Proceedings of the 2012 Winter Simulation Conference
C. Laroque, J. Himmelspach, R. Pasupathy, O. Rose, and A.M. Uhrmacher, eds

ENABLING BEHAVIOR REUSE IN DEVELOPMENT OF VIRTUAL ENVIRONMENT APPLICATIONS

Huaiyu Liu
Mic Bowman
Warrant A. Hunt
Intel Labs
2111 NE 25th Ave, JF2-04
Hillsboro, OR USA

Aaron M. Duffy

Utah State University
5305 Old Main Hill
Logan, UT USA

ABSTRACT

Virtual environments (VEs) provide simulated 3D spaces in which users can interact, collaborate, and visualize in real time. Accordingly, virtual environments have the potential to transform education, creating classrooms that ignore geographic boundaries and immerse students in experiences that would be difficult or impossible to arrange in the real world. A major impediment to the widespread adoption of educational VEs is the high cost of developing VE applications. We believe application development must become tractable for non-expert users in the same way that Web development is no longer the exclusive purview of professional programmers. In this position paper, we describe our experiences in enabling behavior reuse across VE applications. Our approach replaces, whenever possible, application-specific behaviors with general purpose, reusable simulation modules. These modules bootstrap one another until a rich ecosystem develops; thus, VE application development is reduced to *compositing* content and behaviors instead of *developing* them from scratch.

1 INTRODUCTION

3D Virtual Environments (VE) use simulation to create multiplayer (and often massively multiplayer), three-dimensional, persistent virtual environments to provide users with real-time, interactive experiences. Over the years, many educational VE applications have been developed to help students engage and master educational content (AWEDU 2012, EducationGrid 2012, SimTeach 2011, ScienceSim 2012). These applications improve learning experiences in several areas (Daden 2010): supporting distance learning, changing learner dynamics, improving retention (e.g. improving conceptual understanding of complex science subjects by interactive simulations), gaining a subjective view, reducing costs, and doing the impossible (e.g. visualizing events at scales or speeds that make direct observations difficult or modeling extreme cases or exaggerated situations). They also provide environments for educators to collaborate within. Many universities, K-12 schools, international corporations, and government agencies are exploring the use of VE applications to educate their students or employees. Virtual Worlds Watch (2009) estimates that over 95% of UK Universities are using or experimenting with virtual worlds, and the Second Life Education Directory (2012) lists over a hundred US educational institutions.

For all their willingness to embrace the VE technology and bring it into their classrooms, educators face enormous hurdles in developing VE applications cheaply and efficiently. Roughly speaking, VE application development comprises three components: the context in which the application exists, a set of 3D objects (including avatars or agents), and behaviors defined on those objects. Building 3D objects require skills in 3D model creation. Furthermore, to add behaviors to those objects, programming skills are required. As such, designing and developing VE applications is a specialized and time consuming task: the Water Wars civic game (Hirsch 2012), for instance, required more than nine man-months of professional programming time.

The present state of VE application development is reminiscent of the Web circa 1995. At that time, every Web site was built with tightly-coupled content and presentation. Since then, Web development has matured to the point where “anyone” can develop high quality Web content using established standards. Two examples are Cascading Style Sheets (CSS) and Web templates. With CSS, the formatting and style of a Web page are separated from its content, which allows content producers to reuse existing style sheets when presenting novel content. Web templates similarly separate content from presentation via modular components that can be modified independently (such as headers, footers, navigation bars, etc.). In essence, many generic, reusable components and templates are available today, providing "ready-made" Web solutions for the majority of content producers.

For virtual environments to realize their full potential, we believe that VE application development must be similarly transformed to simplify the design and development process for non-expert users. Several modular solutions for 3D content creation already exist, such as the Second Life marketplace (2012), Google Warehouse (2012), and data-driven modeling tools (Chaudhuri and Koltun 2010). In this paper, we describe our experiences enabling reusable behaviors across virtual objects, endowing 3D models in virtual environments with sufficient actions to create immersive, interactive experiences.

We view a VE platform as a set of simulation engines that drive behaviors, where behaviors are essential state changes in a virtual space. The richer the set of simulation engines in a VE platform, the richer the intrinsic behaviors available to VE application developers. However, existing VE platforms, such as Second Life (2012) and OpenSimulator (2012), only include a limited set of simulation engines out of the box. In these platforms, VE application developers are forced to painstakingly and laboriously define behaviors via customized scripts. The more such behaviors can be offloaded to simulation engines, the better the performance and less work in developing VE applications. Our recent work in creating several VE applications followed this trend. For example, Virtual Fernland (Campbell et al. 2010, VPGsim 2012) extended OpenSimulator with five domain specific simulation modules to simulate thousands of virtual ferns that evolve over time in response to environmental changes.

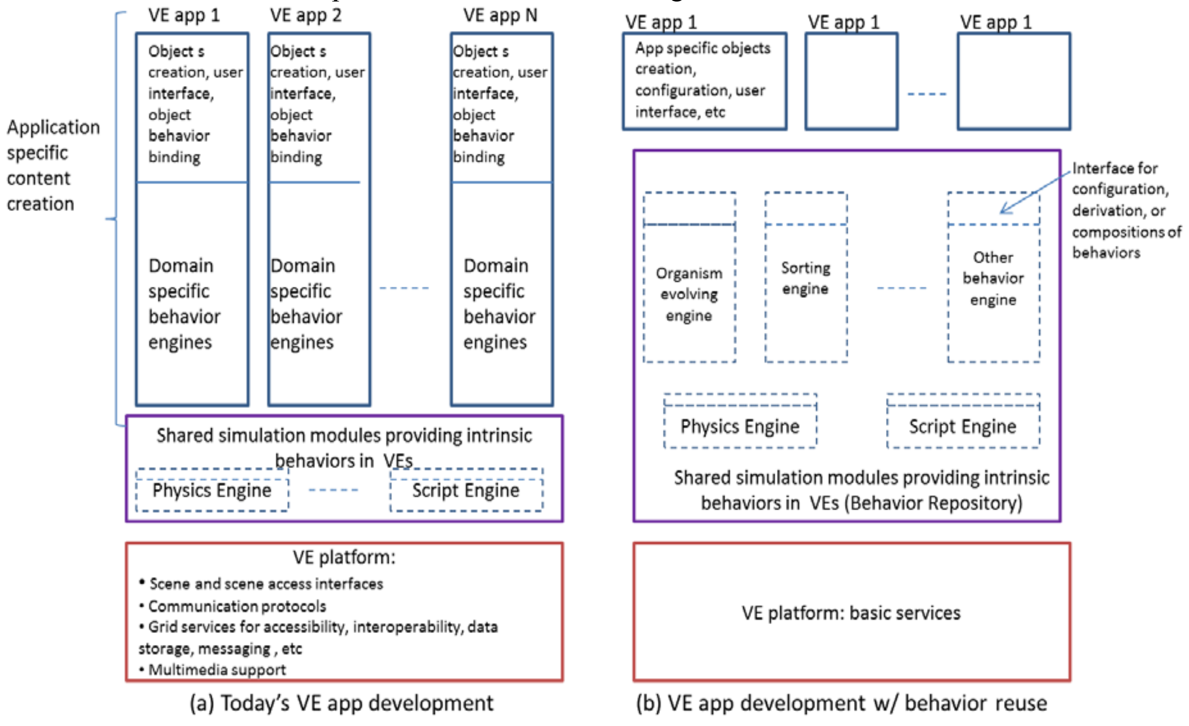

VE app 1
VE app 2
VE app N
Application specific content creation
Object s creation, user interface, object behavior binding
Domain specific behavior engines
Shared simulation modules providing intrinsic behaviors in VEs
Physics Engine
Script Engine
VE platform:
• Scene and scene access interfaces
• Communication protocols
• Grid services for accessibility, interoperability, data storage, messaging , etc
• Multimedia support
(a) Today’s VE app development
VE app 1
VE app 1
VE app 1
App specific objects creation, configuration, user interface, etc
Interface for configuration, derivation, or compositions of behaviors
Organism evolving engine
Sorting engine
Other behavior engine
Physics Engine
Script Engine
Shared simulation modules providing intrinsic behaviors in VEs (Behavior Repository)
VE platform: basic services
(b) VE app development w/ behavior reuse

Figure 1: Transforming VE application development

Yet it is hard to reuse the behaviors defined in the application specific simulation modules. Objects and behaviors are tightly bound, there is no decomposition of generic behaviors from application specific behaviors, and no services from the VE platform are available to support reuse. The 3D application development today looks like what is shown in Figure 1(a), where there is no behavior reuse across applications and each VE application is built as independent, solo effort that requires many, many hours of skillful work. Although there are research projects on modeling object behaviors (Kallmann and Thalmann 1999, Peters et al. 2003) and behaviors of intelligent agents (Perlin and Goldberg 1996, Luengo 2004), not much work has been done in behavior reuse in VEs, especially in enabling reuse across applications on state-of-art VE platforms.

In our work, we found that leveraging the fact that simulation in a VE platform is extensible and using the extensibility in a particular way enables reuse. Our approach includes two major parts. First, we decomposed VE applications into generic simulation modules and application specific modules. The generic simulation modules provide control, data exchange, and event interfaces whereas the application specific modules define the behavior of a specific simulation. Second, we enhanced the VE platform (OpenSimulator in our work) with services that make the above interfaces accessible to other modules and scripts. In short, we replace, whenever possible, application specific code with general purpose (and reusable) simulation engines, which in turn, become a part of the simulation engines composing the virtual platform and can be reused across VE applications. By doing so, we effectively transformed the VE application development to a pattern as shown in Figure 1 (b).

The reminder of the paper is as follows. In Section 2 we review three VE applications we developed for education and training. In Section 3 we describe an approach that enables behavior reuse in a state-of-art VE platform, OpenSimulator. In Section 4, we discuss as future work a potential approach to simplifying VE application development through reusable patterns implemented in a domain-specific language. We then conclude in Section 5.

2 CONTENT CREATION IN EXISTING EDUCATIONAL VIRTUAL ENVIORNMENTS

ScienceSim (2012) is a virtual environment where scientists, educators and students come together to explore VE applications for visualization, education, training and scientific discovery. In this section we review the application development process, in particular the behavior creation process, in three ScienceSim applications we have built, namely, Virtual Fernland (Campbell et al. 2010, VPGsim 2012), N-body Virtualization (2012), and Sort Lab (QuickSort Visualization 2012).

2.1 Backgound: OpenSimulator

The ScienceSim virtual environment is built based on the OpenSimulator (2012) platform. OpenSimulator (OpenSim, for short) is an open source, widely-used multi-user VE application platform, which is also compatible with the Second Life viewer protocol. The core of an OpenSim system is a set of simulators hosting interactive virtual environments and supporting multi-perspective visualization. Each simulator provides a simulation framework that integrates a number of simulation engines.

The functionality of OpenSim simulators can be extended through plugin modules, called region modules. OpenSim provides an interface for region modules to register and be integrated the simulation framework, gain access to objects in the Scene (the data structure stored all objects and their properties in the virtual space), and subscribe to or trigger in-world events to drive state changes. We leveraged the simulation extensibility in OpenSim and developed specific region modules, which we refer to as the simulation modules, to drive application specific behaviors.

2.2 Virtual Fernland

Virtual Fernland was developed by biologists and educators, initially as a tool for modeling gene flow in fern populations, but later to teach principles of population genetics theory through hands-on science inquiry activities. The system simulates a population of virtual ferns that evolves over time in response to natural selection, random genetic drift, migration, and non-random mating. The developers were not 3D experts but recognized that the ability to generate, modify and control large numbers of virtual objects through simulator modules could be used to create rich interactive genetic simulations.

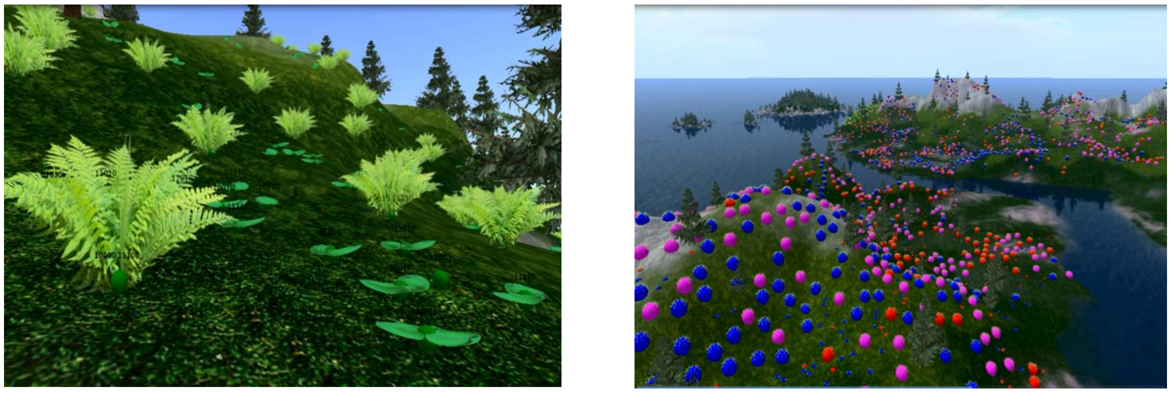


Figure 2: Left: Simulated Ferns with three stages of lifecycle. Right: A population of tens of thousands of virtual ferns, visualized in different colors by the presence or absence of a particular gene.

The Virtual Fernland is implemented as a set of five simulation modules that generate and interact with thousands of individual scripted objects and allow users to control and monitor the simulation through web interfaces (Figure 3):

- Manager module: As each individual scripted plant is generated by the Manager module, it is provided with a virtual genome 'inherited' from its parents (or from a scripted fern planter when the user initiates the simulation).
- Soil module: Each plant receives information about the current environment at its location from a simulation-specific Soil module (drainage, salinity, fertility) and from OpenSim shared modules (light, wind, cloud cover, presence of neighbors).
- Parameters module: It provides the plant with user configuration information. Users interface with the module through web forms to configure several traits that control how well the plant thrives and reproduces and whether each trait is affected by the plant's genome. The plant uses all of this information to simulate its own growth, reproduction and eventual death. Over time, plants with genetic traits that are better adapted to the environment live longer and reproduce more often, so the genes controlling those traits become more common in the population.
- Visualization module: Because the environment varies in both time and space, complex patterns develop across the virtual landscape. Users are able to view these patterns through the Visualization module that modifies the appearance of the plants based on their genomes.
- Summary module: It tracks, logs and plots (on a webpage) data on the population size and genetic makeup. The module also provides data summaries to scripted Heads-Up-Displays worn by each user's avatar.

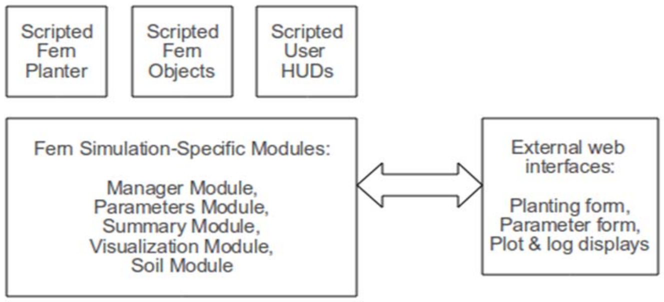

Scripted Fern Planter
Scripted Fern Objects
Scripted User HUDs
Fern Simulation-Specific Modules:
Manager Module, Parameters Module, Summary Module, Visualization Module, Soil Module
External web interfaces:
Planting form, Parameter form, Plot & log displays

Figure 3: Virtual Fernland integration with OpenSim

2.3 N-body Visualization

We developed the N-Body Game to demonstrate how virtual environments enable exploration of concepts at a scale that simply cannot be reproduced in the real world. The N-Body Game uses a relatively simple N-body simulation to demonstrate the effects of gravity. The game was designed for students attempting to create a solar system with planets in stable orbits around a star.

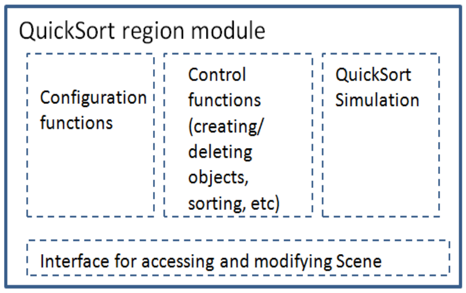
The N-Body game consists of two primary components. The first is a set of objects for the planets and sun. In addition to various effects for appearance, the objects provide an interface for adjusting their mass, position and velocity within the simulation. The second component is a fixed time-scale N-body simulator. The simulator is implemented as a region module that creates the planet objects, simulates the forces between the planet objects, and moves them in the scene based on the forces calculated.

To play the game, the student requests that the N-body simulator creates a set of planets. The student moves the planets to locations around the fixed position of the sun and assigns mass and velocity to each of the planets. Once that is done, the student requests that the simulation begins.

2.4 Sort Lab

The Sort Lab visualization environment was initially built to assist students to study classical sorting algorithms and gain insights by visualizing the sorting process in which objects are sorted and moved around within a sort space. It creates a set of basic “sort” behaviors: objects created in the space are sorted using the quick sort algorithm (Hoare 1961) and a given sort criteria (our initial implementation was based on the hue of each object). The sorting behavior is implemented in a simulation module. Figure 4(a) describes the implementation of QuickSort simulation module. It includes several groups of functions:

- Functions to configure parameters in the sort simulation, such as the number of objects to be created and sorted, to set up a sorting scenario.
- Functions to control the sort simulation, such as invoking the sorting behavior.
- Functions that implement the quick sort algorithm.
- Functions that access the Scene (data structure describing the space and objects in the space) and change its state, for instance, creating or deleting objects, updating object positions as driven by the sort algorithm, etc.


QuickSort region module
Configuration functions
Control functions (creating/ deleting objects, sorting, etc)
QuickSort Simulation
Interface for accessing and modifying Scene

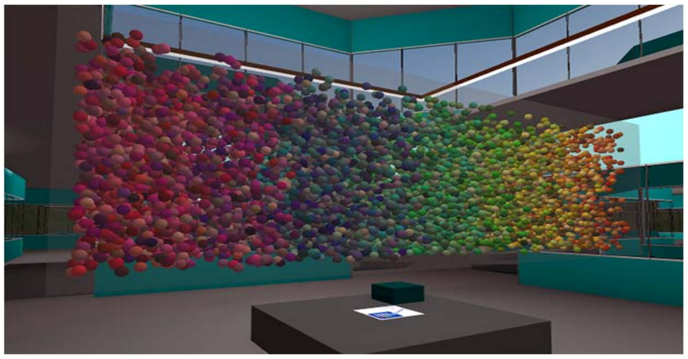


Figure 4: (a) QuickSort Simulation Module, (b) Visualization of clustering of 10,000 objects by applying QuickSort algorithm (sorted by hue).

With QuickSort simulation module integrated into OpenSim, scripts were written to invoke creation of objects and sorting the objects based on their hue values. For instance, Figure 4(b) shows one scenario where 10,000 colorful spheres were created and sorted.

3 ENABLING REUSABLE BEHAVIORS

Each application discussed in the previous section took us many days of work to develop the simulation modules and get behaviors right. Looking through these applications, it is easy to realize that each of them has generic behaviors that are useful to many other VE applications as well. For instance, the sorting behavior from Sort Lab is a common operation in data analysis applications. Also, the basic reproductive, competitive, and environmental interactions that drive simulations of fern genetics from Virtual Fernland are similar to the interactions at work in other organisms. If these generic behaviors can be reused, time and effort to build new VE applications could be dramatically reduced.

The problem, however, is that with the way that VE platforms and VE applications are developed today, it is very hard to extract generic behaviors and reuse them, due to lack of system support and services and lack of application design for reusability.

- The binding of behaviors and objects are usually hard-coded and application specific. (Binding refers to on which set of objects and which of their properties will a simulation module operate on.) For example, in Sort Lab, the set of objects to be sorted and the object property used as sorting criterion were programmed inside the QuickSort module.
- In developing an application specific simulation module, usually due to lack of design for reuse, there is no clear decomposition of the behaviors into generic and application specific functions.
- There is no infrastructure support in VE platforms for sharing and reusing behaviors. For instance, there is no services for behavior registration, discovery, configuration, etc.

As a result, when new VE applications are developed, they go through the same development process and it again requires a lot of time and effort to create their own simulation modules. For example, the developers of Virtual Fernland also created a Virtual Plant Community simulation (VPCsim 2012) to teach ecology. Like Virtual Fernland, individual plant objects in VPCsim interact with simulation modules that control the virtual habitat, provide a user interface, and report on simulation status. However, even though the two systems share a similar structure and requirements, very little of the existing Fernland code could be directly reused in VPCsim, leading to many hours of duplicated effort.

The strong need for behavior reuse in VE application development is quite obvious. The problem is how to enable it. In our work, we took an approach that is similar to the Web Services Architecture (W3C 2004). In such an architecture, there is behavior providers (modules that provide reusable behaviors), behavior requesters (modules or VE applications that reuse or compose behaviors available from the providers), and behavior broker services that assist requesters to discovery and invoke available behaviors. In this section, we first present an overview of the approach, then discuss how we implemented the approach based on OpenSim.

3.1 Behavior Abstraction

We call our approach for a behavior provider to generalize and expose reusable behaviors the "active content design pattern". Here, "active content" refers to the content in a VE application, as the content is a combination of static content and dynamic, interactive behaviors. The approach is illustrated in Figure 5.

- When a new simulation module is developed, its functions are decomposed to generic functions and application specific functions.
- The generic functions from the simulation module provide interfaces for interaction with other modules or scripts:
    - Control interface: A set of APIs for parameter or policy configuration and function invocation.
    - Data exchange interface: the data that the simulation module expects as input and the data it exports during simulation. The data is exchanged either synchronously (data returned when query is issued) or asynchronously (data returned when it becomes available).
    - Event interface: the events the simulation module subscribes to and the events it raises while in operation.

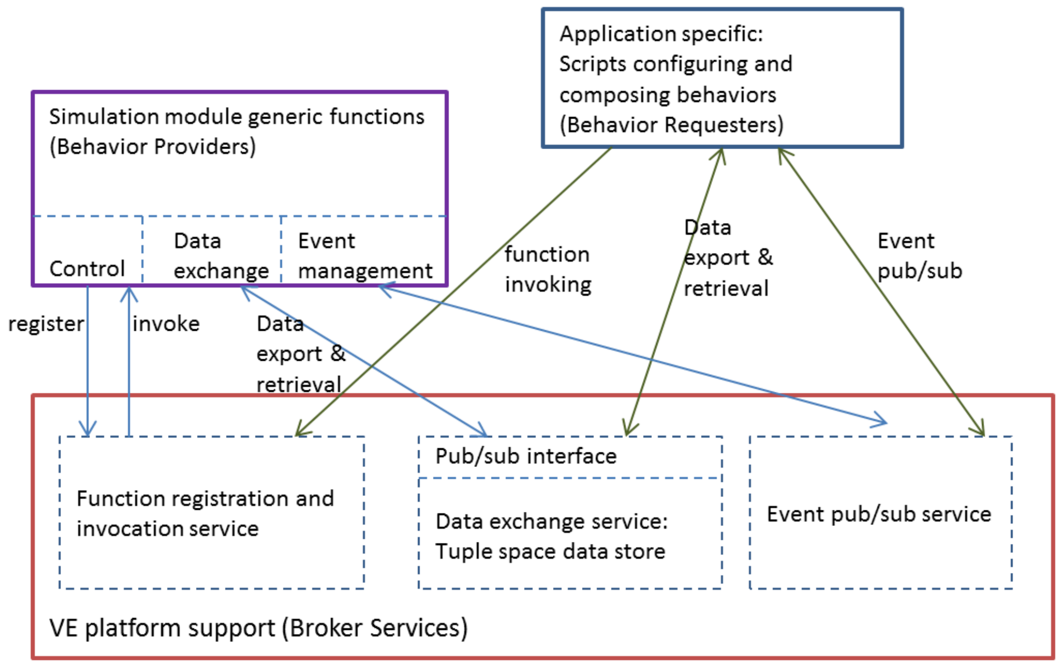


Figure 5: Illustration of "active content" design pattern

3.2 VE Platform Services

Behavior modules are simulation modules, and simulation in a VE platform is extensible (for instance, OpenSim is easily extensible with region modules). Hence a VE platform can be extended with addition behavior modules, where the extensions form a behavior repository. The VE platform then needs to provide broker services to make the interfaces provided by behavior providers accessible to behavior requesters:

- Services for the behavior providers to register and expose its control interface, and for behavior requesters to invoke the registered functions (without having to know who provides the functions).
- A data store with Publish/Subscribe (Baldoni et al. 2003) interfaces for behavior requesters and providers to exchange data. The data store provides a loose binding between simulation and data, so that the binding is no longer hard-coded in simulation modules. The data store supports both synchronous and asynchronous data exchange.
- An event Publish/Subscribe service.

To provide behavior broker services, we enhanced OpenSim with the ModInvoke (2012) family of functions. With ModInvoke support, at system initialization time, a behavior provider registers its generic functions. Then at run time, users can write scripts (behavior requesters) to invoke the registered functions to configure and control simulations and build desired application behaviors.

To support data exchange, we enhanced OpenSim with the JsonStore (2012) service to exchange structured data among simulation modules and scripts. JsonStore is a tuple space data store. It provides functions to create a data store and to write to and read from the store. Values of structured data are encoded as JSON (2012) string for storage and exchange. Data retrieval from a data store can either be synchronous or asynchronous, where in the latter case, JsonStore raises events when the requested data become available.

OpenSim provides a set of event management services for simulation modules and scripts to publish and subscribe to events. In ModInvoke and JsonStore services, we extended OpenSim's event management to enable simulation modules to raise events to communicate with other modules and scripts.

With the VE platform services in place, next we discuss how to re-architect the VE applications discussed in Section 2 to enable behavior reuse.

3.3 Re-architecting VE Applications

3.3.1 Reusable Sorting Behaviors

To enable the sorting behaviors developed in the QuickSort module to be reusable, we re-architected the QuickSort as follows.

First, we replaced the previously hard-coded comparison of hue values and hard-coded collection of objects (created by the QuickSort module solely for sorting purpose) by generic functions. Through the ModInvoke service, the following functions can be invoked through QuickSort's control interface:

- AddSortElement(object), for collecting objects for sorting.
- SortConfig(itemName, itemValue), for configuring sort behaviors, for instance, configuring the module to sort by property "size" instead of "hue".
- SortControl(command), for invoking, pausing, or stopping sorting on the collected objects.

Next, based on the JsonStore service, we extended the QuickSort module to export data produced during the sorting process, for instance, the minimum, medium, and maximum values when sorting is done. The QuickSort module also raises an event when the sorting is done.

With the exposed control, data, and event interfaces from QuickSort module, it is easy to reuse the sorting behaviors in a different VE application. All it needs to do is:

- Define a sort space. Any object moves into this space will trigger AddSortElement.
- Configure the sorting behavior (e.g. writing a script to do configuration) by invoking SortConfig
- Drop objects into the sort space and trigger sorting by calling SortControl.
- If the VE application wants to obtain data exported by QuickSort module, before sorting starts, it can create a data store through JsonStore service and pass the store ID to QuickSort module by invoking SortConfig function. It then just waits for events that indicate the data it subscribes to is ready.

Figure 6 shows a snapshot from a VE application that reuse the sorting behavior by dropping in objects with different sizes and configuring it to sort based on the size of each object.

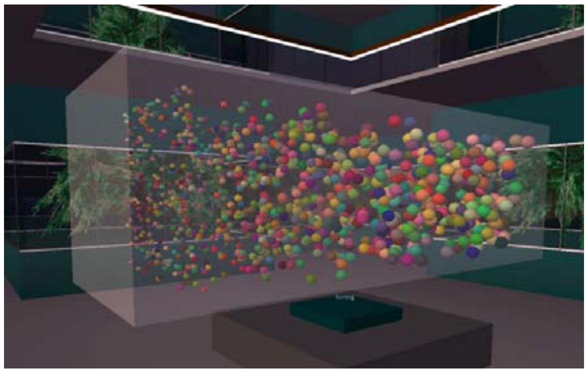


Figure 6: Reuse QuickSort behavior: sorting by size (larger to smaller, from right to left).

3.3.2 Reusable N-body Behaviors

While developing the N-Body Game we were, in parallel, working on ways to visualize large collections of documents. Cosine similarity is a metric frequently used to identify the similarity between documents. The basic behavior in the N-Body Game (simulating the effects of attraction between many planets) seemed like a good starting point. We replaced gravity by document similarity as the attractive force. The resulting application provided a very nice method for visualizing the clusters of similar documents (each document is represented by an object) within a collection (the attraction between similar documents creates clusters over time).

To generalize the N-Body Game into an behavior engine capable of simulating the effects of any attractive force on large numbers of objects required three changes. First, generalize the N-body simulation module with

- Multiple instances
- A suite of configurable attraction functions
- Dynamic time-scale
- Dynamically add and remove objects from the simulation module

Next, define the data interface for sharing the basic properties. Each of the attraction functions in the N-body simulation module use different data values. For example, the gravity attractor uses the mass of the objects to compute attractive force, but the document similarity attractor uses a vector of numbers that represent the occurrence of words in the document. The generality requires a loose, dynamic binding between the application and the N-body simulation module. The JsonStore module provides a means for attaching arbitrary data to an object.

Third, enable the N-body simulation module to provide feedback to the application. As we experimented with the use of the n-body simulation module in applications beyond the N-Body Game, it became apparent that we needed a mechanism for the simulator to provide feedback to the application. For example, it is common for simulations to generate very high velocities that would take objects well outside the boundaries of the simulation. To keep objects within the boundaries of the simulation, we "warp" space at the edges of the simulation so that objects slow down in the visualization. In addition, the n-body simulation module provides information about the warping of objects back to the application. We use this information in the document similarity application to remove documents that are so dissimilar from the rest of the collection that they are pushed out to the edges of the simulation.

Further, properties of the simulation itself like the simulation frequency are provided to the application. This is important to know when the complexity of the simulation begins to degrade the quality of the simulation (and frequently the quality of the visuals). Applications can reduce the number of objects in the simulation or adjust other properties of the n-body simulation module to ensure high quality user experiences.

3.3.3 Reusable Virtual Organism Behaviors

In this section, we discuss that following the active content design pattern, how Virtual Fernland can be re-architected to enable reuse.Each of the simulation modules used in Virtual Fernland could be made more generally applicable and published as reusable libraries. For example, the Parameters module reads a set of fern-specific genetic and life history parameters and provides those parameters to each new fern object so users can change simulation parameters before (or even during) the simulation. The module is currently hard-coded to handle a specific set of parameters in a specific order, stored in an unannotated text file. With the ModInvoke system now available in OpenSim, the module could make use of the data exchange interface to pass any set of parameters required by any virtual organism, rather than being limited to fern-specific parameters. New virtual organisms would request the appropriate parameters through the control interface and register to be notified by the event service when parameters are updated. Similar changes using the 'active content' design pattern could be made to the other Virtual Fernland modules, providing generic modules for summarizing and plotting data about virtual objects, modifying features of a virtual habitat, and visualizing genetics of virtual organisms across a landscape. These more generalized versions of the modules would have drastically reduced the time and effort required to create VPCsim and would allow easier experimentation with a variety of biological simulations.

However, even these changes are not enough to address the biggest limitation of the current implementation of Virtual Fernland: besides the five simulation modules, another big portion of the simulation is driven by the interactions of thousands of individually scripted fern objects. Using large numbers of scripted objects pushes the limits of the scripting engine and makes it difficult to modify plant behaviors or to create new types of virtual organisms. To fully convert Virtual Fernland into a set of reusable tools for creating virtual genetic simulations, it would need to be further re-architected. Rather than having the simulation driven by scripts embedded in the fern objects, those functions would need to be offloaded to a simulation module, which manages the interactions between individual plants and their environment and track their status (health, age, and reproductive readiness). Following the 'active content' design pattern, the system could then be re-architected (similarly to the QuickSort system) so we could:

- Define an evolution space, where any object we place in the space becomes a virtual organism.
- Configure the organism behavior with scripts to define the lifecycle, genetics, and interactions of each type of organism.
- Drop objects into the evolution space and allow them to begin reproducing and interacting with each other.

4 DISCUSSIONS AND FUTURE WORK

One important challenge that occurs when educators or other domain experts attempt to use VEs to build specialized simulations is that they may not know anything about writing software or scripts. For example, researchers writing Virtual Fernland had to learn C# before they were able to make progress implementing their simulation modules to OpenSim. Such a language is complex and may require significant effort before having a solid understanding of its principles and mechanisms.

An approach that can mitigate this overhead is to develop a collection of VE related domain languages. A domain language is a special purpose programming language with carefully chosen syntax and semantics for use in a specific domain. Popular examples of successful domain languages include CSS and SQL. Domain languages typically don't describe computation but instead declare configuration, structure or interconnection within a specific domain. This makes learning, reading, writing and maintaining significantly easier for user familiar with the domain.

We do not propose any specific VE configuration language here but note that such a technology has the potential to greatly reduce the time and effort required to assemble a desired simulation. If we are to reproduce the successes of HTML and CSS with regards to accessibility to novice users, we will need to develop some analogous technologies in the VE domain.

5 CONCLUSION

Enabling behavior reuse in 3D content creation is hard in today's systems, but not impossible. It needs a community effort. VE platform developers need to design and develop the platforms to provide necessary services. Core simulators in a VE platform needs to be extensible by incorporating with new simulation modules. VE application developers, when developing simulation modules to support new behaviors, need to architect the modules in such a way that the generic functions are decoupled from application specific functions and expose their interfaces for reuse. When more and more simulation modules are incorporated into VE platforms, the behaviors available for reuse become richer and richer. VE application developers, in turn, can spend much less time in developing simulation modules, but rather, just write scripts or use other tools to leverage and compose behaviors from existing behavior libraries.

In this paper we shared our experiences in VE application development and in enabling behavior reuse. We noticed that there exists some preliminary work in reusable virtual elements (Luengo and Soto 2006) and behavior reuse for intelligent robots (Duro et al. 2001). We wish the discussion in this paper could invoke more community effort to make the above vision come true.

ACKNOWLEDGMENTS

We thank Jerry Talton, Intel Labs, for his valuable feedbacks. We also thank Stephen L. Gasior, The University of New Orleans, and Eva Comaroski, the Wizardry and Steamworks group, for discussions on development of educational VE applications (both Gasior and Comaroski are also affiliated with Biological Interactive Objects for Science Education, BIO-SE (2012)).

REFERENCES

AWEDU. 2012. The Active Worlds Educational Universe (AWEDU). http://www.activeworlds.com/edu/index.asp. Accessed July 23, 2012.

Baldoni, R.; M. Contenti, and A. Virgillito. 2003. "The Evolution of Publish/Subscribe Communication Systems." Future Directions of Distributed Computing. Springer Verlag LNCS Vol. 2584.

BIO-SE. 2012. Biological Interactive Objects for Science Education. http://wiki.bio-se.info/wiki/Main_Page. Accessed July 23, 2012.

Chaudhuri, S, and Koltun, V. 2010. "Data-Driven Suggestions for Creativity Support in 3D Modeling", *ACM Transactions on Graphics* 29(6).

Campbell, T. Wang, S-K, Hsu, H-Y, Duffy, A.M., and Wolf, P.G. 2010. Learning with web tools, simulations, and other technologies in science classrooms. *Journal of Science Education and Technology* 19:505-511.

Daden Limited 2010. Virtual Worlds for Education and Training. http://www.daden.co.uk/downloads/Virtual%20Worlds%20for%20Training%20and%20Education%2001d2.pdf. Accessed July 23, 2012.

Duro, R. J., Becerra, J. A., and Santos, J. 2001. Behavior Reuse and Virtual Sensors in the Evolution of Complex Behavior Architectures. *Theory in Biosciences*,Volume 120, Numbers 3-4, 188-206.

EducationGrid. 2012. http://theeducationgrid.org/. Accessed July 23, 2012.

Google Warehouse. http://sketchup.google.com/3dwarehouse/. Accessed July 23, 2012.

Hirsch, T. 2010. Water Wars: Designing a Civic Game about Water Scarcity. In *Proceedings of ACM SIGCHI Designing Interactive Systems*.

Hoare, C. A. R. 1961. "Partition: Algorithm 63," "Quicksort: Algorithm 64," and "Find: Algorithm 65." Communications of the ACM 4(7), 321-322.

JSON. 2012. http://www.json.org/ Accessed July 23, 2012.

JsonStore Service. 2012. [JsonStore] http://opensimulator.org/wiki/OSSL_Script_Library/JsonStore. Accessed July 23, 2012.

Kallmann, M., and Thalmann, D. 1999. Direct 3D interaction with smart objects. In *Proceedings of the ACM symposium on Virtual reality software and technology* (VRST '99), 124-130.

Luengo F. and Iglesias A. 2004. Framework for Simulating the Human Behavior for Intelligent Virtual Agents. Part II: Behavioral System". In *Proc. of International Conference on Computational Science*, LNCS 3039, pp.237-244.

Luengo, F., and Soto, C. 2006. Reusable Virtual Elements for Virtual Environment Simulations. *International Journal of Computer Science and Network Security*, VOL.6 No.7A.

ModInvoke. 2012. http://opensimulator.org/wiki/OSSL_Script_Library/ModInvoke. Accessed July 23, 2012.

N-body Space. 2012. http://sciencesim.com/repository/dsg/nbody/

OpenSimulator. 2012. http://opensimulator.org/wiki/Main_Page

Perlin, K., Goldberg, A. 1996. Improv: A System for Scripting Interactive Actors in Virtual Worlds. In *Proc. of ACM Computer Graphics Annual Conference*, 205-216.

Peters, C., Dobbyn, S. , MacNamee, B. , O'Sullivan, C. 2003. Smart Objects for Attentive Agents. In *Proc. of 11th International Conference in Central Europe on Computer Graphics, Visualization and Interactive Digital Media* (WSCG’03).

QuickSort Visualization. 2012. http://sciencesim.com/wiki/doku.php/placestovisit
ScienceSim. 2012. http://sciencesim.com/. Accessed July 23, 2012.
Second Life. 2012. http://secondlife.com/.
Second Life Education Directory. 2012. http://wiki.secondlife.com/wiki/Second_Life_Education_Directory. Accessed July 23, 2012.
Second Life marketplace. 2012. https://marketplace.secondlife.com/. Accessed July 23, 2012.
SimTeach. 2011. Educational Locations in Second Life, http://www.simteach.com/wiki/index.php?title=Top_20_Educational_Locations_in_Second_Life. Accessed July 23, 2012.
VPGsim. 2012. Virtual Population Genetics. http://virtualbiology.usu.edu/vpgsim. Accessed July 23, 2012.
VPCsim. 2012. Virtual Plant Community Simulation. http://virtualbiology.usu.edu/vpcsim.
Virtual World Watch. 2009. http://virtualworldwatch.net/vww/wp-content/uploads/2009/12/Snapshot-7.pdf. Accessed July 23, 2012.
W3C. 2004. Web Services Architecture. W3C Working Group Note 11. http://www.w3.org/TR/ws-arch/. Accessed July 23, 2012.

AUTHOR BIOGRAPHIES

HUAIYU LIU is a research scientist in the Virtual Worlds Infrastructure team, Intel Labs. She holds a Ph.D. in Computer Sciences from the University of Texas at Austin. While at Intel, she had worked on multi-radio networks and energy efficient communications. Her current research is developing scalable infrastructure for multi-user virtual enviornments. Her email is <huaiyu.liu@intel.com>.

MIC BOWMAN is a principal engineer in Intel Labs and leads the Virtual World Infrastructure research project investigating systems architectures for scalable virtual environments. He received his BS from the University of Montana, and his MS and PhD in Computer Science from the University of Arizona. He joined Intel Architecture Lab in 1999. While at Intel, he developed personal information retrieval applications, context-based communication systems, and middleware services for mobile applications. In addition, he led the team that built and deployed the first version of PlanetLab, a global testbed for networking research. His email is <mic.bowman@intel.com>.

AARON M. DUFFY is a Ph.D student in the Department of Biology at Utah State University. His research interests include plant genomics, population genetics, ferns with atypical lifecycles, and teaching scientific concepts and science inquiry through interactive simulations. He has developed 3D simulations that are being used in 8th grade natural science and undergraduate genetics classrooms. His email address is <aaron.duffy@usu.edu>.

WARREN A. HUNT is a Researcher at Intel Labs. He received his undergraduate degrees in math and computer science from Carnegie Mellon University and his PhD from The University of Texas at Austin. His primary fields of study are algorithms, optimization, computer graphics. In addition, he has publications in compilers and artificial intelligence. His email address is <Warren.Hunt@intel.com>.