The Process for Coercing Simulations

Sarah Waziruddin
David C. Brogan
Paul F. Reynolds Jr.
University of Virginia
151 Engineer's Way, P.O.Box 400740
Charlottesville, Virginia, 22904-4740
(434) 982-2296, (434) 982-2211, (434) 924-1039
swaziruddin@cs.virginia.edu, dbrogan@cs.virginia.edu, reynolds@cs.virginia.edu

Keywords: reuse, coercing simulations, optimization, composable simulations

ABSTRACT: *Except under very constrained circumstances, reusing simulations without design or code modifications has proven elusive. Simulation reuse is highly desirable. Benefits of reuse appear in many forms: employing a simulation in an alternative context and/or with different inputs, coalescing (composing) a simulation with others to create a federation with higher objectives, combining a simulation with another to broaden the levels of abstraction represented, and modifying a simulation to exhibit desirable properties, for example better performance. For simulation designers and developers, anticipation of and provision for all possible contexts and uses of a simulation is generally unattainable. As an alternative, we propose and discuss a process that combines subject matter expert insight and semi-automated optimization to coerce a simulation meeting requirement R to instead meet a new requirement R'. In those cases where reuse through redesign or reimplementation is possible, but the costs are unacceptably high, the coercion process we advocate can become a highly cost-effective alternative.*

Our coercion process employs semi-automation to a significant extent. A knowledgeable expert can suggest flexible elements in a simulation, along with an objective function, and then initiate an automated optimization process that establishes the best solution under the constraints specified. If the result is unacceptable, the expert can suggest other flexible elements, adjust the objective function, or employ more conventional forms of modification to coerce the simulation in the direction of meeting the new requirement R'. By introducing automated optimization methods into the process, we improve upon traditional techniques that depend solely on code modifications. By keeping an expert involved in the process, we greatly increase the likelihood that the combination of modification and automated optimization are used to meet the new requirement R' in an efficient manner.

We describe our coercion process and discuss its application to a simple example. We explore its benefits and limitations. We also introduce the concept of coercible simulations, where the coercion process is initiated during the design and implementation phase through the use of programmatic notations that support the automation of the coercion process.

1. Introduction

Quite often simulation practitioners choose to reuse existing simulations, rather than write new ones from scratch, for new applications. A large investment in existing software and the estimated costs associated with starting anew are usually the determining factors. Reuse typically involves code modification, which itself can surprise managers regarding the extent of effort required to make what may seem like a simple set of alterations. Thus the question naturally arises: "Can we do better?" We address that question.

We have begun an investigation of a process we've named "coercing simulations." It involves a combination of code modification and simulation behavior optimization, with the goal of driving the behavior of a simulation in a direction that ultimately satisfies a new set of requirements. Because much of optimization can be automated, it is our hypothesis that a significant portion of the manual code modification that would have otherwise been performed can now be replaced by a semi-automated process, and that the overall transformation of a simulation to meet new requirements can be performed more efficiently. This process of interleaving modification and optimization appears very promising based on tests we have conducted. However, its overall effectiveness remains an unknown (which we are currently exploring). We argue coercing simulations will prove superior to any apparent alternatives.

We present details of the coercion process here, along with an exploration of reasonable alternatives, and then

a rationale for its selection over those alternatives. An introduction to the coercion process can be found in [16]. Details of experiments employing our coercion process can be found in [2] and [6]. A discussion of SimEx, a prototype coercing environment we have implemented, and various approaches to optimization we have considered, can be found in [17]. In this paper we explore the coercion process, P, itself, which can be characterized as a regular expression:

$$P = [\ m^*, o^*\ ]^*$$

that is, zero or more repetitions of: zero or more code modifications followed by zero or more optimizations. Sequences such as *m, o, m, o* and *o, o, m, m, o, m*, and *m*, and *o* are all legal instances of P. Note that to achieve the transformation of a given simulation to conform to a given target set of requirements, we may discover multiple sequences that are all satisfactory. In fact, the sequence *m* is always a possibility, that is the act of only modifying a simulation, with no intervening applications of optimization.

The coercion process we propose has broader application than classical reuse. In the classical sense, reuse involves code, that is code gets reused. Our process applies there, but it also applies in the case where we reuse the behavior of an existing simulation to shape the coercion of a second simulation, perhaps more abstract, perhaps more concrete, perhaps simply satisfying a modified set of requirements. Thus, included in the set of possible coercion processes we consider are those that do not necessarily access or require knowledge of the implementation of a simulation embodying desired target behavior.

Measures of success for any approach to code and/or data transformation for reuse include: 1) minimizing frustration of the person(s) performing the transformation, 2) maximizing efficiency of the transformation process, 3) increasing the confidence of the person(s) performing the transformation in the quality of their product, and 4) improving utility, by providing methods and tools that afford needed insight. Our success in two early experiments [2, 6] suggests simulation coercion will fare very well by these measures. Our analysis of the process itself, appearing in the following sections, bolsters our confidence. However, we have neither analytic proof nor overwhelming experimental evidence to offer a robust argument in favor of coercion. We believe a demonstration of viability requires convincing success in a suite of significant applications.

In the following sections we motivate the need for a coercion process, and we introduce and discuss various candidate processes for accomplishing coercion. After establishing the importance and likely utility of the coercion process P, we discuss details of candidate processes, including different reasons for mixing modification and optimization. This last discussion hinges in part on appreciating the benefits optimization may hold for expediting later actions in a coercion process. We develop that appreciation. Finally, we present a simple example to demonstrate an application of the coercion process, and we discuss future research.

2. Related Work

In its contemporary form, simulation reuse is primarily a manual process due to the complexity of modern systems and the custom needs presented by each user. Research in the simulation reuse community demonstrates that code standardization, functional abstraction, and procedural documentation are tools that facilitate the reuse process. These simulation reuse techniques share a common belief that human creativity and insight are among the scarcest resources, but they differ in how and where they bring human skills to bear on critical issues. In this section, we describe two related areas of research that employ human insight for simulation reuse, simulation abstraction and code modification. We also describe customized instances of simulation coercion and early examples of a general simulation coercion framework.

Davis and Bigelow furnish a compelling argument for accomplishing reuse through simulation abstraction [4, 5]. In their abstraction, called motivated metamodels, the logic contained within a simulation's code is abstracted by numerical and functional models that generate the same simulation output. Consider, for example, how the stochastic models in queuing theory can replicate the automobile flow rates of a high-resolution multiagent traffic simulation. Not only is the queuing model a simplified form of the original simulation, it also provides an intuitive way to be reused for scenarios where the mean automobile arrival rate is varied (perhaps a capability not easily generated through modification of the multiagent simulation).

Davis and Bigelow demonstrate how an unguided statistical metamodeling process will produce poor metamodels that find counterintuitive correlations between variables and fail to represent important features in the input to output mapping. To create motivated metamodels, however, an expert must identify critical components that will be preserved as intuitive parameters of the new representation and serve as a well-understood means to manipulate the metamodel for reuse. Although a formal process for creating motivated metamodels is not provided, the authors identify

important factors an expert should consider. Our coercion process shares this emphasis on the involvement of experts, but code modification and code abstraction are used interchangeably throughout the reuse process and we limit abstractions to those that integrate with automated optimization algorithms.

Grzeszczuk et al. [9] introduce a specific example of simulation abstraction with their NeuroAnimator. Their technique uses a neural network to create an abstraction of the computationally intensive code that implements rigid body simulations. This technique requires repeated execution of the simulation under different initial conditions in order to accumulate a database of performance data. For each instance of simulation performance in the database, a neural network models the numerical integration of the simulated character's equations of motion for one time step. Although no trace of the scientific logic behind rigid body dynamics remains, the neural network representation executes more quickly and, because it is analytically differentiable, it enables the use of gradient-descent search for optimal controllers.

Composability describes the ability to generate simulations from smaller components that are reused and combined to satisfy a new requirement put forth by the user. Through simulation reuse, composable systems aim to decrease development and validation time. The system is envisioned to be totally automatic and is expected to operate correctly, free from human intervention and guidance from the requirement stage to the validation stage. However, "we are discovering that unless models are designed to work together, they don't (at least not easily and cost effectively)" [11]. Research efforts to establish the theoretical foundations of composability are underway [12, 13, 14].

Object oriented methodologies, code and template libraries, and requirements specifications encourage component reuse but human error and system complexity represent significant obstacles to composability. Because of these challenges, manual code modification is the traditional method for reusing and composing existing simulations. Despite the time consumption and complexity of code modification, it remains a very effective way to influence simulation performance. The coercion process utilizes code modification to accomplish reuse, but it emphasizes code modification for the purpose of creating the abstractions needed by the optimization algorithm.

Just as simulation reuse is an important research problem in the modeling and simulation community, performance reuse is a significant research area in the computer animation community [1, 3, 7, 8, 9, 10, 15, 20]. The following examples demonstrate how the temporal sequence of joint angles and limb positions that generate the motion for one character can be reused to animate different articulated characters.

Hodgins and Pollard [10] present one algorithm that adapts the locomotion controller of a physically simulated character so it can be used to control the running motion of characters with different shapes and sizes. In order to correctly accomplish the desired running requirement, the locomotion controller must adjust key parameters for each character depending on such physical aspects as mass, height, and kinematics. The authors' algorithm adapts the locomotion controller from one character to another by first using a heuristic to make an initial estimate of key locomotion controller parameter values and then fine tuning those values using a search process based on simulated annealing. An evaluation function that expresses the quality and appearance of the character in motion is used to guide the search.

This example demonstrates the way expert insight and optimization can be used together to reuse a small portion of code. However, this code reuse method is limited to extremely similar physical simulations where the locomotion controller can be reused without code modification. Coercion also utilizes optimization to tune key parameters, but it allows code modification as a step to make more significant changes to the reused simulation. With these changes, coercion addresses the challenge of reusing general-purpose simulations.

Motion retargeting is another automation technique in the field of computer animation where reusing an animation performance is the goal. Gleicher [7, 8] presents a set of animations of a human walking, climbing a ladder, lifting objects, and dancing. These animations are created by computerized recording (motion capture) of a live actor's performance and can only synthesize the movements of a person with exactly the same limb lengths and style as the actor. To create animations for a computer graphic person of a different size would require building a new computerized recording of a similarly sized actor (complete code modification) or coercing the existing animation using semi-automated optimization techniques. Without either modification or optimization, the motion from a tall actor would lead to an animation of a shorter character that appears to levitate above the floor and to use an abnormal gait.

Gleicher's solution is noteworthy for its combination of computerized automation and manual guidance by a SME (Subject Matter Expert). An automatic process generates the first pass at a coerced simulation.

Through a well-designed user interface, a SME is able to evaluate the results of the first pass and to define simulation errors that must be eliminated, or constrained to have a value of zero. A constrained optimization technique, spacetime constraints, uses the results from the first pass and the SME-defined constraints to eliminate constraint errors while minimizing disruption of the remaining simulation components. The resulting simulation preserves the naturalness of fluid human movement while accomplishing the constraints deemed by the SME to be important for the particular application. Similar to Hodgins' algorithm, Gleicher's solution depends on principles of human motion and is not applicable to all simulations. However, it demonstrates an opportunity to design a coercible system that uses automation and minimal intervention from a SME.

Recognizing the utility of optimization methods (e.g. spacetime constraints), such as those used in the computer animation community, Reynolds first proposed coercion for general simulations in [17]. Drewry et al. [6] demonstrated numerical optimization as an effective technique to coerce an environmental simulation, DOLY, to meet requirements set by the output of a higher resolution simulation, CANOAK. Even though DOLY lacked CANOAK's detailed temporal and spatial resolution, SMEs were able to identify three parameters in DOLY that influenced its ability to match the values generated by CANOAK for a forest's monthly carbon dioxide absorption rate. Optimal values for these parameters, which were initially constants in DOLY, were computed using an automated optimization process. This method succeeded in coercing DOLY to recreate the highly accurate carbon dioxide consumption levels generated by CANOAK without incurring substantial developmental or computational costs.

Carnahan et al. further formalize simulation coercion [2]. The authors specify the roles of simulationist, SMEs, and automation in the coercion process and define an iterative process whereby a SME proposes changes and comparison metrics, a simulationist implements changes, and the SME reviews the effects. By distributing tasks to the specialists, the boundary between the model and its implementation are preserved. An experimental coercion of a low-resolution particle to produce output similar to that produced by a high-resolution physical simulation of a bicyclist validates the process and reveals several important areas of future work: formal definition of simulation assumptions, specification of limitations for any coerced simulation, and the creation of coercion tools.

Simulation Explorer (SimEx) is a tool for coercion [17]. SimEx implements a specific set of coercion paradigms and does not replace the role of the SME or simulationist, but facilitates their tasks. It provides a graphical interface to a simulation through which a SME can visualize data flows and identify important simulation behavioral features, which are marked as formal constraints. By supporting a standard way of executing and comparing simulations, defining constraints and objectives, and optimizing accessible variables, the tool provides the infrastructure within which the coercion process can proceed.

Reynolds, Drewry, Carnahan and Vicaire set a solid foundation for utilizing coercion for general simulations. However, these studies limit themselves to using optimization techniques to coerce an existing simulation to meet a new requirement. We extend the coercion process to include the option of modifying the algorithms in the simulation.

3. The Coercion Process

The coercion process, P, defines a method for transforming an existing simulation, S_0, exhibiting behavior B_0, satisfying requirement R, through a sequence of variations on S_0, and ultimately to a simulation S_n, exhibiting behavior B_n, satisfying requirement R' (a requirement S_0 was not necessarily originally designed to satisfy).

P can be characterized as a regular expression:

$$P = [\ m^*, o^*\ ]^*$$

where m and o represent code modification and optimization respectively. P can consist of any number, including zero, of modifications and/or optimizations in any order. For a coercion culminating in the creation of a simulation S_n, the length of the sequence described by P is n. The sequence:

$$S_0 \rightarrow_m S_1 \rightarrow_o S_2 \rightarrow_o \ldots \rightarrow_m S_{n-1} \rightarrow_m S_n$$

is one in which a simulation S_0 is coerced by way of a modification and two optimizations initially, and ending with two modifications, to become S_n. If the behavior B_n of simulation S_n satisfies requirement R', then the coercion would be considered successful. If not, then the practitioners (a SME working with an expert on the simulation code –a simulationist) could either continue optimizing and/or modifying, return to a prior point in the sequence to resume modifying and/or optimizing, or abandon the effort and start anew with a different strategy. The coercion process is depicted in Figure 1.

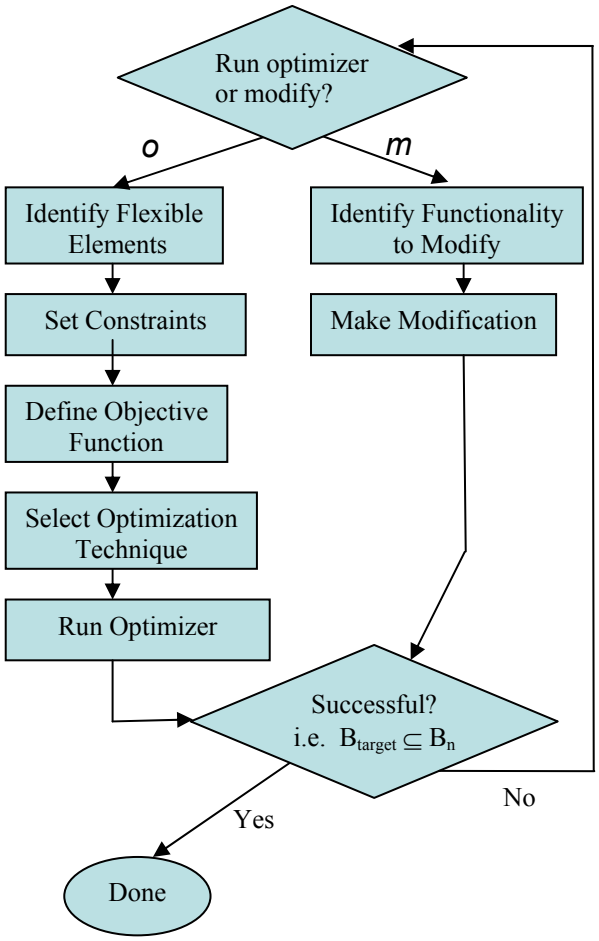

Figure 1: The Coercion Process.

A typical requirement R' can be satisfied by an infinite set of B_n, since there will be degrees of freedom in any instance of B_n which are not addressed by R'. Similarly, a typical behavior B_n can be exhibited by an infinite set of simulations S_n because, again, there will be degrees of freedom in the implementation of S_n not addressed by B_n. Thus, when we speak of coercing a simulation S_0, exhibiting behavior B_0 satisfying requirement R to a simulation S_n, exhibiting behavior B_n satisfying requirement R', there are actually an infinite number of simulations that would qualify as S_n. Conversely, for a given S_n, there are an infinite set of simulations S_0 that can be coerced to S_n. That there is an infinite set of S_n for any given S_0 is of greater interest to us here. It implies that the set of possible coercions from a given S_0 to an S_n exhibiting behavior B_n satisfying requirement R' is infinite, assuming a given coercion is characterized by the resulting S_n. In practical terms this means independent simulation practitioners, starting with the same S_0, and the same target requirement, R', may produce different S_n (or the same S_n by different sequences of *m* and *o* operations). Overall, for a given simulation S_0, its coercion to a new simulation satisfying requirement R' does not lead to a unique process P, a unique resulting simulation S_n or a unique behavior B_n satisfying R'.

For the pair S_0 ,R' we can ask:

- What is the best sequence of operations, *m* and *o*, for coercing S_0 to satisfy R'?
- What is the best S_n?
- What is the best B_n?

Answers to these questions depend on many factors including 1) the nature of S_0, 2) the nature of R', 3) the needs of the stakeholder in a satisfactory S_n, 4) the experience of the practitioners conducting the coercion, 5) the tools available to the practitioners, etc. The coercion process, P, as we define it, is sufficiently flexible to accommodate any set of answers to the questions posed. In fact, we argue that P, because it requires the practitioners to be tightly coupled to the coercion process, enhances the chances of discovering best answers to these questions. The following subsections describe the coercion process in more detail.

Target. For a given target requirement, R', we can define a target behavior B_{target} for any simulation satisfying R'. The goal of coercion is to transform S_0, through a sequence of applications of optimization and/or modifications, to an S_n such that $B_{target} \subseteq B_n$. The relation between B_n and B_{target} is not strictly an equality because S_n may include capability for producing behavior, reflected in B_n, not prohibited by B_{target}.

Coercing S_0 to an S_n, such that $B_{target} \subseteq B_n$ implies the ability to define B_{target} and the ability to relate outputs of S_n with B_{target}. Defining B_{target} can be non-trivial. Often B_{target} is a subset of the behavior exhibited by an existing simulation. For example, consider S_0, a low resolution weather simulation, and S_H an existing high resolution weather simulation. We may be interested only in high quality tornado prediction, a subset of all possible output of S_H. Then B_{target} would be a subset of the behavior of S_H that includes high quality tornado prediction but not necessarily hurricane prediction. Separating the two behaviors, that of hurricanes, and tornadoes, may be involved.

The behavior of a simulation, S_n is a function of its execution properties, its internal variables and its outputs. A target behavior, B_{target} may also be captured as a function of variables, outputs and execution properties of, for example, a second simulation; or it may be a function of a data stream from measurements of real phenomena; or it may be a function of data hypothesized by an expert, etc. Often B_{target} will be a discrete

representation of a continuous process, in which case the practitioner must address sampling issues. Whatever the source of B_{target}, the practitioners must determine a method for comparing B_n and B_{target} to determine if $B_{target} \subseteq B_n$.

Once a representation for B_{target} is established, and a method for relating the behavior of B_n to B_{target} is determined, the practitioners must establish a method for determining how well $B_{target} \subseteq B_n$ has been satisfied. Recall the coercion process P is iterative, involving both optimization and code modification as candidate operations. (See figure 1.) When the most recent step has been optimization, the relation $B_{target} \subseteq B_n$ may never be met strictly. Instead it will be measured by "goodness of fit." An objective function must be identified by the practitioners that indicates when B_n and B_{target} exhibit goodness of fit. Objective functions are discussed below. When the most recent step in a coercion process has been code modification, it is possible that $B_{target} \subseteq B_n$ has been met in a strict sense. The practitioners must provide a method for determining when $B_{target} \subseteq B_n$ has been met strictly.

Optimization. It is our intention that optimization replace a substantial portion of the code modification normally necessary to transform S_0 to S_n. Optimization consists of two parts. The first part, like all optimization techniques, requires some human input, namely identification of the flexible elements the optimizer will be able to vary in order to best satisfy the provided objective function. Flexible elements are variables or constants in a simulation that have values the practitioners deem alterable. Permitting these constants to vary in a well defined range is often reasonable. So in a weather simulation, the force ratings of recorded tornadoes from 1970-1985 in New York City may be stored as constants, but it may be reasonable to allow the values to change slightly due to assessment error. By identifying these constants as flexible elements, the practitioners allow the optimization algorithm to expand its search to include alternative valid values that may lead to the simulation generating behavior closer to B_{target}.

Following the identification of flexible elements, the practitioner must select an optimization method, an objective function and the constraints on the optimizer search space. This is a subjective process and is dependent on experience and familiarity with the simulation subject area, the particular simulation and various optimization techniques [17]. Next, the optimizer can be run and best (or near best depending on optimization method chosen) values for flexible elements are found.

Constraints on the range of flexible element values define the size of the optimizer search space. If the set of possible values for a flexible element is too large, then the optimizer may run for too long, or suffer increased chances of becoming stuck in a local minimum. If the range is too small, the optimizer may not find any solutions satisfactory to the practitioners. Currently all such constraints must be expressed by the practitioners.

The benefit of optimization is directly related to the quality of the flexible elements identified. With only a few, tightly constrained elements the optimization step may prove fruitless and the practitioners would have to resort to code modification. Alternatively, if too many lightly constrained elements are identified, then the coercion process may not be effective if the search space is too vast to search completely.

When performing an optimization step during the coercion process, the practitioners need to identify a measure for deciding whether the simulation behavior is good enough; that is, that $B_{target} \subseteq B_n$ is met with a sufficiently close fit. One possible measure is the sum of errors occurring between a data stream generated by Sn and a corresponding data stream derived from B_{target}. Another error measure could be sum of squares of error. Selection of a good measure is a task that must be performed by the SME engaged in a coercion.

Modification. In the coercion process, P, the alternative to optimization is code modification. As its name implies, this operation involves altering an algorithm to realize a desired behavior. A simulation can always be coerced, using only modification, to satisfy desired target behavior B_{target}. That is, the process P = m is always an alternative. However, it is this modification-only process we propose replacing because we expect the resulting combined optimization and modification process will prove superior on all important measures.

Generally, optimization alone will not lead to discovery of a S_n that satisfies B_{target}. Recognizing this, we offer modification as an optional step in P. In contrast, the computer animation community has avoided explicitly modifying code during a similar coercion process because they have the benefit of working with algorithmically similar physical simulations. We do not expect such good luck in the broader simulation community. Generally, modification should be pursued as a second choice, for example, when a practitioner cannot identify a sufficient set of flexible elements or sufficiently broad constraints. Another case would occur when the practitioner identifies an opportunity to guide the coercion process in an entirely new direction with a set of modifications. In such a case we would

expect modification to be performed in order to enable new explorations with optimization.

Successful Coercion. Up to this point we have focused on details of the operations that together define a coercion process: optimization (*o*) and modification (*m*). Earlier we asked "What is the best sequence of operations, *m* and *o*, for coercing S_0 to satisfy R'?" This question represents issues that occur at a meta-level in the coercion process. A related question is "What can a practitioner do early in a coercion process to make the remainder of the coercion process go well?" We have some insights into this latter question, and believe they offer additional insights into how the former question may be answered.

What can a practitioner do early in a coercion process to make the remainder of the coercion process go well?
Insight 1: The interplay between optimization and modification will often be beneficial. Finding critical flexible elements during an optimization step provides insight into potential opportunities for code modification. Modification leads to a better understanding of a simulation's code, which in turn provides greater opportunity to identify new flexible elements.
Insight 2: A modification performed early may create a broad set of opportunities to allow optimization to explore a behavioral space that includes B_{target}.
Insight 3: Failure of a sequence of optimization steps to lead to satisfactory progress towards B_{target} suggests the focus should be on a set of modifications that create new optimization opportunities.
Insight 4: A knowledgeable practitioner can identify sets of coercions that can be performed marginally, to manage complexity. For example, a weather simulation could first be coerced to predict tornados well, and then be coerced to predict hurricanes well. Clearly, success with this approach depends on the independence of the behaviors.
Insight 5: Sometimes the best coercion strategy will be modification only.
Insight 6: Sometimes, even when experts are doubtful, optimization-only produces excellent results. See [6].

Lastly, we mention *coercible simulations*. Up to this point we have assumed coercion was applied to simulations that offered no embedded support for the process. Given our success with coercions of this nature, it is natural to wonder if programmatic notations captured during the original implementation of the simulation, as well as during subsequent coercions, could be beneficial. Such notations could indicate the flexible elements in a simulation. They could capture relations between program components. They could identify processes in the code that are amenable to exploration with optimization. The goal would be to capture designer knowledge so that more optimization, and therefore automated analysis, becomes possible in a coercion. We are actively exploring the notion of coercible simulations.

4. Coercion Sequence Transitions

In order to provide deeper insight into the coercion process, we discuss the four possible transitions in a coercion process sequence and when they would likely occur.

O $\rightarrow$ O. Sometimes optimization must be pursued iteratively. An initial pass at optimization may fail if it does not converge on a targeted behavior. Perhaps there are too many flexible elements to make the search tractable or the objective function may display chaotic behavior; small changes in test cases result in very different objective function values. In these cases, the practitioner may choose to rewrite the objective function to change the set of flexible elements, or constrain the potential values of the flexible elements.

An O $\rightarrow$ O transition could also occur if the optimizer becomes stuck in a local minimum. Global optimization methods exist, but for efficiency reasons the search space must be constrained to ensure tractability. Expanding the search space iteratively by adding flexible elements may provide a way for the optimizer to seek more attractive solutions away from the local minimum.

The ability to limit coercion to automated optimization is directly dependent on the identification of flexible elements, the constraints on these elements and the objective function. It is possible to employ only a sequence of optimizations in some cases [6].

O $\rightarrow$ M. When optimization fails, and there are no other approaches to optimization apparent, or when the practitioner believes optimization at the current stage of coercion is no longer effective, modification is required. For example, during an optimization step, supporting visualization tools (we provide, see [17]) may reveal discontinuities in the simulation, where simulation behavior changes significantly for specific values of the flexible elements. Most optimization algorithms will have a difficult time converging to optimal values when there are discontinuities in the system and human intervention is required to account for these.

An example of a discontinuity involves a conditional statement in the simulation. For example, the following statement may generate a discontinuity for the

optimizer:

```
If A
  Then B
Else C
```

The output produced by this simulation may vary significantly at the threshold where A is no longer true. The practitioner may decide that for the purpose of producing better behaved output, the conditional statement should never select C. A code modification to edit the selection statement would be performed.

Finally, if the optimizer fails to converge on a value or becomes stuck in a local minimum, and no additional flexible elements exist in the simulation, code modification becomes the only viable option.

M → O. It may be desirable to perform an optimization step after code modification if new flexible elements are uncovered during a modification. A sequence of one or more optimization steps can be performed on these newly identified flexible elements

Optimization may also be pursued after modification when optimal values for flexible elements need to be recalculated as a result of algorithm changes in the simulation.

M → M. Repeated use of code modification is appropriate when the optimization alternative is exhausted (e.g. no apparent flexible elements to search over) or when optimization is too costly, or when the search space is too large. A practitioner may engage in a sequence of small modification steps, in hope of discovering new opportunities to apply optimization as early in the remaining portion of the coercion process as possible. An M → M step may also occur when a modification does not lead to expected behavior and no optimization alternatives are apparent.

5. A simple example

To illustrate the coercion process, P, we apply it to a simple example. Consider a simulation S_0, that accepts one real number, *x*, as input and generates one real number, *y*, as output according to y = -0.25x + 0.5. The behavior, B_0, is a line with a precise slope and y-intercept. The new requirement, R', for S_0 is to output a sine wave (Fig. 2). The target behavior, B_{target}, for S_n is to output *sine(x)* for all 16 sample points, defined by

$$x \in \left\{ \frac{\pi}{4} n; n = 0,1,\ldots,15 \right\}$$

Because we encourage leveraging automation as much as possible, optimization is considered first. Optimization requires the specification of an objective function, flexible elements, and constraints. For this first iteration, the SME defines the objective function to be the absolute value of the sum of all errors in the target behavior and uses basic mathematical insight to identify that the slope and y-intercept are flexible elements in S_0. No constraints are specified and a linear regression optimization method is defined.

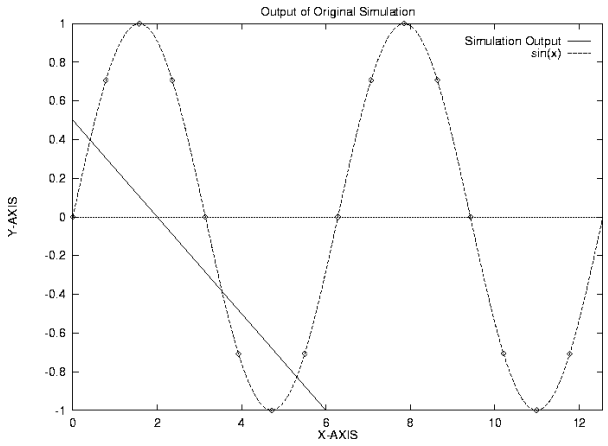

Figure 2. The output of the initial simulation, y = -0.25x + 0.5 and requirement R', y = sine(x).

The optimization algorithm searches for values of the flexible elements that minimize the objective function and converges on the value of zero for both the slope and the y-intercept (objective function equals zero). The coerced simulation, S_1, is therefore $y = 0$ and it generates a line that overlaps the x-axis. It is surprising that the optimization results indicate a straight line can so accurately produce the output of a sine wave and the SME utilizes the available visualization tools to observe that the objective function is inappropriate. In the set of 16 samples stipulated by B_{target}, four have an error of zero, four have an error of $\sqrt{2}/2$, four have an error of $-\sqrt{2}/2$, two have an error of one, and two have an error of negative one (Fig. 3). Even though the absolute value of the sum of errors is zero, the target behavior has not been met.

For the second iteration, the SME changes the objective function to be the sum of the absolute values of all errors. The optimization algorithm searches for values of the flexible elements that minimize the objective function and again the value of zero is computed for the slope and y-intercept. However, the objective function for S_2 returns a value of $4\sqrt{2}+4$. Following a second review of the results in a visualization tool, the SME resolves that in order to produce the sine curve,

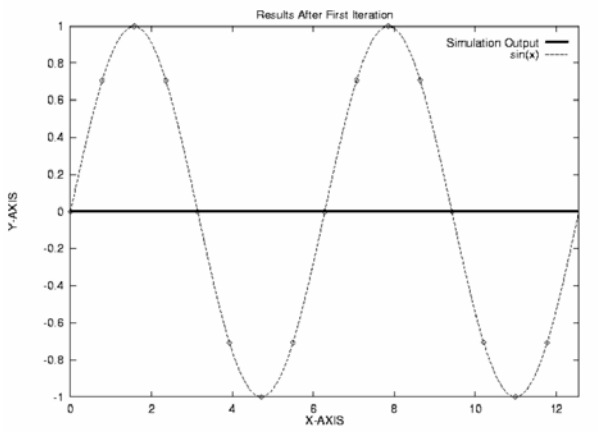

Figure 3. Results after the first iteration of coercion. The output of S_1 is a straight line at y = 0, and the target behavior is indicated with diamonds placed on the sine wave.

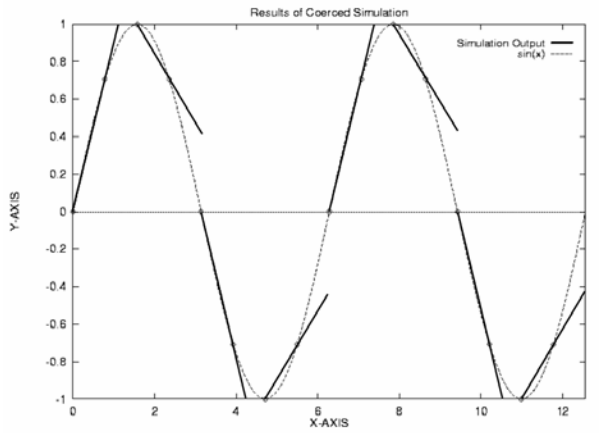

Figure 4. Results of the coerced simulation. The output of S_6 matches the target behavior at all 16 samples.

S_2 must be modified to produce line fragments rather than one line in order to track the wave better. Due to this insight, the SME pursues code modification as the next step.

To create S_3, the practitioners modify S_2 such that the function domain is subdivided into regions of size π. Therefore, the single linear function, $y = 0x + 0$, that was defined for all real values of x is now a set of linear functions, $y_i = 0x + 0;\ i = trunc(x/\pi)$. In order to evaluate the target behavior, four linear functions, all with the form $y = 0x + 0$, are required to span the input range from zero to 3.75π, as required by B_{target}. Upon testing S_3, the SME observes its output is still a straight line that lies atop the x-axis. With the newly added flexible elements, the SME returns to optimization as a solution strategy.

In iteration S_4, The SME converts the constant coefficients of the linear functions $y_i = 0x + 0$ into flexible elements, $y_i = m_i x + b_i;\ i = trunc(x/\pi)$. The optimization algorithm is executed again to find optimal values for the eight unknown values of the flexible elements - the slope and y-intercept for the four line segments that cover the input range between zero and 3.75π. The four lines defined by the eight optimal values cannot match all 16 test samples, but the objective function returns a much lower error value than with S_3 and the coercion process continues.

With the aid of the visualization tools, the SME discovers that S_5 will be constructed by a code modification that doubles the number of subdivisions in the simulation's domain in order to track the sine wave with greater accuracy. The eight line segments that span the input range from zero to 3.75π are defined by 16 coefficients that are fixed (according to an initialization protocol for new flexible elements and inheritance from the previous variation of the simulation). Because these 16 coefficients are unlikely to be correct, the optimizer must once again be executed to find the best values for the slope and y-intercept for each new line fragment.

The optimizer is executed to find optimal coefficients for the eight line segments in S_6 (Fig 4). The optimal coefficients for each line cause it to intersect the two samples from B_{target} that are within its input range. For example, the first line segment is defined by

$$y = \frac{2\sqrt{2}}{\pi}x + 0; 0 \le x \le \frac{\pi}{2}.$$

Every sample in B_{target} is exactly matched by executing S_6 and the objective function returns a value of zero. The SME utilizes the visualization tools to observe that the line fragments track the sine curve and declares that the coercion process has been successful.

The sequence of modifications and optimizations for this coercion process is *o, o, m, o, m, o* and reveals the interplay between code modification and optimization. Optimization was selected when flexible elements had either their initialized values or values obtained with an outdated objective function. Modification was typically selected after failed optimization trials and subsequent visualization of the results identify the specific functionality in S_i that had to be altered in order to coerce S_i to produce output B_{target}. Naturally, a different definition of B_{target} (five samples between 0 and 2.5π) could result in a different coercion process.

6. Discussion

The coercion process we present utilizes human expertise and insight to accomplish simulation reuse through a combination of optimization and code modification. The quantity and form of human interaction is left unspecified as no single solution will work for all reuse requirements. However, we can state that the coercion process lies on the continuum between manual code modification and automated optimization. The exact location on this continuum where practitioners anchor their reuse efforts is dependent on their awareness of opportunities to leverage computerized optimization.

The benefit of coercion over code modification is proportional to how easily optimization is incorporated in the process. The identification of flexible elements, constraints and an objective function are crucial steps that will have significant impact on the performance of the optimization algorithm. Optimization algorithms used in this process present their typical suite of considerations: under/overfitting the requirement, over/underconstraining the solution space, continuous/discrete and differentiable state spaces, local/global solutions, and the curse of dimensionality. However, early success using optimization during coercion reveals the practitioners can surmount these challenges and replace the labor intensive manual code modification process.

Early inspiration for coercion was taken from the automated techniques in the computer animation community that utilize optimization to achieve performance reuse. Because we want to create a process that is applicable to the general class of simulations, not just those that transfer motion from one character to another, we include code modification as an option in our process. However, we have discovered that the way we integrate code modification within an optimization framework actually motivates higher quality code modifications.

The coercion process emphasizes the use of abstractions that are amenable to optimization. The simulation analysis that identifies these abstractions also defines the desired functional properties of the simulation code and may isolate their location within the code. The use of visualization tools is integral to this exploration process. As we saw in the example, the failure of initial optimization steps led to the consideration of an effective code modification. In this way, the optimization and code modification steps are mutually beneficial to each other and both increase the practitioner's knowledge about the simulation by emphasizing the dependencies between various internal elements and the output.

Because of the dependence the optimization step has on an objective function, the translation of reuse requirements to target behaviors is critical. In our example, 16 sample points were selected to validate that the initial line segment simulation had been coerced to a sine wave. If only four sample points had been used (all at $y = 0$), however, a line segment could satisfy this specific target behavior for matching a sine wave. The sample points that define the target behavior must capture the desired features of the requirements. As insights are acquired about the desired features of the requirements, the target behavior will change and the coercion process can be restarted or can continue from its intermediate state towards this new goal.

Coercion is most successful when the unmodified simulation models similar phenomenon to the target behavior. It will always be as successful as code modification because the coercion process can consist of only code modification to coerce a simulation to exhibit any arbitrary behavior.

7. Future Work

The mechanics of the coercing process P are fairly well understood, as we have demonstrated in this paper. However, meta-level questions concerning the process remain largely unanswered: Is there a discipline for employing coercion such that the process itself requires minimal effort? Is there a strategy for the early part of a coercion process that will facilitate activities in the remaining part of the process? For a given target behavior, is there a specific type or instance of S_n that is best for satisfying it, given the initial simulation S_0? These and related questions are a focus in our continuing study of simulation coercion.

In the optimization step of the coercion process, identification of flexible elements has been presented as a manual process, to be carried out by the coercing practitioners. Can the identification of flexible elements be performed automatically, or semi-automatically? At the very least, can identification of non-candidates be performed automatically? We assume that the more the coercion process can be automated, the more all parties involved (coercion practitioners, stakeholders in the final product) will benefit. For this reason, we are studying all possible operations that can be automated, including identification of flexible elements.

We introduced the concept of *coercible simulations*, ones that incorporate notations that convey useful information from previous designers and implementers to coercion practitioners. We expect such annotated

simulations can greatly facilitate the coercion process, and so we are exploring a wide range of issues related to the concept.

8. Conclusion

This paper presents a method for accomplishing simulation reuse through a process called coercion and examines its applicability to a general class of simulations. We showed coercion to be successful with an example reuse scenario and identified the features of the process that were critical to that success. The result of this analysis of coercion is the definition of a framework that capitalizes on the advantages of both manual code modification and automated optimization. The framework specifies the protocol for practitioner (simulationist and SME) involvement such that their skills and intuition are adeptly brought to bear on the reuse problem.

The coercion process is iterative and the protocol describing practitioner involvement and responsibilities specifies when and how necessary information should be provided to accomplish code modification and optimization steps. Together, the participants in the coercion process must identify and define functional abstractions, flexible elements, constraints, and objective functions. The presence and nature of these components control the interplay between code modification and optimization. This paper presents insights regarding the subtle interactions between these components and specifies how these insights trigger particular sequences of modification and optimization.

This research continues with the development of tools that permit the inspection and visualization of simulation performance with respect to the defined flexible elements and objective function. We are encouraged that a coercion process that combines expert involvement and automatic optimization will provide new opportunities for simulation reuse.

9. References

[1] D. Brogan and J. K. Hodgins: "Simulation Level of Detail for Multiagent Control" International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 199-206, 2002.

[2] J. C. Carnahan, P. F. Reynolds, Jr., D. Brogan: "Semi-Automated Abstraction, Coercion, and Composition of Simulations" Submitted to Interservice/Industry Training, Simulation and Education Conference, 2003.

[3] S. Chenney and D. A. Forsyth: "Sampling Plausible Solutions to Multi-body Constraint Problems" SIGGRAPH 2000 Computer Graphics Proceedings, pp. 219-228, 2000.

[4] P. K. Davis and J. H. Bigelow: "Motivated Metamodels" Proceedings of the 2002 PerMIS Workshop, August 2002.

[5] P.K. Davis and J.H. Bigelow: "Motivated Metamodels: Synthesis of Cause-effect Reasoning and Statistical Metamodeling" RAND, Santa Monica, CA, 2003.

[6] D. T. Drewry, P. F. Reynolds Jr., and W. R. Emanuel: "An Optimization-Based Multi-Resolution Simulation Methodology" Proceedings of the 2002 Winter Simulation Conference, pp. 467-475, 2002.

[7] M. Gleicher: "Retargetting Motion to New Characters", SIGGRAPH 1998 Computer Graphics Proceedings, 1998.

[8] M. Gleicher and P. Litwinowicz: "Constraint-Based Motion Adaptation" The Journal of Visualization and Computer Animation, pp. 65-94, 1998.

[9] R. Grzeszczuk, D. Terzopoulous, and G. Hinton: "NeuroAnimator: Fast Neural Network Emulation and Control of Physics-Based Models" SIGGRAPH 1998 Computer Graphics Proceedings, 1998.

[10] J. K. Hodgins and N. S. Pollard: "Adapting Simulated Behaviors for New Characters" SIGGRAPH 1997 Computer Graphics Proceedings, pp. 153-162, 1997.

[11] S. Kasputis and H.C. Ng: "Composable Simulations" Proceedings of the 2000 Winter Simulation Conference, pp. 1577-1584, December 10-13 2000.

[12] E.H. Page and J.M. Opper: "Observations on the Complexity of Composable Simulation" Proceedings of the 1999 Winter simulation Conference, pp. 553-560, December 5-8 1999.

[13] M. D. Petty and E. W. Weisel: "A Formal Basis for a Theory of Semantic Composability" Proceedings of the Spring 2003 Simulation Interoperability Workshop, pp. 416-423, March 30-April 4 2003.

[14] M. D. Petty and E. W. Weisel: "A Composability Lexicon" Proceedings of the Spring 2003 Simulation Interoperability Workshop, pp. 181-187, March 30-April 4 2003.

[15] J. Popovic, S. M. Seitz, M. Erdmann, Z. Popovic and A. Witkin: "Interactive Manipulation of Rigid Body Simulations" SIGGRAPH 2000 Computer Graphics Proceedings, pp. 209-218, 2000.

[16] P.F. Reynolds Jr: "Using Space-Time Constraints to Guide Model Interoperability", Proceedings of the 2002 Spring Simulation Interoperability Workshop, 2002.

[17] P. Vicaire, P. F. Reynolds, Jr., D. Brogan: "Semi-Automated Abstraction, Coercion, and Composition of Simulations" University of Virginia Com-

puter Science Technical Reports, CS-2003-17, August 2003.
[18] A. Witkin and M. Kass: "Spacetime Constraints" SIGGRAPH 1988 Computer Graphics Proceedings, pp. 159-168, 1988.

Acknowledgements

We wish to acknowledge support from the Defense Modeling and Simulation Office, in particular from Sue Numrich, Phil Zimmerman and Phil Berry.

Author Biographies

SARAH WAZIRUDDIN is a graduate student in the Computer Science Department at the University of Virginia. She is an expert in simulation coercion and user interfaces.

DAVID BROGAN earned his PhD 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.

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 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.