VISUALIZING COERCIBLE SIMULATIONS

Joseph C. Carnahan
Paul F. Reynolds, Jr.
David C. Brogan

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

ABSTRACT

The labor intensive aspects of simulation development and maintenance make exploration of reuse essential. However, reuse is generally difficult to achieve in practice due to inflexible assumptions and changing requirements. This paper discusses a technology for simulation reuse called COERCE and the visualization tools that this technology requires. COERCE addresses techniques to make simulations more flexible (coercibility) and the process of transforming a simulation to meet new objectives (coercion). We address the following question: Given that COERCE is a semi-automated technology, what visualization capabilities are necessary to support it? We address this question empirically by studying the role of visualization in the construction of a new coercible simulation and in the coercion of an existing example simulation. Based on this study, we propose a set of requirements for any visualization toolkit meant to support COERCE.

1 INTRODUCTION

A computer simulation is written at a specific time to solve a specific problem, but many simulationists would like to reuse and adapt simulations to serve new purposes. New observational data often becomes available, challenging the validity of a simulation as it was originally developed or offering ways that the simulation could be expanded. Scientists are working to develop dynamic data-driven simulations, which automatically respond to new streams of observational data and incorporate this data in generating new simulation results (Knight 2003). Simulation composability is another important area of research, where users wish to dynamically assemble new simulations using existing simulations as components (Dahmann, Calvin, and Weatherly 1999; Kasputis and Ng 2000; Petty and Wiesel 2003). As with other software, a high level of flexibility and modularity would reduce the costs of developing sim-

ulations by allowing more code to be reused (Mili et al., 2000).

Unfortunately, practical necessity forces simulation designers to build certain assumptions into their simulations. These assumptions include the settings for various simulation parameters, the formulas selected to compute certain values within the simulation, and the format of inputs and outputs to the simulation system. When a simulation is reused in a new setting, these assumptions may no longer be valid.

1.1 COERCE: An Approach to Simulation Reuse

COERCE is a technology for facilitating simulation reuse by addressing the problem of inflexibility in simulations. COERCE is composed of two parts, coercibility and coercion. Simulation coercibility is the study of designing simulations to be more easily coerced. This can be accomplished by enabling the designers to identify flexible points in the simulation. Simulation coercion is the process of manipulating these flexible points, thereby transforming a simulation to meet new requirements. When possible, optimization and other tools are used to partially automate the process (Waziruddin, Brogan, and Reynolds 2003).

In this paper, we explore requirements for visualization tools to support COERCE. Coercing a simulation to meet a new requirement is a semi-automated process: An expert practitioner must be involved to direct the search and provide constraints for the optimization (Reynolds 2002). The expert practitioner directs the process, deciding when to apply optimization techniques and when it is necessary to modify the simulation code directly. When optimization is used, an expert practitioner has to identify precisely which parameters are most relevant to the new requirement and specify ranges of values over which the optimization should search. This semi-automated approach to coercing simulations has been demonstrated successfully by Drewry,
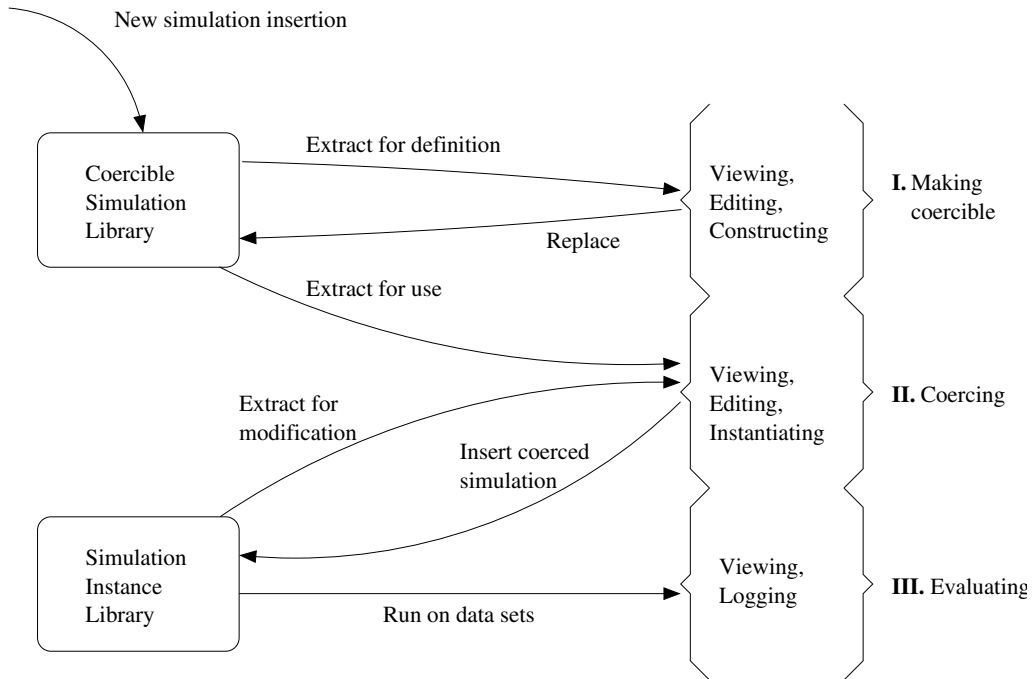
Figure 1: Life Cycle of Coercible Simulations

Reynolds, and Emanuel (2002) and Carnahan, Reynolds, and Brogan (2003).

A natural alternative to our approach to coercibility is the use of parameters for each aspect of the simulation that is expected to change (i.e. a data-driven approach). We have identified the following problems with this approach: 1) The parameter space may become prohibitively large and intractable; 2) A large parameter space can make simulation performance unacceptable; 3) Why simulate for more conditions than planned use may require? 4) Parameterization cannot generally capture the infinitude of patterns, and their effects, in a dynamic flow of data into an executing simulation; 5) An approach involving control and data flexibility in the simulation, coupled with judicious uses of optimization, should prove to be better. We find these points sufficient for rejecting a parameters-only approach, and consequently advocate the identification and capture of opportunities for expansion in a simulation. Our exploration of visualization support for COERCE reflects this conclusion.

The role of expert insight and direction in COERCE means that it is necessary to provide feedback to the expert user. Today, simulations model increasingly complex and adaptive systems, where the user needs to explore and summarize large amounts of high-dimensional data in order to understand the simulation. Because of the difficulty of adequately describing such complex systems, COERCE must address the issue of how the user will receive feedback about the simulations that are being manipulated. Visualization tools are one way to facilitate understanding of these complex simulation systems.

1.2 Challenges of Visualizing Coercible Simulations

As depicted in Figure 1, COERCE requires a visualization capability in each phase of building and using a coercible simulation. In Phase I, *making coercible*, a user indicates opportunities within a simulation for altering its meaning. We refer to this activity as "identifying flexible points." Flexible points include considerations for

- data type widening,
- value substitution for constants,
- changing loop convergence criteria,
- adding and removing stochastic elements,
- modifying branching criteria, and
- rearranging execution order.

To make the coercible simulation useful, the user also identifies the effects of employing selected combinations of flexible points and code modifications. Note that Phase I can either be applied to legacy simulations or concurrently with the development of a new simulation.

In Phase II, *coercing*, a user acquires a simulation from the coercible simulation library or the simulation instance library and experiments with selected bindings to flexible points and code modifications to meet a new set of simulation requirements. Optimization contributes to Phase II by automating the process of finding the best bindings for the flexible points in the simulation. In Phase III, *evaluating*, the user applies a coerced simulation by running it on one or more data sets and examining the results.

Considering the phases of COERCE, we observe numerous opportunities for visualization, such as

- demonstrating the significance of each flexible point identified in Phase I, as well as the complex ways that these flexible points may be related,
- viewing and evaluating the progress of coercing a simulation in Phase II, and
- logging and displaying the behavior of a coerced simulation instance so that it can be evaluated in Phase III.

From this, we can see that COERCE requires at least two types of visualization: Data visualization to display the differences between observed and desired simulation outputs, and software visualization to display the structure and runtime behavior of a simulation, supporting the discovery of flexible points and highlighting which of these points have the most bearing on any new simulation requirements that arise.

In order to better understand the importance of visualization to coercible simulations, we have studied two examples of COERCE in action and documented the role of visualization in each. Based on these examples, we are able to outline the requirements for a COERCE visualization toolkit and evaluate how well existing toolkits satisfy these requirements.

2 VISUALIZATION TECHNOLOGY

As we observed in section 1.2, COERCE requires data visualization and software visualization to support construction and manipulation of coercible simulations. Data visualization and software visualization are both active areas for research in the scientific computing, graphics, and software engineering communities. In this paper, we are not discussing an expansion of either technique, but instead a consideration of the application of these techniques to the challenges of simulation coercibility and coercion. However, before establishing the requirements for a COERCE visualization system, it is important to review the capabilities that modern visualization technology offers.

In data visualization, much of the current research is focused on real-time interaction between the user and large data sources. Flexible interfaces allow the user to dynamically change the way the data is being displayed (North and Shneiderman 2000). As coercion is applied to a data-driven simulation, it will be important for the practitioner to be able to change the mode of visualization while the simulation is running. Researchers are also exploring performance improvements to support rendering of large data sets, including compression, caching, and data-culling techniques (Guthe, Wand, Gonser, and Straßer 2002). As more data becomes available to the user of the coercion system,

these techniques enable the practitioner to make decisions about how to transform the simulation. Finally, research in feature extraction techniques (Bryson et al., 1999) and dimensionality reduction (Bingham and Mannila 2001) will ease the computational burden on the visualization system and the cognitive burden on the user as he or she tries to understand the data.

In the area of software visualization, research continues to focus on how software itself should be displayed, including how to display causal relationships between different threads in a program (Elmqvist and Tsigas 2003) and how to show the ways that different objects in the code are used and how frequently (Wang et al., 2003). It is also important to interface with running programs in a way that does not significantly impede the performance of the system, with different researchers looking at augmented virtual machines (Reiss 2003) and others continuing to focus on compiled-in libraries for visualization (Schroeder, Martin, and Lorensen 1998).

3 COERCE/VISUALIZATION EXAMPLES

Previous work on COERCE references a simulation support tool called SimEx (Carnahan, Reynolds, and Brogan 2003). As part of SimEx, we have developed a customized visualization toolkit, which is available online at <http://www.cs.virginia.edu/~mrm/visualization>.

For our research here we study two applications of COERCE, employing this visualization toolkit:

- Building a coercible simulation of a multi-server machine shop, where the number of repair technicians, number of machines, and other system behaviors can be easily or even automatically customized
- Coercing an agent-based simulation of evolving creatures to a new requirement, namely that the simulation reflect a more realistic relationship between the creatures' size and abilities.

These two simulations are selected based on the following criteria: First, both are conceptually simple and do not require significant domain expertise to understand. Second, the two examples represent different types of simulations, namely a discrete-event simulation and a time-stepped simulation. Finally, they have different levels of familiarity to the researchers. Together, they allow us to verify that our visualization tools are useful for becoming acquainted with a new simulation as well as for gaining insight into simulations that are already familiar.

The first example centers on Phase I of the COERCE life cycle (see Figure 1), making a simulation coercible. The second example consists of coercing a simulation in

Phase II, and both examples require a Phase III evaluation of the results.

3.1 The Multi-Server Machine Shop

Consider a factory or workshop consisting of machines of different types. As long as the machines are running, they produce income. However, they eventually break down and must be repaired. The shop has repair technicians available to fix the machines, but the repair staff must be paid regardless of whether any machines are currently broken. We wish to maximize the shop's profit by finding the optimal balance between the cost of staff salaries and the benefit of having additional staff available to repair machines.

To explore this problem, we can build a model of the shop. This can be implemented as a discrete-event simulation, where machine breakdowns and machine repair completions are the events of interest. Important parameters to this simulation include

- the number of machines of each type,
- the value of the work accomplished by a machine of each type in one hour,
- a distribution that describes the frequency of failures of each type of machine,
- a distribution that describes the time required to repair each type of machine, and
- the number of repair technicians.

Given the problem statement, a simulationist would normally define the number of machines and the hourly income from each machine as simulation constants. The event list would consist of one event for each machine's next failure and one event for each staff member's next repair completion, and the simulation code would sample the specified distributions to generate future failure and repair times. The resulting simulation would be easy to implement and relatively efficient.

However, consider ways that the problem statement could change: Suppose that more machines could be ordered after the shop has already opened or that the number of repair staff could fluctuate. There are an infinite number of ways that this scenario could change, and most of these changes would require a considerable amount of simulation rework. Instead of paying the cost of this rework, we would like to build a coercible version of this simulation in anticipation of future requirements.

3.1.1 COERCE Objectives and Techniques

For this example, we construct a coercible simulation *S'* based on an existing machine-shop simulation *S* and meeting the following requirements:

1. *S'* produces exactly the same results as *S* for the original problem.
2. *S'* simulates the original problem with comparable performance to *S*.
3. *S'* facilitates changing important simulation parameters, even permitting dynamic changes to occur during the execution of the simulation.

To evaluate *S'*, we use a legacy C++ simulation of this problem as *S*.

In part, the coercible simulation improves on the existing simulation by applying general purpose techniques for writing reusable software, such as using a consistent object-oriented design. Data structures such as the event list and the queue of broken machines are abstracted as objects. This abstraction is an important prerequisite to applying our simulation-specific techniques for building a coercible simulation.

In the simulation, all global constants are removed and replaced with instances of classes that bind related simulation parameters together. As an example, the parameters governing each type of machine are defined in *S* as

```
// Mean time to failure
#define TYPE_I_FAILURE      400.0
// Duration of a repair
#define TYPE_I_JOBS         8
#define TYPE_I_JOBTIME      1.0
// Number of machines
#define NUMBER_OF_TYPE_I    300
// Hourly income
#define PROFIT_FROM_I       20.0
```

In the coercible simulation *S'*, these parameters are set by defining a `MachinesI` object and initializing it:

```
FlexMachines MachinesI (400.0,
                        8,
                        1.0,
                        300,
                        20.0,
                        0, &Events);
```

Then, references to simulation parameters are replaced by invoking methods on these objects. Using the `MachinesI` example, the code

```
for (int i = 0;
     i<NUMBER_OF_TYPE_I;
     i++) {
```

is replaced by

```
for (int i = 0;
     i<MachinesI.getNumberOfMachines();
```

```
    i++) {
```

To make the simulation coercible, these objects contain methods that allow the simulation parameters to be changed at run time. These parameter-classes also contain all the code needed to automatically manage dependencies between the different parameters. For example, adding new Type I machines means adding new events to the event list, which the `MachinesI.setNumberOfMachines()` method handles automatically.

3.1.2 Role of Visualization

Visualization tools play an important role in this example of Phase I, creating a coercible version *S'* of a simulation *S*. The first requirement, of course, is that *S'* do everything *S* can do. By instrumenting both *S* and *S'* with a visualization toolkit, quick side-by-side comparisons of the two simulations can be made after every step of developing *S'*. These visualization hooks into a coercible simulation continue to be useful for the entire lifetime of the simulation.

Visualization tools also make it possible to explore the new dimensions of flexibility that *S'* possesses. That is, they enable Phase III evaluation of the coercible simulation. It is crucial to establish what the effects of changing each flexible point are, so that future users of the simulation can determine which points to manipulate in order to coerce the simulation. Using the visualization toolkit, we create plots to show relationships such as

- the optimal number of repair staff for different numbers of type I machines,
- the optimal number of repair staff for different numbers of type II machines,
- the effect of changing the random distribution of repair times for one type of machine on the utilization of the repair staff, and
- the effect on profit of being able to lay off or hire new repair staff at either weekly or monthly intervals.

A collection of visualizations like this makes it extremely simple for future users of *S'* to see the range of workshop scenarios that can be modeled by this coercible simulation.

3.2 Agent-Based Evolution Simulation

Agent-based simulations are a popular and useful tool for studying emergent biological and social phenomena. An example of such a simulation is Achilles, an open-source simulation of evolving virtual creatures available from <http://achilles.sourceforge.net>. Each creature contains a genetic makeup, determining the creature's size, strength, appetite, and behavior. Each creature's behavior is driven by a neural network, which determines where the creature should move and when it should either fight or mate with other creatures that it encounters. Creatures die from either fighting, starvation, or old age, and new offspring are created when two creatures mate. Each new creature's physical and behavioral characteristics are determined by mixing the characteristics of its parents. Since creatures with better attributes are more likely to survive long enough to reproduce their genes, stronger creatures with more intelligent behaviors become more common and other creatures die off.

The Achilles simulation is a very simple model of evolving creatures. For instance, the documentation indicated that each of the creatures' genetic attributes are independent of each other, meaning that there is no relationship between the size, strength, or metabolism of any given creature. So, we propose the following new requirement for this simulation:

Requirement 1: *The size of a creature should be positively correlated with its strength and its metabolism.*

In other words, we are coercing this simulation so that larger creatures are stronger but also require more food than their smaller peers.

3.2.1 COERCE Objectives and Techniques

To begin the coercion process, we instrument the Achilles simulation with the SimEx visualization toolkit, setting up visualizations to display

- average values across the whole population for several individual genes (strength, metabolism, size, natural lifespan, etc.), and
- correlations between strength and size and between metabolism and size across the entire population.

These visualizations are constructed as *software oscilloscopes*, dynamic displays that show the values of selected variables or functions of values on the Y-axis while time advances left-to-right along the X-axis. These oscilloscope displays provide an effective way to visualize multidimensional simulation data: For example, the seven different genes describing each creature's physical characteristics could be displayed as seven different traces on the same oscilloscope screen, making it easy to observe relationships between the genes and to watch the makeup of the population change as new creatures entered and left the simulation.

After deciding to use oscilloscope visualizations, linking the visualization toolkit to the unfamiliar simulation proves to be reasonably easy. The process consists of the following steps:

1. Review the simulation documentation to determine which simulation variables need to be visualized.

2. Locate these variables in the code and create data structures in the visualization code to correspond to each of these variables.

3. Add code to the simulation main-loop to update the values of the visualization data structures every time the simulation updates the corresponding variables.

In the process, we need to locate the simulation main-loop and the simulation variables for the creatures' genes. However, this imposes no additional work, because Requirement 1 relates directly to the values of these variables. Therefore, any transformation of the simulation to meet Requirement 1 would require locating the same sections of code that are used by the visualizations.

Unlike previous simulation coercion examples (Drewry, Reynolds, and Emanuel 2002; Carnahan, Reynolds, and Brogan 2003), this coercion does not involve any optimization. Instead, we apply the following code modifications: The function that returned the strength of a creature is modified so that instead of returning the value of the creature's strength gene, it returns an average of the creature's strength gene and its size gene. Similarly, the function that returns the metabolism of a creature was modified so that instead of returning the value of the creature's metabolism gene, it returns an average of the creature's metabolism gene and its size gene.

So, creatures keep independent size, strength, and metabolism genes, but when the simulation evaluates a creature's strength or metabolism for purposes of fighting or feeding, the creature's size influences the result. With this modification, the simulation meets the new requirement.

3.2.2 Role of Visualization

While this problem is a relatively simple instance of simulation coercion (Phase II), visualization is still essential to the process. Instrumenting the code to set up these visualizations is a useful exercise in itself, since it involves locating the relevant variables and determining where in the simulation their values can change. More importantly, visualization contributes to the overall understanding of the complex system that this simulation represents. Agent-based systems can exhibit discontinuous and even chaotic emergent behaviors, which are difficult to capture with analytical formulas or static data tables. Using visualization, however, we are able to explore interactions between different genetic factors and observe the rate at which the population's genetic makeup changes over time.

Once the simulation has been modified, visualization makes it easy to evaluate the simulation (Phase III) and confirm that the simulation meets its new requirement. Using coercible simulations is an iterative process, and so if the requirement has not been met, then the visualization can provide useful information about the problem that would

guide future coercion steps. For example, consider this variation of Requirement 1:

Requirement 2: *The size of a creature should be correlated with its strength with a correlation value between 0.5 and 0.75.*

The modification that was used to satisfy Requirement 1 would probably not be sufficient: The visualization would show a correlation that is positive but not exactly in the specified range. To meet this new requirement, we return to Phase II and continue coercing the simulation. In this example, we could add a weighting factor when the size gene is averaged with the strength gene, changing the amount of a difference that size makes in the creature's effective strength. Then, we could use optimization techniques to search for a value for this weight-parameter that would yield a correlation in the range between 0.5 and 0.75. Finally, we would use visualization to evaluate the simulation and confirm that this simulation instance met the new requirement.

4 DISCUSSION

These examples illustrate how visualization tools contribute to COERCE technology. In general, visualization supports construction of coercible simulations in three ways:

- Visualizations can be constructed to show the effects of changing each flexible point in the simulation.
- Visualization helps with validation, particularly in cases where another validated instance of a simulation is available for comparison.
- Visualization features in a simulation can be reused to construct new visualizations and support coercing this simulation to new requirements in the future.

Without visualization, future users must rely on the static documentation and on the code itself in order to understand a coercible simulation. With visualization, the developer of a coercible simulation can deliver a dynamic picture of the ranges of capabilities that it offers, overcoming the complexity inherent in flexible simulation systems.

We also demonstrated that visualization supports simulation coercion in several ways:

- Visualizations provide useful information regarding which internal values of the simulation are related to the simulation's new requirement.
- Visualization assists with measuring the progress of the coercion, even in the presence of complex simulation behavior.
- Visualization enables the user to verify that the modification and optimization steps of the coercion process are not interacting with each other or

with existing simulation requirements in undesirable ways.

Visualization accelerates the coercion process by enabling the simulationist to take the coercion steps that will be most effective at making a simulation meet its new requirement.

4.1 Requirements for a COERCE Visualization Toolkit

The examples we studied serve to establish the importance of visualization to COERCE, and they provide insight into requirements for a COERCE Visualization Toolkit (CVT). Taking these examples together with the description of coercible simulations in section 1.2, we enumerate what visualization support is necessary for COERCE:

1. Dynamic data display
2. Save and replay capabilities
3. Application-specific extensions

First and foremost, the CVT must be able to display data dynamically, animating the display as the simulation is running. At a minimum, this is the software oscilloscope capability defined in Section 3.2.1, where an arbitrary number of simulation variables can be plotted against time and updated as the simulation is running. Typically, simulations simulate a sequence of events occurring over time in the order in which the events would occur. As a result, the most intuitive way to understand the phenomenon that is being simulated is to watch the simulation itself unfolding over time. Even for simulations that simulate phenomena without any temporal component, it is still useful to use time as another dimension to analyze the simulation's behavior: For example, a single-processor simulation of a hundred coins being flipped simultaneously would still compute the hundred coin-flips in sequence. Watching the ratio of heads to tails converge toward 1:1 as the simulation ran would still give the user insight into the effect that changing the number of coins has on the simulation, even though the simulated coins are not sequentially dependent on each other.

Second, the CVT must have the ability to save and replay visualizations. This is especially important in Phase I, where a visualization that demonstrates the significance of each of a coercible simulation's flexible points could be stored together with the new coercible simulation in the library. However, this is also useful for Phase II (coercion), where the simulationist would like to apply a coercion step and then compare the modified simulation to the original. The ability to save and replay the visualization from the original simulation would save the cost of having to re-run the original simulation in parallel with the modified version in order to compare them.

Beyond these two baseline capabilities, numerous visualization features and tools enable COERCE in different ways depending on the application. For the Achilles simulation, software oscilloscope visualizations provided all of the information necessary for COERCE to proceed. However, bar graphs and both two-dimensional and three-dimensional scatter plots were used to represent relationships between variables in the multi-server machine shop example. Meanwhile, when coercing simulations with spatial semantics, such as the bicyclist simulation used in Carnahan, Reynolds, and Brogan (2003), a visualization of the trajectory of each object is indispensable for understanding the overall behavior of the simulation. As a result, while none of these additional visualization capabilities (scatter plots, 3-D viewing, etc.) are required for all instances of COERCE, we recognize that each of these capabilities contributes substantially to using COERCE in specific application areas.

The fact that only the first two of these three requirements apply uniformly to all simulations means that many existing visualization toolkits are already sufficient to be used as COERCE visualization toolkits. The SimEx prototype toolkit developed for this paper meets the requirements of a CVT, as do more sophisticated 3-D visualization toolkits such as VTK <http://www.vtk.org>. In fact, a simple oscilloscope-only toolkit such as GScope <http://gscope.sourceforge.net> would still be very useful for many simulation coercion problems.

4.2 Future Work

Supported by visualization, COERCE technology continues to show promise for facilitating simulation reuse. In order to make coercible simulations easier to develop, we are exploring programming language constructs and features that support simulation coercibility. In addition, we are continuing to expand the theoretical foundations and the software tools on which COERCE is built.

4.2.1 Language Support for Coercibility

The machine shop problem is an example of building a coercible simulation using existing language tools and features. Constants and variables were made coercible by replacing them with function calls, so that their values could be easily modified and even changed at run time.

However, there are several questions that arise from this approach to building coercible simulations. The first issue is performance: Substituting a function call for every variable evaluation adds a considerable amount of overhead to a simulation. An ideal framework for building coercible simulations would preserve flexible points for use in the future while optimizing the program's performance for its use in the present.

More importantly, simulation developers need ways to identify and take advantage of flexible points other than simulation constants. In section 1.2, we listed several other

kinds of flexible points, such as modifying branching conditions or rearranging the execution order of an existing algorithm. In order to support new kinds of flexible points, as well as to handle all kinds of flexible points with better performance, language support for coercible simulations is needed.

Note that coercibility language support does not add requirements to those we have already identified for visualization for COERCE. For language provisions such as relaxation of loop convergence criteria, a software oscilloscope supports the display of critical loop variables over time. The impact of relaxing the loop can be observed through graphics already associated with the simulation.

4.2.2 Development of COERCE Theory and Tools

As stated above, a visualization toolkit is only one part of SimEx, a software support system for COERCE. With the development of a useful visualization library, future work should include building a user-friendly system for extracting data from the simulation and relaying this data to the visualization software. In addition, optimization tools that compute the best values for flexible simulation parameters could also use this automatically extracted data. The result is that by linking a single library to a simulation, it will be possible to visually explore the behavior of the simulation while also using optimization to tune simulation parameters for any new requirements that are given.

Meanwhile, important methodological questions remain about how COERCE can be most effective. The interleaved use of optimization and manual code modification saves a considerable amount of developer effort relative to a completely manual approach to simulation transformation, but it would be even better if we were able to recommend exactly when optimization is most beneficial and what the limitations of optimization on a given simulation will be. Similarly, a more theoretical analysis could provide guidelines on how flexible a coercible simulation needs to be in order to meet requirements that differ from its initial requirements in a specified way.

Simulation reuse continues to be the central goal of COERCE research. Visualization tools can help simulationists reach this goal by improving their understanding of the simulations that are being reused and the new requirements that arise in reuse situations. As demonstrated by the examples in this paper, visualization gives the simulationist the information needed to effectively apply tools like optimization, code modification, and coercible-simulation language constructs to a simulation. As a result, visualization tools are an important component of COERCE technology.

REFERENCES

Bingham, E., and H. Mannila. 2001. Random projection in dimensionality reduction: Applications to image and text data. In *Proceedings of the Seventh ACM SIGKDD International conference on Knowledge Discovery and Data Mining*, 245–250: ACM Press.

Carnahan, J. C., P. F. Reynolds, and D. C. Brogan. 2003. An experiment in simulation coercion. In *Proceedings of the 2004 Interservice/Industry Training, Simulation, and Education Conference*. Arlington, Virginia: National Training Systems Association.

Dahmann, J. S., J. O. Calvin, and R. M. Weatherly. 1999. A reusable architecture for simulations. *Communications of the ACM* 42 (9): 79–84.

Drewry, D. T., P. F. Reynolds, and W. R. Emanuel. 2002. 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. 467–475.

Elmqvist, N., and P. Tsigas. 2003. Growing squares: Animated visualization of causal relations. In *Proceedings of the 2003 ACM Vymposium on Software Visualization*, 17–ff: ACM Press.

Guthe, S., M. Wand, J. Gonser, and W. Straßer. 2002. Interactive rendering of large volume data sets. In *Proceedings of the Conference on Visualization '02*, 53–60: IEEE Computer Society.

Kasputis, S., and H. C. Ng. 2000. Composable simulations. In *Proceedings of the 2000 Winter Simulation Conference*, ed. J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick, 1577–1584. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Knight, D. 2003. *Data driven design optimization methodology: A dynamic data driven application system*. Berlin: Springer-Verlag.

North, C., and B. Shneiderman. 2000. Snap-together visualization: a user interface for coordinating visualizations via relational schemata. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, 128–135: ACM Press.

Petty, M. D., and E. W. Wiesel. 2003. A composability lexicon. In *Proceedings of the 2003 Spring Simulation Interoperability Workshop*. Orlando, Florida: Simulation Interoperability Standards Organization.

Reiss, S. P. 2003. Visualizing java in action. In *Proceedings of the 2003 ACM Symposium on Software Visualization*, 57–ff: ACM Press.

Reynolds, P. F. 2002. Using space-time constraints to guide model interoperability. In *Proceedings of the 2002 Spring Simulation Interoperability Workshop*. Orlando, Florida: Simulation Interoperability Standards Organization.

Schroeder, W., K. Martin, and B. Lorensen. 1998. *The visualization toolkit: An object-oriented approach to 3-d graphics*. Upper Saddle River, New Jersey: Prentice Hall PTR.

Waziruddin, S., D. C. Brogan, and P. F. Reynolds. 2003. The process for coercing simulations. In *Proceedings of the 2003 Fall Simulation Interoperability Workshop*. Orlando, Florida: Simulation Interoperability Standards Organization.

AUTHOR BIOGRAPHIES

JOSEPH C. CARNAHAN is a member of the Ph.D. program in Computer Science at the University of Virginia. Joseph earned his B.S. in Computer Science at the College of William and Mary, and has held the position of 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 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, multi-resolution modeling and coercible simulations. He has advised 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>.

DAVID C. BROGAN earned his Ph.D. from Georgia Tech and is currently an Assistant Professor of Computer Science at the University of Virginia. For more than a decade, he has studied simulation, control, and computer graphics for the purpose of creating immersive environments, training simulators, and engineering tools. His research interests extend to artificial intelligence, optimization, and physical simulation. His email address is <brogan@virginia.edu>.