized as a means to introduce abstraction to reduce complexity. With multiview modeling techniques in particular, each view can provide a simple, easy-to-understand view of a system. At the same time, integration of the multiple views must be well-defined to ensure a consistent and comprehensive overall model of a system.

Advanced modeling techniques coupled with model-driven engineering capabilities such as model transformation/refinement and code generation offer a viable means to capture runtime monitoring information, provide decision-making support through runtime model analysis, and enable runtime adaptation of the system through model adaptation³ and corresponding code generation. While current technology may not be able to support all of these activities for all types of systems, data, and modeling technologies,

the vision is in place with promising preliminary results, many of which are reported in this special issue.

The uncertainty posed by ULS and adaptive systems makes it necessary to look outside the traditional computing domain for inspiration in developing modeling techniques, particularly to handle assurance needs. For example, biologically inspired techniques have great potential to provide insight given the complex nature of biological systems. Further, researchers have already used evolutionary computation techniques such as genetic algorithms and digital evolution to generate models for ULS and adaptive systems.⁴⁻⁶

In summary, ULS and adaptive systems pose many interesting and exciting challenges for those working in the area of models@run.time. 

References

1. L. Northrup et al., *Ultra-Large-Scale Systems: The Software Challenge of the Future*, Software Eng. Inst., Carnegie Mellon Univ., 2006.
2. J. Zhang and B.H.C. Cheng, "Model-Based Development of Dynamically Adaptive Software," *Proc. 28th Int'l Conf. Software Eng. (ICSE 06)*, ACM Press, 2006, pp. 371-380.
3. H.J. Goldsby and B.H.C. Cheng, "Automatically Generating Behavioral Models of Adaptive Systems to Address Uncertainty," *Proc. 11th Int'l Conf. Model Driven Eng. Languages and Systems (MoDELS 08)*, LNCS 5301, Springer-Verlag, 2008, pp. 568-583.
4. P. McKinley et al., "Harnessing Digital Evolution," *Computer*, Jan. 2008, pp. 54-63.
5. H.J. Goldsby et al., "Digital Evolution of Behavioral Models for Autonomic Systems," *Proc. 2008 Int'l Conf. Autonomic Computing (ICAC 08)*, IEEE CS Press, 2008, pp. 87-96.
6. A.J. Ramirez et al., "Applying Genetic Algorithms to Decision Making in Autonomic Computing Systems," *Proc. 6th Int'l Conf. Autonomic Computing (ICAC 09)*, IEEE CS Press, 2009, pp. 97-106.

Betty H.C. Cheng is a professor in the Department of Computer Science and Engineering at the University of Michigan. Contact her at betty.hc.cheng@gmail.com.

Runtime models can support adaptation decisions by humans—for example, system managers—through adaptation agents embedded in the system itself or through combinations of both. The trend is to support increasing automation of decision making with respect to adaptation of systems as captured by the drive toward autonomic computing, wherein agents may learn appropriate strategies for effective systems management. We see models@run.time as an important contribution to the field of autonomic computing providing meta-information to drive autonomic decision making.

As a final comment, it is interesting to reflect on the design of modern processors, including multicore architectures, which have the performance to support the concurrent execution of many simultaneous abstractions. For example, the virtualization community notes that

such processors can support up to 100 virtual machine environments potentially supporting different operating system images.⁷ In the same way, this technology is opening the door to the simultaneous execution of systems and models of the system—including potentially sophisticated models—without undue adverse impact on the performance of the system as a whole.

WHAT CONSTITUTES A MODEL@RUN.TIME?

A model can be any useful representation of the system, and it should be no surprise that a variety of techniques have been investigated for models@run.time. Some of the key dimensions include the following:

- *Structure versus behavior.* As in the reflection community, models tend to focus on either the structure of the

underlying system or the associated, more dynamic, behavioral aspects. Structural models tend to emphasize how the software is currently constructed—for example, in terms of objects, inheritance relationships and invocation pathways; components and their connections; or aspects and their patterns of weaving. In contrast, behavioral models emphasize how the system executes in terms of flows of events or traces through the system, or the arrival of events and their pathway to execution—arriving, enqueueing, selection, dispatching, and so on.

- *Procedural versus declarative.* Another key choice is whether the model is procedural, reflecting the actual structures or behaviors in the system (the nuts and bolts), or more declarative—for example, in terms of system goals. There is a clear pressure arising from the need for mirroring the problem space for more declarative models.
- *Functional versus nonfunctional.* Most models that have been considered are based on the underlying system's functionality, but there is an equal need for models that capture and represent nonfunctional characteristics. As an example of the latter, in the software performance field, researchers have looked at how performance models can be used at runtime to optimize system execution. Equally, there is a need for models that capture aspects such as the system's security.
- *Formal versus informal.* Several models are inspired by the mathematics of computation whereas others are derived from consideration of programming models or domain abstractions. This special issue presents examples of both. The advantage of formal models is that they may support automated reasoning about the system's state but equally may not adequately capture or express the required domain concepts so elegantly.

Note that in practice, it is likely that multiple models will coexist and that different styles of models may be required to capture different system concerns.

RESEARCH CHALLENGES

As the work reported in this special issue and elsewhere shows, existing research on models@run.time tends to focus on software structures and procedural representations of such structures. For example, significant bodies of work look at software architecture as an appropriate basis for a runtime model.⁸⁻¹²

Our vision of models@run.time goes further than this. Our aim is to raise the level of runtime model abstraction to that of requirements. For example, an exciting and promising research topic is how a software system's requirements and goals can be observed during execution.^{6,13}


➔ RECONCEPTUALIZING SOFTWARE DEVELOPMENT

Anthony Finkelstein, *University College London*

It is often surprising in software engineering how “sticky” certain concepts are, how hard to change, even when we know they are wrong or, worse, actively misleading. So, for example, we have the conventional systems life cycle in which requirements precede architecture, architecture precedes design, design precedes implementation, and implementation precedes testing. This is clearly wrong, yet so ingrained that it is difficult to talk about software development without this ordering of activities coloring the way we express ourselves.

So it is too with the design, compile, deploy, runtime division. We know that the design of component-based systems can be altered while the system is running, new component instances can be added, and existing component configurations can be rewired. This may happen to address requirements we are only aware of when the system is deployed and running.

Models@run.time is an important step forward in understanding and addressing these issues. It is, however, just the first step. We need a radical reconceptualization of software development in which we rethink the whole notion of design time, compile (or deploy) time, and runtime. Such a reconceptualization should focus on users and their needs and goals, which are constantly changing and reshaping. It should further take into account that global services may require radical changes as they scale, cope with big shifts in usage, or transform to move into new market niches.

With this change we must firmly set aside the last vestiges of “waterfall” thinking and understand that incrementality is not simply the dominant way but the only way large systems will be delivered. Models@run.time is shining a torch in the right direction, but to reach our destination, we might need to step further into the darkness. 

Anthony Finkelstein is professor of software systems engineering, University College London, UK. Contact him at a.finkelstein@cs.ucl.ac.uk.

Anthony Finkelstein refers to this topic as requirements reflection.¹⁴ To achieve this, a model of the requirements of the system should be maintained while the system is running—that is, a runtime model. Clearly, in such systems, maintaining the causal relationship between the model and the running system is much more demanding, but this is, we believe, a key research challenge for the community to investigate with the potential benefits being enormous.

Other challenges include the following:

- How are the current generative technologies used during development different from the synthesis techniques needed when using runtime models during execution? Are the former technologies suitable for dynamic model synthesis?
- What methods and standards for specifying semantics are suited to automated interpretation—that is, done during runtime?

➔ RUNNING AWAY WITH MODELS

Bran Selic, *Malina Software Corp.*

As might be expected, models@run.time is simply the latest manifestation of an old idea: The principle of exploiting the predictive potential of models by integrating them directly into engineering systems was proposed at least 60 years ago in control systems theory.¹ It is new, however, in the world of software engineering, where models were traditionally viewed as inessential informal aids to analysis and design, used primarily for their potential to render complex systems into something intellectually manageable.

Software models@run.time, on the other hand, add a new and important quality to this while retaining all the benefits of engineering models. Namely, rather than being merely informal representations, these models can also serve as specifications of the systems they model. As such, they can be processed online for a variety of purposes, some of which are described in the articles in this special issue.

To support computer-based processing, runtime models are usually formal and unambiguous—unlike the majority of models commonly used in analysis and design. This enables formal couplings between models and the systems they represent, similar to the relationship that exists between a program written in a high-level programming language and its machine-code counterpart. Consequently, these models can be quite accurate, and the results of model analyses are likely to be more trustworthy than with other types of engineering models.

- How can we achieve reversible model transformations to deal with synchronization issues between the run-time model and the running system and between the development models and runtime models?
- What are the right runtime models for systems of systems and ultra-large-scale systems (see the “Using Models@run.time to Manage Ultra-Large-Scale Systems” sidebar by Betty H.C. Cheng) including managing the inherent complexity and dealing with uncertainty?
- What repercussions does models@run.time have for software engineering, and can it act as a catalyst for radical reconceptualization of software development (see the “Reconceptualizing Software Development” sidebar by Anthony Finkelstein)?
- Can runtime models provide support for what-if analysis of software and deal with unanticipated change (see the “Running Away with Models” sidebar by Bran Selic)?


These questions are just a few starting points for research in this exciting topic with potential fruitful results for software engineering.

ABOUT THIS SPECIAL ISSUE

The five papers selected for inclusion in this special issue are varied and complementary, and provide an

The decision-support role of models@run.time resembles their role in analysis and design, with the added twist that, thanks to the efficiency of modern computing technology, models can be consulted on the fly and in real time. For instance, a model can be queried to predict the possible consequences of different response strategies to an event prior to committing to any of them. The final design choices can thus be postponed until the actual circumstances required for making decisions are known—a key enabler for adaptive systems.

Moreover, in reflective systems in which the structure and behavior are defined by underlying resident metamodels, we can dynamically generate new system capabilities to respond to situations unforeseen in the original design simply by changing the metamodel.

So, even though the idea is old, combining it with modern computing technology not only amplifies the benefits but also creates new opportunities. Now is the time to explore and exploit these further. 

Reference

1. H.P. Whitaker, J. Yamron, and A. Kezer, *Design of Model Reference Adaptive Control Systems for Aircraft*, report R-164, Instrumentation Lab, MIT, Cambridge, Mass., 1958.

Bran Selic is the founder and president of Malina Software Corp., a computing services consulting company in Ottawa, Canada. Contact him at selic@acm.org.

excellent representative sample of research in models@run.time.

In “Using Model-Based Traces as Runtime Models,” Shahar Maoz proposes traces of execution as his view of a model@run.time. He successfully abstracts away from low-level system execution by basing the traces on high-level events extracted from design-level sequence diagrams and statecharts. In particular, concrete program executions generate the model-based traces. Maoz also offers a partial family of metrics to analyze the models.

The next two articles focus on the management of variability at runtime, taking inspiration from work on dynamic variability.

“Autonomic Computing through Reuse of Variability Models at Runtime: The Case of Smart Homes” by Carlos Cetina and coauthors outlines their work on reusing design variability models during runtime. The benefits are immediate, as the design knowledge and existing model-based technologies can be reused at runtime. The runtime variability models support the self-reconfiguration of systems when triggered by changes in the environment. Cetina and colleagues have applied their approach to an interesting application in the smart-homes domain, providing valuable validation of their approach.


In “Models@run.time to Support Dynamic Adaptation,” Brice Morin and colleagues also study the role of runtime models to manage runtime or dynamic variability. How-

ever, their research focuses on reducing the number of configurations and reconfigurations that need to be considered when planning adaptations of the application, and they illustrate their approach through a customer relationship management application.

In "Using Architectural Models to Manage and Visualize the Runtime Adaptation," John C. Georgas, André van der Hoek, and Richard N. Taylor revisit the use of architectural models as runtime artifacts to enable adaptation. In particular, they study the use of configuration graphs as runtime representations of the architectural configurations that a system can exhibit during periods of adaptation. The graphs monitor and record information about adaptations for analysis. The key goal of the authors' approach is to regain cognitive control over the runtime adaptation process. The use of runtime models in this area shows significant potential.

Finally, "Mirrors of Meaning: Supporting Inspectable Runtime Models" by Tony Gjerlufsen, Mads Ingstrup, and Jesper Wolff Olsen proposes an interesting approach to the definition of a runtime model based on the novel concept of an H-graph. Interesting contrasts can be made between the H-graph as a runtime representation and the more studied approach based on architectural models (effectively component graphs). For example, is the hierarchical structure implied by H-graphs more understandable from a user's perspective than the alternative component graph structures? This article discusses the experience with applying the H-graph approach in various projects including a software system to support the Tall Ships Races.

The articles selected for inclusion in this special issue collectively demonstrate the application of runtime models in different areas of application, giving an account of the state of the art and of emerging results from the field, and providing a glimpse of future research directions. Much, however, remains to be done. We encourage you to get involved in this newly developing field and contribute your experience in future publication venues, turning the vision of `models@run.time` into reality.

We thank all the authors of the articles included here and also the anonymous reviewers for their hard work and cooperation in developing this special issue. 

References

1. P. Maes, "Concepts and Experiments in Computational Reflection," *ACM SIGPLAN Notices*, vol. 22, no. 12, 1987, pp. 147-155.
2. D.C. Schmidt, "Model-Driven Engineering," *Computer*, Feb. 2006, pp. 25-31.
3. R. France and B. Rumpe, "Model-Driven Development of Complex Software: A Research Roadmap," *Future of Software Engineering*, L. Briand and A. Wolf, eds., IEEE CS Press, 2007.

4. N. Bencomo et al., *Models in Software Engineering: Workshops and Symposia at MoDELS 2008*, LNCS, Springer-Verlag, 2009, pp. 90-96.
5. B. Morin et al., "An Aspect-Oriented and Model-Driven Approach for Managing Dynamic Variability," *Model-Driven Engineering Languages and Systems*, LNCS 5301, Springer, 2008, pp. 782-796.
6. B.H.C. Cheng et al., "Software Engineering for Self-Adaptive Systems: A Research Road Map," 2008; <http://drops.dagstuhl.de/opus/volltexte/2008/1500>.
7. P. Barham et al., "Xen and the Art of Virtualization," *Proc. 19th ACM Symp. Operating Systems Principles (SOSP 03)*, ACM Press, 2003, pp. 164-177.
8. P. Oreizy et al., "An Architecture-Based Approach to Self-Adaptive Software," *IEEE Intelligent Systems*, vol. 14, no. 3, 1999, pp. 54-62.
9. D. Garlan and B. Schmerl, "Using Architectural Models at Runtime: Research Challenges," *Software Architecture*, LNCS 3047, Springer, 2004, pp. 200-205.
10. J. Floch et al., "Using Architecture Models for Runtime Adaptability," *IEEE Software*, vol. 23, no. 2, 2006, pp. 62-70.
11. P. Grace et al., "Engineering Complex Adaptations in Highly Heterogeneous Distributed Systems," *Proc. 2nd Int'l Conf. Autonomic Computing and Communication Systems (Autonomics 08)*, art. no. 27, ICST, 2008.
12. G. Coulson et al., "A Generic Component Model for Building Systems Software," *ACM Trans. Computer Systems*, Feb. 2008, pp. 1-42.
13. J. Whittle et al., "RELAX: Incorporating Uncertainty into the Specification of Self-Adaptive Systems," to appear in *Proc. 17th IEEE Int'l Requirements Eng. Conf. (RE 09)*, IEEE Press, 2009.
14. A. Finkelstein, "Requirements Reflection," presentation, *Dagstuhl Workshop on Software Eng. for Self-Adaptive Systems*; www.cs.ucl.ac.uk/staff/a.finkelstein/talks/reqtsreflection.pdf.

Gordon Blair is professor of distributed systems in the Computing Department at Lancaster University and an adjunct professor at the University of Tromsø, Norway. His research interests include distributed systems architecture, middleware (including reflective and adaptive middleware), mobile and ubiquitous systems, and model-driven engineering techniques applied to adaptive distributed systems. Contact him at gordon@comp.lancs.ac.uk.

Nelly Bencomo is a senior research associate in the Computing Department at Lancaster University. Her research interests include the use of software models and model-driven techniques during the development, execution, and operation of dynamically adaptive systems. She received a PhD in computer science from Lancaster University. She acknowledges the EU FP7 STREP DiVA project for partially supporting this work. Contact her at nelly@acm.org.

Robert B. France is a professor in the Department of Computer Science at Colorado State University. His research interests include model-driven development, aspect-oriented development, and formal methods. He received a PhD in computer science from Massey University, New Zealand. Contact him at france@cs.colostate.edu.



Selected CS articles and columns are available for free at <http://ComputingNow.computer.org>