Proceedings of the 2006 Winter Simulation Conference
L. F. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D. M. Nicol, and R. M. Fujimoto, eds.

REQUIREMENTS FOR DDDAS FLEXIBLE POINT SUPPORT

Joseph C. Carnahan
Paul F. Reynolds, Jr.

Modeling and Simulation Technology Research Initiative
151 Engineer's Way, PO Box 400740
Department of Computer Science, University of Virginia
Charlottesville, VA 22904-4740, U.S.A.

ABSTRACT

Dynamic data-driven application systems (DDDAS) integrate computer simulations with experimental observations to study phenomena with greater speed and accuracy than could be achieved by either experimentation or simulation alone. One of the key challenges behind DDDAS is automatically adapting simulations when experimental data indicates that a simulation must change. Coercion is a semi-automated simulation adaptation approach that can be automated further if elements of the simulation called flexible points are described in advance. In this paper, we use a number of DDDAS adaptation examples to identify the information that needs to be captured about flexible points in order to support coercion.

1 INTRODUCTION

Experimental data has always been one of the most valuable resources for building and validating computer simulations. Simulations are often used to understand phenomena that have not been observed in the real world. However, when real-world experimental data is available, it is usually expected that the simulation agrees with the data. As more experiments are conducted, the simulation must be corrected to reflect each experiment. This leads to an iterative process where

1. a simulation is run to gain insight about a phenomenon,
2. this insight is used to determine what new observations should be collected, and
3. the simulation is adapted to reflect these observations.

Dynamic data-driven application systems (DDDAS) aim to improve this process by automating steps 2 and 3 (Darema 2004). Instead of requiring a subject matter expert (SME) to analyze the simulation output and determine what new experiments should be conducted, the system would perform this interpretation automatically. Likewise, instead of expecting developers to manually modify the simulation code, the adaptation step would also be automated.

Dynamically producing valid simulation adaptations in response to new experimental data poses a significant challenge. Current simulation adaptation techniques usually involve either manually modifying the code or applying optimization to change the value of one or more of the simulation's parameters. Unfortunately, manually modifying the simulation code does not change a simulation fast enough to meet DDDAS' needs. Meanwhile, parameter tuning requires the developers of the simulation to provide parameters to control every possible aspect of the simulation that might be affected by new data. Generalizing software to anticipate every possible way it could change is difficult, and attempting to do so usually interferes with performance and makes the code unmanageably complex (Parnas 1979). Also, even if parameters can be used change the simulation's behavior, it may not be clear which ones need to change in response to the new data nor what all the consequences of changing them might be.

For DDDAS to work, we need improved methods for automatically adapting simulations. In general, automating software adaptation is difficult (Bartholet, Reynolds, and Brogan 2005). However, software engineers have observed that by taking advantage of the properties of a particular domain, the problem of software adaptation can be simplified (Czarnecki and Eisenecker 1999). Most simulations fall into a domain that we call *coercible software*, which has several properties that affect their flexibility (Carnahan, Reynolds, and Brogan 2005).

1.1 Automating Adaptation with Coercion

Coercible software is distinguished by the presence of *flexible points*, which reflect choices in the design of the software about how to represent phenomena outside of the software. Most software implicitly models the context in which it is expected to operate. However, in coercible software, decisions about context modeling tend to be more flexible. For example, in a hardware device driver, the software must issue exactly the commands that the device expects. In contrast, a simulation of a device could represent the device in many different ways, using either a detailed model that tracks each of the device's components or a simple model that replaces the device with a black box that generates events according to a random distribution. At the same time, the choice of a model is not arbitrary. It is made with the simulation's requirements in mind, and it is constrained by expert knowledge about the subject being simulated.

This combination of flexibility and constraints makes it possible to automate many parts of simulation adaptation. Without flexibility, automatic adaptation is impossible because there is no way to know what alternatives should be considered. Without constraints, automatic adaptation is infeasible because there are too many alternatives to consider all of them in a timely fashion. *Coercion* is a semi-automated adaptation approach that exploits the flexibility and constraints of model abstraction opportunities to automate simulation adaptation. Coercion has been demonstrated in several successful experiments (Drewry, Reynolds, and Emanuel 2002; Carnahan, Reynolds, and Brogan 2003). By capturing expert insight about flexible points in formally defined constructs, coercion can be automated even further (Carnahan, Reynolds, and Brogan 2004).

1.2 Applying Coercion to DDDAS

In this paper, we explore coercion as a means to accomplish the simulation adaptation step of DDDAS. By searching for the best abstractions to select at each flexible point, coercion offers the automation that DDDAS requires. At the same time, flexible points can be used to describe complex changes more elegantly than straightforward parameters. Also, by leveraging SME insight about the simulated phenomenon, coercion can also be more efficient than naive parameter sweeps. Hence, coercion provides an excellent starting point for the simulation adaptation component of DDDAS.

As it currently stands, coercion is still a semi-automated adaptation approach, relying on SME input to select flexible points and guide the search process. For coercion to be used in DDDAS, it will be necessary to capture this SME insight in advance, allowing coercion to take advantage of this insight as soon as new data arrives. To accomplish this, formal language constructs for describing flexible points must be developed. Using a number of DDDAS examples as case studies, we have identified key requirements for flexible point language constructs. These include

- connecting flexible points to the part of the model that they represent,
- capturing both direct and indirect effects of flexible points on simulation behavior,
- identifying constraints on how flexible points can change,
- providing implementations so that flexible point changes can be applied automatically, and
- being able to add and remove information about flexible point effects and interactions.

With language tools that meet these requirements, we can use coercion to adapt simulations in support of DDDAS.

2 DDDAS EXAMPLE: SUPPLY CHAIN SIMULATION

To better understand how coercion can be used to adapt simulations for DDDAS, we have analyzed simulations in several domains, including combustion modeling (Zambon 2005) and high-energy physics. For this paper, we will concentrate on a single example domain, namely make-to-order supply chains (Strader, Lin, and Shaw 1998; Ball and Fu 2006). In these systems, customers design the products that they want to order as they order them. The large number of different product configurations makes it prohibitively expensive to keep all of the possible configurations in stock. Therefore, we would like to use simulation to predict which products will be ordered and what parts will be needed in the near future.

The supply chain simulation becomes a DDDAS when we integrate the simulation with incoming reports about customer ordering and product shipping. The simulation would typically be built using past data, with the assumption that purchasing patterns remain roughly the same from year to year. However, factors like changes in the economy or introduction of new products could render the old data obsolete. To maintain the validity of this simulation, it must be continually updated using available data from customer orders and factory status reports.

We have chosen to use the supply chain example in this paper for several reasons. First, many quantities in the system are discrete, allowing us to change how they are computed without concern for introducing floating-point precision errors. Second, there are a number of obvious ways to collect useful data about a supply chain, including reports from store managers, accounting data, and marketing research. All of this data would already be collected as good business practice. Lastly, the supply chain example is relatively accessible, making it a better vehicle for presenting insights that generalize to other DDDAS simulation adaptation problems.

2.1 Details of the Supply Chain Example

An example supply chain is diagrammed in Figure 1. This simple example includes three kinds of parts that are used in assembling products, two plants where products are assembled, and five stores where products are ordered and delivered. The system works as follows:

1. A customer places an order at a store for an instance of a particular product. The order includes a list of included parts and a due date when the product should be delivered.
2. The store handles the order. If an instance of the product is not in the store's inventory, then the store must order at least one instance of the product from its assembly plant. If the store already has the needed product, it may still place an order to the plant to replace the product that is about to be removed from its inventory.
3. When an assembly plant gets an order from a store, it selects the parts needed for the product from its inventory, assembles the product, and ships it to the store. Depending on how many of each part remain in the inventory, the plant may place an order to its suppliers for more parts.
4. Finally, when the order's due date is reached, an instance of the product is removed from the store's inventory. If the store does not have the product, then the store is assessed a penalty to reflect the cost of paying for rushed shipping or the indirect cost of upsetting customers.

In addition to the penalty for delivering late products, there is also a holding cost for keeping items in inventory. Therefore, we would like to find the ideal inventory policies that minimize both the size of the inventories and the number of late product deliveries.

2.2 Modeling the Supply Chain

To simulate the supply chain, we must develop a conceptual model. Event graphs are a well-accepted formalism for discrete event simulation (Schruben 1983). An event graph model of the supply chain is given in Figure 2. From this model, code for an implementation could be generated and run to evaluate different inventory policies. However, a SME on supply chain design is more likely to provide information in terms of the events in this graph than in terms of the generated code.

Note that we have not filled in all of the details of this formal model. For brevity, we have omitted the changes to state variables that occur at each node. Also, note that we could have used a variety of other formalisms, such as process-based modeling, Petri nets, or other representations.

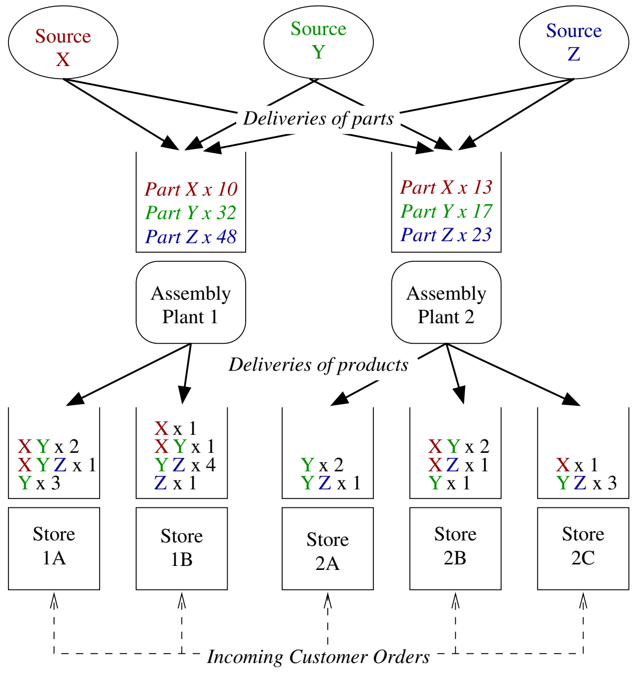

Figure 1: An Example Supply Chain

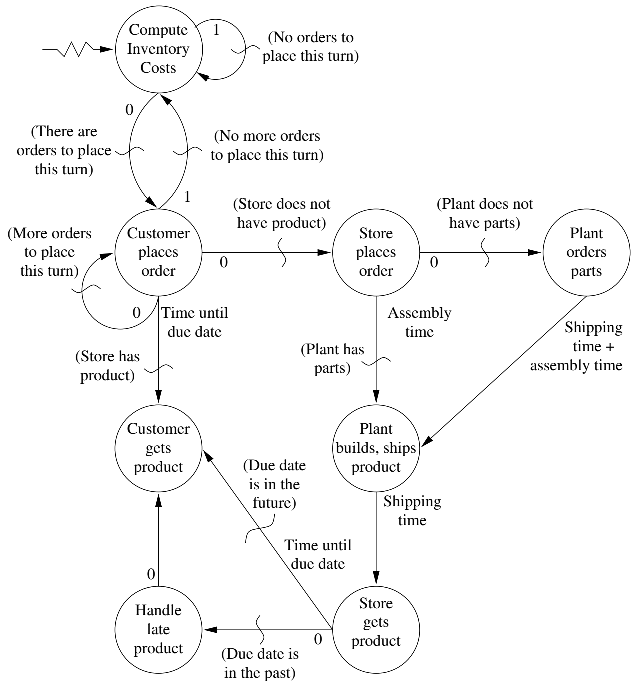
Compute Inventory Costs
1
(No orders to place this turn)
0
(There are orders to place this turn)
(No more orders to place this turn)
1
Customer places order
(More orders to place this turn)
0
(Store does not have product)
0
Store places order
(Plant does not have parts)
0
Plant orders parts
Time until due date
(Store has product)
Assembly time
(Plant has parts)
Shipping time + assembly time
Customer gets product
(Due date is in the future)
Plant builds, ships product
Shipping time
Time until due date
0
Handle late product
0
(Due date is in the past)
Store gets product

Figure 2: Event Graph for the Supply Chain Model

The purpose of this example model is to provide a context for discussing simulation adaptations: As coercion leverages SME insight to automatically adapt simulation code, these adaptations need to be described in terms of a conceptual model that a SME might use.

2.3 Abstraction Opportunities in the Supply Chain Model

There are a number of opportunities for selecting different abstractions in a model of the supply chain described in Figure 2. These model abstraction opportunities include deciding how to represent

- the assembly process,
- the shipping of parts and products,
- customer ordering behavior, and
- the consequences of late deliveries.

Depending on the details of the situation, each of these aspects of the system will have different effects on inventory costs and late product deliveries. Those parts of the system that have the most significant effects need to be modeled both accurately and precisely if the simulation is to be used to evaluate inventory policies and other business decisions.

To build a simulation of the supply chain, a particular abstraction must be selected for each of these model abstraction opportunities. The implementation of each abstraction is a flexible point, meaning that we may want to reevaluate that part of the simulation later when new data prompts us to change our choice of model abstractions.

Table 1 describes some of the flexible points in the supply chain simulation. Let us examine one flexible point as an example, namely *Customer Ordering Behavior*. When building the simulation, we realize that accurately modeling customer ordering behavior will be crucial for minimizing late product deliveries. However, we may not know what structure or equations to use to represent customer ordering. Can we afford to model the number of orders on a given day as a constant, or should we run the simulation for each day many times with randomly chosen numbers of orders? If numbers of orders are picked randomly, then what distribution should we use? Are ordering behaviors more similar to the previous day's orders, the same day's orders on a previous year, or some combination of the two? Each of these alternatives is a possible *binding* for the customer-ordering flexible point.

Note that some flexible points are subsumed by other flexible points. For instance, the question of what distribution to use for determining the rate of orders at each store is only raised when the *Random Ordering* binding is selected for the *Customer Ordering Behavior* flexible point.

3 REQUIREMENTS FOR COERCING THE SUPPLY CHAIN SIMULATION

We are interested in flexible points in the supply chain simulation because we anticipate that new data will require the simulation to adapt automatically. In this section, we consider a number of these adaptations and discuss how simulation coercion could be used to accomplish them. Then, we generalize from these examples to identify information about flexible points that would be needed for coercion to work effectively in DDDAS.

3.1 Correcting Inventory Cost Predictions

Suppose we use the supply chain simulation to select a policy for ordering parts at each assembly plant. We do this because the simulation indicates that a particular policy will minimize average inventory costs. However, accounting reports indicate that the real inventory costs are higher than predicted. This triggers a search for why the inventory cost predictions are incorrect. We will use coercion to adapt the simulation, bringing its predictions into agreement with the real inventory cost data.

First, to leverage SME insight in coercing any simulation, flexible points must be identified in terms that the SME can understand. In the supply chain example, the SME can recognize that inventory costs are affected by specific events in the conceptual model, such as "Customer places order", "Plant orders parts", and "Compute inventory costs". In turn, the coercion process needs to be able to apply this information to automatically determine which flexible points must change, such as *Customer Ordering Behavior*, *Shipping Model*, and *Inventory Unit Cost*.

This leads to the first requirement for flexible point descriptions:

Requirement 1 *Flexible points must be described in terms of changes to the model in a way that can be easily understood by subject matter experts in the domain of the simulation.*

To correct the inventory cost predictions, the coercion process must be able to determine which flexible points affect the inventory cost calculation. This leads to another flexible point description requirement:

Requirement 2 *The effects of changing flexible points must be captured, including*

1. *which simulation variables may take on different values,*
2. *what non-functional characteristics (e.g. performance) may be affected, and*
3. *how the conceptual model may need to change.*

Flexible point effects on variables are of two kinds: direct and indirect. A flexible point has a *direct effect* on a simulation variable if that variable's value is changed by the code associated with the flexible point. Likewise, it has an *indirect effect* on a variable if the variable is not changed in the flexible point's code but is changed through some chain of intermediate variables or other flexible points. These networks of effects can be relatively complex, as depicted in Figure 3.

Flexible Point	*Possible Bindings*	*Related Model Events*
Late Delivery Modeling	Lose customers Pay a flat cost	"Handle late product" and "Customer places order"
Shipping Model	Constant delay Uniform random delay Exponential delay Uniform chance of loss	"Plant orders parts", "Plant builds, ships product", and "Store gets product"
Customer Ordering Behavior	Scripted Random	"Customer places order" and associated edges
Random Ordering Distribution	Normal Uniform Exponential	"Customer places order"

Table 1: A Selection of Flexible Points and Bindings from the Supply Chain Simulation

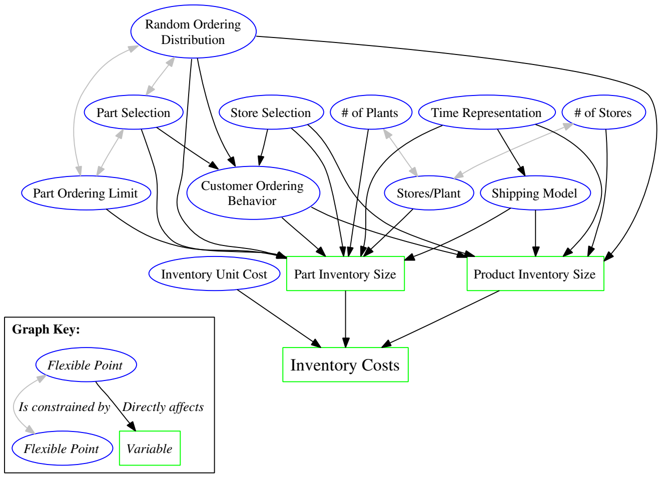

Figure 3: Flexible Points and Relationships to Inventory Cost Predictions

Other flexible point effects of interest include changes to the conceptual model and changes to non-functional characteristics, such as performance and memory requirements. For instance, a possible flexible point change may need to be ruled out because of its effects on performance, while another flexible point may be ruled out because it changing it would remove an event from the model that the SME considers to be essential.

Given that so many flexible points have an impact on the inventory costs, the coercion process needs as may ways as possible to constrain the search for the right set of flexible point bindings. This motivates the next flexible point description requirement:

Requirement 3 *Coercion needs information about flexible point constraints to guide its search for automatic adaptations.*

These constraints can come in several forms. First, we may have expert insight on which flexible points or parts of the model have the largest effects on inventory costs. If the *Shipping Model* flexible point is known to have a greater effect on part inventory sizes than the *Part Ordering Rate*, then the search for a better simulation should start by examining how shipping delays are modeled. Second, some flexible points may make assumptions that depend on other flexible points. For instance, we cannot change the ratio of stores to assembly plants without also changing either the number of stores or the number of plants. Other constraints may be provided by the incoming data, such as information that per-unit inventory costs are correct and should not be changed.

3.2 Handling Variations in Store Popularity

Different stores at the end of the supply chain each handle different numbers of orders each day, which affects the amount of inventory that each store has to maintain. Suppose that we try to simulate the variation between stores by picking a distribution and randomly choosing where to place each order in the simulation based on this distribution. This would allow us to evaluate how well our store inventory policies adapt to some stores being more popular than others.

Eventually, sales data may indicate that stores do not always stay popular or unpopular, as different stores do more business at different times of the year. Our adaptive inventory policies might already handle these variations optimally, but we cannot be certain because the simulation currently represents customer ordering behavior as fixed over time.

Customer Ordering Behavior is a flexible point that we have already identified in this simulation (see Table 1) in connection with the "Customer places order" event (see Figure 2). In order to explore other abstractions for customer ordering, the coercion process must have access to information about other possible valid abstractions. For instance, a subject matter expert might be able to tell us that customer ordering behavior shifts gradually over time. To prevent ordering behavior from changing too quickly, orders at time t would have to be a function of orders at time $t-1$. Also, for comparison of alternative customer ordering abstractions to occur during the coercion process, the implementations of these alternatives must already be available.

Requirement 4 *For analysis of different flexible point alternatives to occur during coercion, alternatives must already be implemented in such a way that they can automatically replace the simulation's current flexible point bindings.*

Requirement 4 could be accomplished a number of ways, such as with object oriented subtyping (Rentsch 1982) or aspects (Kiczales et al. 1997). It could also be accomplished by describing changes to the model (such as adding new event graph nodes and arcs) and automatically generating code from the new model. One way or another, coercion needs a way to automatically integrate its changes with the rest of the simulation code.

3.3 Simulating Customer Reactions to Late Deliveries

Another source of data in the supply chain simulation is marketing research. Suppose we are currently simulating the penalty for late deliveries as a flat cost for each delivery, which could represent either rush-order shipping costs or a discount offered to the customer on late products. However, suppose surveys indicate that customers who receive products late will probably never shop at our stores again. This could have more dire consequences than a flat cost, as it could mean that we are driving customers away faster than we can attract new ones.

This marketing data indicates that our representations of the "Handle late product" and "Customer places order" events need to change (see Figure 2). We have already identified flexible points associated with these events, namely *Customer Ordering Behavior* and *Late Delivery Modeling*. To change these flexible points, the coercion process still needs access to alternative algorithms for representing customer ordering that let the number of customers change over time (Requirement 4). There is also a new dependency between these two flexible points, where representing the number of customers as increasing is only valid if the number of late deliveries is kept below a certain level. To keep track of new dependencies, the coercion process must be capable of verifying or updating the information about flexible points after applying each adaptation.

Requirement 5 *It must be possible to add and remove information about flexible points during and after coercion.*

4 DISCUSSION

As a semi-automated adaptation process, simulation coercion can be improved by increasing the fraction of adaptations that can be accomplished automatically. DDDAS calls for fully-automatic adaptation of simulations in response to experimental data, which defines a goal for coercion technology research. Based on the example DDDAS problems considered in this paper, we observe that the following information about flexible points will be required for the coercion process to proceed automatically:

1. the relationship of the flexible point to the SME's conceptual model,
2. the direct and indirect effects of changing a flexible point,
3. any constraints on how the flexible point may be changed,
4. a description of how to automatically replace the implementation of one flexible point binding with another, and
5. how changing one flexible point potentially affects information that we have captured about other flexible points.

To meet the requirements, we are designing a *coercible software development framework*. This framework includes both the visualization and user interface tools that enable SMEs to describe flexible points and the language tools that enable coercion to automatically process this information and leverage it in performing simulation adaptations. By supporting automatic adaptation through coercion, this framework supports the development of DDDAS.

4.1 Related Work

Coercion is not the only approach to automatic simulation adaptation. *Compositional modeling* uses a similar approach of explicitly identifying model abstractions to search for the right set of equations to use in a simulation (Falkenhainer and Forbus 1991). Compositional modeling supports automatic model construction in polynomial time, given the assumption that the user has complete knowledge of all equations that could ever be used in the model (Nayak and Joskowicz 1996). Unfortunately, compositional modeling's assumptions are too strict for most DDDAS applications, meaning that a more flexible approach is needed.

A number of DDDAS-related research projects are currently underway, including applications of DDDAS to oceanography, geography, structural engineering, meteorology, thermodynamics, combustion modeling, astronomy, medicine, and many other fields (Douglas 2006). Coercion is a cross-cutting technology that can support all of these fields by providing tools and a framework for performing data-driven simulation adaptations.

4.2 Future Work

Given the requirements for flexible point language constructs presented in this paper, the next step is to evaluate existing languages and tools to determine how well they meet these requirements. For instance, can all of the code changes associated with flexible points be described using aspect-oriented languages, or will something more powerful be needed? How well do current techniques for handling uncertainty work for describing the unpredictable effects of flexible point changes? Identifying the right tools will make it possible to implement the adaptations described in this paper.

We are also investigating other ways to improve coercion. These include finding the best visualization tools for interfacing with SMEs, optimization tools to search over the space of possible flexible point bindings, and verification and validation tools to ensure that automatic adaptations do not have any undesirable and unexpected effects.

4.3 Conclusion

Coercion is a semi-automated simulation adaptation process that leverages subject matter expert insight to efficiently search for the best set of abstractions to use in a model. To automate more of the coercion process, language support is needed for describing simulation flexible points. Considering DDDAS' fully-automated adaptation as a goal, we have identified information about flexible points that must be captured to support coercion. With a framework of tools that meet these requirements, it will be possible to develop coercible simulations that can be automatically adapted to incorporate new experimental data.

ACKNOWLEDGMENTS

We gratefully acknowledge support from the DDDAS program at the National Science Foundation (ITR 0426971), as well as from our colleagues in MaSTRI, the Modeling and Simulation Technology Research Initiative at the University of Virginia.

REFERENCES

Ball, M., and M. Fu. 2006, Jan. Dynamic real-time order promising and fulfillment for global make-to-order supply chains (presentation). In *Workshop on Dynamic Data-Driven Application Systems*. Arlington, Virginia: National Science Foundation.

Bartholet, R. G., P. F. Reynolds, and D. C. Brogan. 2005. The computational complexity of component selection in simulation reuse. In *Proceedings of the 2005 Winter Simulation Conference*, ed. M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines, 2472–2481. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Carnahan, J. C., P. F. Reynolds, and D. C. Brogan. 2003, Dec. An experiment in simulation coercion. In *Proceedings of the 2003 Interservice/Industry Training, Simulation, and Education Conference*. Arlington, Virginia: National Training Systems Association.

Carnahan, J. C., P. F. Reynolds, and D. C. Brogan. 2004, Sep. Language constructs for identifying flexible points in coercible simulations. In *Proceedings of the 2004 Fall Simulation Interoperability Workshop*. Orlando, Florida: Simulation Interoperability Standards Organization.

Carnahan, J. C., P. F. Reynolds, and D. C. Brogan. 2005. Simulation-specific properties and software reuse. In *Proceedings of the 2005 Winter Simulation Conference*, ed. M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines, 2492–2499. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Czarnecki, K., and U. W. Eisenecker. 1999. Components and generative programming (invited paper). In *ESEC/FSE-7: Proceedings of the 7th European Software Engineering Conference*, 2–19. London, UK: Springer-Verlag.

Darema, F. 2004. Dynamic data driven application systems: A new paradigm for application simulations and measurements. In *Computational Science - ICCS 2004: 4th International Conference*, ed. M. Bubak, G. D. van Albada, P. M. A. Sloot, and J. Dongarra, Volume 3038. Heidelberg, Germany: Springer-Verlag.

Douglas, C. C. 2006, March. DDDAS: Dynamic data-driven application systems. Web site. http://www.dddas.org, last visited March 22, 2006.

Drewry, D. T., P. F. Reynolds, and W. R. Emanuel. 2002, Dec. An optimization-based multi-resolution simulation methodology. In *Proceedings of the 2002 Winter Simulation Conference*, ed. E. Yücesan and C.-H. Chen. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Falkenhainer, B., and K. D. Forbus. 1991. Compositional modeling: finding the right model for the job. *Artificial Intelligence* 51 (1-3): 95–143.

Kiczales, G., J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. 1997. Aspect-oriented programming. In *Proceedings of the European Conference on Object-Oriented Programming*, ed. M. Aksit and S. Matsuoka, Volume 1241, 220–242. Heidelberg, Germany: Springer-Verlag.

Nayak, P. P., and L. Joskowicz. 1996. Efficient compositional modeling for generating causal explanations. *Artificial Intelligence* 83 (2): 193–227.

Parnas, D. L. 1979, Mar. Designing software for ease of extension and contraction. *IEEE Transactions on Software Engineering* SE-5:128–138.

Rentsch, T. 1982. Object oriented programming. *SIGPLAN Notices* 17 (9): 51–57.

Schruben, L. 1983. Simulation modeling with event graphs. Volume 26, 957–963: ACM Press.

Strader, T. J., F.-R. Lin, and M. J. Shaw. 1998, Mar. Simulation of order fulfillment in divergent assembly supply chains. *Artificial Societies and Social Simulation* 1 (2).
Zambon, A. C. 2005. *Modeling of thermo-acoustic instabilities in the counterflow flames*. Ph. D. thesis, University of Virginia, Charlottesville, VA.

AUTHOR BIOGRAPHIES

JOSEPH C. CARNAHAN is a Ph.D. Candidate in Computer Science and a member of MaSTRI at the University of Virginia. His interests include software engineering and programming language design as they apply to the challenges of flexible and reusable simulation development. He earned his B.S. in Computer Science at the College of William and Mary and previously worked as a scientist at the Naval Surface Warfare Center, Dahlgren Division. His email address is <carnahan@virginia.edu>.

PAUL F. REYNOLDS, JR. is a Professor of Computer Science and a member of MaSTRI at the University of Virginia. He has conducted research in modeling and simulation for over 25 years, and has published on a variety of M&S topics including parallel and distributed simulation, multiresolution models and coercible simulations. He has advised numerous industrial and government agencies on matters relating to modeling and simulation. He is a plank holder in the DoD High Level Architecture. His email address is <reynolds@virginia.edu>.