

A Framework for Prototyping Collaborative Virtual Environments

Clinton Jeffery, Akshay Dabholkar, Kosta Tachtvrenidis, Yosep Kim
{jeffery, adabholk, ktachtev, ykim}@cs.nmsu.edu

Department of Computer Science
New Mexico State

ABSTRACT:

Unicron is a platform for rapidly developing virtual environments that combine two popular forms of collaboration: a 3D collaborative virtual environment fostering meetings, appointments, whiteboard sessions and lectures, along with a 2D development environment including collaborative software design, text editing, and debugging tools. This paper presents novel aspects of the *Unicron* design and implementation.

Keywords: collaborative virtual environments

1 Introduction

Collaborative 3D virtual environments (CVEs) are highly successful in the area of entertainment, in games and social environments such as Everquest and There.com. This success suggests enormous potential in many other domains, especially those involving geographically-dispersed participants, such as large-scale software development efforts, or distance education.

In order to apply CVE's successfully to these other domains, several major obstacles must be overcome. This paper addresses two of those obstacles: the high cost of developing new CVE's which constitutes a barrier to entry, and the importance of bringing existing 2D collaborative tools along to get the "real" domain-specific work done. We have constructed a CVE platform called *Unicron* in order to explore the tools and methods needed to solve these problems.

2 Motivation

Unicron was initiated for the purpose of computer science distance education. Existing distance education tools such as WebCT lack domain-specific support, such as the need for an instructor to look at an individual student's debugging session and explain how to read what the student sees on their screen. Collaborative editors abound, but tools for collaborating over compiler diagnostics or debugger sessions are not common.

We turned to collaborative virtual environments because they have proven highly effective at bonding people together and providing effective substitutes for in-person social necessities such as chatting, making appointments, and holding meetings. Many users of CVEs work productively every day with people they

have never met face to face. The high cost of developing a CVE was seemingly prohibitive, so we developed a research project aimed at reducing this cost. Unicorn's feature set was adapted several times during the construction of our first "low cost" CVE by surprising actual "high cost" tasks encountered during development.

The initial goal for Unicorn was to produce a CVE that brings a Computer Science department experience to remote locations, including as many aspects as possible of the freshman-through-junior year experience. The virtual community is modelled after the appearance and actual dynamic content of the 1st floor of Science Hall at NMSU. Remote students gain access to the department's tools and expertise (such as CS software tools, seminars and meetings, lab assistants).

Access to CS computers and software tools can already be accomplished by existing 2D tools, and seminars and meetings can be held using teleconferencing rooms, so a 3D virtual community is in principle unnecessary, but in practice we feel it offers several advantages. Others' experience with Viras [15] supports the idea that a 3D space provides better social awareness than 2D places of comparable size and complexity. In addition, we believe 3D virtual spaces borrow successfully from users' expertise at remembering and navigating 3D real spaces to provide superior ease of learning and use.

Compared with Viras' use of an abstract and customizable archipelago, it seems unimaginative to base a 3D space on a real academic department. While the Unicorn framework can be used to build abstract spaces, using a real space also has advantages. Using the Science Hall model, transfer students already know the physical environment by the time they arrive at NMSU. Also, faculty and students who become familiar with the real Science Hall, such as prospective students from around the state who visit for a five week summer session, will already know their way around the 3D model when they see it. Together, this helps students and faculty correlate distance education experiences with on-campus CS education experiences, and simplifies their orientation into the NMSU CS department.

A virtual community both complements and contrasts with traditional video teleconferencing, and with web-based tools such as WebCT. While it is impossible to set up cameras from every angle in every room, it is very possible to supplement the traditional distance-learning cameras and digital capture devices in key locations with a 3D model of the department. Special-purpose Internet-based collaborative versions of CS laboratory software, such as text editors and compilers, are part of this virtual CS community. Integration of existing distance-education technology into this 3D virtual community is also important.

To the user, Unicorn looks and feels like an interactive 3D videogame application on their PC, within which they can move around, examine signs and notices, go to class or to an instructor's office hours, and go to a virtual lab to work on assignments. Within the virtual lab, the view often shifts to a higher-resolution 2D view of the tools, but the collaborative nature of the environment (chatting, asking the TA or instructor questions, etc.) remains consistent. Text and voice chat modes can be set to local proximity (seen heard automatically

by those nearby in the 3D environment) or to select individuals by name (closer to a phone or private chat style of communication).

Perhaps the “killer application” targeted by the virtual environment is to see who is available to provide (remote) technical assistance, and then obtain that assistance (say, with debugging a program) in a 2D view of code. The convenience factor of being able to perform that task in real time despite the students or instructor being far away or simply away from the office is tremendous. This replaces long sequences of e-mails in which student and instructor struggle to get to the bottom of the student’s problem which the student may not understand well enough to describe precisely, or which the student may not have the writing skill to describe in less than a long and painful message.

3 Design

Unicon consists of several components that form the middle layer in a rapid prototyping framework for collaborative virtual environments:

- in the bottom layer, a very high level language was augmented with ultra-simple 3D, network, and audio APIs.
- the Unicon class library provides infrastructure for the networked 3D environment, e.g. the behavior of doors, whiteboards, and avatars; the Unicon server enables user interaction and shares state between clients; Unicon also provides simple “builder” tools to generate a virtual environment from inputs such as 2D floor plan data and extract textures from digital camera photos.
- the application layer for any particular CVE generated using Unicon consists of: a 3D model produced semiautomatically using Unicon builder tools; a set of domain collaboration tools; and a set of user accounts, created on the Unicon server.

Figure 1 shows several aspects of the Unicon client-server architecture.

Unicon was developed as a prototyping framework rather than a particular CVE because the same properties that are needed in order to easily modify and experiment with the integration of the CVE and the collaborative 2D tools also make it relatively easy to construct new CVE’s. Objects in the virtual environment are built using a simple 3D API in a very high-level general purpose applications prototyping language. A very high-level language has been employed by other systems for rapid prototyping virtual reality environments, such as Alice [14]. Our emphasis on collaboration is different from that effort’s emphasis on graphics and animation.

3.1 Language Layer

Some of the research effort in developing Unicon was conducted in terms extending a very high level language to support collaborative virtual environments.

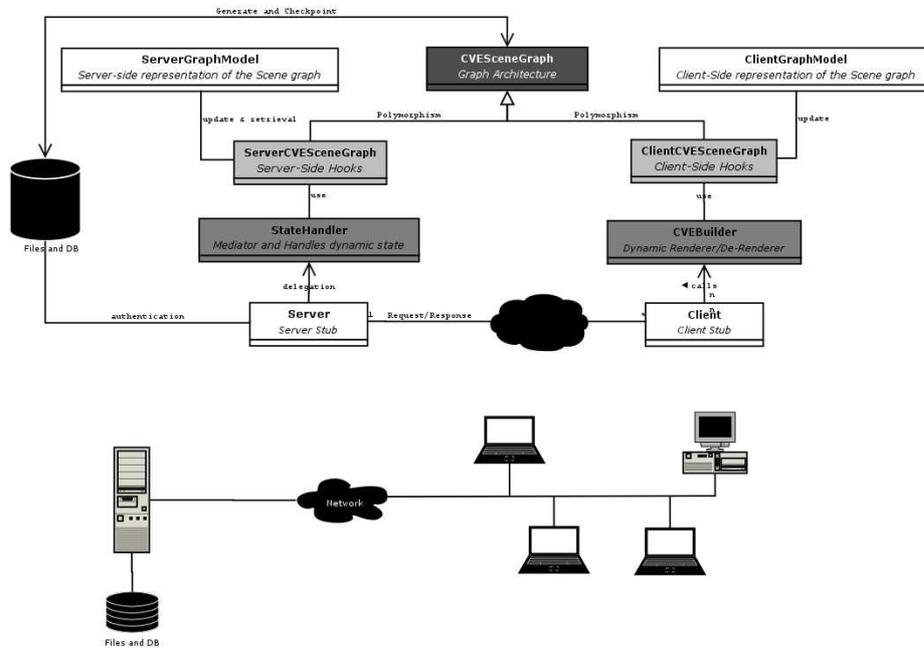


Fig. 1. Unicon CVE Architecture

Simple high level interfaces to graphics, networking, and audio were added or extended to the Unicon programming language in order to build Unicon [10]. Unicon is described at <http://unicon.org>.

Language extension is selected instead of writing libraries or modules when a feature is general, or when it has the need or the potential to interact with other features in the language virtual machine or runtime system. Once a given hardware capability is sufficiently ubiquitous, adding control structures and built-in syntax to access it is not just a notational convenience, but an enabling technology.

3.2 Class Library Layer

The Unicon class library primarily serves to model virtual environment functionality, independent of its views and controls. The research contribution here is not to invent new paradigms, but to explore the simplest implementation techniques that provide sufficient performance on current hardware. This design bias, combined with the very high level language used, make it easy to add new features and conduct experiments. This is appropriate since CVE technology is still in its infancy and only gradually moving towards adolescence: many novel domain features need to be explored in the next decade.

The virtual environment model is a graph data structure, in which “room” objects are nodes, and doors and openings are edges with interesting properties.

It includes relatively static data that is replicated to all clients at install- or load-time and usually not modified during the middle of a user session, as well as dynamic data that can be modified by users. The model is controlled and maintained on a master server, which sends each client that small subset of the dynamic data that is needed for the user's current projection. The server sends dynamic information only to those users' who are in proximity or otherwise need be aware of that information. Where possible, multiple dynamic state changes are bundled together into a single network packet. These approaches reduce the server's network load. As with most CVE's the server remains the main limit to scalability despite these techniques.

We developed our own portable multiplatform representation of the model, after trying many state of the art technologies that we expected to use, such as SQL, XML, VRML, and X3D. We were surprised not to be using VRML or X3D, but they proved much too low-level and cumbersome for our purpose, due to their generality. Avoiding SQL or simpler database technologies such as Access or GDBM allows our server to be brought up trivially in new environments. This aids in testing, makes creating new Unicron-based worlds trivial, and simplifies tasks such as migrating server state to a new architecture, as we found when moving from a 32- to 64-bit platform. The static and dynamic state is all represented using text files in subdirectories in the file system. So far we have not observed a performance problem due to this choice, although one does have to be sensible about how often the server saves frequent dynamic state changes such as avatar moves.

Examples of static and dynamic model data are presented below; first a Room object, then a Door out of that room, then the dynamic state for that door. These files are simple enough to edit or debug easily by hand, and easily generated by a level editor. The coordinate system is a simple cartesian system with units of 1 meter with the origin at the northwest corner of our building.

```
# static properties of a room
Room {
name SH 167
x 29.2
y 0
z 0.2
w 6
h 3.05
l 3.7
floor Rect { texture floor2.gif }
obstacles [
    Box { # window sill
        Rect {coords [29.2,0,.22, 29.2,1,.22, 35.2,1,.22, 35.2,0,.22]}
        Rect {coords [29.2,1,.22, 29.2,1,.2, 35.2,1,.2, 35.2,1,.22]}
    }
]
decorations [
```

```

    Rect { # window
        texture wall2.gif
        coords [29.2,1,.22, 29.2,3.2,.22, 35.2,3.2,.22, 35.2,1,.22]
    }
    Rect { # whiteboard
        texture whiteboard.gif
        coords [29.3,1,2.5, 29.3,2.5,2.5, 29.3,2.5,.4, 29.3,1,.4]
    }
    ]
}

# static properties of a door
Door {
x 33
y 0
z 3.9
height 2.3
plane 3
rooms [SH 167, cooridoor 167]
}

# dynamic state of a door
link {
name link1
openness 1.0
delta 0
direction 1
}

```

The intent of this human readable format was to be concise and human-maintainable as part of the larger goal of reducing the effort to develop CVE's. In practice graphical tools are usually used to generate models, but human-readable output is still useful. The entire Science Hall first floor static model data is around 30KB, making it feasible to send out new models in-session or at program startup as part of the patcher.

3.3 Application Layer

The Unicorn client, called NSH, renders 3D environment sessions in one of many tabbed buffers, which can also include collaborative editing, compilation, execution, debugging, and UML design sessions. The 3D CVE sessions support lecture, lab, and office hour academic functions, serving largely to coordinate more detailed collaborative activities using the other tools. A subgraph of the whole client projection of the CVE is rendered, based on a simple proximity heuristic; as a user moves into a new room, new nodes in the graph become visible, other

nodes are deemed invisible and derendered, and the server changes the set of dynamic data for which the client will be informed.

The initial proximity heuristic (the null heuristic) rendered all static data and all available dynamic data at each step. This worked surprisingly well on reasonable OpenGL implementations, but did not scale well on the typical Windows clients we expect to find in users' homes. A second naive heuristic rendered all nodes within a distance k in the Room graph, unless a closed door prevented visibility. This works sufficiently, but can be improved with very little effort by varying k as the graph is traversed, based on the direction the user is facing and the edge properties being traversed. For example, people cannot see around corners, although they may hear around them. The rendering traversals can be precomputed during initialization when the static data is loaded. Other high performance algorithms such as traversals of binary space partition trees are not implemented in Unicron yet because graphics fidelity has been a lower priority than networking and collaboration aspects of the CVE framework thusfar.

The graphics fidelity of the Unicron virtual environment is on the low side. High graphical fidelity requires an order of magnitude more effort in terms of texture art as well as programming, reducing our ability to add experimental domain-specific features, without changing functionality much from our distance education perspective. This tradeoff is acceptable for Unicron because the CVE serves largely for the coordination of collaborative 2D tool sessions, and because it needs to run on stock PC's without special hardware, rather than a high-fidelity virtual reality system with head mounted display and data gloves or other VR peripherals.

Unicron's network protocol and architecture is simple, but features several hybrid aspects. Unicron's architecture decouples simulation from rendering, as is the case for Alice [14]. The simulation is performed on a shared server cluster which also handles voice transmission and recording processes.

Client messages are classified as vital or transient and sent by TCP or UDP accordingly. Private communications (both chat and audio) are sent peer-to-peer if possible, but a central server is used for forwarding between peers that cannot communicate directly due to firewalls, as well as maintaining shared state of many forms. Secondary servers similar to the booster boxes advocated by IBM Zurich are deployed at sites where substantial bandwidth sharing is enabled by their use [2]. In our case this includes the campuses of our educational partners, a consortium of two-year colleges.

3.4 Avatars

Unicron's avatars are graphically simple, but customizable communication tools (Figure 2). Avatars require about as many graphical primitives as would a LEGO(tm) figure, and feature the ability to point, an identifying label, and a visual indication when audio or text chatting is performed.

Creating an avatar is a simple exercise. Using the GUI, users can customize their clothing using colors or textures, and provide a GIF or JPG image to

present their face in an ordinary rectangle or texturemapped within an egg-shaped head, which is more recognizable from the side or rear.



Fig. 2. Avatars

4 Implementation

Unicon is written in the Unicon programming language [11], a descendant of Icon [7]. Unicon features a set of 3D graphics facilities based on OpenGL that are profoundly simpler and easier to learn and use than OpenGL or Java3D [12].

In addition to those features already described, Unicon's first incarnation as a CS education tool features:

- Audio stream access between selected locations, allowing hands-free verbal communication.
- Dynamic classroom projector and electronic whiteboard output are integrated into the 3D environment
- Passive capture and recording (via microphones and high-resolution digital cameras) of classroom lectures. The audio and conventional black/whiteboard content are captured and integrated into the 3D environment in near real-time.
- A collaborative software development environment, developed to include special purpose, multi-user shared-view versions of compilers (for C, C++, and Java), programmer's editors, debuggers, and software design (UML) diagramming tools.

4.1 An Immersive 3D Graphical Environment

Unicon's graphical environment is cartoon-like (similar to popular games), rather than aiming at being photorealistic as for many CVEs such as *Le Deuxieme Monde*. We believe there will be cultural side-benefits compared with traditional distance learning: for example, students will not have to engage in eye contact, and will have less fear of embarrassment while asking questions. Figure 3 shows example scenes students might see in this environment.

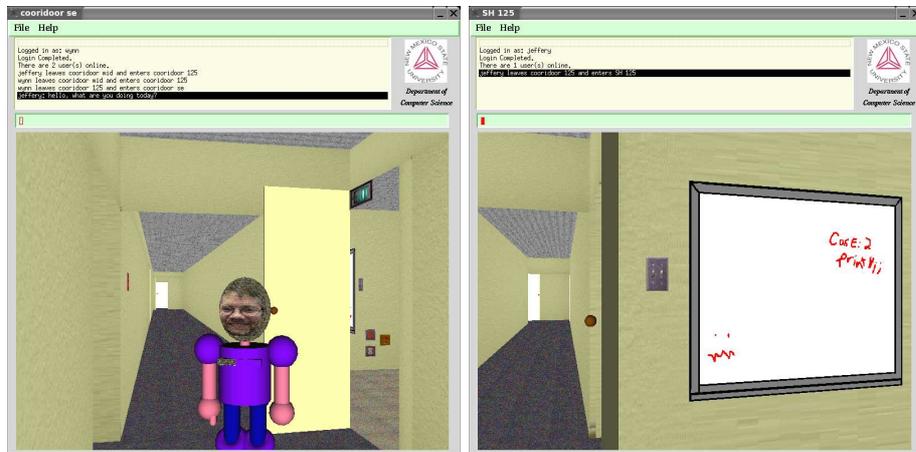


Fig. 3. Virtual Academia with Integrated Whiteboards, Chat and Voice

The virtual whiteboard is a fairly standard collaborative tool; NMSU's electronic classroom features custom local software that feeds the whiteboard content up in Adobe SVG (an XML) format via HTTP, where it is available to the CVE along with other SVG plugin-equipped browsers.

Besides whiteboards and their pens which naturally relate to interactions with 2D drawings, other interactive objects within this environment include avatars (other users), books (on-line documentation), and virtual workstations. Virtual workstations serve to integrate and interconnect with 2D collaborative applications; for example, to join someone's 2D collaborative editor session, walk over to the workstation their avatar is at, and click on their machine. Visible indications of which machines are occupied, and which printers have long wait-queues are other example uses of virtual objects, enabling remote users to select a computer on which to remote login or select a printer to receive a print job.

While open-source image manipulation and 3D tools for artists are abundant, free tools dedicated to simply and rapidly generating 3D virtual buildings from floor plans are scarce. After exploring several alternatives including VRML- and X3D-capable 3D tools, we failed to find what we needed and developed a crude "level wizard" that generates our environment directly from a dialog specifying

the default ceiling, walls, and floors, followed by a semiautomatic room layout extractor. The layout extractor allows the developer to specify the rooms in the model directly from an ordinary floor plan image file, which may have been scanned-in, drawn by hand, or captured from another tool as a screen shot.

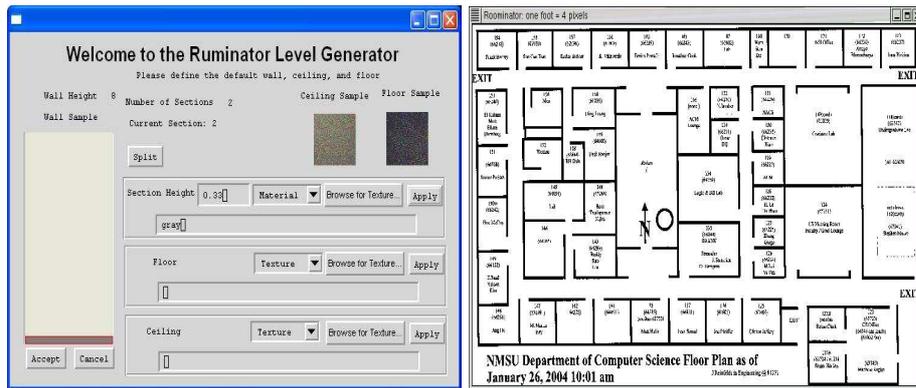


Fig. 4. Layout Editing using Ordinary Floor Plans

4.2 Collaborative Software Development Environment

Computer Science faculty working with students at a distance need more than a bulletin board or chat facility, they need to see the contents of the software development tools that are running on the student's screen, to advise the student about compiler errors and runtime errors they may be experiencing, and show them how to perform problem solving tasks such as debugging. Collaboration software such as Centra and NetMeeting allow shared views of documents, but not within the context of software development. The remote-control genre of commercial software (such as PC Anywhere or RealVNC) provides shared views of applications, but at a considerable cost in bandwidth. For Computer Science instruction, a cross-platform solution (for Linux, UNIX, and Windows) is often needed; specifically, one that is open source, allowing it to be integrated into the virtual community environment; RealVNC might be a good candidate. In New Mexico most of the population is rural and many students still use dialup, where a minimum-bandwidth solution is needed.

Unicon's collaborative development environment prototype called Pegasus is shown in Figure 5. The prototype allows users to edit text, watch each other, and chat. We are in the process of adding hands-free audio communication, and collaborative views of other areas of software development such as UML design diagrams, compilation, execution, and debugging output. Pegasus will also be integrated into the 3D environment in the preceding section. Instructors will be able to walk around a virtual lab, talk with students, and look at their virtual

screens, zooming in to the high resolution collaborative environment view when questions require on-screen details.

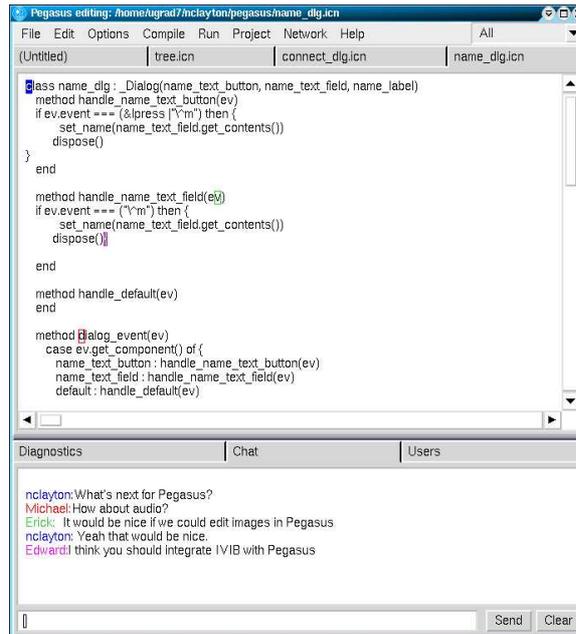


Fig. 5. A collaborative development environment

5 Innovations

The primary innovations thusfar in Unicron are not in the user-visible application experience, nor in the CVE architecture, but in the runtime system support, class library and tools that reduce the programming effort necessary to build a CVE. This support includes both graphics, networking, audio and the integration of these subsystems.

CVE's have a classic problem for event-driven systems, namely, that they have multiple event streams. In Unicron, this consists of a TCP chat connection, a UDP connection for most 3D environment updates, and the window system input handling. Performance requirements dictate that these connections be checked continuously; but polling is too expensive. On UNIX/X11 the `select()` is supposed to be good for this, but on MS Windows `select()` takes only network connections, and on both platforms we found it necessary to write our own runtime system support to avoid polling.

6 Related Work

Unicron is related to both 3D collaborative virtual environments and 2D collaborative applications. Both Unicron's 3D environment and 2D collaborative tools owe much to fundamental internet communication technologies such as chat, mailing lists, and especially MUDs and MOOs featuring different rooms for different topics or tasks [20]. Sony's EverQuest (www.everquest.com) popularized multi-user virtual environments that add an immersive first-person 3D graphical world onto a collaboration tool that still largely consists of MUD-like text chat. EverQuest handles thousands of users and works adequately over 28.8K modems. Digital Space Traveler, available from www.digitalspace.com, demonstrates the potential for voice and 3D sound in virtual environments. Adobe Atmosphere, Apple QuickTime VR, There.com and Meet3D are other commercial efforts. These packages and game engines are attractive, but without source code it is impossible to customize them to integrate CS programming tools such as compilers and editors.

There are many 3D research CVE's, such as MASSIVE (-1/-2/-3)/HIVE [17][6], and DIVE [4]. These systems have often emphasized the avatars at the expense of the work people are trying to collaborate on. Instead of taking a VR-centric view where everything is done in the VR, or the opposite view of VR as just a weak collaboration tool whose main purpose is to augment reality by providing awareness of other users [18], Unicron takes the position of *augmented virtuality*: the VR models real-world places and activities, and 2D tools are integrated into the VR where a person would be turning to a computer or other device (such as a whiteboard) to do some work in the non-VR version of things. This allows for relatively seamless transitions. As much as possible, "integration" of 2D tools means bundling their functionality directly into the CVE, rather than switching to a separate window. For example, the collaborative views of text are provided by tabs in the CVE Window, to minimize the user's cognitive context switching costs, and provide continuity in the overall environment. Text and voice chat controls are uninterrupted by switching between 3D view and code view. See Figure 6. Voice chat buttons are upper left (disable, room-mode, and phone-mode). Text chat in upper right provides context as tabs shift between 3D and 2D main area views. Use of the navigation bar in lower left varies with the current mode.

Some of CVE's and other VR systems are especially interesting because they are open source or otherwise publically available, such as University of Manchester's Maverik [8] (aig.cs.man.ac.uk/maverik/), Planeshift (www.planeshift.it), VR Juggler (www.vrjuggler.org), or Alice [14]. One related CS education project that is using Alice is Saint Joseph University's JABRWOC project, which uses virtual reality as an instructional domain, as opposed to Unicron's goal of creating a virtual community for software development collaborations such as CS distance education. Another related education project is Viras [15], a CVE for education built using Active Worlds (www.activeworlds.com). While it is unclear that Active Worlds is customizable to the extent needed for domain-specific education CVE tasks, it is a tremendous resource for generic CVE construction.

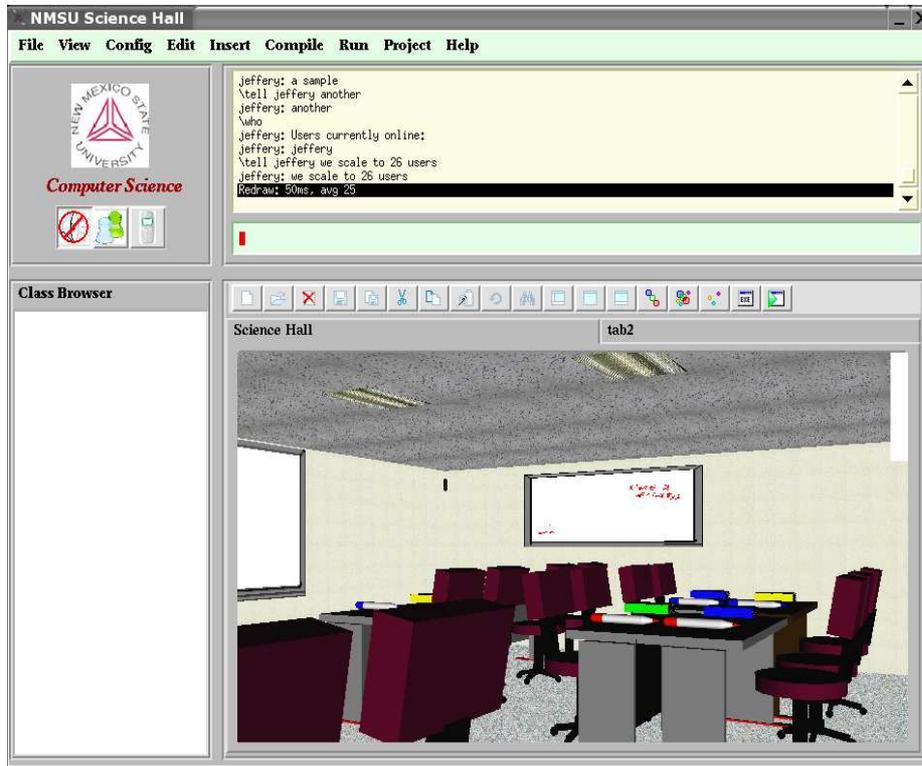


Fig. 6. Integrated view of the IDE and CVE

The term “collaborative programming environment” can refer generically to any tool that assists programmers to work in teams, such as the (asynchronous) document revision features found in Microsoft Word, or the set of software tools provided by Source Forge (www.sourceforge.net). The collaborative software design and programming environment we envision for the virtual community is more closely related to fully interactive systems from the field of computer supported cooperative work (CSCW). Similar systems from that domain include Microsoft’s NetMeeting, NetEdit [21], RECIPE [19], and others. Among 2D collaborative tools, CROCODILE is especially related, as it provides a virtual academic enterprise rendered as hypertext 2D graphs [13]. The CSCL community is also actively exploring uses of 3D environments as educational tools [9]. We are not aware of any systems that integrate a collaborative programming environment into an immersive 3D virtual environment as is the case for Unicron.

7 Experience Gained

Quantitative evaluation of Unicron is ongoing and will be the subject of another paper. Anecdotal experiences learned fall into categories, including (a) exper-

riences building the NSH CVE, (b) discoveries about hardware and software limitations of current mainstream platforms that pertain to CVE applications, and (c) experiences in the integration of domain-specific instructional tools.

The claimed ease of use of the programming language Unicon used to build the Unicon framework has been validated by developer experiences, with caveats. A group of average Master's students were able to learn Unicon and then build, with direction from faculty, both the CVE framework/tools and the working CVE described in this paper. However, the high-level language does not substitute or solve issues of code quality or software engineering, all it does is pose fewer lines of code for instructors or peers to have to code review and debug. Naive student code that works for a few users is different from expert code that runs well for many users.

Many experiences running the CVE have proven useful and occasionally surprising. As expected, performance has been an issue, especially in scaling to larger numbers of users. However, the relatively slow speed of the very high-level language has not been the issue it was expected to be, because CVEs are heavily I/O bound. Because of this, both the client and the server had to be rewritten the same way: in both cases a simple event-driven I/O multiplexor had to be changed to a batch/step processor that minimized the number of interactions with the operating system by processing all input prior to sending all output for each step. This change alone scaled our system from handling 4 users to handling 26 users on midrange Pentium 4 hardware.

We expected the server written in Unicon to be the bottleneck, but found that the server written in a very high level language has not been a problem; server load has seldom exceeded 10% and is generally far less. The CVE built using this framework remains I/O bound, but the network is not the problem any more: current scalability is limited by clients' graphics rendering code, dominated by VM runtime system OpenGL code written in C. Scaling to hundreds of simultaneous users in the same virtual classroom will involve reducing the graphics rendering cost they impose on each other, for example by abstracting groups of far-away users, and disabling their ability to "distract" each other with individual movements. We found the hard way that wireless internet technologies perform substantially worse than wired technologies for such collaborative tools, corroborating reports from gamers. This is unfortunate since NMSU's electronic classroom is equipped with wall-to-wall wireless laptop machines.

Our experiences integrating domain-specific tools are modest so far. Direct integration of the text editor widget and supporting network code for the collaborative IDE was fairly straightforward. Integrating command-line and textual tools is abstracted by means of pseudo-terminals running inside editor widgets. However, many of the tools we want to integrate involve graphical user interfaces and require either reimplementations as a collaborative tool or else they must be open enough to be extended enough to talk to our framework using a mechanism such as [16] [5].

8 Conclusions and Future Work

On the positive side, in rapid prototyping a collaborative virtual environment with Unicon, we were amazed at the ease with which the smoothly animated first-person view of the environment was developed, initially in 300 lines or so. The multi-user interaction capabilities came together similarly rapidly and simply; our first n-user chat client and server were less than 200 lines of code.

Forseeably but inconveniently, to go from a floor plan and digital camera images to a working prototype virtual environment of an entire building or more requires more sophisticated tools for texture management automation. Texture management (reducing the size of textures, making them tile nicely, etc.) is a seemingly straightforward process that should be automatable by many methods, but most tutorials on the subject advocate texture manipulation by hand using a tool such as the Gimp, or PhotoShop [1]. The Gimp is great, but building a virtual world requires a lot of textures. We implemented some simple automated tools based on techniques described by Bourke [3] and are working on further refinements.

The Unicon platform is highly suitable for quickly prototyping CVEs that work well for a modest number of simultaneous users, such as the class sizes of 5-30 that we anticipate using it for. We are gradually improving it to scale to larger numbers of users, but we have not applied the kind of hardware or software engineering resources that go into commercial games that handle hundreds or thousands of users.

Our future efforts include: splitting master server responsibilities across multiple machines to improve scalability; refinement and packaging of open source world-building tools; implementing well-known higher performance graphics algorithms to improve rendering; collaboration support for additional CVE domains and other academic disciplines; and performance tuning and fault resistance, especially to network fluctuations.

9 Acknowledgments

Joe Pfeiffer developed our electronic classroom's whiteboard application, with a Linux device driver written by Mike Wilder. Numerous students assisted with early Unicon development; Wynn Winkler and Nolan Clayton developed noteworthy prototypes. This work was supported in part by the New Mexico Alliance for Minority Participation, and by NSF grants EIA-0220590, EIA-9810732, and DUE-0402572.

References

1. anonymous. *Making Tearable & Seamless Textures*. HighPoly3D.com, 2005.
2. D. Bauer, S. Rooney, and P. Scotton. Network infrastructure for massively distributed games. In *Proceedings of the 1st Workshop on Network and System Support for Games, NetGames 2002*, pages 36–43. ACM, April 2002.

3. P. Bourke. *Tiling Textures on the Plane*. Swinburne Centre for Astrophysics and Supercomputing, Australia, 1992.
4. E. Frecon. Dive: Communication architecture and programming model. *IEEE Communications Magazine*, 42(4):Xpp, April 2004.
5. D. Garlan and E. Ilias. Low-cost, adaptable tool integration policies for integrated environments. In *Proceedings of the fourth ACM SIGSOFT symposium on Software development environments*, pages 1–10. ACM, 1990.
6. C. Greenhalgh and D. Snowdon. Hive distribution api. *Technical Report*, 0:5pp, November 1997.
7. R. E. Griswold and M. T. Griswold. *The Icon Programming Language*. Peer to Peer Communications, San Jose CA, 1997.
8. R. J. Hubbard, X. Dongbo, and S. Gibson. Maverik — the manchester virtual environment interface kernel. In *Proceedings of the 3rd Eurographics Workshop on Virtual Environments*, pages x–x+y, February 1996.
9. R. Hmlinen, P. Hkkinen, S. Jrvel, and T. Manninen. Computer-supported collaboration in a scripted 3-d game environment. In *Proceedings of CSCL 2005*, Taipei, Taiwan, 2005.
10. C. Jeffery, A. Dabholkar, K. Tachtevrenidis, and Y. Kim. Programming language support for collaborative virtual environments. In *Proceedings of 18th International Conference on Computer Animation and Social Agents*. CGS, 2005.
11. C. Jeffery, S. Mohamed, R. Pereda, and R. Parlett. *Programming with Unicon*. Unicon Project, unicon.sf.net, 2004.
12. N. Martinez and C. L. Jeffery. Unicon 3D Graphics User’s Guide and Reference Manual. *Unicon Technical Report*, 9(a):28pp, July 2003.
13. Y. Miao. *Design and Implementation of a Collaborative Virtual Problem-Based Learning Environment*. M.S. Thesis, Technical University of Darmstadt, 2000.
14. R. Pausch and colleagues. Alice: A rapid prototyping system for 3d graphics. *IEEE Computer Graphics and Applications*, 15(3):8–11, May 1995.
15. E. Prasolova-Frland and M. Divitini. Collaborative virtual environments for supporting learning communities: an experience of use. In *Proceedings of the 2003 International ACM SIGGROUP Conference on Supporting Group Work*, pages 58–67. ACM, 2003.
16. S. P. Reiss. Connecting tools using message passing in the field environment. *IEEE Software*, 7(4):57–66, July 1990.
17. D. Roberts and P. Sharkey. Maximising concurrency and scalability in a consistent, causal, distributed virtual reality system, whilst minimising the effect of network delays. In *Proceedings of the IEEE WETICE’97*, pages x–x+y. IEEE, June 1997.
18. M. Robinson, S. Pekkola, J. Korhonen, S. Hujala, T. Toivonen, and M.-J. O. Saari-nen. Extending the limits of collaborative virtual environments. *Collaborative Virtual Environments: Digital Places and Spaces for Interaction*, page XX pp, 2001.
19. H. Shen and C. Sun. Recipe: a prototype for internet-based real-time collaborative programming. In *Proceedings of the Second International Workshop on Collaborative Editing Systems*, 2000.
20. J. Smith. *Basic Information about MUDs and MUDDing (MUD FAQ)*. Oklahoma State University Dept. of Math, Oklahoma, OK, 1999.
21. A. Zafer. *NetEdit: A Collaborative Editor*. M.S. Thesis, Virginia Polytechnic Institute, 2001.