Semi-Automated Simulation Transformation for DDDAS

David Brogan, Paul Reynolds, Robert Bartholet, Joseph Carnahan, and Yannick Loitière

Computer Science Department
University of Virginia

Abstract. Ultimate DDDAS success demands that DDDAS simulations be increasingly reconfigurable and adaptable to a growing variety of runtime sensor feedback. Because we expect a simulation's requirements to change during its lifetime, a new emphasis is placed on designing simulations that are prepared for transformation. In this paper, we address this new interest in designing for transformation. Our technology combines the specialized insight of simulation designers with the principled application of automation techniques to capitalize on untapped human and computational resources. In addressing simulation transformation issues that arise at design time, composition time, and runtime, we demonstrate how a semi-automated process impacts the entire simulation life cycle. The resulting suite of simulation transformation tools supports the crosscutting needs of DDDAS practitioners.

1 Introduction

Dynamic, data-driven application systems (DDDAS) pose challenges that reflect the core of simulation technology's hardest problems: composition, design for reuse, multiresolution modeling, interoperability, and scenario-driven adaptation. DDDAS's complexity, compounded by the uncertainty with which data and resources will be available, requires adaptability - a capability to adjust to match runtime conditions. Many DDDAS research efforts have produced adaptability results suitable for the application under study, and some span even a wider set of applications. We are investigating critical crosscutting adaptability issues that are independent of any particular application. We report on the COERCE effort now underway at the University of Virginia, where our primary focus is a semi-automated transformation process intended to support dynamic simulation modification and refinement to match runtime conditions.

We are exploring simulation transformation because the anticipation of and provision for all specific uses of a simulation, over its lifetime, is generally unattainable. A better method for addressing the uncertainty inherent in DDDAS is to identify effective means to control a simulation and to explore and model the impact of those controls. Because there are so many things that can impact simulation behavior, a designer must identify the specific attributes that permit the

International Conference on Computational Science, pp. 721–728, 2005.

greatest exploration of "structural uncertainty" [1] using semi-automated methods. These explorations can be cataloged using machine learning and utilized at runtime to adapt to unexpected conditions. Because we anticipate DDDAS applications will routinely combine multiple simulation components to achieve a larger DDDAS objective, we are also investigating the design of components with transformation in mind.

We first describe a method for capturing designer knowledge in language constructs at the outset of a simulation's design in order to enable the creation of logically tractable and easily transformed instances of a simulation. We differentiate between capturing the nature of unexpected conditions, which we believe is possible, and the specifics of those conditions, which experience has shown to be difficult to impossible. We argue that capturing the nature of abstract conditions is useful in many ways: 1) Later users can knowledgeably review applicability of a simulation to a specific set of DDDAS conditions, 2) If new conditions merit simulation transformation, those involved will have the best information available to support the effort, and 3) If future transformation is possible, DDDAS simulations can be instantiated to accommodate narrow conditions, thus increasing expectations for cost savings.

Next we present our investigation of "applied component synthesis" - our approach to component-based simulation that keeps the practitioner's objectives in mind. We specifically consider the synthesis of transformable components. All of the component synthesis cost analyses we have seen to date do not take the potential for transformation into account when considering the cost of synthesis. We do. We present our work in the context of contemporary composition and software reuse results from the software engineering and simulation communities. Our analysis of contemporary results identifies the sources of complexity when dynamically composing or reusing simulations and motivates the need for adaptable simulations.

Lastly, we describe a simulation transformation tool that automatically steers a simulation towards desired outputs. These desired outputs typically come from external observations and they serve as corrective feedback for simulations. Without anticipation of such observations, the simulation will not have a preconfigured feedback loop to meet the desired outputs. We demonstrate that offline experimentation can produce an abstraction of simulation behavior that serves to chart a course from the runtime simulation's current output to a desired output.

Our results align well with the four classes of DDDAS research challenges identified by Darema [2]: applications, mathematical algorithms, systems software, and measurements. Our presentation of ways designer insight can be captured and expressed through programming language primitives provides a "multi-modal method for describing the system at hand" in order to handle dynamically injected data. We address the needs of applications to be dynamically composed from families of models according to streamed data and our automatic simulation transformation tool permits well-behaved, runtime steering of a simulation to accommodate external observations.

2 Building transformable simulations

Modular and adaptable software has been the center of software engineering research for decades. However, software reuse remains time consuming and difficult to automate outside of specific application domains [3–5]. We view simulation as another potentially fruitful domain. We posit that designers possess critical knowledge about a simulation's flexibility and limitations at design time, even if runtime circumstances are unpredictable. We focus on methods for capturing this knowledge so transformation can be facilitated. We place special emphasis on capturing insights about time-management and event-generation algorithms –the most commonplace and important of a simulation's attributes [6]. This focus distinguishes our work from mainstream software reuse research.

2.1 Capturing designer insight

Simulations have characteristic elements that lend themselves to automated exploration and manipulation, which we call "flexible points" [7]. Many of these flexible points reflect the design decisions that were made to construct the simulation, such as the use of stochastic sampling, the selection of input distributions, and the level of detail included in the model. In order to facilitate the process of simulation transformation, we are developing categories of flexible points [8]:

- Decisions made during conceptual design of a simulation
- Decisions made regarding low-level implementation details
- Opportunities to select a variable or function assignment from a set
- Opportunities to select a variable or function assignment from an ordered set (for automatic evaluation using local search techniques)
- Opportunities to reorder events
- Opportunities to adjust level of abstraction or remove aspects

Useful information to record for each flexible point includes how its value could be changed (by changing a parameter or by executing an alternative section of code) and what effects the change will have on the simulation's behavior.

Different types of flexible points and different types of information that must be captured define a two-dimensional space. We use language constructs to encode the flexible point descriptions in the simulation code itself. Metalanguages such as XML, for example, can then parse these constructs and use existing libraries and tools to generate, display, and manipulate the flexible points. This simplifies the development of simulation prototypes and the evaluation of the usefulness of different types of flexible points for DDDAS.

2.2 Temporal/Event Manipulation

An attribute that distinguishes simulations from general-purpose software is the role of time and events in dictating the program's semantics. It is possible to represent a simulation as a (possibly repeating) timeline of events or intervals,

reflecting the way the simulated phenomenon itself occurs in time. We are exploring possible sets of operations that can be performed on timelines, such as looping, splicing, and overlaying timelines. From another perspective, simulations can be viewed as sequences of event generation actions (as opposed to event graphs, as captured in [9]). We are developing a transformation language for "event-generating systems" wherein the nature of an event generation may not be fully determined until the time when it is created by a previous event in the simulation.

One of the current difficulties with transformation of DDDAS simulations is the risk that the data may call for an unexpected series of changes, pushing the simulation out of the domain in which it was originally validated. However, by developing a set of well-specified operations on timelines or event systems, it becomes feasible to analyze a set of transformations on a simulation in advance. Without knowing exactly which operations will be applied or in which order, we can still explore whether or not the validity of the simulation is preserved under each of the operations in the transformation language. Such explorations could be conducted using model checking, e.g. [5] , for example. Then it becomes possible to dynamically apply analyzed transformations in response to external observations without risking the stability of the simulation system.

3 Composability

A crosscutting research challenge in DDDAS is the capability to dynamically select and compose models based on runtime conditions [2]. We are investigating model composition. Past research indicates that in the general case component selection is inherently intractable, but we believe there are opportunities to relax typical assumptions in order to reduce the problem to one that is practical in applied modeling systems. We have discovered a relaxed form of composition that is practical, in the context of an efficient means of transformation. We have labeled our approach "applied component synthesis."

3.1 Software Composition

In the software engineering community, composability is typically discussed within the framework of component-based software design (CBSD). Primary CSBD models include Microsoft's Component Object Model Plus (COM+), the Object Management Group's (OMG) Common Object Request Broker Architecture (CORBA), and Sun's Enterprise JavaBeans (EJB). These technologies are similar in that they enforce a binary structure for exposing public interfaces allowing components to provide services to clients, which may or may not themselves be components. While CBSD technologies provide the facilities to communicate and provide services, they make no guarantees about the meaning, reliability, or consistency of the exchanged information.

In his widely read text on component software, Szyperski defines a composition as an "assembly of parts (components) into a whole (a composite) without

modifying the parts" [10], which is consistent with the software engineering view of composition as functional composition – composed black boxes with clearly defined interfaces. Common exemplars are graphical user interfaces (GUIs), mathematical libraries, and UNIX tools. Functional composition is very restrictive for the modeling domain where practitioners typically want to view components as white boxes, and to reason about the semantics of the internals. We are investigating the means to reason about the semantics of models so formal methods of simulation composition can ensure semantic compatibility is preserved [6].

3.2 Simulation Composition

In the simulation community, composability has been defined as "the capability to select and assemble simulation components in various combinations into valid simulation systems to satisfy specific user requirements" [11]. In contrast to Szyperski's definition, this definition admits reasonable amounts of model transformation. Within the simulation community both composability and interoperability have been topics of intense study. Composability prohibits the use of substantial integration efforts to combine components to meet requirements. Interoperability, on the other hand, permits a one-time integration effort as is commonly used in Distributed Interactive Simulation (DIS), Aggregate Level Simulation Protocol (ALSP), and the High Level Architecture (HLA). These technologies provide practical tools for interoperability, but with the same drawbacks seen in current CBSD technologies.

Recent interest in the M&S community has been focused on the simulation composability component selection problem [12–14]. Informally, the component selection problem seeks to find the subset of preexisting components that meet a simulation's objectives (requirements). The component selection problem does not address changing objectives, rather it assumes that all objectives are known at selection time. We believe a more flexible approach, recognizing the reality of changing requirements, is needed. Furthermore, component selection, as currently framed, assumes that components, to be composable, must be both syntactically and semantically composable. We agree with Page et al. that the easier problem is syntactic composability [15] and composability efforts should focus on the hard problem of semantic composability.

Only one theoretical model of simulation composability has been proposed to date [16, 17]. The authors define models as functions, and simulations as the process of stepping through the functions, with each step allowing the model to receive input, give output, and change state. Although this model addresses functional composability and semantic correctness, we find it too limited by traditional functional composition assumptions (e.g. no commutativity).

3.3 Breaking the composability paradigm

We reject the common assumption of immutable components. We do not agree with the assumption that a master set of components exists that can satisfy all possible requirements. In the domain of computational sciences, we believe

practitioners, too, will not permit themselves to be limited by such an assumption. A more practical solution permits transforming existing components when a composition is sufficiently close to satisfying all requirements. Component transformation is a particularly compelling technology when considering the challenge of adapting to requirements that change at runtime.

We are currently investigating a new paradigm of simulation reuse, applied component synthesis, that relies on efficient transformation. Applied component synthesis requires a formal model describing an iterative process of selecting, transforming, creating, validating, and assembling simulation components to meet changing requirements. Initial work regarding component selection algorithms is underway as is the investigation of metrics and means that will guide practitioners in component synthesis. We envision developers making decisions on the importance of different aspects of time and resources, encoding these decisions, and then allowing this encoding to facilitate the process of component selection and transformation.

Our initial results include a new model for the component selection problem – a model that permits consideration of component transformation in a component selection cost analysis. We have demonstrated that cases of this problem are NP hard, and others are polynomial time. We are utilizing our results to identify heuristics and to conduct further reconsiderations of assumptions.

4 Automatic Behavior Modeling

Incorporating data into an executing simulation has the potential to improve the simulation's utility by identifying and correcting errors, steering the simulation towards more effective states, or triggering the reconfiguration of the simulation's underlying computational components. In each of these cases, the infused data may trigger large corrections in the simulation state, which can have undesirable side effects when internal simulation variables are not changed accordingly. Such instabilities should be expected in complex simulation systems because they are composed of a dynamic set of interacting software components, each possessing internal variables and non-linear equations. Because state changes of large magnitude are common sources of such instabilities, we propose a method where a desired change in state is subdivided into multiple smaller changes that avoid simulation discontinuities and perturbations.

Consider how runtime data may reveal that the output O of a simulation in state S is not tracking actual conditions O' of the world and a new simulation state S' will be required to realign the simulation. The simulation (S, O) must be transformed to (S', O'). Recalling that a simulation's variables cannot directly be modified without causing unforeseen complications, we seek an iterative transformation process that permits the simulation to update all state and internal variables synchronously. The challenge becomes one of finding a sequence of states $S_1, S_2, ..., S_n$ that incrementally takes the simulation output from O to O' while avoiding the problems caused by the internal variables. We require a model that provides a means to select the intermediate steps between

O and O' and permits us to map them to the corresponding $S_1, ..., S_n$ that generate them. To build this model, we use a data-driven approach that uses simulation observations acquired during typical execution of the simulation. A particular feature of this model will be to constrain interpolation to outputs that have been observed so that the sequence between O and O' will be supported by valid simulation states.

We use a modeling tool called self-organizing maps to accomplish our simulation transformation goals. Self-organizing maps are a family of neural network-based clustering algorithms developed by Teuvo Kohonen in the 80s [18]. The self-organizing map uses a network of neural network nodes to record simulation outputs observed during a training period and it generates a similarity relationship such that similar outputs get placed in neighboring locations on the map. We benefit from this in that we can use locally guided path finding algorithms to search for an acceptable path from O to O'. While training the self-organizing map, each map node is associated with the simulation outputs that it best matches and their corresponding simulation states. From this mapping, a path through the self-organizing map's output space can be used to generate the desired $S_1, ..., S_n$.

To test this data-driven simulation transformation process, we implemented a curve-plotting program that generates non-linear and counterintuitive relationships between the simulation state and its output. The simulation state is a vector of five continuous parameters and the output is a 64x64-pixel image of a parametric curve. A fully connected 10x10 self-organizing map was trained on a set of 1,000 output images generated by a parameter sweep through four of the five state parameters. Associated with each output image placed in the map is the image's corresponding parameter setting. Given a pair of source and target images, we can then use the map to generate a path (using any path finding algorithm) of similar images between them. In ongoing work, the corresponding sequence of simulation states will be generated to smoothly transition from the source simulation state to a new state satisfying the output criteria.

5 Conclusion

In this paper, we have presented three research projects underway at the University of Virginia in support of the COERCE effort. These projects are united in their combined use of simulation designers and automation to address transformation challenges that would otherwise be insurmountable. Each simulation transformation project addresses a different point in the simulation life cycle and addresses the crosscutting needs of the DDDAS community. We will continue to pursue ways a simulation designer's insight and automation can be brought to bear on transformation's challenges.

Acknowledgments. The authors gratefully acknowledge the support of the NSF under grant 0426971.

References

1. Davis, P.: Exploratory analysis enabled by multiresolution multiperspective modeling. In: Proceedings of the Winter Simulation Conference. (2000)
2. Darema, F.: Dynamic data driven application systems: A new paradigm for application simulations and measurements. In: Proceedings of the International Conference on Computational Science. (2004)
3. Garlan, D., Allen, R., Ockerbloom, J.: Architectural mismatch or why it's hard to build systems out of existing parts. In: Proceedings of the International conference on Software engineering. (1995) 179–185
4. Sullivan, K.J., Knight, J.C.: Experience assessing an architectural approach to large-scale systematic reuse. In: Proceedings of the international conference on Software engineering. (1996) 220–229
5. Batory, D., Johnson, C., MacDonald, B., von Heeder, D.: Achieving extensibility through product-lines and domain-specific languages: a case study. In: ACM Trans. Softw. Eng. Methodol. Volume 11. (2002) 191–214
6. Bartholet, R.G., Reynolds, P.F., Brogan, D.C., Carnahan, J.C.: In search of the philosopher's stone: Simulation composability versus component-based software design. In: Proceedings of the Fall Simulation Interoperability Workshop. (2004)
7. Carnahan, J.C., Reynolds, P.F., Brogan, D.C.: Visualizing coercible simulations. In: Proceedings of the Winter Simulation Conference, Institute of Electrical and Electronics Engineers, Inc. (2004) 411–419
8. Carnahan, J.C., Reynolds, P.F., Brogan, D.C.: Language support for identifying flexible points in coercible simulations. In: Proceedings of the Fall Simulation Interoperability Workshop. (2004)
9. Schruben, L.: Simulation modeling with event graphs. In: Commun. ACM. Volume 26. (1983) 957–963
10. Szyperski, C.: Component Software: Beyond Object-Oriented Programming. 2d edn. Addison-Wesley (2002)
11. Petty, M.D., Weisel, E.W.: A composability lexicon. In: Proceedings of the Spring Simulation Interoperabilty Workshop. (2003)
12. Page, E.H., Opper, J.M.: Observations on the complexity of composable simulation. In: Proceedings of the Winter Simulation Conference. (1999)
13. Petty, M.D., Weisel, E.W., Mielke, R.R.: Computational complexity of selecting components for composition. In: Proceedings of the Fall Simulation Interoperability Workshop. (2003)
14. Fox, M.R., Brogan, D.C., Reynolds, Jr., P.F.: Approximating component selection. In: Proceedings of the Winter Simulation Conference. (2004)
15. Page, E.H., Briggs, R., Tufarolo, J.A.: Toward a family of maturity models for the simulation interconnection problem. In: Proceedings of the Spring Simulation Interoperability Workshop. (2004)
16. Petty, M.D., Weisel, E.W.: A formal basis for a theory of semantic composability. In: Proceedings of the Spring Simulation Interoperability Workshop. (2003)
17. Weisel, E.W., Petty, M.D., Mielke, R.R.: Validity of models and classes of models in semantic composability. In: Proceedings of the Fall Simulation Interoperability Workshop. (2003)
18. Kohonen, T.: Self-Organizing Maps. Springer (1997)