June 2000

Interactive Semi-Qualitative Simulation

Charles A. Erignac
University of Pennsylvania

Postprint version. Published in *Proceedings of the Fourteenth International Workshop on Qualitative Reasoning*, June 2000, 8 pages.

Interactive Semi-Qualitative Simulation

Abstract
In most virtual worlds, users and agents have limited interactions with the artifacts populating their environment. They can at most operate and sense them with a set of action whose consequences have been predetermined. We propose to extend their range of interaction by introducing interactive semi-qualitative simulation. This would allow them to interactively change the structure of the artifacts and use the simulator's internal models to reason about their behaviors. Virtual prototyping, virtual construction sets, and training systems are direct applications of this technology. In this paper we develop an architecture supporting this concept and report on our current and future implementations.

Comments
Postprint version. Published in *Proceedings of the Fourteenth International Workshop on Qualitative Reasoning*, June 2000, 8 pages.

Interactive Semi-Qualitative Simulation *

Charles A. Erignac
Center for Human Modeling and Simulation
University of Pennsylvania, PA 19104-6389, USA
cerignac@graphics.cis.upenn.edu

Abstract

In most virtual worlds, users and agents have limited interactions with the artifacts populating their environment. They can at most operate and sense them with a set of action whose consequences have been predetermined. We propose to extend their range of interaction by introducing interactive semi-qualitative simulation. This would allow them to interactively change the structure of the artifacts and use the simulator's internal models to reason about their behaviors. Virtual prototyping, virtual construction sets, and training systems are direct applications of this technology. In this paper we develop an architecture supporting this concept and report on our current and future implementations.

Introduction

An increasing number of simulations take place in complex virtual worlds where users and intelligent agents interact with each other. They also interact with the environment itself. Beyond simulating basic physical interaction (gravity and collision), the environment allows users and agents to interact with simulated artifacts of various complexity.

Although interaction is initiated by agents or a user interface, it is mainly supported by the model of the simulated artifact. As a consequence, complex interaction is synonymous with complex system behaviors and structures.

Currently, in most applications, interactions are limited to operating and sensing simulated artifacts. In other words, interaction is mediated through a fixed set of inputs (switches, valves, etc.) and outputs (gauges, lights, etc.). Interactions allowing structural system modifications and model-based reasoning are rarely implemented. This is mainly due to their computational costs.

Overcoming these limitations would enable highly interactive applications such as virtual prototyping tools, virtual construction sets, and virtual laboratories. Eventually, every virtual world could contain artifacts supporting a range of interactions which are, for now, only possible in reality.

We believe that interactive semi-qualitative simulation (ISQS) is a key component of such systems. It has the unique ability to dynamically assemble simulation models, generate quantitative physics-based behaviors, and produce qualitative representations of the physical world. This data is suitable for both animating a virtual world and symbolic processing by intelligent agents. It supports all possible man-machine interactions: operating, assembling, sensing, and reasoning. Furthermore, compositional modeling (Falkenhainer & Forbus 1991) techniques allow automated tailoring of simulation models for any scenario.

Our long term goal is to implement real-time environments where intelligent virtual humans would interact with complex artifacts in a realistic fashion. In particular, our current target application is a maintenance simulation system where virtual technicians carry out maintenance procedures. This system could be used to validate procedures, generate digital maintenance manuals, and train students.

In the remainder of this paper we will present our ongoing research on ISQS. First, we motivate our approach, and elaborate on a conceptual ISQS architecture. Afterwards, we present, through a case study, a semi-qualitative simulator implemented as a proof of concept. We conclude by reflecting on the experience acquired through this experiment and we outline future research directions to improve our system.

Why ISQS?

Simulations where the user or intelligent agents interact with simulated artifacts can be found in video-games, virtual worlds, virtual laboratories, and training applications. However, because of modeling and performance limitations, their scenarios provide only limited interaction.

In order to better understand these limitations let us define four agent-artifact interaction categories:

Operational: The agent operates an artifact. It uses

*This research was partially supported by the U.S. Air Force through Delivery Orders F41624-97-D-5002-17, and F33615-99-D-6001-1.

the artifacts controls to change its internal state. For example, flipping a switch turns On or Off a device.

Structural: The agent changes the structure of an artifact. It removes or adds parts or disconnect artifacts. For example, disconnecting a pipe from a tank changes the structure of the system.

Sensory: The agent senses the state of an artifact. For example, reading a gauge is a sensory action.

Cognitive: The agent uses the simulator and its models to reason about the structure and behavior of an artifact. For example, an explanatory agent will analyze a simulation trace and models to answer a query. An agent can also enlist the services of the simulator to run a separate "what if" simulation.

In general, an action belongs to more that one interaction category.

So far, only operational and sensory interactions have been widely implemented. For example, virtual laboratories (Schmid 1999) and training programs (Goad 1999) allow users to navigate through a 3D environment and operate buttons and valves to control physical systems. However, the simulated artifacts cannot be modified.

The tutoring agent Steve (Rickel & Johnson 1999) teaches the operations of a virtual ship engine room. It demonstrates and explains tasks, and it supervises students. Although virtual machinery is simulated, Steve does not use the underlying models to explain principles of operation. It uses, instead, separate causal networks.

Systems, such as the CyclePad (Forbus & Whalley 1994) and the "How Things Work" electronic encyclopedia (Amador 1994), perform real cognitive interaction. Their agents analyze and debug user assembled systems. However, since the simulations run in batch, these applications fall short from being a truly interactive.

The reason for the predominance of operational and sensory interactions is simple. Of the four categories, they are the only ones that can run in real time. In order to achieve acceptable performance, artifacts are simulated with highly optimized ad-hoc numerical models.

An artifact's structure either disappears during optimization or is static. This means that a simulator cannot handle unanticipated structural modifications. Furthermore, these "diluted" models cannot be shared with participating agents.

Building an optimized model supporting structural modification would require enumerating all possible assemblies and compiling each of them ahead of time. Although this solution is feasible for very restrictive scenarios, it is impractical for applications based on unrestricted structural manipulation such as virtual laboratories, and virtual prototyping tools. Their set of possible assemblies are infinite.

Structural and cognitive interactions are related. Predicting interactive structural modifications is untractable. The only alternative is to have the simulator assemble its equation model at run-time. This automated model building and solving is a form of model-based reasoning. It uses explicit structural and behavioral models. They could be shared, via cognitive interaction, with model-based reasoning agents. Up to now, real-time model building has proved impractical. As a consequence, typical run-time simulation models are devoid of any information suitable for either structural or cognitive interaction.

These performance-driven limitations have prevented the development of virtual environments where artifacts have interactively reconfigurable structure and inspectable models. Such a technology would enable virtual environments where users and agents perform configuration, assembly, operation, analysis, and explanatory tasks.

So far only semi-qualitative simulators such as CDME (Iwasaki *et al.* 1997), SIMGEN (Forbus & Falkenhainer 1995), and Pika (Amador 1994), have implemented structural and cognitive tasks in general purpose frameworks. These simulators are respectively used for computer assisted engineering, self-explanatory simulations, and electronic encyclopedias. They are not however adequate for ISQS. They are batch simulators which prohibit user interaction.

More fundamentally, CDME and Pika are interpreters. They will not scale-up. SIMGEN produces compiled simulations in C, but its artifact structures are static.

Finally, their modeling languages use modeling units, *model fragments*, of limited expressiveness. A fragment can only capture one operating mode. A complex device model, having many modes and sub-modes, ends-up being scattered across many units which, according to language semantics, are not related. This complicates model authoring and maintenance as well as model-based reasoning.

Our work focuses on developing an ISQS framework for virtual environments. It is composed of an interaction architecture and a domain independent interactive semi-qualitative simulator[1]. The architecture addresses the problem of integrating the simulator in a virtual environment simulation system and its interfacing with users and agents. The simulator itself is a preliminary answer to run-time performance and modeling issues. We present them in the two following sections.

Interactive Simulation Architecture

The ISQS architecture is simple: a simulation engine can be considered a regular agent in a virtual environment. In order to reach this conclusion, we will first define the nature of agent-artifact interaction. Then by discussing avatars and agent architectures, we will see how an engine can interface uniformly with users and agents alike.

[1]We will also refer to it as a *simulation engine*.

The Nature of Interactions

We define the *interaction* between entities as a multi-directional flow of *actions*. Each action is either a discrete or continuous process initiated by one entity, the *subject*, and directed towards others, the *objects*. Subject and objects can be affected by the action's *effects*.

A discrete action is equivalent to a *message* sent from the subject to the object. For example flipping a switch can be modeled as a discrete action. A continuous action consists of applying constraints until a *termination condition* is satisfied. For example keeping a car key on Start until the engine turns over is a continuous action. Continuous actions are initiated and terminated via message passing.

Actions are either *primitive* or *complex*. Primitive actions correspond to built-in behaviors of the subject and objects. Complex actions are assembled by composing other actions.

We call a set of action definitions a *vocabulary*.

Uniform Interaction Interface

In a virtual world users and embodied agents[2] have a virtual body, giving them a physical existence in the virtual environment. The virtual embodiment of a user is called an *avatar*.

Contemporary agent architectures are layered (Gennings, Sycara, & Wooldridge 1998; Kendall *et al.* 1998). Low-level layers handle sensory input and *reactive* or built-in behaviors such as locomotion, reach, grasp, and attention. Mid-level layers control task execution and monitoring. High-level layers perform communication with other agents and task planning.

Depending on the modeled sensors, the state of the world may be totally or partially *observable*. Many virtual worlds are only partially observable by the agents and users.

The vocabulary of actions an agent can perform is stored in a knowledge base (KB). The reactive layer executes primitive actions. The task execution layer, a.k.a. *executive* layer, uses the KB to carry out complex actions.

We assume that avatars and agent models of the same kind, let us say human, share the same low- and mid-level layers as well as the communication layer. They may also share the same KB. Aside from making them equal in terms of physical abilities this solution allows for the implementation of a *uniform interaction interface*. In other words, agents and avatars share the same action vocabulary and sense the environment in the same way. This includes the effects of actions. They can also act by using the same primitive actions. Furthermore, they use a common communication protocol.

In Object-Oriented terms, the layers that differentiate an avatar from an agent are encapsulated inside the same interface. This means that agents and avatars of the same kind are also perceived the same way (see Figure 1). The only difference is that some high-level layers in agents are replaced by the user command interface in the avatars.

[2]We will use the term *agent* regardless of whether it is embodied or not. We will be more explicit only to prevent ambiguity.

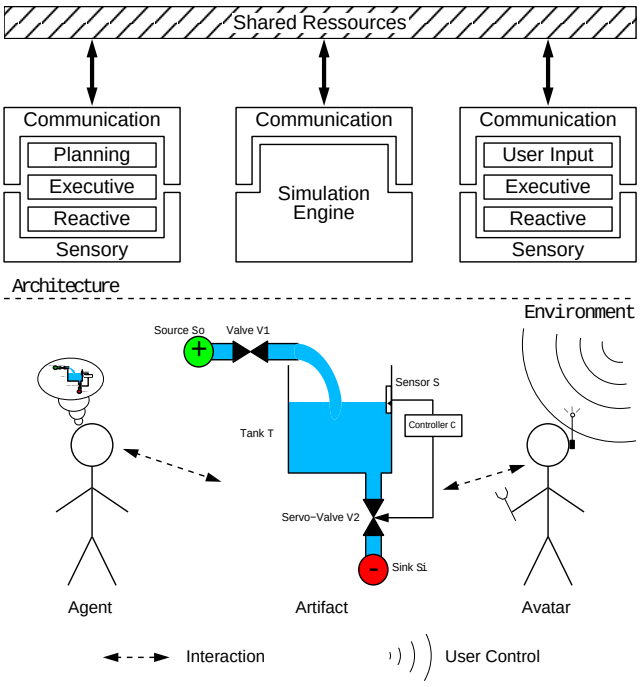

Figure 1: Sensory and communication layers encapsulate agent and avatar architectures within the same interaction interface. The simulation engine uses a standard communication layer to be integrated as a generic agent.

Depending on the interface modalities and the abstraction level of user commands, the avatar will have more or less autonomy to carry out its orders. This autonomy is achieved by sharing with agents some complex behaviors produced in certain high-level layers.

In our framework, avatars and agents of a similar kind can only be distinguished by their degree of autonomy. As far as *interaction* is concerned agents and avatars are identical. For the remainder of this paper will use the term *agent* to refer to both fully autonomous or user controlled entities.

Agent-Simulator Interaction

Each artifact is ultimately modeled and simulated inside a simulation engine. In order to interact with an artifact, an agent must possess a least a minimal vocabulary of primitive actions interpretable by the engine.

We take the position of letting an agent and the engine have their own representation of the same action. In other words, actions are explicitly modeled in the domain theory used by the engine. This maximizes decoupling between the engine and various kinds of agents. Also, actions are reified and can be subjects of agent reasoning. We rely on message passing between agents

and the engine to initiate, terminate, and diagnose actions.

Cognitive actions raise the issue of *observability*. Some agents should only have limited access to the models and unobservable states of the artifacts. The engine should answer agents' queries within the limits of their sensory abilities. However, "extra-sensory" skills are commonly used by explanatory or tutoring agents (Rickel & Johnson 1999).

The nature of an action ultimately depends on the complexity of the agent and artifact models. In particular this determines the amount of feedback an agent requires from the simulator during an action. The feedback indicates whether an action has failed or is successful, and when it is completed. Ideally, feedback should be implemented in the sensory component of the action; that is as a termination condition or a sub-action.

For example, filling a tank to a certain level can be implemented as a discrete action where the engine side of the action simply assigns a value to the fluid level in the tank. It can also be implemented as an autonomous continuous process where a valve is left open until the prescribed level is reached. Finally, it can be a feedback loop where the action is implemented, on the agent side, with an action opening a valve, a wait action whose terminating condition is predicated on the fluid level, and an action closing the valve. The third solution is the most realistic as it decouples the artifact from the agent by only using primitive actions on the engine side.

It is important to note that some actions can be initiated by the engine. An agent can register with the engine its interest into sensing certain events. When the event occurs, the engine sends a message to the agent. This mechanism allows implementing termination conditions without systematic agent polling.

A Simulation Agent

The simulation engine is similar to a regular agent. It maintains internal processes initiated internally or by external events. These processes trigger further interactions with other entities. It also uses the common resources of the virtual world, such as inter-agent communication (messaging), and the scene management system which supports the geometric representation of the world. Indeed each artifact has a geometric model whose appearance, motion, orientation, and position are conditioned by its internal state as modeled in the engine. For example, the value read by a simulated gauge will translate into a rotation angle of the needle in the corresponding gauge 3D model.

Access to these resources is achieved by fitting the engine with a standard communication interface (see Figure 1).

Therefore, as far as the software platform supporting the virtual environment is concerned the simulation engine is an agent. This is the approach we will use to integrate our next generation simulator. Our target platform is the Parametric Action Representation (PAR) system (Badler, Palmer, & Bindiganavale 1999). This system supports a run time environment for avatars and agents. It provides messaging, KB services, shared working memory, action execution, and time sharing.

Case Study

We developed a simulation engine and tested it with an interactive simulation case study. It consists of a web-based application to practice virtual maintenance procedures on hydraulic systems. The application comprises an interactive user interface remotely connected with a simulation server.

We did not fully implement the ISQS architecture. This experiment was meant to evaluate the technical difficulties of introducing interaction into a semi-qualitative simulation.

Scenarios

Two scenarios with different tasks were developed to illustrate various hazard models and complex maintenance procedures. The scenarios are briefly described as follows:

First Scenario: A pipeline composed of a pipe between two isolation valves interconnects a water source and sink. The source and sink are respectively pressurized at 3 and 1 Atmospheres[3]. The maintenance task is to disconnect the pipe from the system (see Figure 2).

Second Scenario: An open tank receives a steady inflow of water from a valve. Its water level is regulated by a controller driving a servo-valve to maintain the appropriate outflow. The maintenance task is to disconnect the servo-valve from the system (see Figure 3).

The first scenario is meant to test basic interaction on a simple system as well as demonstrating hazard detection. These hazards, leaking and sudden decompression, are modeled as processes. The second hazard detects the disconnection of a pipe containing residual pressure. Depending on the order in which the valves were closed pressurized water can be trapped in the pipe.

The second scenario illustrates the complexity of preparatory steps required prior to removing the valve. It also allows the user to interact with a self-regulated (reactive) physical system. The user has a choice between planning the whole task on its own or letting the program execute preplanned complex actions such as draining the tank and removing the servo-valve.

Draining the tank closes the inflow valve, turns off the controller and opens the servo-valve manually. It ends when the tank is empty.

The *remove servo-valve* action starts by calling the *drain tank* action if required, then it disconnects the servo-valve from the hydraulic and electric circuits.

[3]Pressure measurements are absolute.

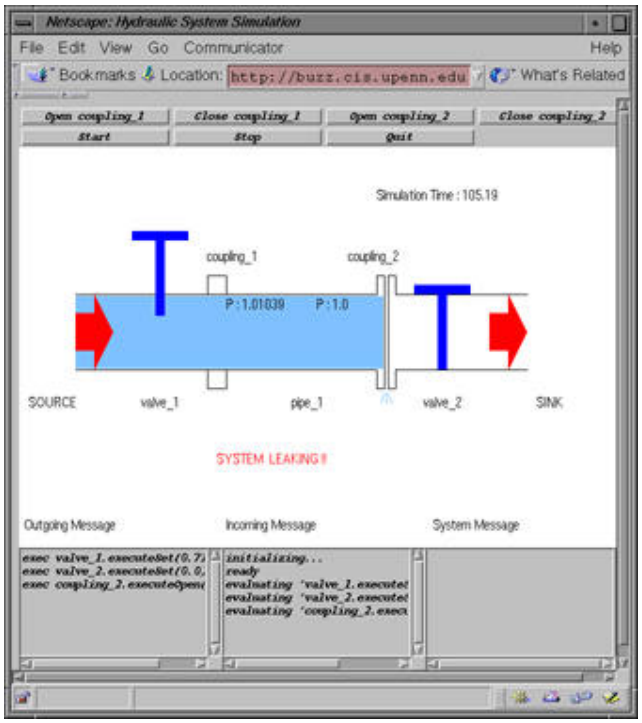

Figure 2: First scenario's user interface.

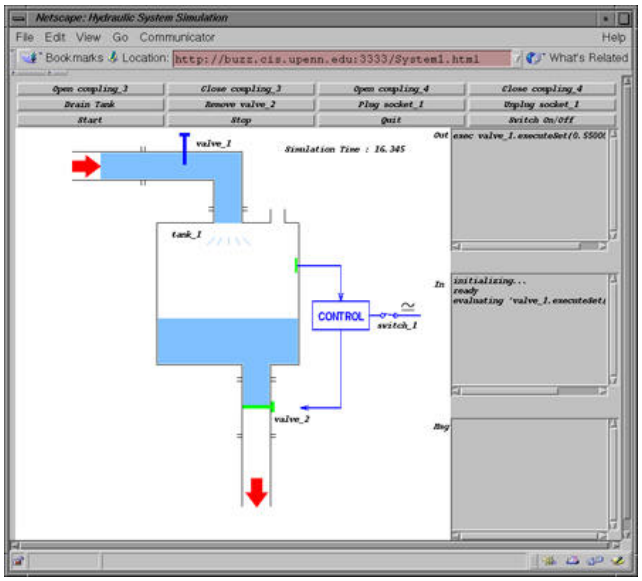

Figure 3: Second scenario's user interface.

Architecture

Both scenarios are presented to the user through a dedicated Java program. Each applet controls the user interface which consists of animated schematics of the hydraulic system, buttons to control the simulation and perform actions, and diverse status text boxes.

The valve schematics can be interactively dragged with the mouse to set their aperture. Both scenarios have buttons to couple and decouple the components of interest from the rest of the system. The second scenario has buttons to start complex actions such as draining the tank and removing the servo-valve.

The simulator uses an Object-Oriented modeling language supporting procedural, rule-based, and constraint-based behaviors. In particular, constraints are logical clauses, algebraic equations, and differential operators. These constraints can be explicitly grouped into qualitative states. User-defined transitions control the hybrid behavior.

Devices, and processes of the hydraulic domain are defined in separate fragments. On the simulator side, actions are directly modeled as procedures or processes.

The applets translate a user action into a procedure call (or any type of statement supported by the language) and ship it in textual form to the simulator which interprets it. Each applet runs in lockstep with a remote server hosting the simulation. After an integration step, the simulator sends update messages for each variable that changed.

The simulator uses a pattern directed inference engine, a LTMS (McAllester 1978) and custom code to build qualitative states. Its solver uses a symbolic Gauss pivoting method and explicit Euler integration. This method allows solving differential algebraic equation (DAE) systems of the form $0 = f(x', x, y, u)$ where the algebraic unknowns y and the state vector derivative x' are in linear form. Although this method only applies to a restricted class of DAE systems it is fast.

Threshold crossing is detected after each pivoting and integration steps[4][5].

Evaluation

This case study is a proof of concept to validate the interactive semi-qualitative simulator we developed for ISQS. We focussed on the following points:

- Interaction Support.

  Operational, sensory, and structural interactions were successfully implemented. In our system, the applet plays the role of an avatar. Since it was just an interface, complex actions were directly modeled in the simulation domain. Cognitive interaction remains to be tested.

- General Purpose Engine.

  The engine is scenario and domain independent. Each scenario is described in a model fragment. It is passed by the client to the server when they initiate their connection. The engine's modeling language can model complex devices, such as a servo-valve or a controller, with finite state machines, and processes, such as liquid flows, with self-instantiating fragments.

[4] A more accurate root finding method, such as bisection (Shampine, Allen, & S. 1996), will be used in the future.

[5] For more information on the simulation engine and its modeling language see (Erignac 1999).

- Suitability of ISQS as a training tool.

  Although we did not implement a complete training system, the user is able to experiment freely with the hydraulic systems and perform maintenance procedures. Other features such as self-explanatory and tutoring functions would be necessary to complete the application.

- Ease of developing a hydraulic and control domain that supports partially assembled systems.

  The hydraulic domain theory we developed models pressurized tanks and single phase flows as in (Collins & Forbus 1987). It is, however, far more complex than their approach, because it supports partially assembled systems. In fact, we model hydrostatic state and residual pressure in pipes. The theory also has an environmental model and associated hazards as in (Catino 1993). Finally, liquid flows are dynamic to include transient behaviors.

  The equation solver of the engine is simple. This made our model fragments quite complex especially since we modeled static and dynamic regimes in pipes.

- Real-time performance of the simulation engine.

  The simulation engine is an interpreter. Although it is able to deliver real-time performance for a given qualitative state, regenerating equation models takes a few seconds during which the simulation is suspended.

The case-study shows that ISQS is feasible. The user is able to operate, remove, and reinsert parts in the hydraulic systems. The environment reacts continuously to these actions with the appropriate physical behaviors (including hazards[6]). We address the outstanding issues in the next section.

Future Work

The proof of concept presented above deals mostly with building an interactive semi-qualitative simulator. Modeling a hydraulic domain and simulating it revealed that the simulator's run-time performance and modeling expressiveness need improvement. However user interactions were successfully modeled and processed by the system.

Implementing the whole ISQS architecture requires developing adequate agent-simulator communication. In particular, we need a common language to enable cognitive interaction, that is exchanging behavior models and simulation traces. This will be our next task once some of the following proposed improvements to the engine are made.

Run-Time Performance

The key factor to improve is run-time performance. Without it ISQS will not be able to leverage its unique interaction capabilities against fast static simulations.

The performance of the engine is affected by the speed at which it can build and solve its equation models. It builds an equation model once along with its resolution plan. This plan is interpreted at each iteration to solve the system. Both models and plans are cached for ulterior reuse during the simulation.

The engine does not deliver good real-time performance because it is interpreted and lacks optimization. We plan to optimize the code dealing with threshold detection, state transition, and truth maintenance. Also, an incremental solver, as in (Amador 1994), would minimize the amount of replanning when a new system is built.

More importantly, we will experiment with adaptive just-in-time compilation of resolution plans. This means converting the most frequently interpreted plan into a C function, compiling it, and loading it as dynamic library. The function would substitute the plan's interpretation.

This method is similar to run-time profiling of Java programs and the compilation of the most used code segments performed in the HotSpot Java Virtual Machine (Sun Microsystems Inc. 1996). Other elements of the simulation, such as threshold detection, can also be compiled. Our proposed compilation scheme only improves run-time performance within a given qualitative state. Latency should be expected the first time a new state is encountered.

In order to truly remain interactive we must predict the next qualitative state and solve its equation model ahead. One solution would be to look ahead in time until a new state is encountered. Because agent interaction is not predictable, this new state might not be the one actually reached. Another solution would be to use a persistent cache to store the qualitative states of a given scenario across simulation sessions.

Modeling Construct Expressiveness

After simulation speed, having expressive modeling constructs is the main concern of our framework. This impacts the complexity of the modeling process as well as the workload of the engine.

The engine's modeling language uses finite state machines to capture different operating modes within one fragment. Although this proved useful to model valves, the sub-modes of the electronic components had to be flattened. This produces multiple states containing the same constraints and complex transition patterns. Complex devices are better modeled with hierarchical finite state machines such as Statecharts (Harel 1987). We will make our states hierarchical to keep our models compact and true to mode and sub-mode hierarchies.

The ease of modeling a wide range of continuous behaviors depends on the power of the engine's equation solver. If the solver is not powerful enough, a model fragment will be burdened with additional states and transitions to ensure that its active constraints are always solvable. Furthermore, operation modes of related

[6] Hazards actually helped us debug our assemblies when we failed to connect certain pipes while building our scenarios.

fragment instances must be kept synchronized to ensure the solvability of the whole equation model. As mentioned earlier our fragment models are quite complex because of the solver's simplicity.

Solution techniques from the field of quantitative simulation could improve the solver's expressiveness. For example, index reduction allows using an algebraic constraint regardless of whether its variables are unknowns or state variables. In the hydraulic domain this equates to using a flow conservation equation for both transient and quasi-static regimes. More generally, index reduction resolves the singularities occuring when assembling certain devices.

Enhancing the solver with such a method would simplify connecting and modeling fragments. Real-time performance however remains to be assessed. Although complex solving methods will slow down the solver, they will require less frequent equation model switching.

Conclusion

We presented a framework for implementing interactive semi-qualitative simulation (ISQS) in virtual environments. This novel application of semi-qualitative simulation breaks the limitations of traditional quantitative simulation where users and agents can only operate and sense precompiled artifact models. It gives them the ability to interactively modify artifact's structure and access their simulation models. We developed a semi-qualitative simulator and built a web-based maintenance simulator around it. This experiment confirmed the feasibility of ISQS provided the simulation engine is upgraded to run at interactive rate. Several methods where proposed and will be implemented.

Acknowledgment

We want to thank Hogeun Shin for developing the web-based architecture of the case study, and the referees whose constructive comments helped us improving this paper.

References

Amador, F. G. 1994. Self-explanatory simulation for an electronic encylopedia. Technical Report TR-94-07-06, University of Washington, Department of Computer Science and Engineering.

Badler, N. I.; Palmer, M.; and Bindiganavale, R. 1999. Animation control for real-time virtual humans. *Communications of the ACM* 42(7):65–73.

Catino, C. A. 1993. *Automated Modeling of Chemical Plants with Application to Hazard and Operability Studies*. Ph.D. Dissertation, Department of Chemical Engineering, University of Pennsylvania, Philadelphia, PA.

Collins, J. W., and Forbus, K. D. 1987. Reasoning about fluids via molecular collections. In *Proc. 6th National Conf. on Artificial Intelligence (AAAI-87)*, 590–594. San Mateo, CA: Morgan Kaufmann.

Erignac, C. 1999. Semi-qualitative simulation for virtual environments. In Price, C., ed., *Thirteenth International Workshop on Qualitative Reasoning*, 73–83.

Falkenhainer, B., and Forbus, K. D. 1991. Compositional modeling: Finding the right model for the job. *Artificial Intelligence* 51(1-3):95–144.

Forbus, K. D., and Falkenhainer, B. 1995. Scaling up self-explanatory simulators: Polynomial-time compilation. In *IJCAI95*, 1798–1805.

Forbus, K., and Whalley, P. 1994. Using qualiative physics to build articulate software for thermodynamics education. In *Proceedings of AAAI-94*, 1175–1182.

Gennings, N.; Sycara, K.; and Wooldridge, M. 1998. A roadmap of agent research and development. *Autonomus Agents and Multi-Agent Systems* (1):7–38.

Goad, C. 1999. A language-level attack on compisitional
simulation. Technical report, The Behavior Engine Company, http://www.besoft.com/cssindex.html.

Harel, D. 1987. Statecharts: A visual formulation for complex systems. *Science of Computer Programming* 8(3):231–274.

Iwasaki, Y.; Farquhar, A.; Fikes, R.; and Rice, J. 1997. A web-based compositional modeling system for sharing of physical knowledge. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, 494–500. San Francisco: Morgan Kaufmann Publishers.

Kendall, E.; Murali Krishna, P.; Pathac, C.; and Sureash, C. 1998. Patterns of intelligent and mobile agents. In *Proc. Int. Conf. on Autonomous Agents*, 92–99. ACM Press.

McAllester, D. A. 1978. A three valued truth maintenance system. (MIT) Artificial Intelligence Memo 473, Department of Computer Science, Massachusettes Institute of Technology.

Rickel, J., and Johnson, L. 1999. Animated agents for procedural training in virtual reality: Perception, cognition,and motor control. *Applied Artificial Intelligence* (7):343–382.

Schmid, C. 1999. A remote laboratory using virtual reality on the web. *Simulation* 73(1):13–21.

Shampine, L.; Allen, R. J.; and S., P. 1996. *Fundamentals of Numerical Computing*. John Wiley & Sons. ISBN: 0471163635.

Sun Microsystems Inc. 1996. The Java HotSpot perfomance architecture. A White Paper About Sun's Second Generation Performance Technology.