Simulation Coercion Applied to Multiagent DDDAS

Yannick Loitière, David Brogan, and Paul Reynolds

Computer Science Department
University of Virginia, Charlottesville, VA 22901, USA
{ycl2r,dbrogan,pfr}@virginia.edu

Abstract. The unpredictable run-time configurations of dynamic, data-driven application systems require flexible simulation components that can adapt to changes in the number of interacting components, the syntactic definition of their interfaces, and their role in the semantic definition of the entire system. Simulation coercion provides one solution to this problem through a human-controlled mix of semi-automated analysis and optimization that transforms a simulation to meet a new set of requirements posed by dynamic data streams. This paper presents an example of one such coercion tool that uses off-line experimentation and similarity-based lookup functions to transform a simulation to a reusable abstract form that extends a static feedback control algorithm to a dynamic, data-driven version that capitalizes on extended run-time data to improve performance.

1 Introduction

Dynamic, data-driven application systems (DDDAS) emphasize the run-time flexibility of data/control systems dynamically composed from complex combinations of sensors, networks, computational resources, simulation software, and human-in-the-loop interaction. Dynamic composition of DDDAS changes the number of interacting components, the syntactic definition of their interfaces, and their role in the semantic description of the entire system. These unpredictable run-time reconfigurations pose significant challenges to the straightforward reuse of complex scientific simulations for DDDAS.

For simulation designers and developers, anticipation of and provision for all possible contexts and uses of a simulation is generally unattainable. Except under very constrained circumstances, reusing simulations without design or code modification has proven elusive. For example, a simulation may be unable to perform its desired role in a DDDAS due to decisions made as early as when an underlying scientific model was selected for its design or as late as when run-time bindings to memory structures capped the maximum amount of data permissible. Despite the challenges of building reusable simulations, additional research is justified by the many forms of benefits: employing a program in an alternative context, composing a program with others to create a new system with greater objectives, combining a program with another to broaden the levels of abstraction represented, and modifying a DDDAS to exhibit such properties as better performance or robustness. We are developing an approach to simulation transformation, COERCE, that tackles the immediate challenges of making simulations work more effectively in existing dynamic, data-driven applications and pursues the long-term challenges of specifying how today's simulations should be designed such that they can easily adapt to the unpredictable needs of tomorrow's DDDAS.

COERCE has the potential to increase the flexibility of simulation components comprising DDDAS. COERCE consists of two elements: coercion and coercibility. Coercion represents a process of transforming a simulation, using a human-controlled mix of semi-automated analysis and optimization, to meet a different set of requirements than those for which the simulation was originally designed. Through coercion, preexisting (legacy) simulations can be transformed to adapt to new or missing data sources, to change fidelity, and to change their roles in the larger data-driven system. Coercibility is a simulation property whereby designer knowledge is documented within the simulation to support future coercion. In ongoing research related to that presented here, coercibility techniques are being used to define necessary simulation metadata, including mechanisms for capturing and representing this data, so that coercion can more easily perform automated analysis and simulation transformation.

Simulation coercion and coercibility are ambitious goals that will immensely benefit the DDDAS enabling technology requirements. Not only is there great breadth in the contributing fields within computer science (software engineering, programming languages, computational science, etc.), but contributions to these goals will come from mathematics (optimization, sensitivity analysis, modeling) as well as from science and engineering. This paper explores one technique of many that must be studied in the context of simulation coercion: simulation transformation through off-line experimentation and similarity-based lookup functions. A simulation of robotic soccer serves as a complex, low-level simulation that must be transformed to interact with a high-level strategic planning algorithm. The transformation process we develop for this problem is data driven and is intended to be applicable to many similar transformations that arise at run time in dynamic applications.

2 Background

In its current form, the process of transforming a simulation to adapt to dynamic run-time data streams is primarily a manual process due to the complex logical interactions and engineering constraints that dictate a simulation's form and function. Research in the simulation reuse community demonstrates that code standardization, functional abstraction, and procedural documentation are tools that facilitate the reuse process. These 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. A growing number of research projects are demonstrating how human-guided, data-centric abstraction processes can re-represent simulation performance to suit new run-time needs.

Davis and Bigelow [2] argue that an abstraction mechanism called motivated metamodels can support simulation transformation. The code that defines a simulation is abstracted to numerical and functional models that generate the same simulation output. Consider, for example, how the stochastic models of queuing theory can replicate the automobile flow rates of a high-resolution 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 the modification of the multiagent simulation directly). No general-purpose process exists to create motivated metamodels, but the authors outline the opportunities and risks of their use.

Grzeszczuk et al. [4] introduce a specific example of simulation abstraction with their Neuroanimator. Their technique demonstrates that a computationally intensive rigid body simulator can be substituted with a trained neural network. This technique requires repeated execution of a simulation under different initial conditions in order to accumulate the database of performance data required for training, but this offline investment is recouped with improved run-time performance. 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 in run-time optimal controllers.

Additional exploration of simulation transformation has been conducted by those studying simulation coercion. Through code modification, Drewry et al. [3] demonstrate a semi-automated process that uses user-guided numerical optimization to retune simulation performance to operate in conditions different from those for which it was originally designed. Waziruddin et al. [6] further formalize simulation coercion as a formal process by constructing a formal language describing its execution and reasoning about provable outcomes in dynamic scenarios. Carnahan et al. [1] contribute to the specification of the coercion process by outlining the roles of simulationists and subject-matter experts and by producing a suite of software tools to support their collaborations. The systems produced through these research efforts are suited to run-time transformation, but currently lack a unifying theory of system transformation that serves the broad community of simulation and designers in DDDAS.

3 Simulation Transformation

This paper studies a transformation strategy for addressing misalignments between data sources. For example, suppose a DDDAS user desires to compose two simulations such that the output of one becomes the input of another. Figure 1 depicts the data relationship between the two simulations. In particular, note that the exact data required in the input vector i_2 must be contained within the output vector o_1 in order for the computational sequence to complete in a seman-

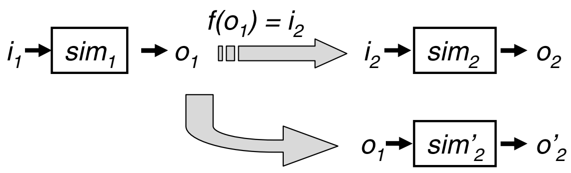

Fig. 1. In the top row, the output from simulation$_1$ aligns with the input to sim_2 to support simulation composition. If no mapping from o_1 to i_2 exists, sim_2 is mapped to a new version, sim_2' that accepts o_1 as input and generates output, o_2'.

tically meaningful way. If the exact data required for i_2 are not in o_1, a mapping function could be created to synthesize the required data from o_1: $f(o_1) = i_2$. With the assistance of a domain expert, such mapping functions may be feasible solutions. If the data in o_1 represents sensor data in the form of units/second and the data in i_2 requires units/minute, a mapping function could simply scale the output data by 60 to create the appropriate input data. Frequently, however, the mismatch between a simulation's input and the available data sources will be more significant.

To integrate two simulations with irreconcilable misalignments between their input and output data, we perform more substantial mappings. Figure 1 depicts a scenario where two simulations must work in a series such that the input to the two-simulation sequence, i_1 cannot be changed and the output from the sequence, o_2, is fixed as well. Because no effective mapping function from o_1 to i_2 exists, we instead consider mapping sim_2 to a new version, sim_2' such that the input to sim_2' is o_1 and the output, o_2', matches the requirements met by o_2. The method we use to accomplish this simulation transformation is data driven.

Our data-driven simulation transformation technique exercises sim_2 in its native environment in order to record its behavior for future abstraction. sim_2 is executed with many input instances of the form i_2. The specific instances of i_2 used during this exploration stage are guided by a user or the run-time execution to reflect conditions the DDDAS will subsequently encounter. Because the accuracy of sim_2' depends on the data generated during this exploration, the sampling of the space represented by i_2 is very important. If the simulation is well behaved in some areas of state space and chaotic or nonlinear in others, the data sampling must capture these behaviors and not the potentially infinite variety of less-interesting ones.

The exploration stage produces a database consisting of mappings from i_2 to o_2. These are point samples of the true functional relationship, $sim_2(i_2) = o_2$, encoded by the second simulation. Because the goal of this transformation process is to produce an alternative functional relationship, $sim_2'(o_1) = o_2$, we must create sim_2'. As an initial step, the data representing i_2 in the database is mapped into a format, i_2', that matches the requirements of output data, o_1. This mapping algorithm is typically easier to describe than its inverse. The transformed i_2' input data is compatible with the o_1 output data from sim_1 and we thus have a database of point samples that demonstrate the mapping sim_2 applies to an input derived from o_1. To be most useful, sim'_2 must be more than a database lookup function and must be converted to a tuned numerical model (neural network, radial basis functions, wavelets) possessing the ability to interpolate and extrapolate the data samples, in a manner semantically consistent with the behavior of sim'_2.

We note that this parameter-mapping approach is inadequate for many cases of simulation incompatibility. For example, no amount of parameter mapping can create new degrees of freedom in sim'_2 or force the frequency of data output from sim_1 to increase in order to meet sim_2's temporal requirements. The simulations will have to adjust to the run-time data properties. Furthermore semantic incompatibilities between simulations require more substantial alterations to unite their data paths. These more complex simulation transformations are currently being studied by the COERCE community.

4 Application to Soccer

We will demonstrate simulation transformation in a DDDAS created to implement a physical simulation of robotic soccer. Simulated robotic soccer teams are showcased each year in the RoboCup competition. Recent RoboCup games demonstrate a high degree of human-like intelligence by the autonomous players. Each team is composed of eleven physically simulated players, all of which are controlled during the game by an autonomous control algorithm that specifies unique player behaviors such as protecting the ball or kicking the ball up the field. The conventional player control algorithms make little use of run-time data about their opponents' behaviors, primarily because each player only has access to local information about the opponent-player locations. We seek to improve the performance of one of these conventional controllers by infusing additional data at run time that contains the locations of all the players.

We propose to improve performance by using the additional run-time data to generate predictions about future player positions. The existing soccer simulator specifies where players will be after one timestep, but it cannot accurately predict player positions further into the future. Due to the aforementioned complexity of transforming a simulation through source code modification, we interface the existing soccer simulation with its data-driven version using our data mapping approach.

The run-time data utilized by the data-driven version of the soccer simulator is obtained from a virtual camera simulation (sim_1 from our earlier example). The virtual camera is observing a simulated soccer game from a bird's-eye position above the field and outputting virtual snapshots (o_1). The soccer simulator (sim_2) must adapt to this new data and improve the quality of its agents' actions (o_2) by specifying where the players will be after one timestep and predicting their locations further into the future.

The DDDAS is a composition of sim_1 and sim_2 and their data are incompatible. The input to sim_2 is the position, orientation, and velocity of all 22 players and the ball as well as additional state variables representing each player's energy level, perceptual model, and locomotion (sprinting, turning, stopping) abilities. The output from sim_1 is a low-resolution image created by quantizing the soccer playing field and rendering colored 2-D Gaussians centered about the 22 players and the ball. Each grid cell of the quantized soccer field is represented by a pixel of the image. The final image of the field is rendered by assigning players of Team A, Team B, and the ball the colors red, green, and blue respectively. The amount of red, green, and blue each entity contributes to a pixel is summed according to the distance between the player/ball and the pixel center.

Because sim_2 cannot utilize the data generated by sim_1 in a straightforward manner, we pursue mapping sim_2 to sim'_2 according to the method described in the previous section. After the mapping, sim'_2 will be able to utilize the run-time data from sim_1 and output predictions of the players' future positions. To create the mapping, the positions of the soccer players are stored after every timestep of a simulated soccer game. Each intermediate game state serves as the simulation output of the previous timestep and the simulation input for the next timestep. This accumulated database of game performance provides the training data required to tune sim'_2. Instead of providing an exact substitution of sim_2's functionality, the transformed sim'_2 we wish to build for this example will map the image of player positions output by sim_1 at one moment to a new state many timesteps in the future in order to provide the desired predictive ability.

The output from sim_1 is semantically incompatible with the input required by sim_2. Because we are unable to transform the output from sim_1 to match the format required by sim_2, we instead transform sim_2 so it accepts the output of sim_1 as input. The database of game states that we created serves to characterize the behavior of sim_2 and we map each state to a format matching the output of sim_1 by using the camera simulation to create the corresponding image for each. Instead of storing data from sim_2's native data format, the database now describes sim_2's execution behavior as a mapping from one image of game state to subsequent images.

4.1 Building sim'_2

The image database represents the performance of sim'_2 in specific circumstances, but to be useful at run time in the DDDAS, sim'_2 must produce correct outputs for inputs that are not in the database. For a unique input, i, sim'_2 produces output in two stages: a matching algorithm uses a similarity metric to first compare i to all images in the database and then a prediction algorithm constructs o'_2 as a function of i's similarity to the database frames and their outputs. Using this process, the transformed simulation, sim'_2, will regenerate the mappings contained within the database when provided with identical inputs and will approximate the mapping for other inputs.

All the images stored in the database are represented by $\mathcal{I}$. Each image of $\mathcal{I}$ is an element of the set of all images, $\mathbf{I}$. The ordering of the image sequences in the database is preserved to maintain time-dependent frame correlations. The matching algorithm compares the camera output image to all the images in the

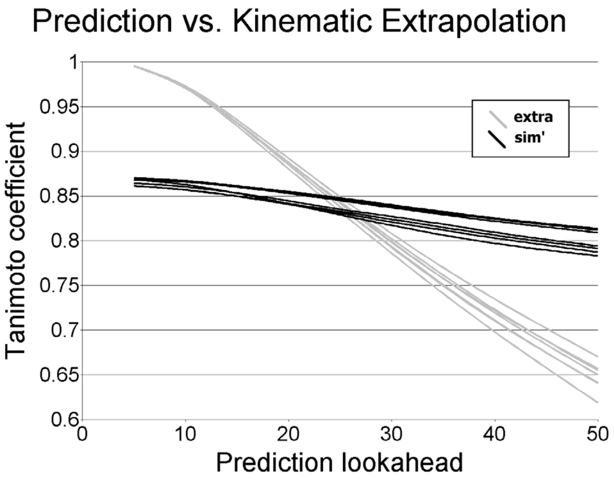

Fig. 2. This graph plots the prediction performance of the data-driven predictor (bold) and the kinematic extrapolator (grey) for seven experiments. The line for each of the seven experiments is generated by plotting the prediction lookahead value (δt) against the average of the similarity measure across all images in that test game log. A perfect score of 1.0 indicates an exact match between the predicted images and the actual future image from the test data.

database and evaluates the similarity of each pair by assigning a match value in the range $[0, 1]$. The similarity $\mathcal{T}_{pq}$ between two images $p, q \in \mathbf{I}$ is measured by computing the Tanimoto coefficient [5] between the vector representation of each image:

$$\mathcal{T}_{pq} = \frac{\overrightarrow{P} \cdot \overrightarrow{Q}}{\overrightarrow{P} \cdot \overrightarrow{P} + \overrightarrow{Q} \cdot \overrightarrow{Q} - \overrightarrow{P} \cdot \overrightarrow{Q}} \tag{1}$$

The measured Tanimoto coefficients are converted into weights, w, that map exactly to $[0, 1]$ in order to better distinguish the most similar images and to further reduce the contribution of low-similarity matches:

$$u_{pq} = \mathcal{T}_{pq} - \min_q(\mathcal{T}_{pq}) \tag{2}$$

$$w_{pq} = (\frac{u_{pq}}{\max_q(u_{pq})})^2 \tag{3}$$

After all images in the database have been assigned a weight, the output image can be computed through a weighted average of all the images.

4.2 Experimental Results

To evaluate the effectiveness of our data-driven prediction technique, we used sim'_2 to predict the game state δt timesteps in the future for every frame of a new soccer game. We constructed sim'_2 from the 6,000 state vectors of player positions obtained during the execution of one simulated soccer game. We use the Tanimoto similarity measure to compare this predicted output to the actual state to determine the accuracy of the transformed simulation. As a reference technique, we constructed a baseline kinematic extrapolator that computes an output image based on the velocities and positions of the player and ball. Because the simulated soccer players are physically simulated, the kinematic extrapolator should perform well for small δt values, but it will fail to predict the accelerations and changes in direction caused by the dynamic nature of the game.

Figure 2 demonstrates the comparison of our data-driven predictor to the kinematic extrapolator in seven different experiments. The length and width of the quantized soccer field image in each is 40 and the number of frames simulated by the two systems, δt, ranges from five to 50. Although the kinematic extrapolator performs better for small δt values, its performance degrades rapidly as δt increases and the data-driven predictor consistently outperforms it for values of δt greater than 26.

5 Conclusion

We have presented a semi-automated method for transforming – coercing – simulations to meet the new requirements that arise at run time in dynamic, data-driven applications. We have discussed an example from RoboCup where we have applied ideas from COERCE to effect data alignment and capitalize on the infusion of new data. We have experienced some encouraging success in the transformation of our position-predicting simulation, as we have reported here. Further, our study of simulation transformation has provided not only another example demonstrating the viability of COERCE-like transformations in data-driven applications, but it has also provided us with insights into how to further develop COERCE technology.

References

1. J. Carnahan, P. Reynolds, and D. Brogan. Semi-automated abstraction, coercion, and composition of simulations. In *Interservice/Industry Training, Simulation, and Education Conference*, 2003.
2. P. Davis and J. Bigelow. Motivated metamodels. In *Proceedings of the 2002 PerMIS Workshop*, 2002.
3. D. Drewry, P. Reynolds, and W. Emmanuel. An optimization-based multi-resolution simulation methodology. In *Winter Simulation Conference*, 2002.
4. R. Grzeszczuk, D. Terzopoulos, and G. Hinton. Neuroanimator: Fast neural network emulation and control of physics-based models. In *Proceedings of SIGGRAPH '98*, pages 9–20. ACM Press, July 1998.
5. T. Tanimoto. Internal report. In *IBM Technical Report Series*, November, 1957.
6. S. Waziruddin, D. Brogan, and P. Reynolds. The process for coercing simulations. In *Fall Simulation Interoperability Workshop*, 2003.