PRIVACY AND INTEGRITY IN COLLABORATIVE, VIRTUAL ENVIRONMENTS

A Dissertation

Submitted to the Graduate School
of the University of Notre Dame
in Partial Fulfillment of the Requirements
for the Degree of

Doctor of Philosophy

by

Timothy E. Wright

Greg Madey, Director

Graduate Program in Computer Science and Engineering
Notre Dame, Indiana
April 2009
PRIVACY AND INTEGRITY IN
COLLABORATIVE, VIRTUAL ENVIRONMENTS

Abstract

by

Timothy E. Wright

Collaborative, virtual environments (CVEs) leverage desktop virtual reality technologies to enable multiple, simultaneous participants to interact and work together. As immersive, multimedia-oriented systems, CVEs typically represent participants as avatars and operate in a variety of areas including, but not limited to: entertainment, commerce, education, and research. Success of endeavors within these areas requires privacy and integrity for both participant activities and the objects upon which participants act. Security mechanisms, however, are often limited in scope and functionality, or lacking entirely. In response, this dissertation investigates and develops controls to protect the privacy and integrity of activities and objects within CVEs. We begin with a survey of 16 technologies that may be used to build a CVE. We then examine two of these technologies further, Croquet and Project Wonderland, giving consideration to their respective architectures, strengths, and weaknesses. Continuing with Wonderland, we design and implement a full discretionary access control solution called WonderDAC. Finally, we carry out a realistic demonstration of Wonderland/WonderDAC with a small number of human participants.
To my wife and children: the music in my life.


CONTENTS

FIGURES ........................................................................... viii

TABLES ............................................................................. xi

CHAPTER 1: INTRODUCTION ........................................... 1

1.1 Collaborative, Virtual Environments ......................... 1
1.2 The Need for Privacy and Integrity ........................... 3
1.3 Croquet and Project Wonderland ......................... 4
1.4 Third Party Interest ........................................ 4
1.5 Contributions and Organization .......................... 5

CHAPTER 2: A SURVEY OF DESKTOP VIRTUAL REALITY TECH-
NOLOGIES FOR BUILDING COLLABORATIVE, VIRTUAL ENVI-
RONMENTS ................................................................. 6

2.1 Abstract .......................................................... 6
2.2 Introduction ................................................. 6
2.3 Related Work .............................................. 10
2.4 Overview of Nomenclature and Approach ............ 13

2.5 APIs ..................................................................... 24

2.5.1 FreeVR .................................................... 24
2.5.2 Java3D ................................................... 25
2.5.3 OGRE ..................................................... 27
2.5.4 OpenGL .................................................. 27
2.5.5 OpenSceneGraph ....................................... 29
2.5.6 OpenSG ................................................ 29
2.5.7 OpenVRML ........................................... 30
2.5.8 X3D Tool Kit .......................................... 31
2.5.9 Xj3D .................................................... 32

2.6 Frameworks .................................................. 33

2.6.1 Crystal Space ......................................... 33
2.6.2 Delta3D ................................................. 34
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.6 Scalability and Benchmarking</td>
<td>193</td>
</tr>
<tr>
<td>6.7 Other Wonderland/WonderDAC Considerations</td>
<td>198</td>
</tr>
<tr>
<td>6.8 Conclusion and Future Work</td>
<td>201</td>
</tr>
<tr>
<td>CHAPTER 7: CONCLUSION</td>
<td>203</td>
</tr>
<tr>
<td>7.1 Summary and Contributions</td>
<td>203</td>
</tr>
<tr>
<td>7.1.1 CVE Technology Survey</td>
<td>203</td>
</tr>
<tr>
<td>7.1.2 Comparison of Peer-to-Peer and Client-Server CVE Architectures</td>
<td>204</td>
</tr>
<tr>
<td>7.1.3 Practical Implementation and Demonstration of CVE Discretionary Access Controls</td>
<td>204</td>
</tr>
<tr>
<td>7.2 Limitations and Future Research</td>
<td>205</td>
</tr>
<tr>
<td>APPENDIX A: VEOC SOURCE CODE EXCERPTS</td>
<td>207</td>
</tr>
<tr>
<td>A.1 XML Handler</td>
<td>207</td>
</tr>
<tr>
<td>APPENDIX B: PROJECT WONDERLAND TEST WORLD XML FILES</td>
<td>215</td>
</tr>
<tr>
<td>B.1 TestRoomA-WLC.xml</td>
<td>215</td>
</tr>
<tr>
<td>B.2 Whiteboard-WLC.xml</td>
<td>237</td>
</tr>
<tr>
<td>B.3 TestRoomB-WLC.xml</td>
<td>238</td>
</tr>
<tr>
<td>B.4 TestRoomC-WLC.xml</td>
<td>239</td>
</tr>
<tr>
<td>B.5 TestRoomD-WLC.xml</td>
<td>240</td>
</tr>
<tr>
<td>APPENDIX C: WONDERLAND SOURCE CODE EXCERPTS</td>
<td>244</td>
</tr>
<tr>
<td>C.1 CellAccessControl</td>
<td>244</td>
</tr>
<tr>
<td>C.2 ClientIdentityManager</td>
<td>249</td>
</tr>
<tr>
<td>C.3 ClientIdentityService</td>
<td>254</td>
</tr>
<tr>
<td>C.4 DAC</td>
<td>255</td>
</tr>
<tr>
<td>C.5 UserGLO</td>
<td>257</td>
</tr>
<tr>
<td>C.6 LDAPAuth</td>
<td>263</td>
</tr>
<tr>
<td>C.7 WonderlandIdentity</td>
<td>269</td>
</tr>
<tr>
<td>C.8 CellGLO</td>
<td>271</td>
</tr>
<tr>
<td>C.9 StationaryCellGLO</td>
<td>282</td>
</tr>
<tr>
<td>C.10 BasicCellGLOHelper</td>
<td>287</td>
</tr>
<tr>
<td>C.11 BasicCellGLOSetup</td>
<td>289</td>
</tr>
<tr>
<td>C.12 WhiteboardCellGLO</td>
<td>294</td>
</tr>
<tr>
<td>APPENDIX D: WONDERDAC DEMONSTRATION SURVEY, GANTT CHART, AND HSIRB PROTOCOL REVIEW</td>
<td>299</td>
</tr>
<tr>
<td>D.1 Demonstration Questionnaire</td>
<td>299</td>
</tr>
<tr>
<td>D.1.1 WonderDAC Basics</td>
<td>299</td>
</tr>
<tr>
<td>D.1.2 Spatial Restrictions</td>
<td>300</td>
</tr>
</tbody>
</table>
### FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>vEOC Architecture under Croquet.</td>
<td>63</td>
</tr>
<tr>
<td>3.2</td>
<td>Joining a Croquet Island.</td>
<td>65</td>
</tr>
<tr>
<td>3.3</td>
<td>Island Replication.</td>
<td>65</td>
</tr>
<tr>
<td>3.4</td>
<td>Maintaining Synchronicity between Island Replicas.</td>
<td>66</td>
</tr>
<tr>
<td>3.5</td>
<td>Message Passing Details in Croquet.</td>
<td>67</td>
</tr>
<tr>
<td>3.6</td>
<td>Avatars within the prototype vEOC virtual environment.</td>
<td>69</td>
</tr>
<tr>
<td>3.7</td>
<td>Participants in the vEOC using text chat.</td>
<td>70</td>
</tr>
<tr>
<td>3.8</td>
<td>The Croquet text whiteboard, showing highlighted text.</td>
<td>71</td>
</tr>
<tr>
<td>3.9</td>
<td>The vEOC simulation script DTD.</td>
<td>73</td>
</tr>
<tr>
<td>3.10</td>
<td>A sample vEOC XML test/simulation script.</td>
<td>74</td>
</tr>
<tr>
<td>3.11</td>
<td>Example message display.</td>
<td>75</td>
</tr>
<tr>
<td>3.12</td>
<td>Looking up at an MPEG display.</td>
<td>77</td>
</tr>
<tr>
<td>4.1</td>
<td>Layers of security in a CVE.</td>
<td>83</td>
</tr>
<tr>
<td>4.2</td>
<td>XML file hierarchy for a test Wonderland world.</td>
<td>91</td>
</tr>
<tr>
<td>4.3</td>
<td>Server state change: communications directly between a client and the server.</td>
<td>93</td>
</tr>
<tr>
<td>4.4</td>
<td>Client-only state change: communications among Wonderland clients (by way of the server).</td>
<td>94</td>
</tr>
<tr>
<td>4.5</td>
<td>The avatar twright looks on at testRoomD and testRoomC.</td>
<td>106</td>
</tr>
<tr>
<td>4.6</td>
<td>The avatar bench-40 looks on at testRoomC, but cannot see testRoomD.</td>
<td>107</td>
</tr>
<tr>
<td>4.7</td>
<td>The avatar bench-40 looks in the direction of testRoomD as twright climbs the stairs.</td>
<td>108</td>
</tr>
<tr>
<td>4.8</td>
<td>The avatar twright looks back at the avatar bench-40 from testRoomD.</td>
<td>109</td>
</tr>
</tbody>
</table>
4.9 The bench-40 participant draws on a local copy of the whiteboard shared with twright.

4.10 Because bench-40 has no alter permission, the twright participant cannot see bench-40's whiteboard updates.

4.11 The twright participant has alter permission and draws on the whiteboard shared with bench-40.

4.12 The final state of the whiteboard from bench-40's perspective.

5.1 Layers of security in a CVE.

5.2 A typical Wonderland architecture configuration.

5.3 The Test World WFS Hierarchy.

5.4 An excerpt from a sample XML cell file.

5.5 A typical Wonderland architecture configuration with secure network communications.

5.6 Because their participants do not have interact permission for the loft, the JUser and Bench-415 avatars cannot see the loft stairs or the T Wright avatar.

5.7 Now granted the appropriate permission, the loft and T Wright avatar appear for the JUser and Bench-415 participants.

5.8 Configuring a Wonderland whiteboard to prevent updates from everyone but the owner.

5.9 Using a Wonderland whiteboard; only T Wright, the owner, is able to alter the whiteboard's contents.

5.10 Creating an ad hoc group to manage a Wonderland cone-of-silence.

5.11 Assigning an ad hoc group to a cone-of-silence.

5.12 Having a private conversation inside a cone-of-silence.

5.13 Muted avatars being addressed by a speaking avatar.

5.14 Configuring an avatar cloak.

5.15 Deploying an avatar cloak.

5.16 The Wonderland client's Tools menu, including the newly added Avatar Cloak and Ad Hoc Group options.

5.17 The Avatar Cloak dialog box.

5.18 The Ad Hoc Group dialog box.

5.19 The WonderDAC configuration dialog box.
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1</td>
<td>A bird’s eye view of the virtual environment used in the demonstration. The team room (with office cubes, common area, and movie screening room) is at the top, while the demo room is underneath in the middle; Loft1 is to the left of the demo room and Loft2 is to the right.</td>
</tr>
<tr>
<td>6.2</td>
<td>A student participates in the Wonderland/WonderDAC demonstration.</td>
</tr>
<tr>
<td>6.3</td>
<td>Participants begin the demonstration in the team room, after logging into Wonderland.</td>
</tr>
<tr>
<td>6.4</td>
<td>TestUser1 looks out from their loft to where TeamB’s loft (Loft2) should be. Because this participant does not have access to Loft2, only a wall in the demo room below is visible.</td>
</tr>
<tr>
<td>6.5</td>
<td>TeamA engages in research—the Firefox web browser is executing in the left window, while OpenOffice Impress is the the right.</td>
</tr>
<tr>
<td>6.6</td>
<td>TeamB engages in research.</td>
</tr>
<tr>
<td>6.7</td>
<td>TeamA presents their research results in the team room.</td>
</tr>
<tr>
<td>6.8</td>
<td>TeamB gives their presentation.</td>
</tr>
<tr>
<td>6.9</td>
<td>The test monitor uses WonderDAC to open up access control for the demo room and its contents.</td>
</tr>
<tr>
<td>6.10</td>
<td>Average participant ratings for the Demonstration Follow-up Survey.</td>
</tr>
<tr>
<td>6.11</td>
<td>A screen snapshot of Top during the high-point of Wonderland activity.</td>
</tr>
</tbody>
</table>

D.1 The WonderDAC demonstration Gantt chart.  
D.2 The Notre Dame HSIRB Protocol Review Form.
TABLES

2.1 SUMMARY OF ACTIVE, DESKTOP-VR TECHNOLOGIES . . 16
2.2 RISK TABLE FOR ACTIVE, DESKTOP-VR TECHNOLOGIES 47
4.1 SUMMARY OF ACCESS CONTROL USE CASES ............. 99
5.1 WONDERDAC PERMISSIONS .................................. 133
5.2 SUMMARY OF ACCESS CONTROL USE CASES .......... 159
6.1 SCHEDULE OF TEST GOALS ................................. 186
6.2 RATING SCALES ................................................. 189
6.3 DEMONSTRATION SURVEY QUESTIONS .............. 190
This dissertation presents research on the problem of ensuring privacy and integrity within collaborative, virtual environments (CVEs). In our examination of the issues at stake, we consider synchronous and atomic participant activities (integrity), as well as the implementation and use of discretionary access controls (privacy and integrity). The bulk of our work, however, focusses on the latter.

In this chapter, we provide a brief overview of CVEs and their growing importance, and discuss the motivation for building privacy and integrity controls. We also introduce the Croquet and Project Wonderland CVEs, which are examined more thoroughly in later chapters.

1.1 Collaborative, Virtual Environments

For our purposes, we use the term CVE to denote a system that:

1. Represents a world with three dimensional computer graphics
2. Is utilized by multiple, simultaneous participants who interact through avatars
3. Forms a virtual reality, accessible by workstation-class computers

Prototypical CVEs include systems that may be used for game playing (e.g., the Quake III engine), commerce and socializing (e.g., Second Life, There, Active-
worlds), or educational and research collaboration (e.g., Croquet, Project Wonderland). While most CVEs may be adapted to any of the aforementioned endeavors, particular systems tend to have strengths and weaknesses that guide their application. Also, some CVEs are built on what we call platforms: complete, underlying systems that already include the characteristics enumerated at the start of this section. However, related technologies can enable a developer to construct a CVE based on various application programming interfaces (APIs) and/or frameworks. Here, we take API to mean a library that enables a developer to access hardware and software resources, while we view a framework as a collection of APIs along with support tools. There are, of course, development risk tradeoffs when building a CVE on a platform, framework, or APIs. These are explored in more detail in Chapter 2, where we survey contemporary CVE technology.

The importance of CVEs as a means of participant interaction is well-chronicled in the literature and noted in chapters 2, 3, and 4. Other measures exist, however. For example, CVEs have become a recent topic for the National Academy of Engineering in one of their Grand Challenges for Engineering: “Enhance virtual reality [100].” Similarly, the Media Grid, an “international collaboration of universities, colleges, research institutes, consortia and companies [57],” has devised the Immersive Education Initiative, which works to encourage the development of CVE technology with regard to “virtual reality and game-based learning and training systems [56].” Also, the New Media Consortium (NMC), an international group of “learning-focused” organizations, has engaged with Sun Microsystems to “develop a range of standards-based, portable open-source educational spaces, content, and objects... [104].” Through the Open Virtual Worlds Project, NMC and Sun plan to extend and leverage Sun’s Project Wonderland. Finally, the
popularity of some commercial CVEs appears to indicate more than faint interest among participants. Second Life, for example, boasts of having roughly 13 million resident accounts [83], Activeworlds claims about two million [3], and There has between 500,000 and one million [85, 94]. While there is some dispute about how accurately these commercial CVE companies are reporting their population statistics [85], the combined total number of registered participant accounts is certainly in the millions.

1.2 The Need for Privacy and Integrity

With a significant number of interested end-users and a diverse range of application areas, the motivation to develop security controls for CVEs is well-founded. Information security typically operates along three axes: confidentiality, integrity, and availability [109]. For the purposes of this research, we choose to address the first two legs of the information security tripod.

Confidentiality, or privacy, denotes restricting information access to authorized parties, while integrity means ensuring that information is not accidentally or maliciously changed. Within the context of CVEs, these concepts can be used to manage a variety issues, including, but not limited to: enabling a consistent collaborative and interactive experience for participants (integrity), controlling into which spaces a participant can see or move their avatar (privacy), controlling what media (i.e., images, videos, sounds) a participant can browse (privacy), and limiting what virtual objects and spaces a participant is allowed to change (integrity). In the absence of these measures it will be difficult, if not impossible, to realize the full potential of activities such as commerce, socializing, education, or research since there would be little to prevent malicious or accidental activities.
1.3 Croquet and Project Wonderland

Our examination of privacy and integrity issues takes place via two CVEs: Croquet and Project Wonderland. The former is a peer-to-peer system written in Squeak, a derivative of the Smalltalk 80 language, while the latter is a client-server system written in Java. Both are complete CVE platforms offering similar capabilities, although some important differences do exist. In the case of Croquet, its peer-to-peer architecture is leveraged to provide true synchronization and atomicity for participant activities. This means the actions of Croquet users always take place in the order in which they occur and are guaranteed to happen for observing participants or not at all. In other words, Croquet offers a way to ensure the integrity of a participant’s experience; we explore this in Chapter 3 where we build a virtual emergency operations center prototype. By comparison, Project Wonderland’s strength lies in its client-server architecture, which provides an authoritative place (i.e., the Wonderland server) to securely implement access controls. Putting such controls into a peer-to-peer CVE like Croquet would entail significant risk, since all participants would have equal access to the resources and information comprising the virtual environment. In Chapter 4, we extend Project Wonderland with the very basics of a discretionary access control system called WonderDAC. In Chapter 5, we evolve and refine WonderDAC into a fully functional solution, which is then used in Chapter 6 for a realistic, live demonstration.

1.4 Third Party Interest

The bulk of our research is directed at the Project Wonderland CVE. For reasons outlined more completely in Chapter 4, Wonderland is well-suited to our work. Moreover, Sun Microsystems, the proprietor of Project Wonderland, has
helped our efforts by providing a source code branch dedicated to WonderDAC and calling attention to our research [126][149]. Sun plans to integrate WonderDAC into Wonderland’s source trunk beginning with the forthcoming 0.5 release in 2009 [28].

1.5 Contributions and Organization

There are three primary research contributions made by this dissertation:

- A CVE technology survey and development risk assessment (Chapter 2)
- A comparison and exploration of peer-to-peer and client-server CVE architectures as they relate to privacy and integrity (chapters 3 and 4)
- A practical implementation and demonstration of CVE discretionary access controls (chapters 5 and 6)

Chapter 2 offers a point of departure with a survey of 16 current technologies that may be used to build CVEs. The survey wraps up with a qualitative assessment to determine and rate the development risks associated with each technology. Chapter 3 examines Croquet and its ability to serve as a simulation and training environment in which all activities are synchronized and atomic. Chapter 4 introduces Project Wonderland and makes use of the Wonderland server to build the WonderDAC prototype. Chapter 5 continues our discussion and exploration of WonderDAC in further detail and offers a more fully realized implementation. Chapter 6 puts WonderDAC to the test in a practical demonstration with real participants.

We conclude the dissertation in Chapter 7 where we summarize our research and outline future avenues of work and exploration.
CHAPTER 2

A SURVEY OF DESKTOP VIRTUAL REALITY TECHNOLOGIES FOR BUILDING COLLABORATIVE, VIRTUAL ENVIRONMENTS *

2.1 Abstract

What viable technologies exist to enable the development of so-called desktop virtual reality (desktop-VR) applications? Specifically, which of these are active and capable of helping us to engineer a collaborative, virtual environment (CVE)? A review of the literature and numerous project websites indicates an array of both overlapping and disparate approaches to this problem. In this chapter, we review and perform a risk assessment of 16 prominent desktop-VR technologies (some building-blocks, some entire platforms) in an effort to determine the most efficacious tool or tools for constructing a CVE.

2.2 Introduction

The benefit of virtual reality (VR) has been understood for some time. An array of training uses for VR are cited by [90], industrial and Internet visualization techniques are respectively demonstrated by [87] and [16], [48] discusses the

* Material from this chapter has been accepted by and is pending publication in the International Journal of Virtual Reality.
effective use of VR and other interactive media in teaching and learning, [150] employs VR for speech therapy, and [31] reviews a decade of VR technologies used to help those with disabilities. This chorus is enhanced by further examples of VR’s utility in artistic, cultural heritage, and scientific visualization deployments [61, 106], as well as the ongoing possibilities for experimentation and research offered by virtual worlds [9]. Despite its many successes and potential, VR remains a technology that has not seamlessly integrated with the average user’s computing interface. For example, regarding the compatibility of different VR systems with one and other, in [106] the authors point out that

in terms of VR applications we are at the Tower of Babel stage. Differences in tracking systems, interface devices, operating systems, VR libraries, graphics libraries, VR authoring systems, and even versions of the same software mean that, unless you have built your VR system with specific applications in mind, not many (or no) existing VR applications will run on it.

This issue is compounded by the fact that different levels of immersion (a participant’s sense of residing within a virtual environment) often require different computing resources. Whereas desktop virtual reality (desktop-VR) typically uses nothing more than a keyboard, mouse, and monitor, a Cave Automated Virtual Environment (CAVE) might include several display walls, video projectors, a haptic input device (e.g., a “wand” to provide touch capabilities), and multidimensional sound. The computing platforms to drive these systems also differ: desktop-VR requires a workstation-class computer, mainstream OS, and VR libraries, while a CAVE often runs on a multi-node cluster of servers with specialized VR libraries and drivers. At first, this may seem reasonable: different levels of immersion require different hardware and software. However, the same problems are being solved by both the desktop-VR and CAVE systems, with specific issues
including the management and display of a three dimensional environment and
user interactions within that environment. Even when we remove the level of
immersion as a factor (i.e., ignore the differences between desktop-VR and CAVE
interfaces), the system designer still faces an array of VR software options—some
overlapping, some different—all geared to accomplish similar tasks.

Regardless of immersion hardware requirements, the incompatibilities and du-
plications of effort among VR software solutions may hinder the acceptance of VR
and its integration as a common computing interface. In an effort to grapple with
this problem, we narrow our focus just to the area of collaborative, virtual envi-
rornents (CVEs) accessible through desktop-VR, and explore 16 software tech-
nologies relevant to building CVEs. For our purposes, then, a CVE is taken to be
a multiuser, 3D graphical environment accessible from, and potentially hosted on,
workstation equipment. Although highly immersive environments, such as CAVEs
and head-mounted display (HMD) systems, offer a more realistic user experience
[144], we restrict our consideration to desktop-VR for the following reasons:

- As desktops have become less expensive and more powerful (especially with
  regard to graphics adapters), viewing and interacting with high quality VR
  environments is more easily managed than in the past [79]

- The use of complex VR worlds for entertainment, commerce, and educational
  purposes is becoming more popular and effective for desktop users [17, 38].
  Typical, modern workstations are able to execute both client and server
  software, though scalability issues may apply

- Giving attention to the area of desktop-VR may help speed up the overall
  integration of VR into the average user’s digital media interface

In this survey, we will work toward identifying the strengths and limitations
of the desktop-VR technologies under review, thereby highlighting the best op-
tion or options for building CVEs. To help assess each technology, we propose
that, for any CVE to function, basic capabilities must be present in the areas of graphics rendering and networking. Secondarily, audio characteristics are desirable. Graphics rendering should include the display of 2D and 3D images, ranging from simple geometries to complex, texture mapped surfaces. Networking should at least provide the ability to remotely access VR content (e.g., images, sounds, scripts); preferably, it should also include features for sharing a virtual environment among multiple, simultaneous users. Last, audio should permit the playing of spatial sound; that is, sound with location, direction, and intensity. It should be observed that not all of the technologies in our survey include capabilities we consider to be primary (i.e., graphics and networking), or primary and secondary (i.e., graphics, networking, and audio). Less equipped software is also considered if it is commonly used as a CVE building-block.\(^1\)

Finally, some VR technologies include out-of-the-box features that are convenient for building a CVE (e.g., avatar support, multi-user capabilities, virtual collaborative tools [such as whiteboards], asset management,\(^2\) an art path,\(^3\) and rigid-body/particle physics). Having these extra features could mean significantly less work for the CVE developer. Therefore, two additional characteristics that we believe to be most significant, avatar support and multi-user capabilities, are taken into account later during a risk assessment of the 16 technologies under investigation.

---

\(^1\)It is noteworthy that all of the software we survey, from the least to most equipped, is graphics oriented. Text-based multi-user environments (e.g., Multi-user Dungeon [MUD] and MUD Object Oriented [MOO] systems) notwithstanding, our definition of desktop-VR requires a 3D graphical display. Thus, one can build a desktop-VR environment without networking or audio—although, absent networking, there would be no collaborative ability; one cannot build a desktop-VR environment without graphics.

\(^2\)For our purposes, this term means tracking the inventory and disposition of objects comprising a virtual world.

\(^3\)This refers to the tools and compatibility available for integrating images, textures, meshes, sounds, etc. into a virtual world.
review.

The remainder of this chapter is structured as follows: the next section summarizes related works; Section 2.4 outlines the nomenclature and approach we use in our review of VR software technologies; sections 2.5, 2.6, and 2.7 include assessments of the technologies (divided into three categories), Section 2.8 provides a qualitative risk assessment of the VR technologies, and Section 2.9 ends the chapter with concluding remarks.

2.3 Related Work

Various desktop-VR technology surveys exist within the literature, a few of which compare tools that can be used to construct a CVE. One such effort, made by [62], outlines a framework for comparing five Web-based visualization technologies, including Scaled Vector Graphics (SVG), Dynamic Hypertext Markup Language (DHTML), Virtual Reality Modeling Language 2 (VRML 2), Extensible 3D (X3D), and Java3D. This framework consists of four high-level categories (technical capabilities, interactivity, support, and application specific) used to measure the strengths and weaknesses of the technologies in question. Although somewhat subjective in places, the framework’s metrics do a reasonable job of capturing the utility and functionality of each technology. Ultimately, the authors select X3D as the best option for displaying 3D visualizations in an experimental project they carry out.

In a survey covering similar territory, [79] offers descriptions of what is termed emerging web graphics standards and technology. Consideration is given to several major areas, including 2D and 3D graphics standards (e.g., SVG and X3D, respectively), approaches to client synchronization and coherency in CVEs, virtual
humans represented on the Web (e.g., as highly-defined “talking heads” or fully rendered human bodies through the Humanoid Animation standard [H-Anim]), and alternative approaches to traditional, projective-geometry-based web graphics. The authors do not offer any significant conclusions in their discussion, except to comment on areas of future research important to the technologies under review. Other work in the same vein, though geared toward highly immersive technologies such as CAVEs, includes an outline by [144] of the potential and difficulties entailed by using VR for scientific visualization, and an entry level review of VR terminology and technology by [106].

In another survey, [20] compares and contrasts different free and commercial implementations of the X3D standard. Specific technologies include the X3D browsers Flux Player, FreeWRL, OpenWorlds Horizon, and Xj3D, and commercial CVEs from Blaxxun, Bitmanagement Software, and Octaga. The authors focus their discussion on how suitable these technologies are for constructing an X3D-compliant networked virtual environment (NVE). Only the contending CVEs are substantively compared with one and other according to ten simple criteria: X3D support, type of architecture, H-Anim support, streaming video support, voice-over-IP support, extensibility, scripting support, presence of X3D authoring tools, supported operating systems (OSs), and cost. The authors point out the risks of developing with commercial software (i.e., cost and lack of options should the vendor decide to change their product), and offer these as reasons for leveraging open source technology instead. Noting that the Xj3D Java libraries offer a “flexible and cross-platform Java architecture,” they use Java and Xj3D in the construction of an X3D browser and the retrofit of an existing, but limited, research CVE.

---

4The term networked virtual environment should be viewed as synonymous with collaborative, virtual environment.
X3D is the focus of [7], too, wherein the standard’s structure and composition are reviewed. Specifically, the authors are interested in applying desktop-VR to the endeavor of software visualization; that is, using a VR context to represent the relationships, statistics, and source code text of software systems. They conclude that the X3D standard offers the means to support software visualization, although they find text rendering (for displays of source code) to be an area of weakness.

Detailed consideration is also given to X3D’s capabilities by [121], where the authors discuss the development of an application to display user-selectable configurations of a given product in high-quality 3D. Several X3D browsers are compared by the authors who note that the lack of reflection mapping and shadowing in the X3D standard may be overcome through appropriate extensions to the standard.

In [58] a broader net is cast by the authors who compare 42 collaborative visualization systems. While some of the reviewed technologies are CVEs, many comprise middle-ware and tangentially related applications. The authors divide the systems into five basic application areas and then attempt to compare them across five categories (number of users, access control, communication architecture, transmitted data, and user synchronization). The primary conclusion reached is that virtual environments (i.e., CVEs) and online games offer the most capability in terms of scaling.

Some desktop-VR technology surveys are more conceptually oriented, choosing to review relevant precepts and foundations as opposed to specific tools or standards. For example, with a stated objective to “provide an awareness of Virtual Reality with respect to simulation,” [11] offers a concise overview of VR basics, including: a definition of VR; brief, background discussions about VR found in the media and as an effective means of simulation; physical mechanisms for interaction
and control within virtual worlds; and a thorough discussion of VR in manufacturing (a combination referred to as “virtual manufacturing”) along with several real-world examples. Similarly, in [105] the authors survey “various [CVE] technology factors and demonstrate their impact on closely-coupled collaboration.” These factors include: task design; immersion, field of view, and navigation; and manipulation technique, workflow, social human communication, user interface, distribution, and task performance. Ultimately, the authors classify these into technology factors, human factors, and application factors. They further note that each grouping must be given attention since many of the contained elements have interrelationships.

While all of these works help to build a picture of desktop-VR technology, little guidance is offered about which technologies are most strategic. We contend, however, that with regard to the engineering of CVEs, there are specific, well suited technologies that developers should strongly consider. Moreover, standardizing on such tools may help deal with the incompatibilities and duplications of effort among some VR systems, thereby leading to better integration with the common digital media experience.

2.4 Overview of Nomenclature and Approach

Our stated goal is to assess 16 desktop-VR technologies applicable to the construction of CVEs. As a result of this survey, we intend to note the most efficacious tool or tools. The following three broad categories offer a convenient way to group and compare the technologies of interest. It is not the case that one category is necessarily better than another: advantages and disadvantages exist for technologies in all three categories.
• Application Programming Interface (API): one or more libraries with documented methods that abstract lower-level resources (e.g., a graphics or audio API)

• Framework: a collection of APIs that provides functionality across multiple domains (e.g., graphics, audio, and networking) and is extended by support tools

• Platform:⁵ a specialized and complete environment within which a software system may be built, tested, and executed; the platform must be deployed anywhere software based upon it is installed

Within each of these categories, only active technologies (i.e., maintained roughly within the past year) that are free and open to developers are considered. Thus, we exclude proprietary options such as the Adobe Flash and Shockwave platforms,⁶ the Octaga suite of tools, the Bitmanagement Software SDK, the Torque Game Engine, etc. We restrict ourselves to active, open source technologies as a way to reasonably limit scope and emphasize software in which the end-users are the primary stakeholders. Also, our intention is not to carry out an exhaustive review of open source VR software: such an undertaking is impractical at best. Instead, we select 16 noteworthy, relevant technologies, any of which can play a significant role in the development of a CVE, and provide simple, concise descriptions of each.

⁵Below, note that we do not include Java3D under the “platform” category even though the Java virtual machine and runtime libraries must be installed anywhere Java3D software is executed. Although Java offers the necessary resources (e.g., networking and audio) to complement Java3D’s graphics rendering and construct a CVE, by itself Java3D is designed to be general purpose in nature. Thus, we believe it does not align well with the platform designation used herein.

⁶Adobe claims that its Flash and Shockwave players have been installed on a majority of all Internet-enabled PCs [5]. The Flash player handles content created by Adobe’s Flash CS3 Professional (2D animations, web advertisements, web interfaces), while the Shockwave player views content created by Adobe’s Macromedia Director (multi-user games, 3D content and product simulations). There is a collection of open source alternatives to CS3 and Director [10], but the Flash and Shockwave specifications remain proprietary and under the control of Adobe.
Each technology is examined to determine its capabilities, level of maturity, adherence to a formal standard, and miscellaneous facts of interest. All of this information is summarized in Table 2.1. Here, capabilities denote whether or not the technology in question provides a graphics rendering engine (G), network communications (N), and audio support (A). Maturity indicates how long a given technology has been in existence—an inception year is provided. Adherence to an open and free standard connotes how easily a technology can interoperate with other tools and 3D models. Of specific interest are VRML 2 and X3D, which became ISO/IEC standards in 1997 and 2004, respectively [24, pp. 2-3]. Maintained by the Web3D Consortium, X3D is the latest iteration of VRML and is backwards compatible with VRML 2. Finally, important or interesting facts are provided in a comments entry for each of the 16 technologies.

Following our review, in Section 2.8 we undertake a qualitative risk assessment to derive what we believe is the best option or set of options for building a CVE. In this assessment, significant factors include support that a given technology may have for graphics rendering and networking (primarily), as well as for audio (secondarily). In addition, software engineering characteristics such as utility (given by the presence of avatar and multi-user support), project documentation and help resources, support for relevant standards, and deployment challenges are all taken into account.
## TABLE 2.1

SUMMARY OF ACTIVE, DESKTOP-VR TECHNOLOGIES

<table>
<thead>
<tr>
<th>Technology (Version)</th>
<th>API, Framework, or Platform</th>
<th>Capabilities</th>
<th>Maturity</th>
<th>Open, Free VR Standard</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>FreeVR (0.5g)</td>
<td>API G</td>
<td>Since 2003</td>
<td>VRML 2</td>
<td>Found online at <a href="http://www.freevr.org">www.freevr.org</a>. Generally directed at, but by no means restricted to, CAVE environments. In general, serves as wrapper around VR input devices and offers multi-processor management</td>
<td></td>
</tr>
<tr>
<td>Technology (Version)</td>
<td>API, Framework, or Platform</td>
<td>Capabilities</td>
<td>Maturity</td>
<td>Open, Free VR Standard</td>
<td>Comments</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------</td>
<td>--------------</td>
<td>----------</td>
<td>------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>Java3D (1.5.1)</td>
<td>API</td>
<td>G-N</td>
<td>Since 1998</td>
<td>VRML 2</td>
<td>Found online at <a href="http://java3d.dev.java.net">java3d.dev.java.net</a>. Sun Microsystems’s 3D graphics API for Java. Uses scene graph data structure to manage graphics information.</td>
</tr>
<tr>
<td>OGRE (1.4.5)</td>
<td>API</td>
<td>G</td>
<td>Since 2001</td>
<td>No</td>
<td>Found online at <a href="http://www.ogre3d.org">www.ogre3d.org</a>. The Object-oriented Graphics Rendering Engine; abstracts OpenGL and Direct3D.</td>
</tr>
<tr>
<td>Technology (Version)</td>
<td>API, Framework, or Platform</td>
<td>Capabilities</td>
<td>Maturity</td>
<td>Open, Free Standard</td>
<td>Comments</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-------------------------------</td>
<td>--------------</td>
<td>------------</td>
<td>---------------------</td>
<td>----------</td>
</tr>
<tr>
<td>OpenGL (2.1)</td>
<td>API</td>
<td>G</td>
<td>Since 1992</td>
<td>No</td>
<td>Found online at <a href="http://www.opengl.org">www.opengl.org</a>. Licensing free for end-users and developers, but not for graphics hardware manufacturers. Libraries packaged with most OSs</td>
</tr>
<tr>
<td>OpenSceneGraph (2.0)</td>
<td>API</td>
<td>G</td>
<td>Since 1999</td>
<td>VRML 2</td>
<td>Found online at <a href="http://www.openscenegraph.com">www.openscenegraph.com</a>. Similar to OpenSG. Abstracts OpenGL libraries to provide scene graph data structure</td>
</tr>
<tr>
<td>Technology (Version)</td>
<td>API, Framework, or Platform</td>
<td>Capabilities</td>
<td>Maturity</td>
<td>Open, VR Standard</td>
<td>Free Standard</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------</td>
<td>--------------</td>
<td>----------</td>
<td>-------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>OpenSG (1.8.0)</td>
<td>API</td>
<td>G</td>
<td>Since 1999</td>
<td>VRML 2</td>
<td></td>
</tr>
<tr>
<td>OpenVRML (0.16.6)</td>
<td>API</td>
<td>G-A</td>
<td>Since 2000</td>
<td>X3D</td>
<td></td>
</tr>
<tr>
<td>X3D Tool Kit (1.2)</td>
<td>API</td>
<td>G-A</td>
<td>Since 2004</td>
<td>X3D</td>
<td></td>
</tr>
<tr>
<td>Technology (Version)</td>
<td>API, Framework, or Platform</td>
<td>Capabilities</td>
<td>Maturity</td>
<td>Open, VR Standard</td>
<td>Free</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------</td>
<td>--------------</td>
<td>----------</td>
<td>--------------------</td>
<td>------</td>
</tr>
<tr>
<td>Xj3D (1.0)</td>
<td>API</td>
<td>G-N-A</td>
<td>Since 2002</td>
<td>X3D</td>
<td></td>
</tr>
<tr>
<td>Technology (Version)</td>
<td>API, Framework, or Platform</td>
<td>Capabilities</td>
<td>Maturity</td>
<td>Open, Free VR Standard</td>
<td>Comments</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------</td>
<td>--------------</td>
<td>----------</td>
<td>------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>Delta3D (1.5.0)</td>
<td>Framework</td>
<td>G-N-A</td>
<td>Since 2004</td>
<td>VRML 2</td>
<td>Found online at <a href="http://www.delta3d.org">www.delta3d.org</a>. Suite of libraries and tools developed by MOVES Institute at Naval Postgraduate School. Used largely for DoD computer simulation and training</td>
</tr>
<tr>
<td>Technology (Version)</td>
<td>API, Framework, or Platform</td>
<td>Capabilities</td>
<td>Maturity</td>
<td>Open, Free VR Standard</td>
<td>Comments</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------</td>
<td>--------------</td>
<td>----------</td>
<td>------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>Quake III Engine (1.32b)</td>
<td>Framework</td>
<td>G-N-A</td>
<td>Since 1999</td>
<td>No</td>
<td>The ioquake3 project was reviewed for this chapter: <a href="http://www.ioquake3.org">www.ioquake3.org</a>; original source found online at <a href="http://www.idsoftware.com">www.idsoftware.com</a>. First released as the Quake III Arena commercial game in 1999, ID Software made the Quake III Engine’s source code available as open source under the GNU GPL license in 2005.</td>
</tr>
<tr>
<td>VR Juggler (2.0.3)</td>
<td>Framework</td>
<td>G-A</td>
<td>Since 1997</td>
<td>VRML 2</td>
<td>Found online at <a href="http://www.vrjuggler.org">www.vrjuggler.org</a>. Middle-ware framework to support VR environments</td>
</tr>
<tr>
<td>Technology (Version)</td>
<td>API, Framework, or Platform</td>
<td>Capabilities</td>
<td>Maturity</td>
<td>Open, Free Standard</td>
<td>Comments</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------</td>
<td>--------------</td>
<td>----------</td>
<td>---------------------</td>
<td>----------</td>
</tr>
<tr>
<td>Croquet (1.0.18)</td>
<td>Platform</td>
<td>G-N-A</td>
<td>Since 2002</td>
<td>VRML 2</td>
<td>Found online at <a href="http://opencroquet.org">opencroquet.org</a>. Enables creation of CVEs with Squeak (derivative of Smalltalk). Peer-to-peer environment in which the SDK/IDE acts as viewer</td>
</tr>
<tr>
<td>Project Wonderland (0.4)</td>
<td>Platform</td>
<td>G-N-A</td>
<td>Since 2007</td>
<td>X3D</td>
<td>Found online at <a href="http://lg3d-wonderland.dev.java.net">lg3d-wonderland.dev.java.net</a>. Enables creation of CVEs with Java. Largely client-server oriented</td>
</tr>
</tbody>
</table>
2.5 APIs

As discussed in Section 2.4, an API is considered to be one or more well-defined libraries that provide access to underlying hardware/software resources. In other words, the API works to abstract lower-level resources by presenting the software developer with a documented set of methods that operate as a wrapper. Although they do not offer as much convenience as frameworks or platforms, APIs do enable a “best of breed” approach toward software engineering (i.e., a developer can draw from multiple, well-conceived technologies). However, compatibility issues could result if implementations of the same CVE rely upon different, underlying APIs.

In the following subsections we examine APIs for FreeVR, Java3D, OGRE, OpenGL, OpenSceneGraph, OpenSG, OpenVRML, the X3D Tool Kit, and Xj3D.

2.5.1 FreeVR

Written in C with support for multi-processing, FreeVR is aimed at providing a uniform, cross-platform means of accessing and integrating with a variety of VR systems and configurations [123]. At present, the FreeVR library compiles on various UNIX platforms, OS X, and Cygwin for Microsoft Windows. Although suitable for use in highly immersive environments such as CAVEs, FreeVR can also be deployed in desktop-VR settings. Graphics rendering is handled by the OpenGL library; the OpenSceneGraph and OpenSG scene graph libraries may also be used.

Some of FreeVR’s notable strengths address interfacing with VR equipment and the administration of FreeVR programs. The FreeVR API makes it possible to map physical mechanisms, such as those used for input, to logical devices. This enables easier programatic access to the mechanisms while abstracting and
simplifying their use. Also working toward ease of use, FreeVR employs runtime configuration (RC) files to specify details about system resources, processes, and program input/output [123]. By editing an RC file, an administrator can change an executing FreeVR program for purposes of debugging, fine tuning, or interfacing with the user differently. Remote, administrative access to a running FreeVR program is permitted through TELNET or a Tcl/Tk GUI, and includes the ability to view and manipulate information and settings. Although this feature does not also allow the editing of RC files, it does enable debugging and the display of program statistics (e.g., frames per second).

Despite its strengths, FreeVR does face some challenges. First, it is missing support for the network and audio components needed to implement a CVE. A variety of libraries external to FreeVR can be drawn upon to fill these gaps (e.g., the FreeVR source code documentation recommends Virtual Sound Server [VSS] or Bergen Sound Server for audio), but the lack of a uniform, homogeneous approach may result in incompatible FreeVR deployments. Next, FreeVR does not have the option of using a VR standard unless OpenSceneGraph or OpenSG, both of which can support VRML 2, is chosen as the graphics renderer. Unfortunately, the updated and more robust X3D standard is not currently an option with either scene graph API. Finally, the knowledge base for FreeVR is still early in its evolution. Materials such as basic documentation and tutorials appear somewhat rough and sparse.

2.5.2 Java3D

Java3D offers the ability to manage and render 3D graphics by way of a scene graph data structure [122, pg. 2]. A scene graph is a directed acyclic graph that
permits efficient storage of, and access/updates to graphics data. More specifically, scene graphs enable “a hierarchical approach to describing objects and their relationship to each other [33],” such that each level of the hierarchy denotes some type of grouping, while the leaves represent the targets to be rendered (e.g., graphics or sound). Because scene graphs enable critical functions like culling (not displaying images that fall outside of the active viewpoint) and sorting, they are used in other desktop-VR technologies, as well (e.g., OpenVRML, the X3D Tool Kit, and Xj3D). One method to implement a scene graph is given by the Composite design pattern, where hierarchical tree structures are built from objects that may be uniformly managed as compositions or individual entities [50, pp. 163-173]. The Java3D API can use OpenGL or Microsoft’s DirectX to render images managed by its scene graph.

In addition to the benefits of a scene graph, Java3D offers the platform independence of the Java Virtual Machine and can execute under Apple OS X, Linux, Sun Solaris, and Microsoft Windows as an application or a web browser applet. Through the use of an extension library, J3D-VRML97 [65], the Java3D API can utilize VRML 2 models; this is also how the API gains basic networking capabilities (i.e., to load remote VR content).

Because Java’s extensive library of packages can be drawn upon to fill in many functionality gaps, Java3D is a viable CVE building block despite its limitations. Its weak areas include a lack of support for audio and X3D, as well as size issues. Though its use of the VRML 2 standard is a benefit, at the same time none of the enhancements and improvements of X3D are available to Java3D (the Xj3D API addresses this—see further below). Also, it may be that a given target host does not possess the Java Runtime Environment (JRE) and/or Java3D. This could
make the installation of Java3D-based CVE software more complicated, since the JRE and Java3D would need to be bundled in the install process.

2.5.3 OGRE

The Object-oriented Graphics Rendering Engine (OGRE) is a C++ API that focuses on modularity and functionality. Available for Linux, Apple OS X, and Windows, OGRE’s dedicated purpose is to “make it easier and more intuitive for developers to produce applications utilising hardware-accelerated 3D graphics [138].” To this end, the API is specifically architected to easily interface with projects that might need 3D graphics abilities. In addition, OGRE itself is designed to be extended through plugins: a number of such add-on mechanisms exists within the OGRE community and provides enhancements ranging from indoor and outdoor scene rendering to bindings for physics engines [137]. Finally, it is notable that OGRE’s documentation and community support are both highly evolved compared to many other open source projects.

Similar to OpenGL below, OGRE is strictly a graphics engine. As such, it would need to be supplemented with network and audio components if used to build a CVE. Also, it has no direct support for VR modeling standards such as VRML or X3D. Nevertheless, because OGRE has been specifically designed as an encapsulated and extensible API these limitations may really be inconveniences more than drawbacks.

2.5.4 OpenGL

Our inclusion of OpenGL in this survey is somewhat contrary to one of our goals: to review only open source technologies. Although freely available to de-
velopers, OpenGL is not open source. However, its ubiquity and pervasive use in most other graphics-related software make it the elephant in the room that we must address.

A creation of Silicon Graphics, Inc., since 1992 the OpenGL API specification has become a de-facto means of handling 2D and 3D graphics on desktop computers. Nearly all significant OSs (Apple OS X, Linux, FreeBSD, Sun Solaris, IBM AIX, HP-UX, Microsoft Windows, etc.) come bundled with, or can easily compile the OpenGL libraries [75]. Underlining this point, each of the technologies reviewed in this survey either uses OpenGL directly or has the option of doing so. Hence, regardless of the tool or tools a developer might pick from our survey, the OpenGL API is almost certainly involved. In addition to its widespread use, the OpenGL libraries are callable from several languages, including C, C++, Python, Perl, Java, and others [76]. Some additional strengths include the ability to handle computationally intense graphics rendering (e.g., programable shading, lighting, texture mapping, non-uniform rational B-spline [NURBS] curves and surfaces), the flexibility to handle graphical elements such as reflections, support for extensions that take advantage of features in specific graphics cards, and a developed knowledge base [74, 124].

Although its ubiquity, capabilities, and maturity make it an attractive technology for constructing CVEs, OpenGL does have some limitations. First, it is an API that only handles graphics: it offers no networking or audio components, and includes no functionality dedicated to the semantic of a multi-user, VR world. As intimated earlier, however, other VR-motivated technologies may be able to work around this issue by incorporating OpenGL into a more complete solution. Finally, the OpenGL specification is not free to all parties [125] (i.e., hardware
platform vendors). Hence, it is possible that software written with OpenGL may
not execute on graphics cards from vendors who do not pay to license the API,
although this may be a low-risk problem due to OpenGL’s widespread nature.

2.5.5 OpenSceneGraph

The OpenSceneGraph (OSG) project implements a scene graph data structure
on top of OpenGL to manage 2D and 3D graphics. Beyond its utility as a scene
graph implementation, other strengths of the OSG API reside in its degree of
portability, language support, and extensibility. Written in C++ with support
for multi-threading and processing, OSG is compatible with the same platforms
as OpenGL (see the previous subsection) through multiple languages including
Java, Python, Tcl, .Net, and others. OSG can also be expanded through what
are termed “node kits:” separate libraries, added at compile- or run-time, that
enhance a project with additional graphics features and special effects. Finally,
OSG supports a wide range of graphics file formats and, through the OpenVRML
API, the VRML 2 standard.

The issues faced by OSG are similar to those of OpenGL. It is an API that
only deals with graphics, necessitating the use of other resources for networking
and sound that may not be as portable as OSG itself. Also, though unrelated to
OpenGL, the utility of supporting the VRML 2 standard is diminished by OSG’s
lack of support for the improved X3D standard.

2.5.6 OpenSG

OpenSG is a project with strong similarities to OSG. Both are OpenGL-based,
scene graph APIs written in C++. Both are compatible with the same extensive
list of platforms, although there appear to be more language bindings for OSG. Both support multi-threading and processing, but OpenSG is explicitly outfitted to handle graphics clusters [113]. Both offer high-performance scene graph management, although OSG offers a compile- and run-time extension feature (node kits). Finally, both support a wide range of 3D file formats, as well as VRML 2—if the OpenVRML API is used.

OpenSG’s challenges are identical to those of OSG; see the previous subsection for details.

2.5.7 OpenVRML

The OpenVRML libraries provide an API for the VRML 2 and X3D standards. Written in C++, OpenVRML is a cross-platform API that uses OpenGL as a graphics renderer and can operate on those platforms where OpenGL is found (see the subsection on OpenGL). Moreover, if the GIMP Tool Kit Plus (GTK+) user interface libraries are available to OpenVRML, a web browser plugin may also be deployed for Mozilla-based browsers. Built around the use of a scene graph, both VRML and X3D offer a script-based means of describing, rendering, and controlling VR scenes. Complete with support for audio, dynamically changing environments, and avatars, VRML and X3D environments are typically accessed through a web browser equipped with an appropriate plugin. Alternatively, a dedicated application, such as one built on top of OpenVRML, may be used.

Although inherently suited to much of the CVE building endeavor, OpenVRML does have a few weak areas. First, it does not include a networking capability. It attempts to manage this by providing a C++ abstract class that the CVE developer can use as an interface to their own networking resources. This
may be an acceptable approach in some circumstances, but it leaves open the possibility of underlying networking components that operate in one environment but not another (possibly leading to CVEs that are not fully cross-platform). Next, despite its implementation of X3D, there is currently no support for X3D’s XML encoding. This is significant since XML is a structured, verbose language capable of being validated against a Document Type Definition (DTD) or XML Schema Reference; it “is the basis of nearly every data language used on the World Wide Web [24, pp. 22-27].” Finally, as with the FreeVR API, the OpenVRML project’s documentation is somewhat thin and in need of expansion and tutorials.

2.5.8 X3D Tool Kit

The X3D Tool Kit is a once-abandoned project now reborn. From 2002 to 2004 the original author developed the API into an implementation of the X3D standard that offered significant support for static X3D scenes—where sensor and collision nodes were not used [53]. In 2006 the project was revived by a group of developers whose stated emphasis was “on proper X3D behavior as specified by the standards committee [112].” Written in C++, the X3D Tool Kit uses built-in scene graph mechanisms to manage and process 3D graphics information, but relies upon OpenGL for image rendering. Among the X3D Tool Kit’s strengths are that its design enables the extension of the API, and its use of C++ and OpenGL permit it to be compiled on the platforms where OpenGL can operate (see the subsection on OpenGL).

The challenges facing this latest incarnation of the X3D Tool Kit are manifold. First and foremost, support for more of the X3D standard, especially collision and sensor capabilities, is needed. Next, the X3D Tool Kit is lacking a network com-
ponent. As with each of the APIs reviewed to this point, the absence of basic networking capabilities can lead to incompatible deployments of CVE software, since different developers may elect to use different networking mechanisms. Finally, the documentation available for the API exists primarily on the original X3D Tool Kit website, which was apparently frozen in 2004. Thus, any changes or updates to the API are not yet being discussed by the new developers.

2.5.9 Xj3D

The Xj3D API is maintained by the Web3D Consortium, the same organization responsible for the VRML 2 and X3D standards. Written in Java, Xj3D can operate anywhere the Java Virtual Machine can, provided there is also support for the graphics rendering engine—the developer may elect to use Java3D or OpenGL (through the Java OpenGL [JOGL] libraries) [147]. Xj3D is a mature implementation of the X3D standard, offering not only a graphics rendering mechanism, but audio and basic network components as well. As with Java3D, because the Xj3D API libraries are simply an addition to Java, the use of Xj3D makes the standard Java environment available to the developer. Other notable strengths of Xj3D include the use of a scene graph to manage VR information, support of XML to encode VR scenes, support of the H-Anim standard to facilitate avatars, compatibility with VRML 2 (increasing the base of 3D models available to X3D systems), and the ability to operate as a Java applet or stand-alone application.

The drawbacks of Xj3D are primarily logistical in nature. Similar to Java3D, the basic Java runtime environment (JRE) may or may not be bundled with a given OS. Hence, the install base for a CVE built with Xj3D may need to include the JRE, Xj3D libraries, Java3D and/or JOGL, and the CVE software itself. If
a CVE client were deployed as an applet, the collective size of all Xj3D libraries (nearly 30 megabytes at the time of this writing) may be a significant obstacle. An alternative option could be to remotely deploy a client application through Java Web Start; the application could then be executed later without need of further downloads.

2.6 Frameworks

As noted earlier in Section 2.4, we define a framework as a group of APIs that abstracts multiple domains of software/hardware resources and is extended by various support tools (e.g., content editors, administration utilities). A robust framework for CVEs might cover several areas within graphics, client-server or peer-to-peer networking, multi-user management, and sound. This enables the software developer to work under the auspices of one API/tool collection, possibly reducing documentation and maintenance overhead. One drawback, however, is that the developer may end up committed to all of the framework authors’ design decisions—those that are good and those that are poor.

In the following subsections we consider the Crystal Space, Delta3D, Quake III, VR Juggler, and Uni-Verse frameworks.

2.6.1 Crystal Space

Since its beginning in 1997, Crystal Space has evolved into a highly modular collection of C++ libraries offering CVE-related functionality in graphics (OpenGL-based or a built-in software renderer), networking, spatial sound, and other areas [139]. Compatible with Linux, Apple OS X, and Microsoft Windows, the Crystal Space framework is intended to operate as an application engine:
through an extensive system of API plugins, a developer is able to include/exclude those software components relevant to a project. In addition, through the optional high-level CEL (Crystal Entity Layer) interface, features important to game/multi-user environments (e.g., networking, game logic) are abstracted and made more accessible [141]. Another high-level interface called CELstart makes it possible to program Crystal Space projects entirely in Python or XML, and to bundle projects into standalone, executable packages [140]. Language bindings for Java and PERL are also available for Crystal Space. Finally, an extension for the Blender 3D modeling software allows content to be created for and directly exported to Crystal Space.

The primary area of weakness for Crystal Space is its lack of support for VR standards, making it difficult to share 3D models and environments with other, unrelated systems. Nevertheless, the integration of proprietary 3D model specifications and a convenient means of exporting content from Blender to Crystal Space may be sufficient for some CVE projects.

2.6.2 Delta3D

Created and maintained by the Modeling Virtual Environment and Simulation (MOVES) Institute at the Naval Postgraduate School in Monterey, California, Delta3D is aimed at VR applications in the realm of defense as well as modeling and simulation. Described on its project web site as “a fully-featured game engine appropriate for a wide variety of uses including training, education, visualization, and entertainment [97],” the Delta3D framework draws upon an array of APIs and support tools. Written in C++, the framework is compatible with Linux and Microsoft Windows with unofficial support for Apple OS X. Operating under the
premise that low-level resources should not be obfuscated by their APIs, Delta3D offers a high-level interface across a range of territory while permitting access to low-level functionality if desired [97]. Utilizing a flat intra-framework organization, there are 14 APIs comprising this interface.

Highlights of the Delta3D framework may be found in the areas of graphics, networking, sound, and support utilities. The core API’s graphics rendering capabilities are based on OpenGL and make use of OpenSceneGraph, thereby integrating the VRML 2 standard. Environmental effects (e.g., clouds, time of day), particle effects (e.g., smoke), and physics (e.g., gravity, collision detection) are also addressed in the core. Delta3D’s networking and game APIs offer client-server or standalone client modes of operation. Sound, both 2D and spatial, is managed by an audio API. Examples of utilities to assist with creating and operating the Delta3D environment include a 3D map editor (the Simulation, Training, and Game Editor [STAGE]), a particle effects editor, and a 3D model viewer. Other strengths of Delta3D include APIs for avatar, weather, and terrain management, and language bindings for Python scripting.

Although Delta3D is a modular framework covering a wide area of functionality it does have limitations regarding VR standards. While it is useful that VRML 2 is supported through OpenSceneGraph, Delta3D is unable to take advantage of the newer X3D standard and accompanying improvements.

2.6.3 Quake III Engine

Initially powering id Software’s Quake III Arena commercial game in 1999, the Quake III engine is also referred to as id Tech 3 [68]. In 2005, id Software released the game engine as open source under the GNU GPL license, as it had done for the
earlier Quake and Quake II engines. We focus on the Quake III Engine because it is the more feature-rich in the series, offering graphics rendering, spatial sound, and network capabilities that all attempt to improve upon the first two engines.

Written entirely in C and compatible with Apple OS X, Linux, and Microsoft Windows, the Quake III Engine comes ready with CVE resources, some of which are also optimized. As a game engine, the graphics rendering and networking functions are designed with performance as a goal: graphics are managed through OpenGL and require 3D hardware acceleration, while networking includes a built-in compression capability to more efficiently transmit data between client and server [66]. Also included in the engine’s bundle is the Q3Radiant map compiler and editor, though a newer version, called GtkRadiant, is freely available, too [44]. Another feature of the Quake III Engine is its use of a virtual machine (referred to as the Quake Virtual Machine, or “qvm”) to readily and securely enable game modifications by end-users. Virtual machine files are written in ANSI C, compiled into assembly code using the freeware LCC compiler, and then converted into qvm byte code via the q3asm assembler—both LCC and q3asm are included with the Quake III Engine bundle [67]. Finally, as a multi-player VR game engine, the Quake III Engine offers full support for avatars and text chat among participants.

Challenges of using the Quake III Engine are relegated to a lack of support for VR standards and somewhat dispersed documentation. Regarding standards, proprietary 3D model formats such as MD3 and BSP are used for avatars and maps, respectively. Such models may be difficult to create or manipulate, and are less interchangeable with other VR systems. Also, the Quake III Engine’s documentation is abundant, but scattered around many locations on the Internet. When id Software released the Quake III Engine as open source, they included
precious little documentation. Presently, most of the information that exists has been voluntarily produced, and there is no authoritative source. Nevertheless, many helpful documents, tutorials, and discussion forums are available, although one must search carefully to find useful resources.

2.6.4 VR Juggler

VR Juggler is a C++ framework that uses APIs in several areas to provide desktop-VR and other levels of VR immersion. There are two basic framework layers: the Juggler Portable Runtime (JPR) API, and the VR Juggler API and micro-kernel. The JPR resides on top of the OS and works to abstract critical system resources such as threads, I/O, and sockets [145]. Resting on the JPR, the Juggler API/micro-kernel layer “acts as ‘glue’ between all the other Juggler components [70].” This two-layer foundation allows a developer to build application objects in C++ that are executed by the Juggler micro-kernel. One of the paramount goals of this framework is to operate as a middle-ware between applications and underlying VR systems. In so doing, software can be built against the VR Juggler APIs with little regard for the equipment underneath; such programs become inherently portable among different VR Juggler environments.

Rounding out the framework, several more APIs and tools exist within the Juggler API/mico-kernel layer, including:

- Gadgeteer – a tool that facilitates VR device management
- The Juggler Configuration and Control Library (JCCL) – configuration and performance monitoring tools for the Juggler system
- Tweek – APIs to build extensible, Java-based GUIs that communicate with underlying Juggler applications
- PyJuggler – language bindings that permit the use of Python Juggler application objects

- VRJ.NET – language bindings that permit the use of C# and VB.NET Juggler application objects

- Sonix – an API for audio hardware and other audio APIs

VR Juggler relies on the use of OpenGL, OpenSceneGraph, or OpenSG to render images. If either OpenSceneGraph or OpenSG are utilized, then support of the VRML 2 standard is brought along. Regarding audio rendering, although the Sonix API is capable of playing sound on its own, other audio APIs, such as OpenAL, may be used for more full-featured sound options. Finally, a tool called Maestro [69] is employed to control the execution of VR Juggler applications on a cluster of machines. Maestro includes a GUI to assist an administrator with the task of overseeing and monitoring Juggler activities.

Despite VR Juggler’s breadth of capabilities, it does face some challenges. To begin with, it offers no network component—save that required to facilitate its support of graphics clusters. As discussed in earlier subsections, while it is possible for a developer to utilize a network API of their choosing, doing so may result in software that is incompatible with other VR environments. Also, support for a VR standard is absent in VR Juggler unless OpenSceneGraph or OpenSG is used for rendering graphics. However, as noted for FreeVR and Delta3D, the updated and more robust X3D standard is not currently an option with either of these scene graph APIs.
2.6.5 Uni-Verse

Founded on the Verse [129] networking protocol, the Uni-Verse framework attempts to “create an open source Internet platform for multi-user, interactive, distributed, high-quality 3D graphics and audio [143].” To realize this premise, Uni-Verse is cross-platform (Apple OS X, Linux, Microsoft Windows), client-server oriented, and includes several C APIs and support tools. The Verse protocol is designed to efficiently share 2D and 3D image data among applications, thereby creating a distributed graphics processing environment. The Uni-Verse server, clients, and various support tools leverage Verse to communicate graphical information, while the Uni-Verse Sound Rendering Protocol is used for audio data. Image rendering is handled through OpenGL with support for OpenSG under Microsoft Windows. With OpenSG, the VRML 2 standard is available if the OpenVRML libraries are used. Sound rendering is facilitated by an audio/video programming environment called Pure Data [107] (a separate project from Uni-Verse) that must be installed on a given Uni-Verse client machine.

The Uni-Verse server and client are collectively made of a Verse server and the client software suite known as Quel Solaar. Noteworthy tools packaged in Quel Solaar include a rendering client also named Quel Solaar, a 3D modeling tool called Loq Airou, and a graphical 3D scene editor called Connector. The Quel Solaar client is based on OpenGL and described by the Uni-Verse project as a high-quality graphics renderer for desktops [142]. Loq Airou, also based on OpenGL, provides a means for creating 3D objects on a Uni-Verse server through an informal “sketch pad” user interface. Finally, Connector enables the editing and management of objects stored on a Uni-Verse server.

In addition to the server and client software, other Uni-Verse system compo-
nents include, but are not limited to:

- Enough and Ample APIs – written in C and C++, respectively, these libraries help in the development of Uni-Verse programs

- PyVerse – a language binding to write Verse programs in Python

- Purple – a tool that implements a scripting environment for Uni-Verse

- Saver and Loader – tools to save and load, respectively, a Uni-Verse server’s state

- Network security – clients may be authenticated by a Uni-Verse server using a cryptographically strong cipher (RSA)

Although Uni-Verse offers a full-featured environment, it does face some issues. First, sound rendering is complicated by the need for the Pure Data programming environment to be installed on a Uni-Verse client machine. Second, support for the VRML 2 standard, while useful, falls short of the capabilities offered by the newer X3D standard.

2.7 Platforms

Our definition for platform picks up where that of framework leaves off. In Section 2.4 we specify that a platform is a fully contained, specialized environment within which a solution may be developed, tested, and executed. As such, wherever a platform-based solution is deployed, the platform must also be installed; this may be an onerous task for particularly large platforms. Of the 16 technologies surveyed in this chapter, only Croquet and Project Wonderland are considered to be platforms.
2.7.1 Croquet

Built on top of Squeak (an open source derivative of Smalltalk) Croquet’s aim is to provide a distributed, virtual environment that emphasizes realtime, multiuser collaboration. The Croquet platform employs OpenGL-based graphics, OpenAL spatial sound, and a peer-to-peer networking/synchronization architecture (TeaTime) that facilitates object sharing and communication [36]. Due to Squeak’s cross-platform nature, Croquet is available for Apple’s OS X, Linux, and Microsoft Windows. It leverages Squeak’s integrated development environment (IDE) to allow the creation, testing, debugging, and operation of Croquet software. Squeak’s late-bound characteristics (inherited from Smalltalk [128]) make it is possible for Croquet program code to be accessed and altered not only prior to, but during execution. Moreover, the TeaTime architecture immediately propagates such changes to all relevant, distributed end-users. Croquet is designed under the premise that its applications will operate across small and large scale computing environments; in TeaTime, this is supported through object replication and synchronization. Another of Croquet’s compelling strengths is that, out-of-the-box, it serves as a functional CVE: several of its example programs are fully operational CVEs with support for avatars, text chat, and the collaborative sharing of applications (e.g., text editors).

Relative to the goal of constructing a CVE, Croquet comes armed with an extensive set of features and capabilities. Nevertheless, some challenges and limitations remain. First, the current state of Croquet favors use by developers as opposed to typical end-users: the platform operates from within Squeak’s IDE and, thus, may not be suitable for deployment to a general user population. To this point, the project web site notes that “in its current state of development,
Croquet should be thought of as an enabling technology intended for use by software developers to create their own applications or customized OpenGL-based game engines—complete with user interfaces and application-specific features and functionalities [37]." Next, the prospect of providing Croquet CVE software to an end-user is troubled by the need for the whole Croquet platform to go along. At the time of this writing, Croquet comes packed into a 72 megabyte archive, regardless of the target OS. In this regard, it is similar to Java3D, Xj3D, or Project Wonderland (below), with the exception of having no Web Start or web browser applet channel for delivery. Last, Croquet requires a more modern workstation to perform well: the use of a late-bound language coupled with the multimedia factors of VR software means that extra graphics power, CPU speed, and memory will lead to better performance. Along these lines, the project FAQ suggests that a workstation “less than two years old” should be adequate to operate Croquet [35].

2.7.2 Project Wonderland

Sun Microsystems’ Project Wonderland was motivated by problems in commuters’ collaborative experiences. Specifically, Sun wanted to leverage CVE technology as a means to better integrate remote employees with their actual place of business and related work resources [134]:

Organizations should be able to use Wonderland to create a virtual presence to better communicate with customers, partners, and employees. Individuals should be able to do their real work within a virtual world, eliminating the need for a separate collaboration tool when they wish to work together with others.

Implemented in Java and leveraging the Java3D API for graphics rendering, Project Wonderland offers a substantial platform for building CVEs. Though
not as mature as Croquet, it nevertheless includes the same staple components: graphics (OpenGL or DirectX), network support, spatial sound, and multi-user capabilities with avatars. In addition, Java’s cross-platform nature makes Project Wonderland available for users of Apple’s OS X, Linux, Sun Solaris, and Microsoft Windows. Similar to Croquet, Project Wonderland comes as a fully functional CVE and also includes the ability to collaboratively share X Window applications. However, unlike Croquet, it is largely client-server oriented in its architecture. Its client and server components are based on two other open source Sun products: Project Looking Glass and Project Darkstar. Project Looking Glass and Java3D are used in combination to provide the 3D rendering for Project Wonderland; more specifically, Looking Glass offers APIs for building 3D applications and windowing environments. The Project Darkstar game server provides a “fully distributed, fault tolerant communication and event processing system [131].” It is through Darkstar that state and persistence are managed for all objects comprising a given virtual world.

The challenges facing Project Wonderland continue the similarities with Croquet. First, like Croquet, Project Wonderland is not currently in a polished state. Deployment and troubleshooting of the server components require some level of Java programming and systems administration skill. Also, even though the Wonderland client can be automatically distributed through Java Web Start, there may be need for troubleshooting and system configuration by the end-user. Next, depending on the platform, the client bundle for Project Looking Glass can range from 110 to 120 megabytes—this does not include the size of the Java runtime.

Conceptually, Project Wonderland uses peer-to-peer communications in some areas. For example, avatar motion data are transmitted among participants through peer-to-peer channels[132]. These channels, however, are managed by the Wonderland server component.
environment, should it be missing. Finally, Project Wonderland’s features make
it a resource intensive system: the need for a modern, speedy CPU (1.5 GHz or
better), a graphics adapter with hardware acceleration, and a gigabyte or more of
RAM precludes the use of older equipment [134].

2.8 A Simple Risk Assessment

As mentioned earlier, the aim of this survey is twofold: first, to concisely review
several prominent, active desktop-VR technologies, and, second, to recommend
the technology or technologies most well suited to building a CVE. In this section
we make our recommendation.

Pressman notes that “the software engineering environment supports the project
team, the process, and the product. But if the environment is flawed, it can be
the source of significant risk [111, pp. 139-140].” Hence, it is important that
our recommendation be risk savvy—that is, it should consider the development
environment risks that effect each of the technologies we survey in earlier sections.
To accommodate this, we undertake a simple, qualitative risk assessment of the
16 technologies. Our assessment is not all encompassing: for accessibility and
expedience we look at the major development environment risks comprising the
limitations of the technologies. Also, because this is a qualitative exercise, there
is some degree of subjectivity involved. Nevertheless, we believe there is value
in the assessment, since the risks we measure are easy to understand, gage, and
compare across the 16 technologies. Table 2.2 captures our assessment.

We calculate the overall risk for a given technology by adding up the products
of likelihood and impact across several specific risks:

\[ Risk = \sum_i (Likelihood_i \times Impact_i) \]

Likelihood indicates the perceived chance of something unwanted happening and impact denotes how significant (on a scale of 1 to 5, with 1 being minimal) the outcome will be. The risks we measure include:

- Primary Features Missing – likelihood of 50% for either feature absent (graphics [G] and network [N]); impact of 5

- Audio Missing – likelihood of 100% if audio is absent (A); impact of 2

- Lack of Utility – likelihood of 50% for each CVE-oriented feature missing (proprietary, H-Anim, or other avatar support [V]; multi-user support [M]); impact of 3

- Support Missing – likelihood of 25% for each type of support material missing (basic user/administrator/developer documentation [D], multiple tutorials [T], one or more forums to post questions and interact with a development community [F], and a means to track bugs [B]); impact of 4

- VR Standard Missing or Not Current – likelihood of 100% if there is no standard, 50% if a standard is not up-to-date; impact of 3

- Immature Technology – likelihood of 10% for each year less than ten; impact of 2

- Deployment Challenges – likelihood of 33.\(\frac{1}{3}\)% for each constraint (install package is 20 megabytes or more [L], special expertise required of the end-user [E], extra computing resources recommended [X]); impact of 1

Regarding the Deployment Challenges risk, it is difficult to accurately estimate how large an API, framework, or platform will be when installed: the inclusion or exclusion of optional features, utilities/tools, and optimizations can lead to
very different footprints for the same technology. Hence, a best effort is made to determine whether the size of a default installation falls above or below a 20 megabyte watermark. This watermark was arrived at by presuming a download rate of 128,000 bits per second (a very modest high-speed line), which is equivalent to 15.6 kilobytes/sec, or 20 megabytes in a little more than 20 minutes. For a 56K dial-up modem (presuming 52,000 bits per second due to good line conditions), this works out to 20 megabytes in less than an hour. Our belief is that most end-users would not wait more than 20 minutes to download CVE (client) software, and fewer still would wait an hour.
<table>
<thead>
<tr>
<th>Technology</th>
<th>Primary Features Missing</th>
<th>Audio Missing</th>
<th>Utility Lacking</th>
<th>Support Missing</th>
<th>VR Standard Missing or Not Current</th>
<th>Immature Technology</th>
<th>Deployment Challenges</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>FreeVR</td>
<td>N: 50% x 5</td>
<td>A: 100% x 2</td>
<td>V-M: 100% x 3</td>
<td>T-F-B: 75% x 4</td>
<td>VRML 2: 50% x 3</td>
<td>6 Years Old: 40% x 2</td>
<td>Negligible</td>
<td>12.8</td>
</tr>
<tr>
<td></td>
<td>2.5</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>1.5</td>
<td>.8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Java3D</td>
<td>A: Negligible</td>
<td>M: 100% x 2</td>
<td>VRML 2: Negligible</td>
<td>L: 50% x 3</td>
<td>Negligible</td>
<td>33.3% x 1</td>
<td>2</td>
<td>5.33</td>
</tr>
<tr>
<td>Technology</td>
<td>Primary Features Missing</td>
<td>Audio Missing</td>
<td>Utility Lacking</td>
<td>Support Missing</td>
<td>VR Standard Missing or Not Current</td>
<td>Immature Technology</td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>--------------------------</td>
<td>---------------</td>
<td>-----------------</td>
<td>----------------</td>
<td>-----------------------------------</td>
<td>---------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OGRE</td>
<td>N:</td>
<td>A:</td>
<td>M:</td>
<td>Missing</td>
<td>8 Years Old: L:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>50% × 5</td>
<td>100% × 2</td>
<td>50% × 3</td>
<td>Negligible</td>
<td>100% × 3</td>
<td>20% × 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.5</td>
<td>2</td>
<td>1.5</td>
<td>3</td>
<td>0.4</td>
<td>0.33</td>
<td>9.73</td>
<td></td>
</tr>
<tr>
<td>OpenGL</td>
<td>N:</td>
<td>A:</td>
<td>V-M:</td>
<td>B:</td>
<td>Missing:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>50% × 5</td>
<td>100% × 2</td>
<td>100% × 3</td>
<td>25% × 4</td>
<td>100% × 3</td>
<td>Negligible</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.5</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>Negligible</td>
<td>11.5</td>
<td></td>
</tr>
<tr>
<td>OSG</td>
<td>N:</td>
<td>A:</td>
<td>M:</td>
<td>VRML 2:</td>
<td>L:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>50% × 5</td>
<td>100% × 2</td>
<td>50% × 3</td>
<td>Negligible</td>
<td>50% × 3</td>
<td>Negligible</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.5</td>
<td>2</td>
<td>1.5</td>
<td>1.5</td>
<td>0.33</td>
<td>33.3% × 1</td>
<td>7.83</td>
<td></td>
</tr>
<tr>
<td>Technology</td>
<td>Primary Features Missing</td>
<td>Audio Missing</td>
<td>Utility Lacking</td>
<td>Support Missing</td>
<td>VR Standard Missing or Not Current</td>
<td>Immature Technology</td>
<td>Deployment Challenges</td>
<td>Risk</td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------------------</td>
<td>---------------</td>
<td>-----------------</td>
<td>-----------------</td>
<td>-----------------------------------</td>
<td>---------------------</td>
<td>----------------------</td>
<td>------</td>
</tr>
<tr>
<td>OpenSG</td>
<td>N:</td>
<td>A:</td>
<td>M:</td>
<td>VRML 2:</td>
<td>L:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>50% × 5</td>
<td>100% × 2</td>
<td>50% × 3</td>
<td>Negligible</td>
<td>50% × 3</td>
<td>Negligible</td>
<td>33.3% × 1</td>
<td>7.83</td>
</tr>
<tr>
<td></td>
<td>2.5</td>
<td>2</td>
<td>1.5</td>
<td>1.5</td>
<td>.33</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OpenVRML</td>
<td>N:</td>
<td>M:</td>
<td>T:</td>
<td></td>
<td>9 Years Old:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>50% × 5</td>
<td>Negligible</td>
<td>50% × 3</td>
<td>25% × 4</td>
<td>Negligible</td>
<td>10% × 2</td>
<td>Negligible</td>
<td>5.2</td>
</tr>
<tr>
<td></td>
<td>2.5</td>
<td>1.5</td>
<td>1</td>
<td>.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X3D Tool Kit</td>
<td>N:</td>
<td>V-M:</td>
<td>D-T</td>
<td></td>
<td>5 Years Old:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>50% × 5</td>
<td>Negligible</td>
<td>100% × 3</td>
<td>50% × 4</td>
<td>Negligible</td>
<td>50% × 2</td>
<td>Negligible</td>
<td>8.5</td>
</tr>
<tr>
<td></td>
<td>2.5</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Technology</td>
<td>Primary Features Missing</td>
<td>Audio Missing</td>
<td>Utility Lacking</td>
<td>Support Missing</td>
<td>VR Standard Missing or Not Current</td>
<td>Immature Technology</td>
<td>Deployment Challenges</td>
<td>Risk</td>
</tr>
<tr>
<td>--------------------</td>
<td>--------------------------</td>
<td>---------------</td>
<td>-----------------</td>
<td>-----------------</td>
<td>------------------------------------</td>
<td>---------------------</td>
<td>------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Xj3D</td>
<td>M:</td>
<td>Negligible</td>
<td>50% × 3</td>
<td>Negligible</td>
<td>7 Years Old: 7 Years Old: L:</td>
<td>30% × 2</td>
<td>33.3% × 1</td>
<td>1.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Negligible</td>
<td></td>
<td>Negligible</td>
<td></td>
<td></td>
<td></td>
<td>.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>.33</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2.43</td>
</tr>
<tr>
<td>Crystal Space</td>
<td>Missing:</td>
<td>Negligible</td>
<td></td>
<td>Negligible</td>
<td>L:</td>
<td>100% × 3</td>
<td>33.3% × 1</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Negligible</td>
<td></td>
<td>Negligible</td>
<td></td>
<td></td>
<td></td>
<td>.33</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Negligible</td>
<td></td>
<td>Negligible</td>
<td></td>
<td></td>
<td></td>
<td>3.33</td>
</tr>
<tr>
<td>Technology</td>
<td>Primary Features Missing</td>
<td>Audio Missing</td>
<td>Utility Lacking</td>
<td>Support Missing</td>
<td>VR Standard Missing or Not Current</td>
<td>Immature Technology</td>
<td>Deployment Challenges</td>
<td>Risk</td>
</tr>
<tr>
<td>------------</td>
<td>--------------------------</td>
<td>---------------</td>
<td>-----------------</td>
<td>----------------</td>
<td>-----------------------------------</td>
<td>---------------------</td>
<td>----------------------</td>
<td>------</td>
</tr>
<tr>
<td>Delta3D</td>
<td>B: 5 Years Old: L:</td>
<td>Negligible</td>
<td>Negligible</td>
<td>25% × 4</td>
<td>50% × 3</td>
<td>50% × 2</td>
<td>33.3% × 1</td>
<td>1.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Negligible</td>
<td>Negligible</td>
<td>1</td>
<td>1.5</td>
<td>1</td>
<td>.33</td>
<td>3.83</td>
</tr>
<tr>
<td>Quake III</td>
<td>Missing: L:</td>
<td>Negligible</td>
<td>Negligible</td>
<td>100% × 3</td>
<td>Negligible</td>
<td>33.3% × 1</td>
<td>.33</td>
<td>3.33</td>
</tr>
<tr>
<td>VR Juggler</td>
<td>N: V-M: T: VRML 2: L:</td>
<td>50% × 5</td>
<td>Negligible</td>
<td>100% × 3</td>
<td>25% × 4</td>
<td>50% × 3</td>
<td>Negligible</td>
<td>33.3% × 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Negligible</td>
<td>25% × 4</td>
<td>50% × 3</td>
<td>Negligible</td>
<td>33.3% × 1</td>
<td>.33</td>
<td>8.33</td>
</tr>
</tbody>
</table>

TABLE 2.2

Continued
<table>
<thead>
<tr>
<th>Technology</th>
<th>Primary Features Missing</th>
<th>Audio Missing</th>
<th>Utility Lacking</th>
<th>Support Missing</th>
<th>VR Standard Missing or Not Current</th>
<th>Immature Technology</th>
<th>Deployment Challenges</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uni-Verse</td>
<td>V-M: Negligible Negligible</td>
<td>F-B: 100% × 3 50% × 4</td>
<td>50% × 3</td>
<td>50% × 2</td>
<td>Negligible</td>
<td>Negligible</td>
<td>7.5</td>
<td></td>
</tr>
<tr>
<td>Croquet</td>
<td>VRML 2: Negligible Negligible Negligible Negligible</td>
<td>50% × 3</td>
<td>30% × 2</td>
<td>100% × 1</td>
<td>1.5</td>
<td>.6</td>
<td>1</td>
<td>3.1</td>
</tr>
<tr>
<td>Wonderland</td>
<td>2 Years Old L-E-X: Negligible Negligible Negligible Negligible Negligible</td>
<td>80% × 2</td>
<td>100% × 1</td>
<td>1.6</td>
<td>1</td>
<td></td>
<td>2.6</td>
<td></td>
</tr>
</tbody>
</table>
Our desire is to select a technology that minimizes the overall risk. The largest quantity that could theoretically appear in Table 2.2’s Risk column is 20—given by the sum of the impacts for each individual risk. Happily, most of the technologies in our survey score around or below half of this maximum, with the least risky being Xj3D (2.43) and the most risky being FreeVR (12.8). Thus, we believe the technology that is most conducive to building and deploying CVEs is Xj3D.

There are, however, some remarks to be made about this selection. First, it is noteworthy that among the three least risky technologies (Xj3D, Wonderland, and Croquet), deployment risk counts for much of the overall score. In particular, if we ignore the need for end-user expertise and additional computing power, the risks for Wonderland and Croquet drop to 1.93 and 2.43, respectively. Putting Wonderland in first place and tying Croquet with Xj3D may be a more accurate reflection of overall risk if we are unconcerned about our end-users’ capabilities and resources. Next, this risk assessment tries to compare apples with apples across all technologies in the survey; this means an important feature specific to a given technology may not factor into the assessment. In the case of Croquet, for example, atomicity and synchronicity of multi-user activities are guaranteed: such a characteristic may be critical for some CVE applications. Finally, although a best effort is put forth to make reasonable choices, one should bear in mind that this is a qualitative risk assessment. Even a modest change in perspective on the 16 technologies could lead to different selections for likelihood and impact and, thus, a different outcome.
2.9 Conclusion

The boon of VR has been somewhat muted as a result of technology solutions that are redundant and/or incompatible. This circumstance has worked against the acceptance and integration of VR as a common computing interface. In an effort to untangle some of the overlap and differences among significant desktop-VR technologies, we have surveyed 16 current, open source solutions (grouped as APIs, frameworks, and platforms). During this review we discussed the strengths and challenges for each technology relative to the goal of constructing a CVE. We then undertook a simple, qualitative risk assessment to make a determination of which technologies were most well-suited to building a CVE. A given technology’s risk was found by summing the products of likelihood and impact along an axis of seven development environment risks. This axis was drawn from the issues and limitations uncovered by our earlier review of the 16 technologies, and included consideration for the presence of CVE-oriented features that could aid the development process. Although we concluded that Xj3D is the technology most well-suited to building and deploying a CVE, even a minor shift in what is considered important about the surveyed technologies could change the risk assessment’s outcome. For example, Project Wonderland becomes the least risky technology if we ignore the deployment needs of end-user expertise and computing power.
CHAPTER 3

A PROTOTYPE VIRTUAL EMERGENCY OPERATIONS CENTER USING A COLLABORATIVE VIRTUAL ENVIRONMENT *

3.1 Abstract

In the realm of emergency operations, planning and training is a critical ingredient for success. The use of virtual environments can offer a convenient means of practicing and simulating activities in an emergency operations center (EOC). Although many virtual environments strive to offer realism in their simulations of weather, population, and incident happenings, they often fall short in terms of collaboration among simulation participants: unless participants are at the same physical location, their ability to see and interact with one another is limited. Moreover, the interactivity that is possible may not be truly synchronous (e.g., network lag can cause activities to happen out of order). These are compelling drawbacks to computer-based EOC simulators/trainers, since collaboration is a

cornerstone for successful EOC teams. To address these integrity problems, in this chapter we present our prototype for a virtual EOC. Our work aims to provide a collaborative, virtual environment that enables interactivity among participants while executing synchronous, script-driven tests and simulations.

3.2 Introduction

In the United States, the 2002 National Research Council (NRC) report on technology and terrorism clearly notes the importance of modeling and simulation techniques for disaster planning and training [101]. Subsequent to this, in early 2003, a U.S. Homeland Security Presidential Directive was issued creating the National Incident Management System (NIMS). The primary function of NIMS is to provide a framework that can be leveraged by government, private-sector, and other organizations to “prepare for, prevent, respond to, recover from, and mitigate the effects of incidents regardless of cause, size, location, or complexity [41].” Throughout NIMS, training is noted as an essential ingredient of preparedness, and skills development and maintenance. Of course, the U.S. is hardly alone in the ambition to leverage technology for disaster management: subsequent to the Indian Ocean tsunamis in 2004, the Tsunami Evaluation Coalition (TEC) recommended the use of open source software to help prepare for disaster response [40]. Following the guidance of both the 2002 NRC report and TEC, one method for emergency operations centers (EOCs) to address their training needs is to employ virtual environments for testing and simulating emergency scenarios.

Within the context of EOC training, virtual environments offer a way to model incidents of different types and severity levels while testing EOC team members’ abilities to respond. Though realism and completeness vary from one virtual
environment to the next, the use of computer-simulated emergencies can be an
efficient and expeditious means of training [29, 60]. For example, scenarios can be
electronically created, saved, edited, and re-used. Also, a similar amount of effort
is required whether running drills for an individual or a group. Finally, many
virtual environments include substantial capabilities regarding the simulation of
weather, populations, and events such as fires, floods, chemical spills, etc. Despite
all of these relevant characteristics and features, we believe that collaboration is
significantly limited within current emergency management virtual environments.

Most commercially developed EOC virtual environments are geared toward
the participation of centrally located team members. Products such as Alion Sci-
ence’s Emergency Command System™ Training & Exercising Tool (ECS), Break-
Away’s Incident Commander™, ETC Simulations’ Advanced Disaster Manage-
ment Simulator Command (ADMS-COMMAND™), and MASA-SCI’s Simula-
tion for Wargaming Operation Research and Doctrine—Critical Infrastructure
(SWORD-CI™) all include robust virtual environment and simulation mecha-
nisms. However, they are also best suited for individuals or groups of participants
operating in a local area network (LAN) context. The reasons for this are time
and synchronicity. In a stand-alone (single participant) situation there are no
timing or synchronicity issues to consider: one’s desktop is always in lock step
with itself. For groups of participants, however, unless everyone is in the same
geographic location sharing the same LAN (so that timing issues are negligible if
not imperceptible), experiences with a virtual environment can include network
latencies and disruptions. In discussing the potential use of virtual reality game
engines for incident management training, Jain and McLean [72] capture this issue
succinctly by noting that:
The primary challenge is the development of mechanisms for communications and time synchronization between and among simulation modules and game modules. Major issues associated with distributed multiplayer games are how and when players receive information on fellow players’ actions. Time lags may occur between when a player initiates an action and when other players see the action. This latency causes problems in the execution of distributed games.

Because collaboration is such an important component on EOC teams, we contend that timing and synchronization are essential ingredients for any virtual environment used in training and simulation. This is not to diminish the other roles played by the virtual environment. However, timing and synchronization problems lead to integrity issues (i.e., a group experiences the same simulation in different and erroneous ways) and can limit the potential of training and simulations for geographically dispersed participants. In turn, this reduces convenience and cost effectiveness, since collaboration may suffer unless resources are expended to bring all participants to the same physical location.

We believe that a virtual EOC (vEOC) predicated on synchronous collaboration and capable of useful testing and simulation is readily achievable. Following the previous chapter’s review and risk assessment of many popular open source tools for building collaborative, virtual environments (CVEs), we have settled on the Croquet Software Development Kit to implement the initial prototype of a vEOC. This prototype represents a step in the first phase of Project Ensayo, an endeavor to construct a large-scale system to research, simulate, and train for emergency incident management [13]. Written in Squeak, a derivative of Smalltalk-80, Croquet was selected for several reasons, including its built-in collaboration, timing, and synchronization mechanisms; OpenGL 3D graphics; support for avatars; text messaging communication; and platform independence. The vEOC prototype includes only very basic, but important, features: fully synchronized collaboration
(through Croquet) among distributed participants to ensure the integrity of simulations and training, an Extensible Markup Language (XML) interface to control the vEOC during tests and simulations, and a three-dimensional virtual environment with programmable video and still-image displays.

The remainder of this chapter is structured as follows: the next section summarizes related works; Section 3 provides further background and details about Croquet, its core architecture, and how this relates to the prototype vEOC; Section 4 introduces the prototype vEOC including its 3D room (modeled after the Miami-Dade EOC); Section 5 discusses some of the drawbacks discovered while using Croquet; and Section 6 ends the chapter with summary remarks and a brief discussion of what lies ahead for the vEOC.

3.3 Related Work

A variety of different commercial products exists in the area of emergency management training and simulation. Although not an exhaustive list, the four offerings noted earlier, ECS, Incident Commander, ADMS-COMMAND, and SWORD-CI, represent a range of approaches in virtual environment technology. All of these products have several common, basic goals: realistic simulations, engaging scenarios that can be customized, single- and multi-participant support, geographic information system (GIS) functionality, and, of course, an overall effective training and simulation experience. Differences are largely manifested in the degree of immersion (the sense a participant has of being in the virtual environment) and proximity to the simulated incidents. For example, Incident Commander offers a 2D “shadow box” display and abstract user interface, while ADMS-COMMAND provides a 3D experience that is on-the-ground and in first-person. Meanwhile,
ECS and SWORD-CI integrate several components such as 3D visualizations, training and simulation workflow mechanisms, features to manage and choreograph scenarios, and, in the case of ECS, the interweaving of real-world elements (e.g., faxes, emails, video news clips) into the training or simulation.

An interesting, recent trend is that of adapting pre-existing game engines to the purposes of emergency management simulation. One study on homeland security training and simulation software notes that cooperative team training for “geographically dispersed organizations” may benefit from CVE systems found in online games such as the Sims Online and EverQuest [127]. McGrath and McGrath point out that established game technology can help make immersive virtual environments more affordable and available for emergency management training and simulation endeavors [96]. To this end, they profile two relevant projects developed with the commercial Unreal Tournament game engine: Unreal Triage (a first-responder arriving at the scene of an airline accident) and Unreal Tunnel (remote EOC interactions with responders to a highway traffic tunnel incident). Also, Jain and McLean devise an architecture for integrating game engines with the needs of emergency management training [72], and then construct a prototype illustrating this architecture. Their prototype, which executes a dirty-bomb scenario, incorporates a number of simulation modules (plume, crowd, traffic, healthcare, and transportation) and two gaming modules (triage [based on Unreal Triage] and incident management strategy) for incident response and management training.

Jain and McLean [72] as well as McGrath and McGrath [96] point out that one of the drawbacks to re-purposing game engines is the potential for software license costs. Further issues identified by Jain and McLean [73] include, but
are not limited to, difficulties integrating disparate simulation modules, enabling communications between these modules and the game engine, and, of course, timing and synchronization among training and simulation participants.

Other efforts in the area of emergency management simulation and training have emphasized the simulation aspect. For example, Loper and Presnell use complex adaptive systems and agent-based models to simulate a state-level EOC’s behavior and information flows [86]. Along similar lines, Pollak, Falash, Ingraham, and Gottesman employ discrete event simulation models to investigate activities during an anthrax attack scenario. Specifically, they model an EOC, hospital, medical care distribution center, and civilian population. In building their simulation, an operational analysis framework is devised to integrate the various models, providing a solution to some of the issues (noted above) raised by Jain and McLean [110]. A different framework approach, suggested by Bowers and Prochnow, leverages simulations developed at the Department of Defense together with the IEEE’s High Level Architecture (HLA) [47]. Here, a combination of the Joint Theater Level Simulation (JTLS—an operational-level warfare simulator), the Joint Conflict and Tactical Simulation (JCATS—an interactive, entity-level simulator), and HLA to integrate these simulations, is offered as a possible framework to meet the needs of emergency response simulation and training.

3.4 Croquet and Teatime

The vEOC prototype establishes a basic distributed architecture within the open source Croquet environment. Croquet is designed for “creating and deploying deeply collaborative multi-user online applications on multiple operating systems and devices [34].” Croquet employs its TeaTime architecture as the means
for communication and synchronization of local and remote system objects [36]. Implemented as a peer-to-peer network, TeaTime enables objects to share information and interact through simple message passing. By maintaining a universal time that is strictly coordinated among all peers, TeaTime enforces synchronicity and, through a two-phase commit process, atomicity in the objects and activities hosted by peers. This means that vEOC participants, regardless of their proximity to one another, will experience their virtual environment in a coherent and simultaneous manner. The imperative goal of establishing the vEOC prototype within Croquet, then, is to derive fundamental EOC training and simulation capabilities that have integrity through synchronicity and atomicity. Specific capabilities include: a basic means of scripting and testing scenarios; support for local and remote, simultaneous users; and support for a desktop virtual reality (desktop-VR) interface including avatars and some form of participant communication.

A simple network diagram of the prototype vEOC system is provided in Figure 3.1. Here, we see that all participants connect to and replicate an initial Croquet “island” (a collection of objects) containing the vEOC system; from then on, these participants and the initial island are synchronized peers of one another who communicate through message passing. For security purposes, a virtual private network (VPN) tunnel is used to protect remote accesses to the prototype vEOC’s LAN.

TeaTime’s peer-to-peer network approach also brings efficiency and resiliency to the prototype vEOC. Efficiency is gained as a result of replicating (at peer sites) not only the island containing vEOC components, but the computations as well. This may not seem intuitive at first. By having smart copies of the initial island on all peers, small messages describing state and activities are transmitted
during a training/simulation session, instead of shuffling around large collections of data. This also means that high-latency participants will not slow down other peers. All peers operate against a coordinated time system, which maintains strict synchronicity. Slower peers will end up experiencing sluggish performance as they work to stay in synch, but all replicated islands will remain identical. Adding to this resiliency, Croquet objects track state information to enable recovery from system faults. This is a product of the two-phase commit protocol that Croquet uses to ensure atomicity of behaviors among peers.

To better explain Croquet’s TeaTime architecture, figures 3.2, 3.3, and 3.4 depict a simplified example of a participant connecting to an island of objects. First, Host 2 notifies Host 1 that it intends to join Host 1’s island (Figure 3.2); Host 1 responds by replicating its island (objects and state) on Host 2 (Figure 3.3). Later, when object B in Host 2’s island replica changes state (e.g., due to user
interaction), a message is communicated back to Host 1 to maintain synchronicity between the island copies (Figure 3.4). Because Host 1 is the originator of the island, it also includes a Croquet mechanism known as a router (pictured in Figure 3.5). A second, related mechanism, referred to as a controller, is also operated by the island originator, as well as all other participants (again, see Figure 3.5).

The router is responsible for managing the clocks of the initial and all subsequent island replicas. Whenever a message is transmitted from one replica to another (steps 1 and 2 in Figure 3.5), that message must travel through the router to be time-stamped. The controller, then, is in charge of aggregating and handling messages in order (step 3 in Figure 3.5). It is by way of these components and processes that synchronicity and atomicity are enabled. Also, not all object state changes need to be communicated among all island replicas. This is the case, for example, when rendering images to a participant’s screen. In summary, each Croquet island replica is identical and operates from a known starting point with the same inputs; thus, all replicas are deterministically the same; through simple message passing, replicas maintain their identical nature over time. It is this property that enables Croquet to deliver a truly synchronous experience, making it well suited to collaboration among distributed participants.
Figure 3.2. Joining a Croquet Island.

Figure 3.3. Island Replication.
Figure 3.4. Maintaining Synchronicity between Island Replicas.
Figure 3.5. Message Passing Details in Croquet.
Finally, the desktop-VR interface provided by Croquet makes interactions within the prototype vEOC more intuitive and natural. The use of VR in this fashion can offer advantages in terms of user interfaces, training, and collaboration [48, 90, 105]. Again, the idea is to enhance the integrity of the vEOC’s collaborative characteristics, thereby creating a better training/simulation experience.

3.5 Tour of the vEOC

The prototype vEOC, built on top of Croquet’s KidsFirst Application Toolkit (KAT) demo, may be broken down into three major components: fully synchronized mechanisms for collaboration (already built into Croquet), an XML interface to uniformly specify tests and simulations, and a three dimensional virtual environment with video and still-image displays that can be programmed through the XML interface.

3.5.1 Collaboration Mechanisms

We have already reviewed Croquet’s inherent synchronization capabilities provided by the TeaTime architecture. Nevertheless, this is such an important topic that it is worth reiterating briefly. Croquet’s guarantee of synchronicity and atomicity among participants begins by ensuring that each participant’s copy of the virtual environment is identical. Next, Croquet enforces a universal clock for participants by rigidly controlling and time-stamping all messages they pass. Such communications take place at various moments, including whenever an object in one participant’s virtual environment changes state. Under this circumstance an update message is broadcast to all other participants, since their environments also contain the same object. The implication is that if all participants’ environ-
ments are the same and begin execution from a known, synchronized state, then they can maintain their synchronicity through peer-to-peer communications that are regulated by Croquet’s universal clock. Atomicity becomes a by-product of this system, implemented as a two-phase commit process such that a change is aborted for all island replicas if it fails to be applied in any one of them.

Croquet offers several other collaborative tools, two of which, text messaging and text whiteboards, are used by the prototype vEOC. Before looking at these further, however, it is important to note that Croquet’s 3D virtual environment itself (based on OpenGL) is a collaboration mechanism by which participants’ avatars can see and interact with one and other. Figure 3.6 shows us two avatars standing in the vEOC’s main operations area.

Text messaging within Croquet is similar to other CVEs: participants may direct simple messages to one and other or to groups. Figure 3.7 shows the same
two participants from Figure 3.6 engaged in a brief discussion (note that both participants joined the vEOC environment under the same, general moniker of “everyone”).

The text whiteboard offers one convenient way for participants in the prototype vEOC to collaboratively track information. To use the whiteboard, a participant merely clicks on it and enters/edits text. Text may also be copied from programs outside of Croquet and pasted directly onto the whiteboard. Figure 3.8 demonstrates the text whiteboard in action (from the first-person perspective of one of the avatars).

3.5.2 The XML Interface

We elected to use XML as a means of describing vEOC tests and simulations in an accurate and uniform way, as well as to make these descriptions portable.
At the present time, Croquet supports document type definitions (DTDs), but not XML schemas. Despite the limitations of DTDs (e.g., they are not written in XML and can specify only text data types for content [120]), we find the use of a vEOC DTD helpful in guiding the construction of test/simulation scripts. Also, the XML interface acts as a form of abstraction for the entire vEOC environment: in future iterations of the prototype vEOC, an external game engine could feed a continuous, dynamic XML stream into Croquet. In this fashion, simulations and tests that take advantage of external resources, such as agent-based modeling, could be implemented. Currently, however, the prototype vEOC follows a static, XML script to carry out tests and simulations.

---

1It should be noted that going outside of the Croquet environment for test or simulation directives creates synchronicity challenges. For example, a dynamic XML feed may experience network problems, such that one participant can access the feed while another cannot. This situation could disrupt Croquet’s synchronicity and atomicity characteristics.
The basic DTD is patterned after the injection script format used by the Miami-Dade EOC. Under this format, individual script items are viewed as messages, although a message may be information, an activity, or a state of some sort. Primary script fields include time (an offset from the start of a test), injection number (a unique integer denoting the sequence in which scripted items take place), sender/receiver of a message, type of message (e.g., information, resource request, incident), message text, and expected action. We have extended this format to accommodate timed displays of MPEG videos and GIF/JPEG images (see the following sub-section). The DTD is provided in Figure 3.9 and a sample XML script follows in Figure 3.10 (see Appendix A for an excerpt of the Squeak source code comprising the XML interface).

When the prototype vEOC is initially executed, it reads an XML test/simulation script. As the script’s directives are processed, video and image resources are loaded and stored until they are needed for display later. Similarly, test and simulation message injects are built and set aside for use at the appropriate times. In the current prototype vEOC, message injects are displayed outside of the 3D window (but within the Croquet environment); presently, there is no input solicited from a participant in response to a message. This is, of course, not desirable, and is intended as a simple placeholder for future functionality. Ultimately, participants should be able to react to a given message and have their response critiqued. Figure 3.11 shows an example message display.

---

2Injections are simply pieces of narrative that describe a test’s or simulation’s activities and state. Inject messages, then, are provided to vEOC participants as a means of describing events that unfold.
<!-- Document Type Definition for vEOC Simulation Scripts -->
<!ELEMENT simulationScript ((msgRecord|rsrcRecord)+)
<!ELEMENT msgRecord (time,sender,receiver,message,action,explanObj)
<!ELEMENT rsrcRecord (time,device,file)
<!ELEMENT time (#PCDATA)
<!ELEMENT sender (agency+)
<!ELEMENT receiver (agency+)
<!ELEMENT agency (#PCDATA)
<!ELEMENT message (#PCDATA)
<!ELEMENT action (#PCDATA)
<!ELEMENT explanObj (#PCDATA)
<!ELEMENT file (#PCDATA)
<!ELEMENT device (#PCDATA)

<!-- A message record includes a type attribute -->
<!ATTLIST msgRecord type (infoRequest|infoUpdate|actionRequest|incident|resourceRequest) "infoUpdate"/>

<!-- Time consists of offset (from some starting point) and injection number attributes -->
<!ATTLIST time offset CDATA #REQUIRED>
<!ATTLIST time injectNum CDATA #REQUIRED>

<!-- A device consists of type (video or photo display) and ID number attributes -->
<!ATTLIST device type (video|photo) "video">
<!ATTLIST device devID CDATA #REQUIRED>

Figure 3.9. The vEOC simulation script DTD.
Figure 3.10. A sample vEOC XML test/simulation script.
Figure 3.11. Example message display.
3.5.3 Video and Photo Displays

As found in the various figures preceding, the prototype vEOC is equipped with video and image displays that hang down from the ceiling. During observations of a Miami-Dade EOC test activation in May 2007, it was observed that such displays were used to purvey a variety of information (e.g., news programs, weather and GIS maps, and slide shows containing EOC team updates). We decided to incorporate eight of these displays (four MPEG displays and four JPEG/GIF displays) into the prototype vEOC. Each display is programmable via the XML test/simulation script facility: specific videos and still images can be loaded at the start of a test or simulation and then rendered at specific times and on specific displays. While the still images remain static (until they are replaced with different images), the MPEG videos may be started, paused, and re-started by participants clicking on a given display. When a video plays through to the end, it automatically rewinds to its beginning. Figure 3.12 shows an avatar looking up at the MPEG of a hurricane as seen by a weather satellite.

3.6 Croquet Risks

Croquet’s strong emphasis on collaboration and its OpenGL-based 3D environment make it a good choice for implementing a vEOC trainer/simulator. However, the use of Croquet is not without costs. Croquet offers an array of useful resources through its website (www.opencroquet.org) and mailing lists (croquet-user and croquet-dev). Also, it comes out-of-the-box with several demonstration CVEs—the prototype vEOC was, in fact, built on top of one of these, which provide both example code and a starting place for budding Croquet developers. Unfortunately, the demos are poorly commented (or not commented at all, in some cases), and
often have overlapping functionality that is not uniformly implemented. As soon as a developer steps off the path embodied by these demos, there must be a significant amount of time spent unraveling and learning about the complex Croquet hierarchy of Squeak objects. This is often necessary because official Croquet documentation, though extensive, is still incomplete. Of course one can, and often must, appeal to the croquet-dev mailing list for expert help. However, assistance is offered on a volunteer basis and may require some time to materialize. Other Croquet weaknesses include very limited support for avatars with humanoid movements (e.g., walking, running, jumping), and incomplete support for content built in the Virtual Reality Modeling Language (VRML)—there is currently no support for Extensible 3D (X3D), VRML’s successor. Avatar limitations can directly constrain the collaboration experience in Croquet, since avatars that are less realistic diminish the sense of immersion. Also, non-VRML/X3D content may be
less portable and require learning complicated 3D graphics software to construct.

Perhaps the most serious risk manifested in Croquet is the difficulty of using resources outside the TeaTime architecture. Synchronicity can be broken if, for example, a resource is available to one island replica but not another. This is significant to the endeavors of the vEOC since the use of existing, external simulation and database tools may be desirable. For example, Fiedrich and Burghardt [46] note how agent-based simulation (ABS) may be used to create and analyze complex disaster scenarios, and as a means of decision support during a disaster response. Having to build agents within Croquet, as opposed to leveraging pre-existing ABS technology, may be tedious, costly, and contrary to good software engineering practice. In fact, this issue may be compelling enough to warrant consideration of CVE options other than Croquet (e.g., the Java-based Wonderland system). Weaker synchronicity/atomicity could be tolerated in exchange for greater integration and support of outside resources.

It is important to note that Croquet is billed as a software development kit, not a polished system for the masses. Understanding Smalltalk is a requirement for doing anything meaningful with Croquet; this alone can be a challenge for those raised on a diet of languages that are not truly object-oriented.

3.7 Conclusions and Future Work

Within the realm of emergency management training and simulation, there is a growing precedent to leverage collaborative virtual environments. Such environments enable the creation, customization, and re-use of scenarios, as well as the ability to support multiple participants. This can be a potent tool for EOC teams that look to collaborative training as a critical ingredient of success. However,
many virtual environments are aimed at participation on a LAN and may not support synchronized and atomic activities among distributed participants. For EOC training and simulation, this can have a negative impact on the integrity of the collaborative experience. In response to this problem we have built a prototype virtual EOC based upon the Croquet software development kit. Our prototype takes advantage of Croquet’s TeaTime architecture, which guarantees synchronicity and atomicity among all participants—regardless of their proximity to each other. Moreover, the prototype vEOC makes use of additional Croquet collaboration tools such as text chat, text whiteboards, and general 3D capabilities. We have also added an XML interface as a convenient, uniform way to accurately describe tests and simulations. Finally, within the vEOC we have implemented video and still-image displays that are programmable via the XML interface.

As a first prototype, our current vEOC implementation demonstrates only the most rudimentary capabilities required to test and simulate EOC scenarios. Clearly, this is a long way from being a useful training tool. For example, the interaction between participants and scenario messages still needs to be implemented: presently, messages are displayed to participants at times that are defined in the XML test/simulation script, but there is no response or evaluation mechanism. There are also no specific roles for participants to play during a test or simulation. Presumably, when roles are put into effect, messages can be directed to specific participants as opposed to everyone. Other important EOC functions are missing at the moment, too; for example there is no GIS capability. Finally, administrative and account management tools need to be devised, in particular with regard to creating and editing the vEOC test/simulation scripts, as well as setting up role-based participant accounts.
CHAPTER 4

DISCRETIONARY ACCESS CONTROLS FOR A COLLABORATIVE, VIRTUAL ENVIRONMENT *

4.1 Abstract

As collaborative virtual environments (CVEs) are more widely used, participant access to CVE objects and information becomes a significant concern. In virtual reality games, storefronts, classrooms, and laboratories, for example, the need to control access to spaces and objects is integral to the security of activities taking place there. However, limited access controls are typically available in CVEs. Often, such controls are course-grained, only protecting against movements by unauthorized participants into specific areas. In answer to this deficiency, we offer a discretionary access control (DAC) system based on traditional concepts of users and groups, and tailored to the needs of a CVE. Our system, called WonderDAC, includes the ability to restrict movement into areas, as well as control interactions with objects. A basic WonderDAC prototype has been implemented within the Project Wonderland CVE.

* Material from this chapter is currently under review by the International Journal of Virtual Reality.
4.2 Introduction

We take the term “collaborative virtual environment” (CVE) to mean a graphical, multi-user, virtual reality environment capable of operating on a typical, modern workstation. Often such environments are associated with game playing (e.g., the Quake series of games, massively multiplayer online role-playing games [MMORPGs] such as World of Warcraft), though commerce, educational, research, and social networking systems are also being realized through CVE technology [14, 19, 21].

The essential quality of any CVE is, of course, the collaboration among participants; however, this can also serve as an avenue for problems. Players in a game may elect to cheat, patrons of a virtual store might try to steal, researchers could inadvertently damage information or interfere with an experiment, and participants of a social networking system might have sensitive, personal information exposed. Problems of this ilk have manifested themselves in commercial CVEs such as Second Life. For example, the December 2006 malicious disruption of a CNET interview taking place in Second Life [136], the March 2007 defacement attack carried out on John Edwards’ presidential campaign headquarters [23], the April 2007 attack on Toyota’s Scion storefront [146], or the myriad of other, similar attacks and disruptions caused by miscreants (also called “griefers”) in Second Life [42, 45]. Malware exploits within CVE worlds are also a concern, as demonstrated by the November 2007 QuickTime vulnerability within Second Life: an attacker could leverage a bug in the QuickTime player to “crash or exploit the Second Life viewer [80].” Through the presence of better privacy and integrity controls, incidents and weaknesses such as these could have been substantially mitigated, if not outright prevented.
To manage the integrity and privacy risks of a CVE, we note that security works best in layers, and that there are at least three layers where controls should be employed. The use of layered security controls, or defense-in-depth, is a well-understood means to reduce the chances of malicious and accidental damage to information systems [22, 117, 119]. The basic concept is that multiple controls can operate in a synergistic fashion, reducing risk more broadly as a group than as individual measures. Moreover, if one control becomes disabled or compromised, the other controls will still be in effect and may compensate.

For expedience in discussing access control, as well as for alignment with our search of the literature, we choose to divide a CVE into the following convenient layers: network communications, in-world access, and user experience. The network communications layer includes those mechanisms that transmit data between a CVE client and server, or among CVE peers (depending on whether a client-server or peer-to-peer architecture is employed). The in-world access layer encompasses course-grained access to a CVE system, its map, and resources. Finally, the user experience layer deals with the immersive qualities of a CVE, including the sense of awareness and privacy imparted by a virtual world’s architecture, and a participant’s virtual ability to see and hear. Figure 4.1 depicts the three layers.

We choose to focus on security in the middle layer for two reasons. First, network layer security is already an area of long-standing, significant research. Regardless of whether a CVE or some other system resides on top of this layer, the options and techniques to secure network communications are the same. Second, although there is no strong coupling between the network communications and in-world access layers, there is between the top two layers. In particular, the means of authentication and authorization used for the in-world access layer should be
leveled by the user experience layer, too. Hence, this dependency compels us to deal with the in-world access layer first.

We believe that, within the in-world access layer, one way to bring about CVE privacy and integrity controls improvements is through a simplified, yet wide-ranging access control solution. Contrary to what is found in many commercial and open source CVEs, access control should be streamlined and ubiquitous so as to enable correct, consistent application across all virtual assets.

Our approach to securing the in-world access layer follows that of classic, UNIX-style, file system discretionary access control (DAC). We imitate this type of DAC primarily because it is well-known in many general operating systems, easy to comprehend, and easy to deploy. We avoid more complex access control list systems, such as that found in Microsoft Windows XP/2000/2003, because their complexity can lead to misconfiguration [54].

The archetypal, UNIX-style DAC is based upon the assignment of permissions to three roles: the owner of a file system object, the group to which that object
belongs, and all other system users. This basic DAC employs read, write, and execute permissions to permit or deny access to a directory or file; we propose something analogous though even simpler. The tenets of our system may be concisely stated as follows.

- Access controls must always be inherently simple to understand and manage.

- The CVE’s virtual world should be viewed by access controls as a collection of objects: terrain, avatars, decor, accessories, media (i.e., sound, video, still image), etc.

- Access should be managed through two different permissions: interact and alter.

- All virtual world objects must have an assigned owner and participant group; a default other role includes all participants who are neither the owner nor in an object’s group.

- Interact and alter permissions should be specified for an object’s group and all other participants—the owner of an object will always have full access to the object.

- Any participant should be able to determine an object’s ownership;\(^1\) the owner of an object should be able to set that object’s permissions, and may transfer ownership to a different participant.

Semantics of the interact and alter permissions are comparable to the classic read, write, and execute file system capabilities. Interact denotes read and execute, while alter is equivalent to write. In a VR context, interacting with an object entails viewing/hearing (“reading”) and possibly using (“executing”), while

\(^1\)Unrestricted access to ownership information may raise privacy concerns. However, we believe the availability of owner IDs is in keeping with the classic, UNIX style of file system access control. Moreover, the benefit of making these IDs private is outweighed by the cost of increased system complexity (potentially leading to confusion and access control misconfiguration).
altering an object involves changing, deleting, or updating in some way ("writing"). We considered splitting the interact permission into observe and interact, where observe would be equivalent to read and interact would be equivalent execute, but this seemed unnecessary: a participant should probably not interact with an object they cannot also observe, and if a participant is allowed to observe an object, there is probably an intention that they also have interact capabilities. Thus, collapsing observe and interact into just interact has merit for our purposes. Along with the roles of owner, group, and other, we believe the interact and alter permissions are sufficient to handle nearly all CVE access control situations. With any fewer permissions there is a significant loss of utility; more permissions, on the other hand, entail a risk of complexity, which may lead to improper access control configuration.

We have implemented a prototype of this DAC system, called WonderDAC, by extending Sun Microsystems’ Project Wonderland version 0.3. We selected this CVE because of three important factors: first, Wonderland is at an early, formative stage, making it easy to integrate the components of our DAC system; next, Wonderland’s approach to handling VR objects (called cells in Wonderland parlance) maps very well to the tenets we state above; and, finally, Wonderland’s server component offers a logical, secure place to implement access controls [148].

The remainder of this chapter is organized as follows: the next section summarizes related works, Section 4.3 presents background and details about Project Wonderland and its core architecture, Section 4.4 outlines critical use case scenarios the WonderDAC prototype should enable, Section 4.5 illustrates our prototype implementation, and Section 4.6 ends the chapter with summary remarks and a brief discussion of further research and development.
4.3 Related Work

As discussed in the previous section, we find there to be three basic layers in which CVE security controls should be implemented: network communications, in-world access, and user experience. Protecting the network communications layer has been a carefully studied problem in many other research endeavors, the outcome of which is applicable to CVEs or any other systems that transmit data using network protocols. For example, among other techniques, [43, 51, 116] all propose encrypting the network protocols that underly their respective CVEs. Other network research peculiar to CVEs has touched on the integrity of the gaming experience. Along these lines, [12, 30, 102] each consider how to provide consistent and fair participant interactions through methods such as dead-reckoning and bandwidth reduction.\(^2\)

Regarding the middle CVE layer, in-world access, we find a limited number of efforts that deal with security controls. For example, [25] derives a spatial access control scheme, called SPACE, and notes that through SPACE’s adoption “a natural part of the environment is exploited, making it possible to hide explicit security mechanisms from end users through the natural spatial makeup of the environment.” SPACE works by leveraging an access graph (a structure denoting constraints when moving from one location to another) to build adjacency and classification matrices. The adjacency matrix serves as a map, while the classification matrix determines the permissions a participant needs in order to move about a CVE. In [108] a more refined, but complex, approach to access control is

\(^2\)Dead-reckoning is a widely used method to improve the appearance of movement among participants and other objects in a virtual reality context. In essence, the position of a moving object is predicted based on it’s velocity and previous location. Updated information about the object’s position is only required if it differs substantially from the prediction.
proposed, wherein CVE participants can interact with VR objects and exchange messages with one and other in order to gain access to resources. Central in this approach is the use of keys by participants to prove their identities and authority when undertaking some action.

Other means of in-world access control exploit VR metaphors. In [27], so-called privacy lamps and vampire mirrors are utilized to affect and verify object access. Privacy lamps shine a spot light on virtual objects that are to be hidden from view, while vampire mirrors reveal objects that are private (such objects won’t show up in the mirror) and can make an object private when a participant touches its mirror image.

Finally, some commercial, social networking CVEs offer in-world access controls, although these are typically geared toward the ownership, sale, and rental of virtual property. Second Life, for example, has extensive controls dealing with these issues, as do There, and Activeworlds [1, 2, 4, 81–83, 91–93]. The concept of role-based access (through a group) is also available in Second Life relative to land management. All three CVEs provide for other access control mechanisms that are tailored to the social networking experience (e.g., permissions to restrict flying and building). Often such controls require expertise with the CVE in question in order to be correctly applied and managed.

In the top CVE layer, user experience, consideration has been given to how structural divisions and social conventions play a role in participant behavior and privacy. Each of [59], [64], and [63] explores how the design of virtual spaces can enable or discourage privacy by controlling views into work areas. The intersection of technical and social components in media environments (such as CVEs) is also examined by [99], wherein it is noted that successful virtual communities must
adapt and evolve practices and conventions from the real world. In so doing, visual cues (e.g., the layout of a space, signs) can affect participant behavior and the respect for others’ virtual world privacy. Along these lines, [114] notes that the privacy of a CVE participant (i.e., the right of that participant not to be hounded by other users in the CVE) may sometimes be an issue, too. Unfortunately, resetting a participant’s virtual identity (their name and avatar appearance) may disrupt associations with friends or fail to work if the new identity becomes linked to the old one.

4.4 Project Wonderland

4.4.1 Background

Project Wonderland is an open source, Java-based CVE that is constructed around a client-server model. Developed primarily by Sun Microsystems, Wonderland draws upon the technologies of four other open source projects (also operated by Sun):

- Project Darkstar: the server component for Wonderland; designed to offer a scalable, distributed, transaction-based, game server infrastructure

- jVoiceBridge: the audio server and client for Wonderland; provides spatial, stereo sound, as well as the ability to place telephone calls through the Session Initiation Protocol (SIP)

- Java 3D: responsible for providing the Wonderland client’s 3D graphics and scene graph

- Project Looking Glass: used by the Wonderland client for 3D scene management (e.g., to create the client’s user interface)

Compared to similar technologies, such as Croquet or Second Life, Wonderland is a relatively new offering in the realm of CVEs. First available in January 2007,
this system incorporates a significant set of features, including, but not limited to: a scalable, virtual environment specified through a simple, XML file hierarchy; a transaction-based, fault-tolerant server that enables atomic participant activities; built-in avatar support (including a simple editor); spatial sound and audio chat; a virtual whiteboard; the ability to operate and share X Window applications within the Wonderland virtual world; and the means to connect a virtual phone with any SIP compliant phone system. The impetus for these features is a result of Sun’s vision that Wonderland be “an environment that is robust enough in terms of security, scalability, reliability, and functionality that organizations can rely on it as a place to conduct real business [134].” These goals are meant to encompass remote office, educational, and other types of multi-user collaboration.

4.4.2 Client-Server versus Peer-to-Peer

Our ability to extend Wonderland with a DAC mechanism is enabled by Wonderland’s client-server architecture. The client-server approach gives us a central authority, in the form of a server, that can be protected from malicious parties. By contrast, a peer-to-peer environment complicates the use of security controls, since each participant is on equal footing with full access to resources and information; serious problems can arise if one or more peers is under the control of a malicious user. As discussed in [102], with regards to cheating in a peer-to-peer CVE-based game, numerous obstacles and issues must be resolved to enable reliable security controls:

First, it may be very complicated to apply cheating resistance extensively to a whole peer-to-peer architecture, that is mostly made of uncontrolled hosts. Moreover, it is unclear where to place the mechanisms to be used: on all the devices that take part to the game, or only on a subset of them? In this latter case how is this subset se-
lected? Aside from this, in the case of a fully decentralized approach, the management of trust between the devices and between the players is a complex issue. How can we be sure that a device or a player acts faithfully? How is trust established and maintained? How is slandering handled?

Endpoint trust, as one of the obstacles just touched upon, is an area of significant concern. In a client-server environment, clients must determine whether or not to trust servers and vice-versa; in a peer-to-peer environment, peers must determine whether or not to trust each other. Although we recognize the importance of endpoint trust in helping to establish access control, we believe it to be a matter that resides at the network communications layer, and, thus, is outside our stated scope of interest. Nevertheless, the area of public key infrastructure (PKI) offers well-studied solutions for client-server architectures. Peer-to-peer schemes also exist (e.g., [18] and [130]), but entail more complexity to compensate for the lack of a central authority.

4.4.3 Wonderland Cells and the Wonderland File System

Our implementation of the WonderDAC prototype takes place only in Wonderland’s server components. However, as we will demonstrate in Section 4.6, there is still a need to enhance the client-side code in order to hide some of the cosmetic side-effects of WonderDAC. As mentioned above in Section 4.1, a Wonderland “...virtual world is composed of a collection of ’cells’, each of which represents a 3D volume in the world [132].” A cell can denote a single decorative object in a virtual room, the room’s architecture, or the entire virtual world. Hence, there is a hierarchical, parent-child structure at work in the use of cells, where a given cell may contain zero or more other cells. Moreover, this structure is represented by the XML files that prescribe the layout of a Wonderland virtual world and
make up the Wonderland file system (WFS) [133]. Figure 4.2 provides an example WFS hierarchy comprising a simplistic world, while Appendix B includes the XML source code for these files. Each XML file is actually the serialization of a Java bean (called BasicCellGLOSetup) that handles the fundamentals of creating a generic cell. At the root of any WFS is a directory with a name ending in the -wfs suffix; within this directory XML files that end with a -wlc.xml suffix correspond to individual cells. It is possible to load up a cell file with an array of virtual object models: in this case, all objects are constituents of the cell. Alternatively, one can establish a parent-child relationship by creating a directory named after a given cell, but with a -wld suffix, and then populating that directory with new cell files (this is seen in Figure 4.2 where testRoomA is the parent of the cell given by Whiteboard-wlc.xml. All cells have virtual world coordinates that are relative to their parents’ coordinates.
4.4.4 Communications

Two forms of communication are employed within the Wonderland client-server architecture: direct and publish/subscribe channels [132]. The former is primarily used in activities such as a client’s initial connection to the Wonderland server, error message transmission, the management of shared applications, and handling virtual phone calls. The latter facilitates communications between clients and the server, as well as among clients (by way of the server). In the publish/subscribe channels, communications involve participants entering and leaving the Wonderland environment, cell management, avatar setup and movement, and server management (through a dedicated server management client).

The driving force behind using a combination of direct and publish/subscribe communications is efficiency: some situations call for the transmission of data between a given client and the server, while other situations may not involve the server in a logical sense. For example, if a participant enters or leaves the Wonderland environment, the server must be notified directly; if a participant causes a cosmetic change that only they and other nearby participants should see (e.g., a virtual object lights up when a mouse hovers over it) there is no need to update information on the server. In other words, there can be state changes on a client, and state changes on the server—figures 4.3 and 4.4 illustrate this. In Figure 4.3, a server state change has taken place (step 1) as a result of some activity by Client A. In this circumstance, Client A directly contacts the server with an appropriate message (step 2); Client B is then updated by the server over a channel to which Client B subscribes (step 3). In Figure 4.4, however, a cosmetic change has taken place within a cell that Client A is accessing (step 1). Since there is no corresponding change in server state for the cell, the cell
publishes an update to its channel (step 2a), which is subscribed to by Client B (step 2b).

4.5 Use Case Scenarios

In Section 4.1, our list of tenets for the WonderDAC prototype started with the goal of simplicity and ease of understanding. We bear this in mind as we now consider five representative use case scenarios for CVE access control.

Our general approach with WonderDAC is modeled after the classic, UNIX-style DAC system, and, for a given VR object, works by assigning access permissions to three roles: owner, group, and other. As opposed to read, write, and execute, WonderDAC employs the VR analogs of interact (similar to read/execute) and alter (similar to write). By having the interact permission for an object,
one can use, but not change, that object. By having the `alter` permission, one can change (i.e., update or delete) the object. As a participant operates within Wonderland, the privileges they are assigned for a given object are mutually exclusive: if the participant is a member of an object’s group, then they will be assigned only the object’s group permissions; permissions for an object’s other role are assigned to a participant when the `group` role cannot be applied. This means that an object’s `group` role can be used to grant or deny access to a select list of participants.

Not all of the following use cases have been implemented in the Wonderland prototype. Instead, we have focused on very basic components for use cases 1 and 2, laying the foundation for handling the remaining use cases in future work (see Section 4.6).
4.5.1 Use Case 1: Spatial Object Restrictions

A participant should only be able to enter, hear, and see into a virtual space if they are the owner of the space, a member of the space’s group (and the group has interact permission), or the space is defined with interact permission for the other role. When a participant’s avatar has entered a space, the avatar should only be visible and heard by other participants who also have interact permission (but may or may not be inside the space).

A participant should only be able to change a virtual space (by speaking in, or deleting/updating the space) if they are the owner of the space, a member of the space’s group (and the group has alter permission), or the space is defined with alter permission for the other role. Although it is possible for a space to have only the alter permission assigned to the group or other roles, the result of such a configuration would be meaningless, since, to begin with, non-owners would be incapable of entering the space.

4.5.2 Use Case 2: Non-spatial Object Restrictions

A participant should only be able to view and, if applicable, hear a non-spatial object (e.g., a whiteboard, phone, or X Window application) if they are the owner of the object, a member of the object’s group (and the group has interact permission), or the object is defined with interact permission for the other role. For example, a participant with only interact permission for a whiteboard can view the whiteboard and its contents, but cannot change or add to those contents. Similarly, a participant with only interact permission for a conference phone, can view and hear the phone, but cannot join in the conversation.

A participant should only be able to change/utilize a non-spatial object if they
are the owner of the object, a member of the object’s group (and the group has
*alter* permission), or the object is defined with *alter* permission for the *other* role.
Similar to Use Case 1, although it is possible for a non-spatial object to have only
the *alter* permission assigned to the *group* and *other* roles, such a configuration
is meaningless since non-owners would be unable to view/hear the object in the
first place.

4.5.3 Use Case 3: Audio Conversation Restrictions

Two or more participants engaged in audio chat should be able to restrict
their conversation to themselves (somewhat like having a “cone of silence” at
their disposal). Other participants should be able to solicit the initial group to
join in the conversation if the group wishes. This amounts to a specialized version
of Use Case 1 except that an ephemeral, invisible space is created around the
participants:

1. A temporary group role, to which all of the conversation participants belong,
is created and assigned to the invisible space

2. *Interact* and *alter* permissions are assigned to the space’s group role

3. New participants may be added to the temporary group role as desired by
the initial conversation participants

It should be possible to assign/remove the *interact* and *alter* permissions for
the invisible space’s *other* role. This could enable, through the assignment of only
the *interact* permission, members of the temporary group to converse, while all
others can listen but not interrupt. The use of *alter* permissions alone for the
invisible space’s *other* role may or may not make sense, depending on the desires
of the conversation participants: a participant in the *other* role would be able to
speak in, but not hear, the conversation.

It is noteworthy that this use case addresses privacy and integrity concerns
explored in other SIP/VR research. For example, [6] and [32] discuss the use
of narrowcasting within SIP environments to implement granular restrictions on
audio conference participants. Of particular interest is how [6] employs a limited,
Java3D-based CVE to configure, through narrowcasting, the speaking and hearing
attributes of participants represented by avatars. Another example is found in
[15] where virtual analogs for the social components of in-person communication
are leveraged. Here, abstractions such as aura, awareness, and adapter are used
to facilitate the ability of participants to speak with one and other. Similarity
with this use case may be found in our employment of access-controlled space:
somewhat akin to a combination of aura (a subspace that bounds participant
interactions) and adapter (a mechanism that can modify aura and awareness) to
protect a conversation. The means to control which conversants can hear and/or
speak, and to whom they may listen and/or speak is an important tool—not
only for security, but for improving the effectiveness of a conference with many
participants.

4.5.4 Use Case 4: Avatar Cloaking

A participant should be able to disguise their avatar’s appearance. Each avatar
should have a *group* role that is unique to its owner. By assigning or removing
*interact* permissions to/from this group or to/from the avatar’s *other* role, the
appearance of the avatar should be controllable by its owner. A participant should
be able to see an avatar’s true image if they are a member of the avatar’s group
(and the group has interact permission), or the avatar is defined with interact permission for the other role. Otherwise, a generic image should be displayed by the avatar. The use of alter permissions should be ignored, here: only the owner should be permitted to make changes to an avatar’s appearance. This approach helps to manage any risk of avatar defacement by deferring to the owner to facilitate all avatar image changes.

4.5.5 Use Case 5: Permissions and Ownership Changes

All participants should be able to determine the ownership of a VR object. Also, a participant should be able to change permissions for the group and other roles of any object they own, or, if desired, assign ownership of the object to another participant. The interface to carry out these activities should be simple and accessible through mouse interactions with the object in question.

4.5.6 Summary Use case Table

Table 4.1 presents a summary of the use case scenarios discussed above, along with the semantics behind permissions settings.
<table>
<thead>
<tr>
<th>Use Case</th>
<th>Permissions</th>
<th>Meaning for Group Role</th>
<th>Meaning for Other Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spatial Object</td>
<td>I A</td>
<td>Any member of the object’s group can enter, see, and hear into the space. All such constituents can speak within and change the space, too.</td>
<td>All participants who are not members of the object’s group can enter, see, and hear into the space. All such constituents can speak within and change the space, too.</td>
</tr>
<tr>
<td>I -</td>
<td>Any member of the object’s group can enter, see, and hear into the space. All such constituents are unable to speak within and change the space.</td>
<td>All participants who are not members of the object’s group can enter, see, and hear into the space. All such constituents are unable to speak within and change the space.</td>
<td></td>
</tr>
<tr>
<td>- A</td>
<td>Meaningless: a member of the object’s group can speak within and change the space, but they are unable to enter the space.</td>
<td>Meaningless: all participants who are not members of the object’s group can speak within and change the space, but they are unable to enter the space.</td>
<td></td>
</tr>
<tr>
<td>Use Case</td>
<td>Permissions</td>
<td>Meaning for Group Role</td>
<td>Meaning for Other Role</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
<td>---------------------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>- -</td>
<td></td>
<td>No member of the object’s group can enter, see, or hear into the space. All such constituents are unable to speak within and change the space.</td>
<td>All participants who are not members of the object’s group cannot enter, see, or hear into the space. All such constituents are unable to speak within and change the space.</td>
</tr>
<tr>
<td>Non-spatial Object</td>
<td>I A</td>
<td>Any member of the object’s group can see and, if applicable, hear the object. All such constituents can change the object, too.</td>
<td>All participants who are not members of the object’s group can see and, if applicable, hear the object. All such constituents can change the object, too.</td>
</tr>
<tr>
<td>I -</td>
<td></td>
<td>Any member of the object’s group can see and, if applicable, hear the object. All such constituents are unable to change the object.</td>
<td>All participants who are not members of the object’s group can see and, if applicable, hear the object. All such constituents are unable to change the object.</td>
</tr>
<tr>
<td>- A</td>
<td></td>
<td>Meaningless: a member of the object’s group can change the object, but they are unable to see or, if applicable, hear the object.</td>
<td>Meaningless: all participants who are not members of the object’s group can change the object, but they are unable to see or, if applicable, hear the object.</td>
</tr>
</tbody>
</table>
**TABLE 4.1**

*Continued*

<table>
<thead>
<tr>
<th>Use Case</th>
<th>Permissions</th>
<th>Meaning for Group Role</th>
<th>Meaning for Other Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>- -</td>
<td>-</td>
<td>No member of the object’s group can see, change, or, if applicable, hear the object.</td>
<td>All participants who are not members of the object’s group cannot enter, see, change, or, if applicable, hear the object.</td>
</tr>
<tr>
<td>Audio Conversation</td>
<td>I A</td>
<td>All member of the conversation’s temporary group can hear and speak in the conversation.</td>
<td>All participants who are not members of the conversation’s temporary group can hear and speak in the conversation.</td>
</tr>
<tr>
<td>I -</td>
<td>Not allowed: all members of the conversation’s temporary group can hear the conversation, but not speak. This would prevent the conversation from taking place.</td>
<td>All participants who are not members of the conversation’s temporary group can hear, but not speak in, the conversation.</td>
<td></td>
</tr>
<tr>
<td>Use Case</td>
<td>Permissions</td>
<td>Meaning for Group Role</td>
<td>Meaning for Other Role</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
<td>--------------------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>- A</td>
<td>Not allowed: all members of the conversation’s temporary group can speak in, but not hear the conversation. This would prevent the conversation from taking place.</td>
<td>All participants who are not members of the conversation’s temporary group can speak in, but not hear, the conversation. This may or may not be useful, contingent on the desires of the conversation’s temporary group members.</td>
<td></td>
</tr>
<tr>
<td>- -</td>
<td>Not allowed: no member of the conversation’s temporary group can speak in or hear the conversation. This would prevent the conversation from taking place.</td>
<td>All participants who are not members of the object’s group cannot speak in or hear the conversation.</td>
<td></td>
</tr>
<tr>
<td>Use Case</td>
<td>Permissions</td>
<td>Meaning for Group Role</td>
<td>Meaning for Other Role</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td>------------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>I -</td>
<td>Any member of the object’s group can see the true appearance of the avatar. (Effectively the same as I A permissions above)</td>
<td>All participants who are not members of the avatar’s group can see the true appearance of the avatar. (Effectively the same as I A permissions above)</td>
<td></td>
</tr>
<tr>
<td>- A</td>
<td>No member of the avatar’s group sees the avatar’s true image; instead, they see a generic disguise. The alter permission is ignored: only the owner is allowed to change an avatar.</td>
<td>All participants who are not members of the avatar’s group cannot see the avatar’s true image; instead, they see a generic disguise. The alter permission is ignored: only the owner is allowed to change an avatar.</td>
<td></td>
</tr>
<tr>
<td>- -</td>
<td>No member of the avatar’s group sees the avatar’s true image; instead, they see a generic disguise. (Effectively the same as - A permissions above)</td>
<td>All participants who are not members of the avatar’s group cannot see the avatar’s true image; instead, they see a generic disguise. (Effectively the same as - A permissions above)</td>
<td></td>
</tr>
</tbody>
</table>
4.6 Prototype Implementation

As discussed in Section 4.5, our focus during the implementation of this prototype was on very basic components for use cases 1 and 2. To begin with, we leveraged Wonderland’s ability to use the lightweight directory access protocol (LDAP) as the means of authenticating a participant.\textsuperscript{3} We extended this to include authorization by storing a group membership list in each participant’s LDAP record. When a participant logged into Wonderland to authenticate, their membership list was accessed and parsed into an array of group names, which was then stored in an associated WonderlandIdentity object. Due to constraints in Darkstar, we had to implement a special Darkstar service to enable access to a participant’s identity information during the login process.

Next, we took advantage of a pre-existing, empty, class stub that Wonderland developers had inserted into their source code as a placeholder for future access control checks. We replaced this stub (called CellAccessControl) with a new class that could compare a given cell’s group with the group membership of a participant, and then, based on the outcome, perform subsequent permissions checks. This new class put into effect the mutually exclusive assignment of permissions discussed at the outset of Section 4.5: “...if the participant is a member of an object’s group, then they will be assigned only the object’s group permissions; permissions for an object’s other role are assigned to a participant when the group role cannot be applied.”

Finally, we made modifications to other central Wonderland classes. This included, but was not limited to: CellGLO, the parent class for all Wonderland cells;

\textsuperscript{3}This ability also includes support for strongly encrypted communications between the LDAP and Wonderland servers.
BasicCellGLOSetup, a utility class to aid in configuring a cell; and Whiteboard-CellGLO, the server component of a Wonderland whiteboard cell. Our modifications to BasicCellGLOSetup enabled us to add the following property tags to the XML files that specify Wonderland cells:

- **accessOwner** – The name of the cell’s owner
- **accessGroup** – The name of the cell’s group
- **accessGroupPermissions** – A number indicating the interact and alter permissions assigned to the cell’s group (“I A” = 3, “I -” = 2, “- A” = 1, “- -” = 0)
- **accessOtherPermissions** – A number indicating the interact and alter permissions assigned to the cell’s other role

All Java source code modifications and contributions are included in Appendix C.

4.6.1 Spatial Object Access

Here, we demonstrate how the WonderDAC prototype was able to control spatial access. For this situation, there were two cells (spaces) of interest: a two-story room referred to as testRoomC (see Appendix B, Section B.4), and, within this space, a second story loft with stairs leading up, referred to as testRoomD (see Appendix B, Section B.5). For the purposes of this demonstration, testRoomC was configured as follows: (twright, admin, 2, 2). This tuple is shorthand for denoting that the cell’s owner is twright, the cell’s group is admin, the group permissions are I -, and the permissions for the other role are I -. The loft, testRoomD, was configured with: (twright, admin, 0, 0), where a 0 denotes no permissions granted. In Figure 4.5, we see two avatars, twright and bench-40,
Figure 4.5. The avatar twright looks on at testRoomD and testRoomC, looking at the spaces created by testRoomC and testRoomD; this figure is from twright’s perspective. Because twright is the owner of both cells, he is able to see both rooms. Figure 4.6, however, is from bench-40’s perspective, and does not include the presence of testRoomD. This is because bench-40 is a guest user who does not belong to the admin group, and testRoomD permits neither interact nor alter access for its group or the other roles.

Although the bench-40 participant is unable to view testRoomD, our current WonderDAC prototype stops short of concealing from them the avatars that enter testRoomD. Such concealment is required to fully implement Use Case 1, and is planned future work. Figure 4.7 depicts what happens when twright enters into
Figure 4.6. The avatar bench-40 looks on at testRoomC, but cannot see testRoomD.
Figure 4.7. The avatar bench-40 looks in the direction of testRoomD as twright climbs the stairs.

the testRoomD cell as bench-40 (an avatar that cannot enter this space) watches. The same situation is pictured from twright’s point of view in Figure 4.8; again, twright is the owner of the testRoomD cell.

4.6.2 Non-spatial Object Access

To demonstrate non-spatial object access, we created a whiteboard cell in our test Wonderland environment (see Appendix B, Section B.2) and configured it in the same manner as testRoomC: (twright, admin, 2, 2). Both the twright and bench-40 participants were able to see the whiteboard object, including im-
Figure 4.8. The avatar twright looks back at the avatar bench-40 from testRoomD.
ages drawn there. However, only twright was supposed to be able to change the contents, since, as the owner, twright was the only participant with alter permissions for the whiteboard object. This proved to be an interesting situation for Wonderland due to its use of publish/subscribe channels. In particular, when a participant uses a whiteboard, they are operating on a local copy maintained by their Wonderland client. Periodically, their client publishes updates of the whiteboard’s state so that other clients observing/participating with the whiteboard may stay synchronized. If a participant lacks interact permission for the whiteboard they will, of course, be unable to see or use it in any way. If they lack just alter permission, they will still be able to draw on their local copy, even though the Wonderland server will ignore any whiteboard updates their client tries to send. Hence, there is a cosmetic issue: even though no other participants will see an unauthorized participant’s whiteboard updates, the unauthorized participant can continue to use their whiteboard locally. Figures 4.9 and 4.10 illustrate this happening with twright and bench-40.

As an authorized participant uses a whiteboard, their updates will be added to the local copies of all participants—some of whom may not have alter permission, but have drawn on their local whiteboards nonetheless. Figures 4.11 and 4.12 depict this situation.

The place to fix this issue is at the Wonderland client. Because there is no security risk involved (just the nuisance of a cluttered whiteboard for some participants), enhancing the client to block whiteboard use when a participant has no alter permission should be a reasonable solution. A complete implementation of Use Case 2 requires such an enhancement, and is planned for future work. We do not believe a similar circumstance exists for Wonderland’s virtual phone.
Figure 4.9. The bench-40 participant draws on a local copy of the whiteboard shared with twright.
Figure 4.10. Because bench-40 has no *alter* permission, the twright participant cannot see bench-40’s whiteboard updates.
Figure 4.11. The twright participant has *alter* permission and draws on the whiteboard shared with bench-40.
Figure 4.12. The final state of the whiteboard from bench-40’s perspective.
4.7 Conclusion and Future Work

As CVEs continue their evolution into more ubiquitous, useful technologies, we find them enabling entertainment, commercial, educational, and scientific endeavors. A critical component to the success of any collaborative technology is the control of access to information and resources. For most CVEs (commercial and open source) such control is often limited in scope and not designed to fully leverage the assignment of roles to a participant. In answer to this, we have implemented a limited prototype called WonderDAC: a discretionary access control system for the Project Wonderland CVE. We have designed WonderDAC not only to be an effective means of access control, but also to serve as a simple, ubiquitous mechanism that may be consistently applied by CVE participants. We believe this contrasts with other CVEs (both commercial and open source) where discretionary access controls may be narrowly focussed, complex, and/or significantly lacking in capabilities.

WonderDAC is modeled after a classic, UNIX-style DAC system, and treats all virtual objects according to three roles: owner, group, and other. Interact and alter permissions are assigned to, or removed from the group and other roles to control access by CVE participants, while an object’s owner always maintains full privileges. The interact permission is analogous to read/execute in a UNIX-style DAC system, while alter is similar to write. By defining five representative use case scenarios for managing CVE access control, we formulated a guide for our prototype implementation.

Although our prototype is limited to just the first two use cases (access control for spatial and non-spatial objects), it provides an important foundation for future work. We propose undertaking additional research to expand upon WonderDAC,
including short-term and long-term objectives. The short-term work deals with cosmetic issues raised by the prototype: hiding avatars that reside in privileged spaces from the view of unauthorized participants, and modifying the Wonderland client to prevent unauthorized changes to a participant’s local whiteboard and other virtual objects. The long-term work addresses the remaining use case scenarios and includes building an interface for participants to view and manage object ownership and permissions information.
CHAPTER 5

WONDERDAC: AN IMPLEMENTATION OF DISCRETIONARY ACCESS CONTROLS WITHIN THE PROJECT WONDERLAND CVE *

5.1 Abstract

Both proprietary and open source collaborative, virtual environments (CVEs) offer extensive means for social, scientific, and commercial collaboration. To safeguard the objects and activities that comprise these virtual worlds, in the previous chapter we propose an in-world discretionary access control (DAC) system called WonderDAC. The simplicity and role-based nature of this system have already been demonstrated through a limited prototype. In this chapter, we continue our work by expanding upon the WonderDAC prototype to build a more complete and functional DAC solution. Although integrated with Project Wonderland, WonderDAC's concepts and design could be adapted to any similar CVE platform.

* Material from this chapter is currently under review by the Journal of Virtual Reality and Broadcasting, and appears as a technical report at the University of Notre Dame: Wright, T. and Madey, G., Wonderdac: An implementation of discretionary access controls within the project wonderland cve, Technical Report TR 2008-15, University of Notre Dame, College of Engineering, University of Notre Dame, Notre Dame, IN 46556, November 2008.
5.2 Introduction

We begin with a reference to the previous chapter, wherein we define a collaborative, virtual environment (CVE) as a “graphical, multi-user, virtual reality environment capable of operating on a typical, modern workstation.” A few well-known, proprietary CVEs include Second Life, There, and Activeworlds, while Project Wonderland (also referred to as Wonderland) and Croquet are among the more popular open source CVEs. Typical uses for these systems range from socializing, education, and commerce, to game playing and scientific research.

Despite their flexibility, most CVE platforms lack in-world access controls that are simple in design, yet effective and encompassing in functionality. Proprietary systems, for example, offer extensive means for their subscribers to own, rent, and sell virtual land [1, 2, 4, 81–83, 91–93], but such controls are more limited with regard to other types of objects (e.g., of a non-spatial nature). Also, the user interface to configure and manage access controls in these CVEs can be complicated, requiring a certain level of participant expertise. For open source platforms, such as Croquet and Wonderland, access controls are generally relegated to protecting the user login activity alone.

To answer to these deficiencies, the previous chapter proposes the creation of a discretionary access control (DAC) system specifically engineered for CVEs. This system is intended to be easy to use and capable of protecting any object found in a CVE, including, but not limited to, three-dimensional spaces and assets, still images, videos, sounds, and avatars. In this earlier work, we identify three CVE layers (see Figure 5.1) where security controls should be applied, with the top two of these (in-world access and user experience) subject to any DAC system. In addition to protecting these two layers, the simplicity and ubiquity of our approach
is intended to ensure that a participant willingly and correctly employs access controls. Regarding this, our previous research devises five use case scenarios to capture the essence of what our DAC system should be capable, and then partially implement the first two scenarios in the form of a limited prototype. Restated here, for convenience, are the five use cases:

- Spatial Object Restrictions—access control for three dimensional spaces
- Non-spatial Object Restrictions—access control for three and two dimensional assets, as well as images, videos, and sound (excluding audio chats)
- Audio Conversation Restrictions—access control for audio chats
- Avatar Cloaking—access control for an avatar’s image (i.e., permit/deny others to see and interact with one’s avatar)
- Permissions and Ownership Changes—user interface to effectively manage roles and permissions for all objects
Our DAC prototype is implemented within Project Wonderland 0.4, an open source, Java-based CVE maintained by Sun Microsystems. There are two compelling reasons behind this choice. First, Wonderland utilizes a client-server model, which enables the server end to act as an authority for the management of access controls and protected resources. This is in contrast to a peer-to-peer model, such as Croquet’s, where each peer has a copy of the same data as every other peer. Under the peer-to-peer model, a malicious user could simply attack their own workstation to obtain unauthorized access to information and resources. Because of this, safeguarding such an environment is an inherently complicated problem [102]. Second, Wonderland is at a very early stage in its development life-cycle, making it relatively easy to integrate a DAC system of the robustness we intend. Resulting from our use of Wonderland, the name our prototype, WonderDAC, is a shortened form of Wonderland Discretionary Access Controls.

Although available only since January 2007, Wonderland is founded atop other, more mature open source technologies and includes a variety of important features [134]. The most significant example of this is found in Wonderland’s use of the Project Darkstar game server, which provides a transaction-based virtual environment that is also highly scalable and fault-tolerant. Another technology, jVoice-Bridge, equips Wonderland with spatial, stereo sound and the ability to place telephone calls through the Session Initiation Protocol (SIP) to the outside world. Completing the list, Wonderland utilizes Java3D and Project Looking Glass to handle its 3D visualizations.

By evolving the WonderDAC prototype into a full implementation, we demonstrate in this chapter that an easy, encompassing approach to CVE access control is viable.
The remainder of this chapter is organized as follows: the next section summarizes related works; Section 5.4 looks at the basics of how Wonderland and WonderDAC operate; Section 5.5 considers the role that network communications security plays relative to WonderDAC, Section 5.6 presents several examples of WonderDAC in use; and Section 5.7 ends the chapter with summary remarks and a consideration of future work.

5.3 Related Work

Though limited in number, efforts to manage access control for CVEs span a range of approaches that cover the in-world access and user experience layers from Figure 5.1. For example, a scheme based on the movement of a user’s avatar assigns permissions to spaces through a so-called access graph [25]. Two other approaches exploit VR metaphors: one by using specific virtual objects to enable and manage privacy [27], and another by leveraging structural divisions and social conventions to guide participant behavior and privacy [59, 63, 64, 99]. Still other means of access control operate through message passing (to gain access to resources) [108], embodied interaction (where the state of an object and the avatar, along with the programmed functionality of the virtual environment determine access) [39], and simple, static authorization granted at login [88, 89].

Subscription-based CVEs including Second Life, There, and Activeworlds generally offer an access control list (ACL) approach to securing in-world resources. As was noted in Section 5.2 above, such controls are full-featured regarding the management of virtual land, but can often be complex to apply and limited with regard to other VR assets. Although WonderDAC is also ACL-based, it differs significantly from the access control found in commercial CVEs as a result of its
simplicity and homogeneous application to virtual objects.

A noteworthy alternative to the in-world access control solutions discussed here is object-capability security [118]. An object capability may be thought of as a key that can open a lock: the possession of an object’s capability allows a user to interact with that object in some specific manner. Capabilities can enable discretionary access control by being exchanged, assigned, and removed as users see fit. Object-capability security attempts to resolve some of the problems and shortcomings of traditional ACL solutions, for example: ACL-based security is server-bound and, thus, a potential bottleneck, participants cannot react dynamically to ACL changes, authentication is required for ACL solutions, and ACLs cannot operate at a fine-grained level without incurring noticeable overhead (related to the bottleneck issue).

We agree that object-capability security holds promise and should be explored further. However, we also believe that WonderDAC’s use of ACLs remains practical and viable. First, although ACL-based security is server-bound, this does not necessarily make it a bottleneck. Wonderland uses the Darkstar game server in which “work is broken into small units, each corresponding to one action in a game, and each of which can be processed anywhere on the server-side network. Individual actions are then automatically distributed across all available threads and any number of server machines for parallel execution [131].” Next, in the case of WonderDAC, changes to the ACLs of any VR object (a “cell” in Wonderland) have an immediate, dynamic affect on nearby users and the environment. For example, if a participant loses or gains permissions to interact with a cell, they will immediately see the cell disappear or materialize, respectively. Regarding the need for authentication in ACL solutions, object-capability security must grapple
with forged and re-played capabilities, that is: a malicious party could generate fake capabilities or monitor communications and steal copies of capabilities for later use. Encryption and digital signatures offer a viable way to deal with these issues, but require that some sort of public key infrastructure be established and maintained. Not only is this solution authentication-based, it also leads to the overhead and central authority that object-capability security hopes to avoid. Finally, on the point of fine-grained operation, WonderDAC is applicable at the level of a Wonderland cell, where a cell can be a single, 3D virtual asset, or a massive hierarchy of other cells. As a result, WonderDAC applies equally to the smallest and largest dimensions of Wonderland.

5.4 WonderDAC Basics

5.4.1 System Architecture

Our discussion of WonderDAC begins with a high-level view of Wonderland’s system architecture. Figure 5.2 shows us a diagram of components that include:

- Wonderland Server – Provides the underlying CVE core mechanisms; WonderDAC is implemented, here

- jVoiceBridge – Handles Wonderland’s sound content, including audio chat

- LDAP Server – Maintains all participant and system accounts, including related DAC information

- Web Server – Stores and serves model and texture files to Wonderland clients via hypertext transfer protocol (HTTP); any standard Web server can be used

- Client – The client application that enables a participant to interact with Wonderland
Figure 5.2. A typical Wonderland architecture configuration.

It is noteworthy that Wonderland’s design enables its major services to be distributed among multiple server machines: in the figure, we see a different icon for Darkstar, jVoiceBridge, an LDAP database, and a Web server. Although not depicted in the figure, it is also possible for Darkstar to be deployed across multiple servers (as noted at the end of Section 6.3). Of course, all of these services may just as easily reside on one physical machine, if that is desired.

WonderDAC leverages Wonderland’s client-server arrangement in order to keep a participant from being in direct contact with the DAC programming. Our DAC “authority,” then, is free to be situated on one or more secured hosts, enabling a high degree of assurance to be associated with the Wonderland server. Moreover, by maintaining our DAC solution in one logically central place, it becomes trivial to manage multiple perspectives of the same virtual world. For example, different levels of access may cause participants to see and interact with their
surroundings differently. On the other hand, a decentralized CVE approach, such as that found in a peer-to-peer environment, would have inherent difficulties securing DAC mechanisms and enabling participants to see different versions of the same virtual world simultaneously. In the absence of centralized management, the ability to handle DAC and state must be pushed to CVE participants. For access controls, this means an increased risk of tampering; and for state, it means somehow governing what is seen by participants who have different privileges in the same virtual environment.

As we have just noted, our implementation of WonderDAC’s critical components takes place only within server code. Nevertheless, a client-side to WonderDAC does exist—it’s function, however, is purely cosmetic. Specifically, it prevents a participant from making changes to their local representation of the virtual world, if these changes would not also be allowed in a global context. This client-side code is never relied upon for DAC purposes; if a malicious participant were to tamper with their client, they would be limited to altering their own, private representation of the CVE (they would not be able to change the server’s copy of their access control profile).

5.4.2 System Walkthrough

A simplified narrative of how Wonderland functions without the WonderDAC modifications should prove useful at this juncture. The client begins by transmitting a username and password to the Wonderland server. In response, the server consults an LDAP directory to lookup the participant and validate the given password. If the credentials are invalid or the participant doesn’t exist, the client is given two more chances to log in before their connection to the Wonderland server
is closed.

Upon validation of the username and password, the Wonderland server communicates back and forth with the client to instantiate server- and corresponding client-side Java objects. These represent the participant’s avatar and all of the Wonderland cells visible in the participant’s view of the virtual world. Moreover, they transmit messages between one and other to facilitate the activities of a given cell. This step also includes the establishment of a session between the client and jVoiceBridge server, enabling the participant to hear sound and engage in audio chats. As the client builds a representation of the virtual world, it queries a Web server to retrieve model and texture files for rendering.

Once logged in and initialized, the client communicates with the Wonderland server through both direct and publish/subscribe channels, and with other clients through publish/subscribe channels. All of these channels exist on the Wonderland server; however, data for sound and audio chat flow between clients and the jVoiceBridge server.

As the participant guides their avatar about the virtual world, the server tracks which cells become visible and which fall outside of the client’s perspective. Newly visible cells are configured on the server and client, and then rendered in the client’s display or, if audio-related, played over the client’s speakers.

When the client exits Wonderland, the participant departs from all of the communication channels joined earlier, and an orderly disposal of the Java objects tracking the participant’s state and avatar takes place.
5.4.3 The Wonderland Cell

As mentioned in Section 6.3, the virtual world in Wonderland is comprised of cells, where each cell “represents a 3D volume in the world [132].” Because cells are hierarchical, one root cell corresponds to the entire virtual world, while child cells throughout the hierarchy are associated with more fine-grained elements (or assets) of that world. By adding DAC attributes and methods to the basic Wonderland cell class (CellGLO), we are able to apply access control in a truly homogeneous and ubiquitous manner. This is key to maintaining the simplicity of our DAC solution throughout Wonderland.

The Wonderland cell hierarchy itself is implemented as another hierarchy of XML files. Stored on the Wonderland server or elsewhere, these files describe the layout of the virtual world and are collectively referred to as the Wonderland file system (WFS) [133]. For the purpose of testing WonderDAC, we devised the WFS depicted in Figure 5.3. As with the composition of all WFS hierarchies, our test WFS includes a root directory with a name ending in the \textit{-wfs} suffix, child directories with names that end in the \textit{-wld} suffix, and XML cell files with names ending in \textit{-wlc.xml}. It is not the case that one cell file must correspond to only one object within the virtual world. Instead, an array of object models can be included within a cell’s XML specification, thereby enabling more efficient handling of all the objects involved. Of course, WonderDAC operates at the level of a cell, so a multi-model cell’s objects would be protected as a group under one access control arrangement.

All XML cell files are serializations of a Java bean called BasicCellGLOSetup. This bean is responsible for the fundamentals of building a generic cell, and, so, has been modified by WonderDAC to include several basic access control elements,
Figure 5.3. The Test World WFS Hierarchy.
each represented by a Java String object: accessOwner, accessGroup, accessGroupPermissions, and accessOtherPermissions. The first String, accessOwner, denotes the participant who owns a cell in question; this account implicitly has full control over the cell. The next String, accessGroup, denotes the group that is assigned to the cell. Stored in the third String, accessGroupPermissions, are the permissions that determine how group members can access the cell. Similarly, the final String, accessOtherPermissions, maintains the permissions that determine how those who are neither the owner nor members of the cell’s group may access the cell. Permissions are specified as numeric values between 0 and 3 (see the table in Subsection 5.4.4.1 for an explanation of these numbers). Figure 5.4 shows an excerpt from an XML file with these four strings included. The CellGLO class also includes modifications that mirror the Java String objects found in BasicCellGLOSetup.

Regarding cell hierarchies, if the parent of a group of cells is made inaccessible to a participant, then all of the cells underneath that parent also become inaccessible, regardless of how their permissions are configured. For example, suppose a participant can access a room and the virtual objects contained therein, and the room’s cell is the hierarchical parent for those objects. Losing access to the room would immediately cause the entire space (including the contained objects) to disappear from the participant’s local perspective of the virtual world. Only by restoring access to the room’s cell would the room and its objects become visible again. Of course, the room’s objects could have a parent other than the room’s cell, although this might not be an intuitive virtual world design. In this circumstance, the objects would not necessarily disappear if access to the room were lost.
Figure 5.4. An excerpt from a sample XML cell file.
An important characteristic of Wonderland cells that has not yet been discussed is that they may overlap in the virtual world. For spatial cells, this turns out to have significant consequences regarding a DAC solution. For example, how do we determine access rights for a virtual room that falls partially or completely inside another space with different permissions? In the spirit of simplicity, we elected to use the following approach with WonderDAC.

When a participant’s avatar moves into an area of overlapping spatial cells, the group and permissions for the most restrictive cell at that location are applied to the avatar for the time spent in the area. Avatars are similar to other cells in that they are subject to the same DAC treatment. By dynamically assigning their permissions, the avatars of participants who can access a space will be invisible to those who cannot access the same space. For example, if avatar X enters a spatial cell that avatar Y’s participant cannot access, then the space and avatar X will be invisible to avatar Y’s participant. Avatar X will reappear only if it moves into a location accessible to avatar Y’s participant. (See figures 5.6 and 5.6.) When overlapping spatial cells have the same permissions, the temporary, dynamic assignment of a group and permissions to an avatar is handled in the following order:

1. If a space is owned by the participant, the participant’s avatar takes that spatial cell’s group and permissions

2. If a participant does not own any of the overlapping cells, but belongs to one of the cells’ groups, the participant’s avatar takes that cell’s group and permissions

3. Otherwise, the participant’s avatar takes the group and permissions of the first spatial cell examined by the server

When entering a non-overlapping space, an avatar temporarily adopts that
space’s group and permissions straight away.

5.4.4 Permissions and Roles

The approach embodied by WonderDAC is patterned after an idealized form of UNIX file system access control. A Wonderland cell may be thought of as a file system entity to which we assign roles with permissions. In keeping with our theme of simplicity, there are only three roles (owner, group, and other) with two permissions (interact and alter) per role possible. While this design may seem overly blunt (e.g., limiting how finely the attributes and capabilities of a cell may be restricted), we believe that a more complex solution would either be avoided or used incorrectly by participants, and, thus, would negate the whole purpose of having a DAC implementation. Because the basic tenets of UNIX file system access controls have become so engrained in modern operating systems, WonderDAC’s approach should seem natural and familiar to anyone who has had a reasonable level of interaction with a typical workstation.

We consider the WonderDAC permissions first, followed by the roles.

5.4.4.1 Permissions

The interact permission is an analog to the read and execute UNIX file system privileges, while the alter permission is an analog to write. Interact allows a participant to see and, in a limited sense, use a cell,\(^1\) while alter enables the participant to make changes to the cell. For example, if a participant has interact permission for a whiteboard cell, they will be able to see the whiteboard and any

\(^1\)Another way to view the interact permission is that it enables a participant’s local representation of the virtual world to receive a given cell’s updates.
image drawn there. If, in addition, the participant has *alter* permission, they will be able to draw upon or erase the whiteboard cell. Having *alter* permission without simultaneously having *interact* permission for a cell is logically the same as having no permissions. This is because it is impossible to make changes to a cell with which one cannot be in contact.

WonderDAC permissions are programmatically represented by numeric values in the following manner:

<table>
<thead>
<tr>
<th>Permissions</th>
<th>Numeric Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>IA</td>
<td>3</td>
</tr>
<tr>
<td>I -</td>
<td>2</td>
</tr>
<tr>
<td>- A</td>
<td>1</td>
</tr>
<tr>
<td>None</td>
<td>0</td>
</tr>
</tbody>
</table>

5.4.4.2 Roles

The *owner* role implicitly maintains full control of a given cell. Simply put, a cell’s owner always has *interact* and *alter* permissions for the cell. The only time this can change is when ownership is willingly transferred to a different participant,
whereupon the new owner immediately assumes full control over the cell.

The group role is both static and dynamic in nature. The static part of this duality includes three system groups (admin, users, and world) that help manage access along administrative, regular user, and fully public fronts, respectively. Wonderland users only become members of these groups through the action of the system administrator.\(^2\) One of the groups, world is hardcoded in WonderDAC as a convenience; the remaining two, however, exist only within the Wonderland LDAP user account database. For this reason, any number of other appropriately named static groups may also be employed.\(^3\) The only requirement is that referential integrity be maintained among all of the user accounts and system groups.

Regarding the dynamic nature of groups, WonderDAC supports the ability for participants to create and manage their own ad hoc, personal groups. These may be applied by a given user to any cells they own, and, thus, enhance the discretionary element of access control. A participant may create an ad hoc group by naming the group and adding to its membership at least one other user; the moment the group contains no users it is automatically deleted by the Wonderland server. Any cells that employ a defunct ad hoc group must be assigned to a different group by the owner, or they will appear to other participants as though they belong to an unknown group.

The other role is simply a catch-all that applies to those participants who are neither the owner nor group members of a given cell.

Finally, the permissions assigned to WonderDAC roles are mutually exclusive:

\(^2\)The unique, dummy accounts generated by Wonderland’s built-in test mechanism (e.g., Bench-1, Bench-2, ..., Bench-N) are automatically added to the world group.

\(^3\)WonderDAC enforces specifications for name composition: no more than 15 alpha-numeric characters and no spaces.
a participant can only reside in one role with respect to any given cell. During access checks, a participant is determined to be the owner of a cell, a member of the cell’s group, or part of the other role. This enables roles to enforce the addition or removal of privileges for specific groups of participants regarding a particular cell.

5.4.5  System Walkthrough Redux

Returning to our earlier narrative of how Wonderland functions (Subsection 5.4.2), we are now ready to include WonderDAC in our walkthrough.

The client begins by transmitting a username and password to the Wonderland server. In response, the server consults an LDAP directory to lookup the participant and validate the given password. If the password is invalid or the participant doesn’t exist, the client is given two more chances to log in before their connection to the Wonderland server is closed. Upon validation of the username and password, the participant’s group membership information is obtained from the LDAP directory and stored on the Wonderland server in a user profile.

Communication between the server and client then takes place to instantiate server- and corresponding client-side Java objects. These objects represent the participant’s avatar and all of the Wonderland cells visible from the participant’s view of the virtual world. Moreover, they transmit messages between one and other to facilitate the activities of a given cell. However, the messages can be ignored by the server when the participant does not have alter permission for the cell in question. As the server discovers cells that might be visible to the client, it determines the participant’s level of access; only cells for which the participant has interact permission are setup. During all of this initialization, the client also
establishes a session with the jVoiceBridge server, enabling the participant to hear sound and participate in audio chats. As the client builds a representation of the virtual world, it queries a Web server to retrieve model and texture files for rendering.

Once logged in and initialized, the client communicates with the Wonderland server through both direct and publish/subscribe channels, and with other clients through publish/subscribe channels. All of these channels exist on the Wonderland server; however, sound and audio chat data flow exclusively between clients and the jVoiceBridge server.

While the participant guides their avatar about the virtual world, the server tracks which cells become visible and which fall outside of the client’s perspective. As in the initial setup of the virtual world, only those cells with which the participant can interact are made available to and rendered by the client. Cells that become inaccessible are immediately removed from the client’s and server’s representations of the virtual world for the participant. Newly visible cells are configured on the server and client, and then rendered in the client’s display or, if audio-related, played over the client’s speakers.

When the client exits Wonderland, the participant departs from all of the communication channels joined earlier, and an orderly disposal of the Java objects tracking the participant’s state and avatar takes place.

5.5 Network Communications Security

It is important to bear in mind that WonderDAC operates at the top two layers (in-world access and user experience) depicted in Figure 5.1. This isn’t to suggest that network communications, the bottom layer, has nothing to do with
Figure 5.5. A typical Wonderland architecture configuration with secure network communications.

access control. At this layer, encryption and non-repudiation play the critical, respective roles of assuring the privacy and integrity of transmitted data, and guaranteeing that client and server endpoints are who they claim. Wonderland does not currently support such controls, making it vulnerable to eavesdropping, spoofing (where a malicious agent pretends to be a server or client), and integrity loss. More to the point, WonderDAC could be subverted by a malicious participant who wished to carry out any of these attacks. Hence, it would seem natural to add network communications controls to Wonderland in support of WonderDAC. Although we have elected to focus solely on WonderDAC in our work, we believe that enhancing Wonderland at the network communications layer is a conceptually straight-forward task as Figure 5.5 shows.

The central idea behind this figure is to employ asymmetric key cryptography to protect lines of communication and identify users and services (the use of such
mechanisms in a CVE context is discussed by [115], for example). Presumably, a protocol such as secure sockets layer (SSL) or transport layer security (TLS) could be used to authenticate clients and servers to one and other and negotiate symmetric keys for data stream encryption. For convenience, the LDAP server could store public keys as part of user account records (private keys would have to be securely maintained by participants and Wonderland system administrators, respectively). Finally, the Web server would need to be capable of communicating with the Wonderland server to prevent unauthorized access to model and texture files (i.e., the Web server could validate access attempts using pre-existing Wonderland mechanisms). Thus, a new communications channel, indicated by a dashed connector line in Figure 5.5, would have to be established between these two servers.

5.6 WonderlandDAC by Example

Prior to the implementation of the WonderlandDAC prototype, we discerned five areas that we believed should be addressed by any successful CVE DAC solution. These were captured as the use case scenarios listed in Section 5.2, and only partially treated by the prototype. The following subsections reiterate these scenarios (in slanted text), as they were stated in our earlier work, and include brief technical discussions and examples of how each scenario is now handled by our completed WonderlandDAC implementation.

5.6.1 Use Case 1: Spatial Object Restrictions

A participant should only be able to enter, hear, and see into a virtual space if they are the owner of the space, a member of the space’s group (and the group
has interact permission), or the space is defined with interact permission for the other role. When a participant’s avatar has entered a space, the avatar should only be visible and heard by other participants who also have interact permission (but may or may not be inside the space).

A participant should only be able to change a virtual space (by speaking in, or deleting/updating the space) if they are the owner of the space, a member of the space’s group (and the group has alter permission), or the space is defined with alter permission for the other role. Although it is possible for a space to have only the alter permission assigned to the group or other roles, the result of such a configuration would be meaningless, since non-owners would be incapable of entering the space to begin with.

In WonderDAC, spatial cells are identified as one of two server-side classes: SimpleTerrainCellGLO or MultiModelCellGLO (both extend another class called StationaryCellGLO, which represents cells that cannot move). As noted in Subsection 5.4.3, WonderDAC needs to detect where a participant’s avatar is located in order to manage access to spaces. Both the detection and the DAC assessment take place in a method called revalidate, which belongs to the server-side class UserCellCacheGLO. This method is very important to WonderDAC’s implementation: not only does it deal with spatial access control, it reviews all other cells visible to the participant and determines those cells’ interact and alter accessibility. Each Wonderland participant has an associated UserCellCacheGLO object that builds, maintains, and prunes all of the visible (and, therefore, accessible) cells comprising that participant’s local perspective of the virtual world.

When a participant lacks interact permission for a spatial cell, that cell isn’t
made available to the participant’s client for rendering, or, if already rendered, it is removed from the cache of cells associated with the client. If the participant lacks *alter* permission for a spatial cell they become muted when their avatar resides in that space.

In figures 5.6 and 5.7 we see the result of a private space (a second floor loft) being made accessible relative to two participants. Initially, these participants can see neither the space, nor the T Wright avatar residing in the space. Once both participants are granted *interact* permission, however, the space and avatar become visible.

### 5.6.2 Use Case 2: Non-spatial Object Restrictions

A participant should only be able to view and, if applicable, hear a non-spatial object (e.g., a whiteboard, phone, or X Window application) if they are the owner of the object, a member of the object’s group (and the group has *interact* permission), or the object is defined with *interact* permission for the other role. For example, a participant with only *interact* permission for a whiteboard can view the whiteboard and its contents, but cannot change or add to those contents. Similarly, a participant with only *interact* permission for a conference phone, can view and hear the phone, but cannot join in the conversation.

A participant should only be able to change/utilize a non-spatial object if they are the owner of the object, a member of the object’s group (and the group has *alter* permission), or the object is defined with *alter* permission for the other role. Similar to Use Case 1, although it is possible for a non-spatial object to have only the *alter* permission assigned to the group and other roles, such a configuration is meaningless since non-owners would be unable to view/hear the object in the
Figure 5.6. Because their participants do not have *interact* permission for the loft, the JUser and Bench-415 avatars cannot see the loft stairs or the T Wright avatar.
Figure 5.7. Now granted the appropriate permission, the loft and TWRight avatar appear for the JUser and Bench-415 participants.
Between this and the first scenario, almost all possible Wonderland cell types are covered. As in the first use case, a participant’s access to a non-spatial cell is determined by the `revalidate` method found in the UserCellCacheGLO class. Unlike spatial cells, however, non-spatial cells required us to implement additional handling mechanisms on the server and client-side. For example, Wonderland’s PDF viewer, video viewer, and VNC client cells can all be shared by multiple, simultaneous participants; removing `interact` permission for some users while they are working with these cells means having to deal with server-side control and synchronization issues. Also, as discussed in Subsection 5.4.1, WonderDAC’s client-side handles cosmetic issues that result when `alter` permission is not granted for a non-spatial cell. For example, this prevents a user from drawing on their local copy of a Wonderland whiteboard when they do not have permission to update the whiteboard globally.

Figures 5.8 and 5.9 demonstrate how WonderDAC can restrict access to a whiteboard cell. The same principle applies equally to all other Wonderland cells.

5.6.3 Use Case 3: Audio Conversation Restrictions

Two or more participants engaged in audio chat should be able to restrict their conversation to themselves (somewhat like having a “cone of silence” at their disposal). Other participants should be able to solicit the initial group to join in the conversation if the group wishes. This amounts to a specialized version of Use Case 1 except that an ephemeral, invisible space is created around the participants:
Figure 5.8. Configuring a Wonderland whiteboard to prevent updates from everyone but the owner.
Figure 5.9. Using a Wonderland whiteboard; only T Wright, the owner, is able to alter the whiteboard’s contents.
1. A temporary group role, to which all of the conversation participants belong, is created and assigned to the invisible space.

2. Interact and alter permissions are assigned to the space’s group role.

3. New participants may be added to the temporary group role as desired by the initial conversation participants.

It should be possible to assign/remove the interact and alter permissions for the invisible space’s other role. This could enable, through the assignment of only the interact permission, members of the temporary group to converse, while all others can listen but not interrupt. The use of alter permissions alone for the invisible space’s other role may or may not make sense, depending on the desires of the conversation participants: a participant in the other role would be able to speak in, but not hear, the conversation.

Ultimately, this use case was handled by coalescing two elements: our implementation of the first scenario (spatial object restrictions) and Wonderland 0.4’s introduction of an actual cone-of-silence cell.\(^4\) By adding roles and ad hoc groups to the newly available cone-of-silence cell, it becomes possible to restrict audio conversations in the way we desire. However, minor differences between this and our third use case’s original specifications do exist. For example, Wonderland’s cone-of-silence is not an ephemeral, invisible space, as prescribed by the use case scenario; it is a translucent cone that stays suspended over the floor to mark off an area for protected audio chats.\(^5\) In addition, all participants of a protected chat

\(^4\)The cone-of-silence cell is implemented on the server by the ConeOfSilenceCellGLO class, which extends the EnterExitCellGLO class. EnterExitCellGLO is a child of StationaryCellGLO.

\(^5\)Participants inside the cone-of-silence can only hear each other, while those outside cannot hear inside the cone-of-silence.
must have *interact* permission for the cone-of-silence cell, or else the cell will not exist in their client-side representation of the virtual world. With only *interact* permission (i.e., no *alter* permission), a participant will see the cone-of-silence object, but be unable to hear or speak in the protected audio chat. This is the most significant departure from the use case, which specifies that a participant may be able to speak in a conversation they cannot also hear. Because the need for this ability is, at best, rare, and simplicity is served by streamlining functionality, we decided to drop this part of the use case implementation. Another, less significant departure from the use case is that the cone-of-silence cell must already exist and be owned by a conversation participant. There is no reason why future versions of Wonderland cannot simply allow a participant to dynamically create/destroy cones-of-silence as they see fit.

Figures 5.10, 5.11, and 5.12 demonstrate how an ad hoc group may be created and assigned to a cone-of-silence cell in order to enabled a private conversation among two participants (others, of course, may be added to the group and conversation as desired).

The cone-of-silence is not an appropriate tool to configure a space that simply allows one or more participants to speak while others can only listen. Instead, such an area is created by enabling a spatial cell’s *interact* and *alter* permissions for a group of speakers, and just the *interact* permission for the cell’s *other* role. A space of this type could facilitate presentations, after which questions could be asked of the presenter by enabling the *alter* permission for the spatial cell’s *other* role. Figure 5.13 depicts the avatar T.Wright addressing two other avatars who are not allowed to speak in the area where they reside (as indicated by the square brackets around their names).
Figure 5.10. Creating an ad hoc group to manage a Wonderland cone-of-silence.
Figure 5.11. Assigning an ad hoc group to a cone-of-silence.
Figure 5.12. Having a private conversation inside a cone-of-silence.
5.6.4 Use Case 4: Avatar Cloaking

A participant should be able to disguise their avatar’s appearance. Each avatar should have a group role that is unique to its owner. By assigning or removing interact permissions to/from this group or to/from the avatar’s other role, the appearance of the avatar should be controllable by its owner. A participant should be able to see an avatar’s true image if they are a member of the avatar’s group (and the group has interact permission), or the avatar is defined with interact permission for the other role. Otherwise, a generic image should be displayed by the avatar. The use of alter permissions should be ignored, here: only the owner should be permitted to make changes to an avatar’s appearance. This approach helps to manage any risk of avatar defacement by deferring to the owner to facilitate all avatar image changes.

Figure 5.13. Muted avatars being addressed by a speaking avatar.
As with the last use case, our implementation of this scenario departs somewhat from the specifications given; there are two issues driving this. First, we discovered that Wonderland uses participant IDs when naming some of the communication channels created between clients and the server. Because of this, a malicious participant could monitor communications between their Wonderland client and the server to figure out the identity of a cloaked individual. Second, mandating that an avatar have a group role unique to its owner seems unnecessarily strict: as with all Wonderland cells, an owner should be able to assign any group they wish to their avatar.

Our answer to the above issues is, as always, guided by our deference to simplicity: we treat the avatar like any other Wonderland cell. If a participant does not have interact permission for a particular avatar, that avatar will not exist in the participant’s local perspective of the virtual world. This entails removing (or never establishing) all traces of the avatar and its participant, including related communication channels, relative to unauthorized participants. The group role issue is resolved as a natural consequence of treating an avatar like other cells and enables a participant to apply ad hoc groups to their avatar, as well.

An interesting side effect of being able to handle the spatial access control situations outlined at the end of Subsection 5.4.3, is that we need two sets of roles for each avatar. One set helps WonderDAC sort through access to spaces that may or may not overlap, while the other handles avatar cloaking.

Figures 5.14 and 5.15 demonstrate how an avatar cloak may be configured—in this case, the TWright avatar seems to disappear completely from the virtual world, as far as the other participants are concerned.
Figure 5.14. Configuring an avatar cloak.
Figure 5.15. Deploying an avatar cloak.
5.6.5 Use Case 5: Permissions and Ownership Changes

All participants should be able to determine the ownership of a VR object. Also, a participant should be able to change permissions for the group and other roles of any object they own, or, if desired, assign ownership of the object to another participant. The interface to carry out these activities should be simple and accessible through mouse interactions with the object in question.

This use case is handled as stated, but with an additional interface to permit the creation and management of ad hoc groups. For the purpose of confidentiality, when a participant who is not the owner of a cell reviews that cell’s DAC configuration, they will only see a group role name if they happen to be a member of the group. This is a matter of “least privileges,” whereby a user can access only the information and resources needed to carry out their authorized activities. Also, WonderDAC handles all user input sanity checking on the server, although, for convenience, the Wonderland client engages in simple composition checks of anything the participant enters in the interface dialogs.

Figure 5.16 shows the Wonderland client’s Tools menu, where two WonderDAC options are included: Avatar Cloak and Ad Hoc Groups. Sample dialog box images for both are provided in figures 5.17 and 5.18.

The Avatar Cloak dialog box (Figure 5.17) is the simplest of the WonderDAC interfaces. The participant can apply any group they wish (system or ad hoc) to their avatar, and then configure the interact permission for this group and the other role as desired.

The Ad Hoc Group dialog box (Figure 5.18) enables a participant to manage any pre-existing ad hoc groups or create new ones. Management consists of adding
Figure 5.16. The Wonderland client’s Tools menu, including the newly added Avatar Cloak and Ad Hoc Group options.

Figure 5.17. The Avatar Cloak dialog box.
users to or removing users from a particular group. By entering a name in the "New Group" text box, an ad hoc group can be initiated; adding at least one user to the group and clicking the "Save" button will create and store the group on the LDAP server. As soon as an ad hoc has no members it is automatically removed by Wonderland server—it is up to the owner of any cells that use a defunct group to assign new groups as appropriate. All Wonderland user accounts have a built-in ad hoc group named after the user ID; this group can never be removed.

The WonderDAC configuration dialog box (Figure 5.19) is invoked by a right mouse click on whatever cell a participant wishes to manage or review. More specifically, two-dimensional windows (e.g., PDF viewer, movie viewer, whiteboard, VNC client, X Window application) have a small, translucent, yellow button in their upper-left corner that opens this dialog upon a right mouse click; all other Wonderland cells respond to a right mouse click anywhere within their
boundary. The contents of the WonderDAC configuration dialog include the current owner and group for the cell in question, along with permissions settings for the group and other roles. There is also a “Recursive” checkbox to indicate whether or not a new DAC configuration should be applied to the cell and its children. Whenever a new owner is selected for a cell, the group is automatically forced to be “world.” This ensures that the cell is never assigned to a group in which the new owner is not a member.

5.6.6 Summary Use Case Table

Table 5.2 summarizes the use case scenarios discussed above, including the semantics of permissions settings.

---

6In the current version of WonderDAC, only one level of children is recursively updated.
<table>
<thead>
<tr>
<th>Use Case</th>
<th>Permissions</th>
<th>Meaning for Group Role</th>
<th>Meaning for Other Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spatial</td>
<td>I A</td>
<td>Any member of the object’s group can enter, see, and hear into the space. All such</td>
<td>All participants who are not members of the object’s group can enter, see, and hear into the space. All such constituents can speak within and change the space, too.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>constituents can speak within and change the space, too.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>I -</td>
<td>Any member of the object’s group can enter, see, and hear into the space. All such</td>
<td>All participants who are not members of the object’s group can enter, see, and hear into the space. All such constituents are unable to speak within and change the space.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>constituents are unable to speak within and change the space.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- A</td>
<td>Meaningless: equivalent to a participant having no <em>group</em> permissions for the object.</td>
<td>Meaningless: equivalent to participants having no <em>other</em> permissions for the object.</td>
</tr>
<tr>
<td></td>
<td>- -</td>
<td>No member of the object’s group can enter, see, or hear into the space. All such</td>
<td>All participants who are not members of the object’s group cannot enter, see, or hear into the space. All such constituents are unable to speak within and change the space.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>constituents are unable to speak within and change the space.</td>
<td></td>
</tr>
<tr>
<td>Use Case</td>
<td>Permissions</td>
<td>Meaning for Group Role</td>
<td>Meaning for Other Role</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
<td>----------------------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Non-spatial Object</td>
<td>I A</td>
<td>Any member of the object’s group can see and, if applicable, hear the object. All such constituents can change the object, too.</td>
<td>All participants who are not members of the object’s group can see and, if applicable, hear the object. All such constituents can change the object, too.</td>
</tr>
<tr>
<td></td>
<td>I -</td>
<td>Any member of the object’s group can see and, if applicable, hear the object. All such constituents are unable to change the object.</td>
<td>All participants who are not members of the object’s group cannot enter, see, change, or, if applicable, hear the object.</td>
</tr>
<tr>
<td></td>
<td>- A</td>
<td>Meaningless: equivalent to a participant having no group permissions for the object.</td>
<td>Meaningless: equivalent to participants having no other permissions for the object.</td>
</tr>
<tr>
<td></td>
<td>- -</td>
<td>No member of the object’s group can see, change, or, if applicable, hear the object.</td>
<td>All participants who are not members of the object’s group cannot enter, see, change, or, if applicable, hear the object.</td>
</tr>
<tr>
<td>Use Case</td>
<td>Permissions</td>
<td>Meaning for Group Role</td>
<td>Meaning for Other Role</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------</td>
<td>----------------------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Audio Con-</td>
<td>I A</td>
<td>All member of the cone-of-silence’s group can hear and speak in the conversation.</td>
<td>All participants who are not members of the cone-of-silence’s temporary group can hear and speak in the conversation.</td>
</tr>
<tr>
<td>versation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I -</td>
<td></td>
<td>All members of the cone-of-silence’s temporary group can see the cone-of-silence cell,</td>
<td>All participants who are not members of the cone-of-silence’s temporary group can see the cone-of-silence cell, but cannot use the cell.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>but cannot use the cell.</td>
<td></td>
</tr>
<tr>
<td>- A</td>
<td></td>
<td>Meaningless: equivalent to a participant having no <em>group</em> permissions for the cone-of-silence cell.</td>
<td>Meaningless: equivalent to participants having no <em>other</em> permissions for the cone-of-silence cell.</td>
</tr>
<tr>
<td>- -</td>
<td></td>
<td>No member of the cone-of-silence’s group can see and use the cell.</td>
<td>All participants who are not members of the cone-of-silence cell cannot see and use the cell.</td>
</tr>
<tr>
<td>Use Case</td>
<td>Permissions</td>
<td>Meaning for Group Role</td>
<td>Meaning for Other Role</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------</td>
<td>----------------------------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Avatar Cloak</td>
<td>I A</td>
<td>Any member of the object’s group can see and interact with the avatar. The alter permission is ignored: only the owner is allowed to change an avatar.</td>
<td>All participants who are not members of the avatar’s group can see and interact with the avatar. The alter permission is ignored: only the owner is allowed to change an avatar.</td>
</tr>
<tr>
<td>I -</td>
<td></td>
<td>Any member of the object’s group can see and interact with the avatar. (Effectively the same as I A permissions above)</td>
<td>All participants who are not members of the avatar’s group can see and interact with the avatar. (Effectively the same as I A permissions above)</td>
</tr>
<tr>
<td>- A</td>
<td></td>
<td>No member of the avatar’s group can see or interact with the avatar. The alter permission is ignored: only the owner is allowed to change an avatar.</td>
<td>All participants who are not members of the avatar’s group cannot see or interact with the avatar. The alter permission is ignored: only the owner is allowed to change an avatar.</td>
</tr>
</tbody>
</table>
5.7 Conclusion and Future Work

WonderDAC is intended to address a deficiency in state-of-the-art CVEs: the lack of functional, comprehensive, yet simple means of access control. Although some commercial CVE systems offer DAC capabilities, their controls are often limited in scope (e.g., oriented toward the management of virtual land) and somewhat complex in nature. Open source CVEs are arguably worse in that they tend to offer little if any DAC capabilities.

Patterned after idealized, UNIX file system access controls, WonderDAC extends Project Wonderland by introducing role-based access controls at the level of a Wonderland cell. Access to a cell is handled according three roles: owner, group, and other. Two permissions, interact and alter, may be enabled or disabled for the group and other roles, while the owner role always maintains full control over a given cell. Five use case scenarios were defined and partially implemented for the WonderDAC prototype we created in the previous chapter. In moving from

<table>
<thead>
<tr>
<th>Use Case</th>
<th>Permissions</th>
<th>Meaning for Group Role</th>
<th>Meaning for Other Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>No member of the avatar’s group can see or interact with the avatar. (Effectively the same as - A permissions above)</td>
<td>All participants who are not members of the avatar’s group cannot see or interact with the avatar. (Effectively the same as - A permissions above)</td>
<td></td>
</tr>
</tbody>
</table>
prototype to full solution we have had to adjust and streamline our handling of the third and fourth use cases (Audio Conversation Restrictions and Avatar Cloaking, respectively); the remaining use cases, however, have been implemented as specified.

Despite the broad scope and relative completeness of WonderDAC’s current iteration, much work remains. First, Wonderland 0.4’s integration of shared X Window applications, while functional, is not optimal. This has led to constraints for WonderDAC, such as the inability to control who is allowed to start or stop an X Window application within a given space. Next, WonderDAC’s user interface should eventually be migrated from Java Swing to something that is rendered within the Wonderland client window (i.e., as a part of the virtual environment’s graphic display). Also, a variety of administrative tools need to be created for WonderDAC to more easily enable activities such as password changes and account management—at the moment, all such undertakings are handled by the Wonderland administrator through an LDAP browser/editor. Finally, security needs to be implemented at Wonderland’s network communications layer in order to ensure the privacy and integrity of WonderDAC’s in-world access control.
CHAPTER 6

WONDERDAC IN PRACTICE *

6.1 Abstract

The use of discretionary access control (DAC) within collaborative, virtual environments (CVEs) has been a limited endeavor for both proprietary and open source systems. Yet, as virtual worlds become more useful and engaging to our computing society, the need to safeguard access to virtual objects (e.g., spaces, three-dimensional assets, two-dimensional images and videos, avatars, sound) becomes increasingly important. In the previous two chapters we have proposed and implemented a DAC extension called WonderDAC for the Project Wonderland CVE. Here, we demonstrate how WonderDAC can play a useful role in managing access within a virtual world adapted to educational purposes. In our demonstration, a small group of students researches a topic, builds presentations, and, finally, delivers talks—all within the Wonderland CVE. Along the way, WonderDAC operates effectively to enable and ensure the privacy of research activities and orderly presentations afterwards.

* Material from this chapter appears as a technical report at the University of Notre Dame: Wright, T. and Madey, G., WonderDAC in practice: a demonstration of discretionary access controls within the project wonderland cve, Technical Report TR 2008-16, University of Notre Dame, College of Engineering, University of Notre Dame, Notre Dame, IN 46556, November 2008.
6.2 Introduction

Desktop virtual reality (i.e., virtual reality [VR] applications and systems that are accessible from a standard, desktop workstation) has introduced avenues of entertainment, commerce, education, and science that have begun to take root within our computing society. A central theme of the desktop VR paradigm is one in which multiple, simultaneous users, represented by avatars, can remotely interact within a virtual world. We refer to such realms as collaborative, virtual environments (CVEs) and note the existence of popular commercial systems like Second Life, There, and Activeworlds, as well as open source platforms such as Croquet, OpenSim, and Project Wonderland.

In the previous two chapters, we extended Project Wonderland with a discretionary access control (DAC) system called WonderDAC. This was an effort to grapple with the problem of weak and limited access control found in both commercial and open source CVEs. Our solution was designed according to five use case scenarios: spatial object restrictions, non-spatial object restrictions, audio conversation restrictions, avatar cloaking, and permissions and ownership changes. The first two of these deal with protecting access to spaces (three-dimensional) and objects (three- and two-dimensional), respectively. For example, enabling/disabling a CVE participant’s ability to enter, view, and hear into a space, as well as their ability to see and use some asset. The third use case, audio conversation restrictions, is aimed at managing the capacity of a participant to speak within an area. This can be applied to situations where one participant is addressing a group of others and should not be interrupted, or used to enforce vocal silence in a particular space. The fourth use case, avatar cloaking, addresses the need for participants to control who can and cannot know they are present in a CVE. For
example, a participant may only wish that a particular group can see and interact with their avatar. Finally, the last use case, permissions and ownership changes, proposes that a simple, effective graphical user interface be employed to handle DAC configuration management by CVE participants.

In this chapter, we discuss a demonstration in which WonderDAC’s privacy and integrity mechanisms were put to the test. Several participants were invited to spend a short time investigating and presenting a simple topic entirely within the Wonderland CVE. At the outset, they were divided into two teams, with each team given access to a separate, private space for research activities; neither team was permitted to access the other’s space. To aid in their work, each team had its own Web browser, presentation builder, and whiteboard. Thirty minutes were allotted for research activities, after which the teams relocated to a common space and used another 20 minutes to present their findings to each other. To guide these presentations, the teams alternated as audience and presenters, with only the presenting team being able to speak and manipulate a virtual PDF viewer. Also, a virtual microphone (accessible by the presenting team) enabled speakers to be heard throughout much of the CVE—in case an audience member temporarily left the space where the presentations were given. Finally, the teams were allowed some additional time to socialize and walk about the CVE. In total, the demonstration required an hour and 15 minutes: the length of time for a typical college class meeting.

Following the demonstration, a ten question survey was administered to the participants. The goals of the survey were to solicit opinions about how well WonderDAC seemed to function and how sensible its approach and interfaces were. Depending on context, one of two five-point rating scales was used for each
survey question.

The remainder of this chapter is organized as follows: the next section summarizes related works, Section 6.4 looks at our approach to evaluating WonderDAC’s usability, Section 6.5 reviews and interprets the results of our evaluation, Section 6.6 discusses scalability and benchmarking issues, Section 6.7 considers other important issues faced by Wonderland and WonderDAC, and Section 6.8 concludes the chapter with summary remarks and a brief discussion of future work.

6.3 Related Work

The field of human-computer interface (HCI) study has been active for decades and offers established methods for designing and testing various interface types [8, 26, 98]. However, the treatment of VR within HCI is still relatively new, and not yet formalized [84, 95]. Nevertheless, we are interested in approaches that can evaluate the usability of desktop CVEs, in particular as this relates to access control. Although our search of the literature has revealed some information about CVE usability testing, we have found nothing to addresses the role that might be played by access control. Hence, our review of related works is focused on those efforts that we can learn from and adapt into our own evaluation method.

There are largely two categories into which we may divide related works: usability tests and usability evaluation approaches. Examples of CVE-related tests are often aimed at comparisons of activities and/or interfaces relative to the virtual world. Typically, there is both a quantitative and qualitative analysis involved. In [103], the authors evaluate how efficiently ten users can search for books within a virtual library. A quantitative analysis is carried out to compare different clustering and query highlighting techniques for the books; a qualitative analysis of
user behavior within the virtual library is also undertaken. [78] presents an assessment of the physical interfaces used in the navigation of a three-dimensional terrain system. Here, in an experiment with 24 participants, quantitative and qualitative comparisons are made by the authors for mouse, speech, gesture, and multimodal (speech and gesture) navigational means. Quantitatively, the time to locate a target with each interface and the number of correctly recalled locations are measured and evaluated. Qualitatively, test participants are asked to rate various aspects of the terrain system (e.g., ease of learning and use, sense of presence, comfort). A purely quantitative CVE assessment is found in [55], in which the authors discuss a model for transitioning between augmented reality (AR) and VR. They present two experiments in which they evaluate the ability of a user to navigate a maze through AR and VR interfaces. Analysis of their experiments is quantitative and focused on the efficiency differences and gains of using AR or VR.

Regarding evaluation approaches for CVE usability, sometimes an approach is derived to assist with the construction and evaluation of a specific system; other times, an approach is independent of any CVE. For example, in [52] the authors build and evaluate a medical training CVE according to three methods: expert heuristic, formative evaluation, and summative evaluation. The first two are applied in alternating fashion during the initial design of the CVE. Collectively, these leverage the knowledge of an expert to resolve usability problems and improve the CVE’s design. The third method, summative evaluation, is applied last to make statistical comparisons of how well interfaces, operations, and components work. Along similar lines, [49] build and evaluate a battlefield visualization environment according to what they call user-centered design. This follows a methodology
akin to that above, but with an initial user-task analysis step to explicitly call out the need for “identifying a complete description of tasks, subtasks, and methods required to use a system, as well as other resources necessary for user(s) and the system to cooperatively perform tasks.” No CVE is built or assessed in [95], where the author works from an established HCI perspective (for two-dimensional interfaces) in order to formulate a new approach for evaluating VR systems. Along the way it is noted that “tasks with VR systems are not only performed with the VR application, but also within or inside the virtual environment.” This, then, becomes the primary difference between VR systems and standard two-dimensional interfaces, and necessitates a different, augmented approach for VR usability evaluation. In [84], the authors present a substantial framework (what they call a handbook) for evaluating VR applications. To assist the typical researcher or student with carrying out a meaningful usability evaluation of a VR system, they propose a number of simple guidelines and include two brief illustrative case studies. Finally, in [71] the authors devise a new HCI approach called Reality-Based Interaction (RBI). RBI is intended to better enable the evaluation and understanding of emerging, post-WIMP (window-icon-menu-pointing device) interfaces such as VR. The theme of RBI is that such interfaces are unified in their employment of real-world interactions: interfaces may be mobile, and, as with VR, navigation and perception are often driving forces. RBI goes on to postulate that the most effective interface designs are those that maximize power (functionality and efficiency) and reality.
6.4 Evaluating WonderDAC

Our evaluation of WonderDAC comes with several caveats. First, WonderDAC is an extension to Project Wonderland 0.4. This version of Wonderland is intended to support only small groups of participants on a well-appointed server under moderate load. Next, we have no interest in evaluating Wonderland and its interfaces. Instead, we wish to evaluate how successfully WonderDAC can control access to objects and spaces within Wonderland. Finally, to ensure that the use of Wonderland is not an impediment in our assessment of WonderDAC, we decided to handpick a small group of technically savvy participants.

Ultimately, these stipulations diminish the utility of a highly quantitative experiment. In fact, because our participants are not probabilistically sampled, we are prevented from making any reliable statistical inferences about WonderDAC’s operation [77]. However, we contend that this does not reduce the value of our assessment, so long as we have designed a realistic demonstration and have elicited useful feedback from the participants. To these ends, we have carried out a demonstration of WonderDAC’s capabilities within a Project Wonderland 0.4 CVE according to the background and narrative provided below. As a follow-up to the demonstration, we administered a short, participant survey to gauge perspectives and opinions about how WonderDAC operated and whether or not its approach to access control made sense. In designing this demonstration, we have adapted guidelines from the *Handbook for Evaluation Studies in Virtual Reality* devised by [84].
6.4.1 WonderDAC Demonstration Background

The essential research question we wish to address is, does WonderDAC effectively and usefully control access to spaces and objects within the Wonderland CVE? To answer this, we constructed a simple, virtual world wherein a small group of participants could engage in two educationally motivated activities: researching a topic and then presenting this research to others. Here, WonderDAC’s role would be to enable the privacy of the participants as they conducted their research, and then to permit orderly presentations by controlling who is able to speak and who has access to the presentation tool.

The demonstration environment included four primary spaces in which the participants operated: a team room where avatars first appear when a client logs into Wonderland, a demonstration room that shows off some of Wonderland’s features, and two second-floor lofts where participants can carry out research and build their presentations. In addition to this basic layout, the team room is further divided into a movie screening room, a central meeting area, and two moderate-sized cubicles (for purposes of ambiance). The team room’s central meeting area is where presentations may be given, and is equipped with a virtual PDF viewer, tape recorder, and microphone.

The participants selected for this demonstration were all graduate students in Computer Science and Engineering. Researching, creating, and delivering presentations are activities with which the participants had familiarity. We settled on a group of six in order to permit two small teams to simultaneously work on the task of researching and creating presentations. Given the capacity limitations in Wonderland 0.4, we decided to keep the number of participants as low as possible so that we could include more assets within the virtual environment (e.g., there
were several X Window applications, whiteboards, PDF viewers, and various other objects available in the virtual world). Unfortunately, one of the participants was unable to attend the demonstration, leaving us with a team of three and a team of two.

Another person operated in the role of virtual test monitor to observe and record (from their avatar’s perspective) research activities and the presentations. One guest observer was also present during the demonstration, although their avatar employed WonderDAC’s cloaking feature to remain invisible to the other participants until after the first presentation.

The Wonderland 0.4 CVE used in the demonstration was installed on an Intel Core 2 Quad 2.5 gigahertz workstation with four gigabytes of RAM operating Ubuntu 8.04 LTS. The participants’ Wonderland client software was installed on Intel Core 2 Duo 2.13 gigahertz workstations, each with two gigabytes of RAM, ATI Radeon X1600 graphics adapters, and 256 megabytes of video RAM. All participant workstations operated clones of the same Windows XP image. Unlike the participants, the guest observer employed OS X on an Intel Core 2 Duo 2.33 gigahertz MacBook Pro with three gigabytes of RAM, an ATI Radeon X1600 graphics adapter, and 256 megabytes of video RAM. Finally, the test monitor also used an OS X MacBook Pro, but with an Intel Core 2 Duo 2.5 gigahertz processor, four gigabytes of RAM, an NVidia GeForce 8600M GT graphics adapter, and 512 megabytes of video RAM.

Prior to the demonstration, all participants were given a 15 minute introduction to Wonderland/WonderDAC and the chance to try out the Wonderland CVE with the WonderDAC extensions. After the demonstration, the participants were de-briefed and asked to answer a simple questionnaire.
6.4.2 WonderDAC Demonstration Narrative

The following narrative acted as a flight-plan during the demonstration and is summarized as a schedule of test goals in Subsection 6.4.3.

6.4.2.1 Environment Preparations Phase (completed before demonstration)

Before the demonstration commences, the Wonderland system administrator (who may also be the test monitor) logs into the CVE and ensures that the environment is properly configured:

1. All necessary user accounts have been appropriately created

2. Two groups, TeamA and TeamB, have been created; half of the participants have been added to TeamA, and the other half to TeamB

3. The test monitor’s and guest observer’s accounts have been added to both the TeamA and TeamB groups (to enable full access to each team’s space)

4. The Loft1 space has been assigned to the TeamA group; the Loft2 space has been assigned to the TeamB group

5. A Web browser (Firefox) and presentation tool (OpenOffice Impress) have been enabled in both lofts and assigned to each respective loft’s group (i.e., TeamA or TeamB)

6. The rest of the environment has been left in its default, initialized state

Test goals: the system administrator should be able to create the two necessary, ad hoc groups and add the participants, test monitor, and guests to these groups as required; also, the lofts and any objects within the lofts (i.e., X Window applications and whiteboards) should be assigned to their respective groups to enable spatial privacy and integrity.
Figure 6.1. A bird’s eye view of the virtual environment used in the demonstration. The team room (with office cubes, common area, and movie screening room) is at the top, while the demo room is underneath in the middle; Loft1 is to the left of the demo room and Loft2 is to the right.
Figure 6.2. A student participates in the Wonderland/WonderDAC demonstration.

6.4.2.2 Login Phase (5 minutes)

The test monitor and guests log into the Wonderland CVE first, then the test monitor prepares to document the demonstration. The participants login next and proceed to their respective team lofts. Because the members of one team cannot see or access the other team’s loft, locating the correct destination is trivial.

Test goals: the participants should only be able to see and access their respective team’s loft; when members of either team enter their loft, they should disappear from the perspective of the other team; none of the participants should be able to see/hear any guest observers who have logged into the CVE.
Figure 6.3. Participants begin the demonstration in the team room, after logging into Wonderland.
Figure 6.4. TestUser1 looks out from their loft to where TeamB’s loft (Loft2) should be. Because this participant does not have access to Loft2, only a wall in the demo room below is visible.
6.4.2.3 Research Phase (30 minutes)

Once all participants are ready, the research phase of the demonstration begins. Both teams spend the allotted time researching the history of hypertext and building a six minute presentation. At their disposal in each loft are two X Window applications (the Firefox Web browser and OpenOffice Impress) and a virtual whiteboard. The resulting presentations are saved as PDFs in locations provided by the test monitor.

At the end of this phase, the test monitor alters the group and permission configurations of both lofts to permit full access by all participants. The X Window applications are then closed to save on system resources.

Test goals: status quo—the teams should continue to have access only to their respective lofts; neither team should be able to see/hear the team in the other loft until the phase has ended; guest observers should remain cloaked.

6.4.2.4 Presentation Phase (25 minutes)

Each team makes a six minute presentation (allowing each member a chance to talk) and answers questions for four minutes at the end. Permission to use (i.e., alter) the team room’s PDF viewer and microphone should be granted to the presenting team while the other team acts as an audience. Five minutes are allotted to switching the teams from presenters to audience members. All guest observers should reveal themselves at the end of the first presentation so that they may participate in questioning the presenters—they should stay de-cloaked for the remainder of the demonstration.

Test goals: the test monitor should configure the group and permissions of the team room PDF viewer and microphone to enable control by the presenting
Figure 6.5. TeamA engages in research—the Firefox web browser is executing in the left window, while OpenOffice Impress is the right.
Figure 6.6. TeamB engages in research.
team (in the interests time and efficiency, this may take place at some convenient moment in the last phase); the group and permissions of the team room space should be configured to permit only the presenting team to speak; the virtual audio recorder should be used to capture the presentations, although only the test monitor should be able to operate the recorder; the guest observers should correctly appear for all of the participants.

6.4.2.5 Wrap-up Phase (15 minutes)

This phase is used to handle any overrun during the research or presentations, and offers the chance for participants to mill around, socialize, and try things out.
Figure 6.8. TeamB gives their presentation.
in Wonderland. This includes watching the movie in the screening room, starting up any of the X Window applications, and interacting with any other Wonderland feature.

Test goals: the test monitor should recursively change the permissions for the demo, team, and screening rooms to enable any participant to interact with and alter the objects contained therein. (The X Window applications default to world-accessible permissions when created.)

6.4.3 Schedule of Test Goals

Table 6.1 provides a concise schedule of all the test goals discussed above. In essence, this table is a check list for elements of the demonstration. Headings along
the top include the demonstration phase followed by the five use case scenarios that informed WonderDAC’s design.
<table>
<thead>
<tr>
<th>Phase</th>
<th>Spatial Objects</th>
<th>Non-spatial Objects</th>
<th>Audio Conversations</th>
<th>Avatar Cloaking</th>
<th>GUI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Environment Preparations</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>The test monitor creates/manages/assigns team groups</td>
</tr>
<tr>
<td>Participant Login</td>
<td>Teams can access/view only their respective lofts</td>
<td>Teams can access only their respective lofts' objects</td>
<td>Teams cannot hear into each other's lofts</td>
<td>Guest observers are hidden from teams</td>
<td>Guest observers enable avatar cloaking</td>
</tr>
<tr>
<td>Research</td>
<td>Teams can access/view only their respective lofts</td>
<td>Teams can access only their respective lofts' objects</td>
<td>Teams cannot hear into each other's lofts</td>
<td>Guest observers are hidden from teams</td>
<td>-</td>
</tr>
<tr>
<td>Phase</td>
<td>Spatial Objects</td>
<td>Non-spatial Objects</td>
<td>Audio Conversations</td>
<td>Avatar Cloaking</td>
<td>GUI</td>
</tr>
<tr>
<td>-------</td>
<td>-----------------</td>
<td>---------------------</td>
<td>---------------------</td>
<td>----------------</td>
<td>-----</td>
</tr>
<tr>
<td>Presentation -</td>
<td>Only presenting team can use PDF viewer and mic in team room. Only test monitor can use audio recorder</td>
<td>Only presenting team can speak in team room</td>
<td>Only presenting team can speak in team room</td>
<td>Guest observers reveal themselves after first presentation</td>
<td>Test monitor manages groups/permissions for team room, PDF viewer, and mic; operates audio recorder</td>
</tr>
<tr>
<td>Phase</td>
<td>Spatial Objects</td>
<td>Non-spatial Objects</td>
<td>Audio Conversations</td>
<td>Avatar Cloaking</td>
<td>GUI</td>
</tr>
<tr>
<td>---------</td>
<td>-----------------</td>
<td>---------------------</td>
<td>---------------------</td>
<td>----------------</td>
<td>----------------</td>
</tr>
<tr>
<td>Wrap-up</td>
<td>All participants can access all rooms</td>
<td>All participants can use all objects</td>
<td>All participants can speak in all rooms</td>
<td>Test monitor recursively changes groups/permissions for team, screening, demo rooms to enable full access by all participants</td>
<td>Test monitor recursively changes groups/permissions for team, screening, demo rooms to enable full access by all participants</td>
</tr>
</tbody>
</table>
6.4.4 Demonstration Follow-up Survey

In addition to putting WonderDAC’s functionality to the test, we intended that our demonstration draw out opinions and perspectives regarding WonderDAC’s use and basic approach. For example, how did participants find access restrictions while using the Wonderland CVE? Were such restrictions useful or in the way? Also, did WonderDAC’s use of roles (i.e., owner, group, and other) and permissions (i.e., interact and alter) make sense? Finally, was WonderDAC’s user interface sensible and easy to use? The general content of these questions was addressed at the end of our demonstration in the form of a short survey. The five participants were asked to respond to ten questions by using one of two rating scales for each question, depending on context. The scales are given in Table 6.4.4:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Disagree</td>
<td>1</td>
<td>Ineffective</td>
</tr>
<tr>
<td>2</td>
<td>Somewhat disagree</td>
<td>2</td>
<td>Somewhat ineffective</td>
</tr>
<tr>
<td>3</td>
<td>Have no opinion</td>
<td>3</td>
<td>Neutral</td>
</tr>
<tr>
<td>4</td>
<td>Somewhat agree</td>
<td>4</td>
<td>Somewhat effective</td>
</tr>
<tr>
<td>5</td>
<td>Agree</td>
<td>5</td>
<td>Effective</td>
</tr>
</tbody>
</table>

189
The survey questions, given in Table 6.3 and included in Appendix D, are generally phrased as statements with which the participant was asked to register some form agreement or disagreement. The results of the demonstration and survey are provided in Section 6.5.

### TABLE 6.3

DEMONSTRATION SURVEY QUESTIONS

<table>
<thead>
<tr>
<th>Question</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Discretionary access control based on roles (owner, group, and other) and only two permissions (interact and alter) seems straight-forward and intuitive.</td>
</tr>
<tr>
<td>2</td>
<td>How effective was access control for entering, seeing, and hearing into spaces (e.g., the team lofts and the screening room)?</td>
</tr>
<tr>
<td>3</td>
<td>Spaces for which one has no interact permission are completely removed from one’s perspective of the virtual world. In practice, this metaphor worked well.</td>
</tr>
<tr>
<td>4</td>
<td>How effective was access control for viewing and using non-spatial objects (e.g., X Window applications, whiteboards, PDF viewers, video viewers, 3D models)?</td>
</tr>
<tr>
<td>5</td>
<td>Non-spatial objects for which one has no interact permission are completely removed from one’s perspective of the virtual world. In practice, this metaphor worked well.</td>
</tr>
<tr>
<td>6</td>
<td>How effective was access control for restricting audio conversation in the team room (during presentations) and in the screening room (during the wrap-up phase)?</td>
</tr>
</tbody>
</table>
TABLE 6.3

Continued

<table>
<thead>
<tr>
<th>Question</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>The avatar cloak may be used to restrict the ability of participants to see and hear one’s avatar. Avatar’s for which one has no interact permission are completely removed from one’s perspective of the virtual world. In practice, this metaphor worked well.</td>
</tr>
<tr>
<td>8</td>
<td>By right-clicking on the walls of a space, on a 3D non-spatial object, or on the DAC button of a 2D window, one can activate the WonderDAC Object Configuration dialog box. This interface was sensible and concise.</td>
</tr>
<tr>
<td>9</td>
<td>By selecting Avatar Cloak from the Wonderland client Tools menu, one can activate the Avatar Cloak Properties dialog box. This interface was sensible and concise.</td>
</tr>
<tr>
<td>10</td>
<td>By selecting Ad Hoc Groups from the Wonderland client Tools menu, one can activate the Ad Hoc Group Builder/Maintainer dialog box. This interface was sensible and concise.</td>
</tr>
</tbody>
</table>

6.5 Evaluation Results

The outcome of our demonstration offered two perspectives of the Wonderland CVE extended by WonderDAC. One addresses operational correctness, while the other is concerned with the end-user experience. By operational correctness, we mean WonderDAC’s ability to function in expected and accurate ways. Neither the participants nor the guest observer experienced any significant events that appeared unusual or incorrect. Also, the test monitor had no problems configuring
access control within the virtual world before and during the demonstration. Participants were able to complete all scheduled activities and deliver coherent (if not brief) presentations about the history of hypertext. Afterwards, they were able to experiment and interact with various WonderDAC dialogs. In short, the demonstration went according to plan, regarding WonderDAC’s capabilities. Moreover, as Figure 6.10 indicates, participant responses to the survey questions were basically positive about the demonstration experience and WonderDAC’s features.

The outlier in this figure is Q10, the final question of the survey that asks about WonderDAC’s ad hoc groups interface. Here, all but one of the participants answered the question with “Have no opinion,” which maps to a value of three. During the last fifteen minutes of the demonstration, participants were
given the chance to experiment with the virtual environment and more directly access WonderDAC’s features. Because this phase of the demonstration was not as regimented as the others, we believe that most of the participants did not take the opportunity to try out WonderDAC’s ad hoc groups feature. This, in turn, lead to a response that was closer to the middle of the scale.

6.6 Scalability and Benchmarking

Generally speaking, end-user experience relative to Wonderland (how participants perceive the CVE and its immersive qualities) is not a significant factor for WonderDAC—other parties are involved in improving this aspect of Wonderland. Nevertheless, we are at least concerned that WonderDAC not introduce any undue delay or resource contention within Wonderland. To that end, we include a brief discussion of scalability issues along with a synopsis of the performance we observed in our demonstration.

To begin with, it is important to keep in mind that Wonderland, although functional, is still at a very early stage of development. Version 0.4 (which WonderDAC extends) is designed to handle at most 25 simultaneous participants whose avatars are visually near one and other. Sun Microsystem’s Wonderland development team notes that their goal is to double this number in the forthcoming version 0.5 and to explore methods of significantly increasing this number in future releases. Methods to improve scalability might include restricting avatar level-of-detail and movement when there is a large crowd present, and the use of federation to simulcast events (such as presentations) into dispersed virtual areas [135].

The bottleneck for concurrent participants has both client and server components. On the Wonderland client, any given participant’s workstation has lim-
its to what its graphics adapter can support in terms of rendering avatars and other content. On the server, whenever an avatar moves, a revalidation of the corresponding client’s visible world takes place. This entails sending an update message to those clients of nearby avatars that can ”see” the movement; as the number of nearby avatars increases, so do the revalidation activities and update messages. Wonderland’s audio and shared X Window capabilities can further add to the scalability issues, although their impact may be throttled by participants selecting low-quality sound and closing unused X Window programs.

In terms of scaling with the number of concurrent users, WonderDAC adds incrementally to Wonderland’s performance. Behind the server revalidation and update messages just described, a loop is triggered for a given client’s server-side thread whenever that client’s avatar moves. This loop refreshes both the client and nearby client perspectives of the virtual world as a result of the movement. WonderDAC hooks this loop so that, whenever a client and its surroundings are revalidated, permissions checks simultaneously take place. On occasion, execution of the revalidation loop is forced as a result of certain WonderDAC activities. For example, if a participant should choose to cloak their avatar, revalidation would take place for those nearby clients who should no longer be able to see the cloaked avatar. On the Wonderland client, WonderDAC operates in a purely cosmetic fashion to prevent a participant from making local changes to an object for which they have no update permissions. (If a malicious user were to bypass these cosmetic controls, they could change their locally rendered object, but any update messages transmitted to the server would be ignored. This would lead to a situation where the object is rendered correctly for every client except the malicious user.)
Because we do not perceive WonderDAC to be a limiting factor for the overall performance of Wonderland 0.4, we did not attempt to benchmark a non-WonderDAC system against a WonderDAC system. However, we did track CPU and swap space usage during our demonstration and believe the results correspond to what we describe above. Specifically, server load averages for one, five, and 15 minute intervals never came close to the threshold value of one; this indicated that, within the periods sampled, no processes were ever waiting for service by the CPU. Moreover, at no time did the server utilize swap space. Figure 6.11 provides a screenshot of the Top utility operating on the Wonderland server halfway through the demonstration.

Unfortunately, the speediness of the Wonderland server did not translate to the client programs used by the participants and guest observer—although we do not believe this was due to WonderDAC. The underlying client operating systems appeared to function well during the demonstration, but off-and-on sluggishness was present for some participants. A few times, participants had to quit and restart their Wonderland clients due to freeze-ups. Differing with these experiences, the test monitor’s Wonderland client exhibited continuous, responsive behavior; in this case, however, a more powerful graphics adapter was at the test monitor’s disposal (see subsection 6.4.1). We believe the cause of the client-side sluggishness was due to the combination of Java3D and the ATI Radeon X1600 graphics adapters used by the participants and guest observer. Prior to the demonstration, in fact, we had to upgrade the participant workstations’ ATI drivers to the latest release in order for the Wonderland client to execute at all. Even then, however, there were consistent issues with textures and colors rendered on the participants’ and guest observer’s machines.
Figure 6.11. A screen snapshot of Top during the high-point of Wonderland activity.
In the end, WonderDAC appears able to protect Wonderland components and content as intended. Regarding scalability, Sun claims that Wonderland 0.4 can scale to no more than about 25 simultaneous users. Version 0.5 should approximately double this number, while future versions will attempt more significant improvements. We believe WonderDAC’s implementation has only incremental affect on the Wonderland server. This is a result of WonderDAC’s access control checks taking place during the execution of the server-side revalidation loop. Beyond Wonderland 0.4’s known user limit, the graphics adapters (Sun recommends NVidia cards and drivers) used by client workstations play a significant role in the immersive qualities experienced by participants.

Despite the outcome of our demonstration and the known limitations of Wonderland 0.4, we intend to carry out more extensive scalability testing beginning with version 0.5. The basic test scenario will remain the same, although we would repeat the test for twelve, twenty-four, and forty-eight users, with and without WonderDAC. To better accommodate the increase in users and to more accurately reveal stress on any server components, we will distribute our Wonderland server resources over three separate machines: the primary server, the immersive audio server, and the Web server (used to transmit texture and model files to the client). Measurements of both server and client activities will be handled in a fashion external to the Wonderland/WonderDAC software (i.e., source code debugging and profiling will be disabled in order to allow a more realistic picture of load and scalability to emerge). Standard system monitoring tools can be used to track server and client workstation performance. In the event that a significant issue is detected, follow-up tests will be carried out with source code profiling capabilities engaged.
6.7 Other Wonderland/WonderDAC Considerations

Although the evaluation results generally indicate that WonderDAC is effective at access control, there are several other issues we must also consider. We begin with Wonderland’s use of X Window applications within its virtual environment. The utility of this feature was heavily leveraged in our demonstration, and clearly offers a convenient means for extending Wonderland’s collaborative nature. However, by integrating X Window applications within Wonderland, dangerously easy and direct access is provided to the underlying server. Furthermore, X Window applications may potentially be executed by Wonderland’s Server Master Client (the SMC provides server-side X Window capability) under the same operating system user ID as other primary Wonderland components (i.e., the server and jVoiceBridge). In our demonstration, for example, a participant could have used FireFox or Impress to browse and possibly edit various system- and Wonderland-related files. This risk can be somewhat mitigated by operating the SMC within a so-called chroot jail on an independent host.\(^1\) Of course, having to manage a separate server for X Window integration while correctly and securely maintaining a chroot jail significantly increases the administrative overhead to a Wonderland system.

The non-existence of a root-level (i.e., administrator) account within WonderDAC is another important issue to address. In theory, WonderDAC should never require such an account: as a truly discretionary solution, the management of all access control is left to the owners of virtual objects comprising a Wonderland environment. Additionally, the presence of a root account introduces significant

---

\(^1\)A chroot jail is a directory that, for affected users, is treated as the system’s root, thereby thwarting access to sensitive resources outside of the jail.
risk: root is often the target of malicious activities, and, in the wrong hands, provides unfettered control over an underlying system. Without a root account, if a Wonderland administrator finds themselves in a pinch, they can always edit the role and permission fields within the XML files that describe most virtual objects and spaces. Unfortunately, there is a problem regarding access control of Wonderland’s server-side X Window capabilities. Owned by the SMC dummy account and existing outside of the XML infrastructure of other virtual objects, server-side X Window applications cannot be restricted by a normal participant. For our demonstration, we had to modify WonderDAC to treat a specific account as though it had root privileges; this enabled us to fully control access to the Firefox and Impress applications used by the participants. However, we feel this is an unacceptable solution to the problem. It would be safer and more elegant for a newly created X application to assume the roles and permissions of the space where it resides. We believe this should become possible in the forthcoming 0.5 release of Project Wonderland.

Continuing, we note that there is insufficient logging capabilities within Wonderland’s system components. Although error, informational, and debugging messages may be written to the terminal screen and/or stored in local files, this is a poor substitute for a secure, aggregated logging facility. We hasten to add that our implementation of WonderDAC is just as lacking in this area. Events that should be captured by an appropriate logging facility include, but are not limited to: system accesses (successful and unsuccessful participant logins), all WonderDAC configuration changes, error messages, informational messages, Wonderland

---

2It should be noted that Wonderland’s client-side X Window capability (where a participant using Linux or Solaris starts an X application on their workstation) does not suffer from this problem. Such applications are automatically configured by WonderDAC with the initiating participant as the owner.
management activities (e.g., configuration changes, system restarts), and system exits (participant logoffs).

Next, we consider the WonderDAC user interface. Although it works in a practical sense, WonderDAC’s GUI includes minor aesthetic and functional weaknesses, too. As part of the client software, the WonderDAC GUI is implemented in Java Swing with simple menus and text boxes. A more visually pleasing way to handle access control management would be through graphical dialogs that operate within the Wonderland client’s window (as opposed to on top of or around this window). This would also align better with the operation of virtual objects such as the PDF, video, and VNC viewers which already include in-world controls. Functionally, it is desirable that right-clicking on any virtual object brings up the WonderDAC Object Configuration dialog box. Unfortunately, due to the limited way mouse events are handled by objects enclosed in two-dimensional frames (e.g., the whiteboard, the PDF viewer, X Window applications), a button that specifically invokes the WonderDAC dialog had to be added to all such objects. This work-around compels a participant to go a little out of their way when configuring access control for two-dimensional assets. Also, there is no direct means for a participant to determine the system and ad hoc groups to which they belong. To some extent, this information can be gleaned by examining the DAC configuration of virtual objects: the participant belongs to any group listed. However, this is clearly not an expedient way to obtain one’s group affiliations.

Finally, as was discussed in the previous chapter, Wonderland is missing the network security features requisite for privacy and non-repudiation. Data stream encryption can enable privacy and integrity at the network level, while digital certificates could be employed to help prove the identities of participants and
Wonderland system components. The absence of these security measures under-
mines the ability of WonderDAC to operate reliably and with assurance.

6.8 Conclusion and Future Work

We devised and implemented WonderDAC in response to the limited and some-
times absent means of access control within CVEs. As an extension to Project
Wonderland, WonderDAC enables access control management by the owners of
virtual objects inside a given Wonderland world. All objects are included: spaces,
three-dimensional assets, two-dimensional images and videos, avatars, and sound.
In addition, WonderDAC offers a simple, ubiquitous way to handle objects by
configuring two permissions (interact and alter) for each of three possible roles
(owner, group, and other). This approach is further enhanced by allowing par-
ticipants to create and manage ad hoc groups and apply these groups to objects
they own.

In an effort to explore WonderDAC’s utility, we conducted a demonstration
in which a small group of graduate students divided into two teams, researched a
topic, and then presented their research—all within the Wonderland CVE. During
the demonstration, WonderDAC was used to maintain privacy and integrity within
the virtual world and to ensure an orderly progression of activities during the
presentations (e.g., only one team at a time could speak or use the PDF viewer).
Survey responses by the demonstration’s participants were largely positive about
WonderDAC’s operation and interface.

Despite what we perceive to be a successful demonstration, important ancil-
lary issues remain to be tackled before WonderDAC is a fully secure solution. For
example, the ability to deploy X Window applications in Wonderland introduces
great flexibility. For X applications that originate on the server-side, however, this feature can provide a malicious participant with a doorway to wherever the application is executing, and introduces access control management difficulties. Also, secure logging of Wonderland and WonderDAC events is lacking. This makes it quite difficult to track down and resolve anomalies and issues of abuse. The WonderDAC interface needs improvements, too. Here, it is desirable that a participant use dialogs and menus integrated within the graphical, three dimensional Wonderland environment—as opposed to the Java Swing dialogs that occur on the participant’s desktop. Finally, network security features such as data stream encryption and digital certificates should be employed to protect Wonderland communications and ensure non-repudiation. WonderDAC can only be as reliable as Wonderland’s communications are secure.
CHAPTER 7

CONCLUSION

7.1 Summary and Contributions

This dissertation investigates controls that can protect the privacy and integrity of objects residing in collaborative, virtual environments (CVEs). Such environments leverage desktop virtual reality technologies to enable collaborative activities among multiple, simultaneous participants. Popular uses of CVEs include, but are not limited to: entertainment, commerce, education, and research. Success in each of these endeavors requires the means to ensure privacy and integrity for participant activities and the objects upon which participants act. Our exploration of CVE privacy and integrity controls has led to the following research contributions.

7.1.1 CVE Technology Survey

We began with a survey of 16 current technologies that may be used to build a CVE. We grouped these along three axes (APIs, frameworks, and platforms) and then undertook an analysis to determine the least risky (from a development perspective) for constructing a CVE.
7.1.2 Comparison of Peer-to-Peer and Client-Server CVE Architectures

Next, we considered integrity with respect to the interactive experience of a CVE participant. Specifically, we looked at the Croquet peer-to-peer CVE and its ability to guarantee synchronicity and atomicity in activities. We took advantage of this to build a virtual emergency operations center prototype.

We then explored integrity and privacy issues related to VR objects (e.g., three-dimensional spaces and assets, avatars, images, sounds) within the Project Wonderland client-server CVE. Here, we leveraged the Wonderland server to implement a prototype, discretionary access control (DAC) system, called WonderDAC, capable of restricting VR object interactions and updates.

7.1.3 Practical Implementation and Demonstration of CVE Discretionary Access Controls

Continuing our work with Wonderland, we refined WonderDAC to evolve it from a simple prototype to a functioning, usable, access control solution. We gave further consideration to the details of its implementation and discussed some key areas of weakness. Sun Microsystems, the proprietor for Project Wonderland, has plans to integrate WonderDAC into Wonderland’s forthcoming 0.5 release.

Finally, we completed our investigation of Wonderland and WonderDAC with a realistic, but small-scale demonstration. Here, a modest group of participants was asked to research a topic, build presentations, and deliver talks—entirely within the Wonderland CVE. We highlighted how, during these activities, WonderDAC provided privacy and integrity that otherwise would have been absent. In effect, WonderDAC helped to protect and even guide the demonstration.
7.2 Limitations and Future Research

We end with a brief discussion of Wonderland/WonderDAC limitations and possible, future research efforts including the exploration of a novel approach to CVEs and access control.

Further development related to Wonderland and WonderDAC is needed to address a variety of important issues, such as handling X Window applications, logging, improving user interfaces, and securing network communications. Wonderland’s current server-side use of X Window leads to X application objects that cannot be appropriately restricted by WonderDAC. Significant work is required to enhance the security of X Window applications in Wonderland and better integrate them with WonderDAC. Logging is currently handled in a crude fashion, with debugging and informational messages displayed to a terminal screen and/or stored in local files. Improved logging is needed to more accurately track Wonderland/WonderDAC activities and problems—securely centralizing logs on a dedicated server should be investigated. The WonderDAC user interface dialogs should be moved from Java Swing into the Wonderland client’s graphical window. Ideally, this would lead to greater parity with the look and feel of other Wonderland object controls. Also, WonderDAC is missing a way for participants to interact with their accounts (e.g., to change a password); at present, such activities may be carried out on the participant’s behalf by a Wonderland administrator using an LDAP browser. Lastly, the security of network communications among Wonderland server components and between the Wonderland client and server must be addressed. This includes encrypting data traffic and providing some means of non-repudiation (most likely through the use of digital certificates).

Although we found Wonderland to be an effective CVE and WonderDAC to
be a viable access control solution, we note that, in some circumstances, the two may be excessive. For example, CVE platforms such as Project Wonderland are appropriate when a permanent virtual world is needed to provide service to an established group of simultaneous participants. Within such a CVE, an access control approach similar to WonderDAC’s is reasonable, given the depth and scope of the virtual world. But what about more simplistic needs? What if only a handful of participants, who may even know one and other, wish to collaborate in a temporary virtual world for just a few hours? We believe that this scenario may be a common one, and while there are certainly tools available to facilitate some short-lived, collaborative interactions (e.g., web cameras, text and audio chat, shared virtual whiteboards), these may not be a useful substitute for the immersive qualities of a CVE. Thus, we propose investigation into a novel type of virtual environment that we term the micro-CVE.

A micro-CVE would include the collaborative power of a normal CVE, but only be capable of hosting a small, virtual world and an equally small number of simultaneous participants (e.g., no more than ten). Architecturally, the server component of this system would be streamlined and compact enough to operate on a typical, modern desktop. Internal access control would be patterned after WonderDAC, but simplified to be commensurate with the requirements of an ephemeral, virtual world: no need for an external database to track accounts, only ad hoc group roles, no avatar cloaking, and no need to permanently store DAC configurations. Implementation options must be carefully explored. One possibility would be to significantly scale down Project Wonderland—the benefit, here, is the possible reuse of Wonderland objects in a micro-CVE system, as well as potentially being able to connect to a normal Wonderland world in some fashion.
APPENDIX A

VEOC SOURCE CODE EXCERPTS

A.1 XML Handler

The following excerpt comprises the XML parser created for the vEOC project on the Croquet platform. All of the code is Squeak, a derivative of the Smalltalk 80 language.

Object subclass: #ImportXML
instanceVariableNames: 'errors domParser simulationXMLElement directiveList fileName'
classVariableNames: ''
poolDictionaries: ''
category: 'Wisconsin-Worlds'

directiveList
^ directiveList.
dumpDirectives
"Dump the directives in the directiveList SortedCollection."
directiveList doWithIndex: [:element :index |
Transcript cr.
Transcript show: '===================================='; cr.
initialize
errors := OrderedCollection new.
directiveList := OrderedCollection new.
simulationXMLElement := nil.
domParser := nil.
simulationXMLElement := nil.

parseFileNamed: parseFileName
| aStream |

fileName := parseFileName.

"Open the file and parse it."

aStream := (CrLfFileStream readOnlyFileNamed: fileName) ascii.

[self parseStream: aStream] ensure: [aStream close].

"Print out problems parsing the file and cleanup if necessary."

(errors size > 0) ifTrue: [
self showErrors.
].

parseStream: aStream

"Parse through the file."

domParser := XMLDOMParser parseDTDFrom: aStream.

self processFile.
processDirectives

| timeOffset timeInject deviceType deviceNumber filePath directive sender
| receiver messageType message action explanObj |

sender := OrderedCollection new.
receiver := OrderedCollection new.
simulationXMLElement tagsNamed: #simulationScript do: [ :xmlRecords |
xm1Records tagsNamed: #rsrccRecord do: [ :xmlRecord |
"Resource record time."

xmlRecord tagsNamed: #time do: [ :timeValue |
timeOffset := timeValue attributeAt: #offset ifAbsent: [
errors add: 'No attribute ''offset'' in rsrcRecord time tag.',
String cr.
].
timeInject := timeValue attributeAt: #injectNum ifAbsent: [
errors add: 'No attribute ''injectNum'' in rsrcRecord time tag.',
String cr.
].
].
"Resource record device ID."

xmlRecord tagsNamed: #device do: [ :deviceID |
deviceType := deviceID attributeAt: #type ifAbsent: [
errors add: 'No attribute '''type''' in rsrcRecord device tag.',
String cr.
].
deviceNumber := deviceID attributeAt: #devID ifAbsent: [
errors add: 'No attribute 'devID' in rsrcRecord device tag.', String cr.
]
]

filePath := xmlRecord contentStringAt: #file.

"Build a resource record directive object and add it to our ordered collection. (Note that the record type [resource or message] is implicitly handled by the ResourceRecord object, but the device type is not.)"
directive := ResourceRecord new.
directive time: timeOffset.
directive injectNumber: timeInject.
directive device: deviceNumber.
directive deviceType: deviceType.
directive file: filePath.

"Add this new directive to the SortedCollection 'directiveList'."
directiveList add: directive.
]

xmlRecords tagsNamed: #msgRecord do: [ :xmlRecord |

messageType := xmlRecord attributeAt: #type ifAbsent: [ errors add: 'No attribute 'type' in msgRecord tag.', String cr. ].

"Message record time."
xmlRecord tagsNamed: #time do: [:timeValue |
timeOffset := timeValue attributeAt: #offset ifAbsent: [errors add: 'No attribute "offset" in msgRecord time tag.';
String cr.].
timeInject := timeValue attributeAt: #injectNum ifAbsent: [errors add: 'No attribute "injectNum" in msgRecord time tag.';
String cr.].
].

"Message record sender."

xmlRecord tagsNamed: #sender do: [:sendingAgency |
sender add: (sendingAgency contentStringAt: #agency).].

"Message record receiver."

xmlRecord tagsNamed: #receiver do: [:receivingAgency |
receiver add: (receivingAgency contentStringAt: #agency).].

"Message record message content."

message := xmlRecord contentStringAt: #message.

"Message record action content."

action := xmlRecord contentStringAt: #action.
"Message record explanation object code."

explanObj := xmlRecord contentStringAt: #explanObj.

"Build a message record directive object and add it to our ordered collection. (Note that the record type [resource or message] is implicitly handled by the ResourceRecord object, but the message type is not.)"

directive := MessageRecord new.
directive time: timeOffset.
directive injectNumber: timeInject.
directive messageType: messageType.
directive sender: sender.
directive receiver: receiver.
directive message: message.
directive action: action.
directive explanObj: explanObj.

"Add this new directive to the SortedCollection 'directiveList'."
directiveList add: directive.
].
].

processFile

"Get the parsed XML document."
simulationXMLElement := domParser document.
"Build the simulation directives."
self processDirectives.

"Dump the directives in the directiveList SortedCollection."

"self dumpDirectives."

showErrors
| str |

"Display any errors that have accumulated."

errors do: [ :error | Transcript show: error ].

"Last and Final message."

str := 'Errors occurred in file: ', fileName, String cr.
str := str, 'vEOC directives not configured!!', String cr.
Transcript show: str.
APPENDIX B

PROJECT WONDERLAND TEST WORLD XML FILES

B.1 TestRoomA-WLC.xml

The following listing comprises the XML code to describe testRoomA in the Wonderland virtual test world from Chapter 4.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<javaversion="1.5.0_13" class="java.beans.XMLDecoder">
<object class="org.jdesktop.lg3d.wonderland.darkstar.server.setup.BasicCellGLOSetup">
  <void property="cellGLOClassName">
    <string>org.jdesktop.lg3d.wonderland.extracells.server.MultiModelCellGLO</string>
  </void>
  <void property="origin">
    <double>50.0</double>
  </void>
  <void property="origin">
    <double>50.0</double>
  </void>
  <void property="boundsRadius">
    <double>100.0</double>
  </void>
  <void property="boundsType">
    <string>BOX</string>
  </void>
  <void property="accessOwner">
    <string>twright</string>
  </void>
  <void property="accessGroup">
    <string>admin</string>
  </void>
</object>
</java version="1.5.0_13" class="java.beans.XMLDecoder">
<void property="accessGroupPermissions">
        <string>3</string>
    </void>

<void property="accessOtherPermissions">
        <string>2</string>
    </void>

<void property="cellSetup">
    <object class="org.jdesktop.lg3d.wonderland.extracells.common.MultiModelCellSetup">
        <void property="models">
            <array class="org.jdesktop.lg3d.wonderland.extracells.common.MultiModelCellModel">
                <object class="org.jdesktop.lg3d.wonderland.extracells.common.MultiModelCellModel">
                    <void property="modelFile">
                        <string>models/TestRoomA.j3s.gz</string>
                    </void>
                </object>
            </array>
        </void>
    </object>
</void>

<-- Common Space Tables and Chairs-->

<object class="org.jdesktop.lg3d.wonderland.extracells.common.MultiModelCellModel">
    <void property="modelFile">
        <string>models/default_building/teamroom_table.j3s.gz</string>
    </void>
    <void property="location">
        <array class="double">
            <double>-20.0</double>
            <double>0.5</double>
            <double>45.0</double>
        </array>
    </void>
</object>

<object class="org.jdesktop.lg3d.wonderland.extracells.common.MultiModelCellModel">
    <void property="modelFile">
        <string>models/default_building/chair.j3s.gz</string>
    </void>
    <void property="location">
        <array class="double">
            <double>-1.0</double>
            <double>1.1</double>
            <double>-5.2</double>
        </array>
    </void>
    <void property="instanted">
        <boolean>true</boolean>
    </void>
</object>

<object class="org.jdesktop.lg3d.wonderland.extracells.common.MultiModelCellModel">
    <array class="double">
        <double>-20.0</double>
        <double>0.5</double>
        <double>45.0</double>
    </array>
</object>

<object class="org.jdesktop.lg3d.wonderland.extracells.common.MultiModelCellModel">
    <array class="double">
        <double>-1.0</double>
        <double>1.1</double>
        <double>-5.2</double>
    </array>
</object>

<object class="org.jdesktop.lg3d.wonderland.extracells.common.MultiModelCellModel">
    <array class="double">
        <double>-20.0</double>
        <double>0.5</double>
        <double>45.0</double>
    </array>
</object>

<object class="org.jdesktop.lg3d.wonderland.extracells.common.MultiModelCellModel">
    <array class="double">
        <double>-1.0</double>
        <double>1.1</double>
        <double>-5.2</double>
    </array>
</object>
<object class="org.jdesktop.lg3d.wonderland.extracells.common.MultiModelCellModel">
    <void property="modelFile">
        <string>models/default_building/teamroom_table.j3s.gz</string>
    </void>
    <void property="location">
        <array class="double">
            <double>1.0</double>
            <double>1.1</double>
            <double>-3.3</double>
        </array>
    </void>
    <void property="rotation">
        <array class="double">
            <double>0.0</double>
            <double>1.0</double>
            <double>0.0</double>
            <double>3.9</double>
        </array>
    </void>
    <void property="instance">true</void>
</object>
<array class="double">
  <double>-20.0</double>
  <double>.5</double>
  <double>52.0</double>
</array>
</object>
</object>
<array class="double">
  <double>1</double>
  <double>1.1</double>
  <double>3.2</double>
</array>
</object>
<array class="double">
  <double>0.0</double>
  <double>1.0</double>
  <double>0.0</double>
  <double>2.62</double>
</array>
</object>

</object>

<!-- Workstation on Teamroom Table -->
<object class="org.jdesktop.lg3d.wonderland.extracells.common.MultiModelCellModel">
  <void property="modelFile">
    <string>models/default/building/chair.j3s.gz</string>
  </void>
  <void property="location">
    <array class="double">
      <double>-1</double>
      <double>1.1</double>
      <double>5.8</double>
    </array>
  </void>
  <void property="rotation">
    <array class="double">
      <double>0.0</double>
      <double>1.0</double>
      <double>0.0</double>
      <double>2.62</double>
    </array>
  </void>
  <void property="instanced">
    <boolean>true</boolean>
  </void>
</object>
<object class="org.jdesktop.lg3d.wonderland.extracells.common.MultiModelCellModel">
  <void property="modelFile">
    <string>models/default/building/chair.j3s.gz</string>
  </void>
  <void property="location">
    <array class="double">
      <double>-1</double>
      <double>1.1</double>
      <double>5.8</double>
    </array>
  </void>
  <void property="rotation">
    <array class="double">
      <double>0.0</double>
      <double>1.0</double>
      <double>0.0</double>
      <double>2.62</double>
    </array>
  </void>
  <void property="instanced">
    <boolean>true</boolean>
  </void>
</object>
<object class="org.jdesktop.lg3d.wonderland.extracells.common.MultiModelCellModel">
  <void property="modelFile">
    <string>models/default_building/teamroom_filecabinet.j3s.gz</string>
  </void>
  <void property="location">
    <array class="double">
      <double>-14</double>
      <double>1.55</double>
      <double>-9</double>
    </array>
  </void>
  <void property="rotation">
    <array class="double">
      <double>0.0</double>
      <double>-1.0</double>
      <double>0.0</double>
      <double>3.2</double>
    </array>
  </void>
  <void property="instanced">
    <boolean>true</boolean>
  </void>
</object>

<object class="org.jdesktop.lg3d.wonderland.extracells.common.MultiModelCellModel">
  <void property="modelFile">
    <string>models/default_building/teamroom_filecabinet.j3s.gz</string>
  </void>
  <void property="location">
    <array class="double">
      <double>-14</double>
      <double>1.55</double>
      <double>-9</double>
    </array>
  </void>
  <void property="rotation">
    <array class="double">
      <double>0.0</double>
      <double>-1.0</double>
      <double>0.0</double>
      <double>3.2</double>
    </array>
  </void>
  <void property="instanced">
    <boolean>true</boolean>
  </void>
</object>
<!-- Common Space File Cabinets -->

<object class="org.jdesktop.lg3d.wonderland.extracells.common.MultiModelCellModel">
  <void property="modelFile">
    <string>models/default_building/teamm_room_filecabinet.j3s.gz</string>
  </void>
  <void property="location">
    <array class="double">
      <double>-5</double>
      <double>1.55</double>
      <double>3.2</double>
    </array>
  </void>
  <void property="instancedException">
    <boolean>true</boolean>
  </void>
</object>

<object class="org.jdesktop.lg3d.wonderland.extracells.common.MultiModelCellModel">
  <void property="modelFile">
    <string>models/default_building/teamm_room_filecabinet.j3s.gz</string>
  </void>
  <void property="location">
    <array class="double">
      <double>-5</double>
      <double>1.55</double>
      <double>4.4</double>
    </array>
  </void>
  <void property="instancedException">
    <boolean>true</boolean>
  </void>
</object>

<object class="org.jdesktop.lg3d.wonderland.extracells.common.MultiModelCellModel">
  <void property="modelFile">
    <string>models/default_building/teamm_room_filecabinet.j3s.gz</string>
  </void>
  <void property="location">
    <array class="double">
      <double>-5</double>
      <double>1.55</double>
      <double>5.6</double>
    </array>
  </void>
  <void property="instancedException">
    <boolean>true</boolean>
  </void>
</object>

<object class="org.jdesktop.lg3d.wonderland.extracells.common.MultiModelCellModel">
  <void property="modelFile">
    <string>models/default_building/teamm_room_filecabinet.j3s.gz</string>
  </void>
  <void property="location">
    <array class="double">
      <double>-5</double>
      <double>1.55</double>
      <double>5.6</double>
    </array>
  </void>
  <void property="instancedException">
    <boolean>true</boolean>
  </void>
</object>
<string>models/default_building/tearoom_filecabinet.j3s.gz</string>
</void>
</void>
<array class="double">
  <double>-5</double>
  <double>1.55</double>
  <double>-2</double>
</array>
</void>
<void property="instanced">
  <boolean>true</boolean>
</void>
</object>
<object class="org.jdesktop.lg3d.wonderland.extracells.common.MultiModelCellModel">
  <void property="modelFile">
    <string>models/default_building/tearoom_filecabinet.j3s.gz</string>
  </void>
  <void property="location">
    <array class="double">
      <double>5.1</double>
      <double>1.55</double>
      <double>-2</double>
    </array>
  </void>
  <void property="rotation">
    <array class="double">
      <double>0.0</double>
      <double>-1.0</double>
      <double>0.0</double>
      <double>3.2</double>
    </array>
  </void>
  <void property="instanced">
    <boolean>true</boolean>
  </void>
</object>
<object class="org.jdesktop.lg3d.wonderland.extracells.common.MultiModelCellModel">
  <void property="modelFile">
    <string>models/default_building/tearoom_filecabinet.j3s.gz</string>
  </void>
  <void property="location">
    <array class="double">
      <double>5.1</double>
      <double>1.55</double>
      <double>3.2</double>
    </array>
  </void>
  <void property="rotation">
    <array class="double">
      <double>0.0</double>
    </array>
  </void>
</object>
<object class="org.jdesktop.lg3d.wonderland.extracells.common.MultiModelCellModel">
  <void property="modelFile">
    <string>models/default_building/teamroom_filecabinet.j3s.gz</string>
  </void>
  <void property="location">
    <array class="double">
      <double>5.1</double>
      <double>1.55</double>
      <double>4.4</double>
    </array>
  </void>
  <void property="rotation">
    <array class="double">
      <double>0.0</double>
      <double>-1.0</double>
      <double>0.0</double>
      <double>3.2</double>
    </array>
  </void>
  <void property="instanced">
    <boolean>true</boolean>
  </void>
</object>

<object class="org.jdesktop.lg3d.wonderland.extracells.common.MultiModelCellModel">
  <void property="modelFile">
    <string>models/default_building/teamroom_filecabinet.j3s.gz</string>
  </void>
  <void property="location">
    <array class="double">
      <double>5.1</double>
      <double>1.55</double>
      <double>5.6</double>
    </array>
  </void>
  <void property="rotation">
    <array class="double">
      <double>0.0</double>
      <double>-1.0</double>
      <double>0.0</double>
      <double>3.2</double>
    </array>
  </void>
  <void property="instanced">
    <boolean>true</boolean>
  </void>
</object>
<object class="org.jdesktop.lg3d.wonderland.extracells.common.MultiModelCellModel">
  <void property="modelFile">
    <string>models/default_building/teamroom_desk.j3s.gz</string>
  </void>
  <void property="location">
    <array class="double">
      <double>8.1</double>
      <double>9.0</double>
      <double>-0.45</double>
    </array>
  </void>
  <void property="rotation">
    <array class="double">
      <double>0.0</double>
      <double>1.0</double>
      <double>0.0</double>
      <double>3.1</double>
    </array>
  </void>
  <void property="instanced">
    <boolean>true</boolean>
  </void>
</object>

<object class="org.jdesktop.lg3d.wonderland.extracells.common.MultiModelCellModel">
  <void property="modelFile">
    <string>models/default_building/teamroom_desk.j3s.gz</string>
  </void>
  <void property="location">
    <array class="double">
      <double>14.1</double>
      <double>9.0</double>
    </array>
  </void>
  <void property="rotation">
    <array class="double">
      <double>1.41</double>
      <double>0.9</double>
      <double>0.6</double>
    </array>
  </void>
  <void property="instanced">
    <boolean>true</boolean>
  </void>
</object>

<object class="org.jdesktop.lg3d.wonderland.extracells.common.MultiModelCellModel">
  <void property="modelFile">
    <string>models/default_building/teamroom_desk.j3s.gz</string>
  </void>
  <void property="location">
    <array class="double">
      <double>14.1</double>
      <double>9.0</double>
    </array>
  </void>
  <void property="rotation">
    <array class="double">
      <double>0.0</double>
      <double>-1.0</double>
      <double>0.0</double>
      <double>1.5</double>
    </array>
  </void>
  <void property="instanced">
    <boolean>true</boolean>
  </void>
</object>
<double>-6</double>
</array>
</void>
<void property="rotation">
<array class="double">
<double>0.0</double>
<double>-1.0</double>
<double>0.0</double>
<double>1.5</double>
</array>
</void>
<void property="instanced">
<boolean>true</boolean>
</void>
<object class="org.jdesktop.lg3d.wonderland.extracells.common.MultimodelCellModel">
<void property="modelFile">
<string>models/default_building/teamroom_desk.j3s.gz</string>
</void>
<void property="location">
<array class="double">
<double>-8.1</double>
<double>.9</double>
<double>.45</double>
</array>
</void>
<void property="instanced">
<boolean>true</boolean>
</void>
</object>
<object class="org.jdesktop.lg3d.wonderland.extracells.common.MultimodelCellModel">
<void property="modelFile">
<string>models/default_building/teamroom_desk.j3s.gz</string>
</void>
<void property="location">
<array class="double">
<double>-14.1</double>
<double>.9</double>
<double>.6</double>
</array>
</void>
<void property="rotation">
<array class="double">
<double>0.0</double>
<double>-1.0</double>
<double>0.0</double>
<double>-1.52</double>
</array>
</void>
<void property="instanced">

<boolean>true</boolean>
</void>
</object>

<object class="org.jdesktop.lg3d.wonderland.extracells.common.MultiModelCellModel">
  <void property="modelFile">
    <string>models/default_building/teamroom_desk.js.gz</string>
  </void>
  <void property="location">
    <array class="double">
      <double>-14.1</double>
      <double>-0.9</double>
      <double>-6</double>
    </array>
  </void>
  <void property="rotation">
    <array class="double">
      <double>0.0</double>
      <double>-1.0</double>
      <double>0.0</double>
      <double>-1.52</double>
    </array>
  </void>
  <void property="instanted">
    <boolean>true</boolean>
  </void>
</object>

<object class="org.jdesktop.lg3d.wonderland.extracells.common.MultiModelCellModel">
  <void property="modelFile">
    <string>models/default_building/teamroom_desk.js.gz</string>
  </void>
  <void property="location">
    <array class="double">
      <double>-8.1</double>
      <double>-0.45</double>
    </array>
  </void>
  <void property="rotation">
    <array class="double">
      <double>0.0</double>
      <double>1.0</double>
      <double>0.0</double>
      <double>3.1</double>
    </array>
  </void>
  <void property="instanted">
    <boolean>true</boolean>
  </void>
</object>
<object class="org.jdesktop.lg3d.wonderland.extracells.common.MultiModelCellModel">
    <void property="modelFile">
        <string>models/default_building/chair.j3s.gz</string>
    </void>
    <void property="location">
        <array class="double">
            <double>-8.1</double>
            <double>1.1</double>
            <double>-2</double>
        </array>
    </void>
    <void property="instanced">
        <boolean>true</boolean>
    </void>
</object>

<object class="org.jdesktop.lg3d.wonderland.extracells.common.MultiModelCellModel">
    <void property="modelFile">
        <string>models/default_building/chair.j3s.gz</string>
    </void>
    <void property="location">
        <array class="double">
            <double>-13</double>
            <double>1.1</double>
            <double>-6.1</double>
        </array>
    </void>
    <void property="rotation">
        <array class="double">
            <double>0.0</double>
            <double>1.0</double>
            <double>4</double>
        </array>
    </void>
    <void property="instanced">
        <boolean>true</boolean>
    </void>
</object>

<object class="org.jdesktop.lg3d.wonderland.extracells.common.MultiModelCellModel">
    <void property="modelFile">
        <string>models/default_building/chair.j3s.gz</string>
    </void>
    <void property="location">
        <array class="double">
            <double>-8.1</double>
            <double>1.1</double>
            <double>2</double>
        </array>
    </void>
    <void property="instanced">
        <boolean>true</boolean>
    </void>
</object>
<object class="org.jdesktop.lg3d.wonderland.extracells.common.MultiModelCellModel">
  <void property="modelFile">
    <string>models/defualt_building/chair.j3s.gz</string>
  </void>
  <void property="location">
    <array class="double">
      <double>-13</double>
      <double>1.1</double>
      <double>6.1</double>
    </array>
  </void>
  <void property="rotation">
    <array class="double">
      <double>0.0</double>
      <double>1.0</double>
      <double>0.0</double>
      <double>4</double>
    </array>
  </void>
  <void property="instanced">
    <boolean>true</boolean>
  </void>
</object>

<object class="org.jdesktop.lg3d.wonderland.extracells.common.MultiModelCellModel">
  <void property="modelFile">
    <string>models/defualt_building/chair.j3s.gz</string>
  </void>
  <void property="location">
    <array class="double">
      <double>8.1</double>
      <double>1.1</double>
      <double>-2</double>
    </array>
  </void>
  <void property="rotation">
    <array class="double">
      <double>0.0</double>
      <double>1.0</double>
      <double>0.0</double>
      <double>4</double>
    </array>
  </void>
  <void property="instanced">
    <boolean>true</boolean>
  </void>
</object>

<object class="org.jdesktop.lg3d.wonderland.extracells.common.MultiModelCellModel">
  <void property="modelFile">
    <string>models/defualt_building/chair.j3s.gz</string>
  </void>
  <void property="location">
    <array class="double">
      <double>8.1</double>
      <double>1.1</double>
      <double>-2</double>
    </array>
  </void>
  <void property="rotation">
    <array class="double">
      <double>0.0</double>
      <double>1.0</double>
      <double>0.0</double>
      <double>4</double>
    </array>
  </void>
  <void property="instanced">
    <boolean>true</boolean>
  </void>
</object>
<double>13</double>
<array><double>1.1</double>
<array><double>6.1</double>
</array>
</void>
<void property="rotation">
<array class="double">
<array><double>0.0</double>
<array><double>1.0</double>
<array><double>0.0</double>
<array><double>4</double>
</array>
</array>
</void>
<void property="instanced">
<boolean>true</boolean>
</void>
</object>

<object class="org.jdesktop.lg3d.wonderland.extracells.common.MultiModelCellModel">
<void property="modelFile">
<string>models/default_building/chair.j3s.gz</string>
</void>
<void property="location">
<array class="double">
<array><double>8.1</double>
<array><double>1.1</double>
<array><double>2</double>
</array>
</array>
</void>
<void property="instanced">
<boolean>true</boolean>
</void>
</object>

<object class="org.jdesktop.lg3d.wonderland.extracells.common.MultiModelCellModel">
<void property="modelFile">
<string>models/default_building/chair.j3s.gz</string>
</void>
<void property="location">
<array class="double">
<array><double>13</double>
<array><double>1.1</double>
<array><double>-6.1</double>
</array>
</array>
</void>
<void property="rotation">
<array class="double">
<array><double>0.0</double>
<array><double>1.0</double>
<array><double>0.0</double>
<array><double>1.5</double>
</array>
</array>
</void>
<object class="org.jdesktop.lg3d.wonderland.extracells.common.MultiModelCellModel">
    <void property="modelFile">
        <string>models/default/building/teamroom/table.j3s.gz</string>
    </void>
    <void property="location">
        <array class="double">
            <double>-10.0</double>
            <double>0.5</double>
            <double>55.0</double>
        </array>
    </void>
</object>

<object class="org.jdesktop.lg3d.wonderland.extracells.common.MultiModelCellModel">
    <void property="modelFile">
        <string>models/default/building/teamroom/table.j3s.gz</string>
    </void>
    <void property="location">
        <array class="double">
            <double>-28.0</double>
            <double>0.5</double>
            <double>55.0</double>
        </array>
    </void>
</object>

<object class="org.jdesktop.lg3d.wonderland.extracells.common.MultiModelCellModel">
    <void property="modelFile">
        <string>models/default/building/teamroom/table.j3s.gz</string>
    </void>
    <void property="location">
        <array class="double">
            <double>-10.0</double>
            <double>0.5</double>
            <double>41.0</double>
        </array>
    </void>
</object>

<object class="org.jdesktop.lg3d.wonderland.extracells.common.MultiModelCellModel">
    <void property="modelFile">
        <string>models/default/building/teamroom/table.j3s.gz</string>
    </void>
    <void property="location">
        <array class="double">
            <double>-10.0</double>
            <double>0.5</double>
            <double>41.0</double>
        </array>
    </void>
</object>

<object class="org.jdesktop.lg3d.wonderland.extracells.common.MultiModelCellModel">
    <void property="modelFile">
        <string>models/default/building/teamroom/table.j3s.gz</string>
    </void>
    <void property="location">
        <array class="double">
            <double>-10.0</double>
            <double>0.5</double>
            <double>41.0</double>
        </array>
    </void>
</object>
<array class="double">
  <double>-28.0</double>
  <double>-28.0</double>
  <double>41.0</double>
</array>
</object>

<--- Cube Workstations --->

<object class="org.jdesktop.lg3d.wonderland.extracells.common.MultiModelCellModel">
  <void property="modelFile">
    <string>models/default/building/workstation.j3s.gz</string>
  </void>
  <void property="location">
    <array class="double">
      <double>8.1</double>
      <double>1.48</double>
      <double>-0.45</double>
    </array>
  </void>
  <void property="rotation">
    <array class="double">
      <double>0.0</double>
      <double>-1.0</double>
      <double>0.0</double>
      <double>1.5</double>
    </array>
  </void>
  <void property="instanced">
    <boolean>true</boolean>
  </void>
</object>

<object class="org.jdesktop.lg3d.wonderland.extracells.common.MultiModelCellModel">
  <void property="modelFile">
    <string>models/default/building/workstation.j3s.gz</string>
  </void>
  <void property="location">
    <array class="double">
      <double>14.1</double>
      <double>1.48</double>
      <double>6</double>
    </array>
  </void>
  <void property="rotation">
    <array class="double">
      <double>0.0</double>
      <double>-1.0</double>
      <double>0.0</double>
      <double>3</double>
    </array>
  </void>
</object>
<object class="org.jdesktop.lg3d.wonderland.extracells.common.MultiModelCellModel">
    <void property="modelFile">
        <string>models/default_building/workstation.j3s.gz</string>
    </void>
    <void property="location">
        <array class="double">
            <double>-8.1</double>
            <double>1.48</double>
            <double>.45
        </array>
    </void>
    <void property="rotation">
        <array class="double">
            <double>0.0</double>
            <double>-1.0</double>
            <double>1.5</double>
        </array>
    </void>
    <void property="instance">true</void>
</object>

<object class="org.jdesktop.lg3d.wonderland.extracells.common.MultiModelCellModel">
    <void property="modelFile">
        <string>models/default_building/workstation.j3s.gz</string>
    </void>
    <void property="location">
        <array class="double">
            <double>-14.1</double>
            <double>1.48</double>
            <double>6</double>
        </array>
    </void>
    <void property="rotation">
        <array class="double">
            <double>0.0</double>
            <double>-1.0</double>
            <double>0</double>
        </array>
    </void>
    <void property="instance">true</void>
</object>

<object class="org.jdesktop.lg3d.wonderland.extracells.common.MultiModelCellModel">
    <void property="modelFile">
        <string>
        </string>
    </void>
</object>
<string>models/default_building/workstation.j3s.gz</string>

</void>

<array property="location">
  <double>-8.1</double>
  <double>1.48</double>
  <double>-0.45</double>
</array>

</void>

<array property="rotation">
  <double>0.0</double>
  <double>1.0</double>
  <double>0.0</double>
  <double>1.5</double>
</array>

</void>

<void property="instanced">
  <boolean>true</boolean>
</void>

<object class="org.jdesktop.lg3d.wonderland.extracells.common.MultiModelCellModel">
  <string>models/default_building/workstation.j3s.gz</string>
</object>

<object class="org.jdesktop.lg3d.wonderland.extracells.common.MultiModelCellModel">
  <string>models/default_building/plant.j3s.gz</string>
</object>

<!-- Plants -->

<object class="org.jdesktop.lg3d.wonderland.extracells.common.MultiModelCellModel">
  <string>models/default_building/plant.j3s.gz</string>
</object>
<object class="org.jdesktop.lg3d.wonderland.extracellulars.common.MultiModelCellModel">
  <void property="modelFile">
    <string>models/default_building/plant.j3s.gz</string>
  </void>
  <void property="location">
    <array class="double">
      <double>-14</double>
      <double>-1.7</double>
      <double>-2</double>
    </array>
  </void>
  <void property="instanced">
    <boolean>true</boolean>
  </void>
</object>

<object class="org.jdesktop.lg3d.wonderland.extracellulars.common.MultiModelCellModel">
  <void property="modelFile">
    <string>models/default_building/plant.j3s.gz</string>
  </void>
  <void property="location">
    <array class="double">
      <double>-14</double>
      <double>-1.7</double>
      <double>-2</double>
    </array>
  </void>
  <void property="instanced">
    <boolean>true</boolean>
  </void>
</object>

<object class="org.jdesktop.lg3d.wonderland.extracellulars.common.MultiModelCellModel">
  <void property="modelFile">
    <string>models/default_building/plant.j3s.gz</string>
  </void>
  <void property="location">
    <array class="double">
      <double>-14</double>
      <double>-1.7</double>
      <double>-2</double>
    </array>
  </void>
  <void property="instanced">
    <boolean>true</boolean>
  </void>
</object>
The following listing comprises the XML code to describe the whiteboard in testRoomA in the Wonderland virtual test world from Chapter 4.

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<java version="1.5.0_13" class="java.beans.XMLDecoder">
<object class="org.jdesktop.lg3d.wonderland.darksstar.server.setup.BasicCellGLOSetup">
  <void property="cellGLOClassName">
    org.jdesktop.lg3d.wonderland.whiteboard.server.cell.WhiteboardCellGLO
  </void>
  <void property="cellSetup">
    org.jdesktop.lg3d.wonderland.whiteboard.common.WhiteboardCellSetup
  </void>
  <void property="preferredWidth">
    1024
  </void>
  <void property="preferredHeight">
```
B.3 TestRoomB-WLC.xml

The following listing comprises the XML code to describe testRoomB in the Wonderland virtual test world from Chapter 4.

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<java version="1.5.0_13" class="java.beans.XMLDecoder">
  <object class="org.jdesktop.ig3d.wonderland.darkstar.server.setup.BasicCellGLOSetup">
    <double>768</double>
  </object>
  </void>
</java>
```
The following listing comprises the XML code to describe testRoomC in the Wonderland virtual test world from Chapter 4.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<java version="1.5.0_13" class="java.beans.XMLDecoder">
<object class="org.jdesktop.lg3d.wonderland.darkstar.server.cell.BasicCellGLOSetup">
   <void property="cellGLOClassName">
      <string>org.jdesktop.lg3d.wonderland.darkstar.server.cell.SimpleTerrainCellGLO</string>
   </void>
   <void property="origin">
      <double>50.0</double>
   </void>
   <double>50.0</double>
   <void property="boundsRadius">
      <double>100.0</double>
   </void>
   <string>BOX</string>
   <void property="accessOwner">
      <string>twright</string>
   </void>
   <string>admin</string>
   <void property="accessGroupPermissions">
      <string>3</string>
   </void>
   <void property="accessOtherPermissions">
      <string>2</string>
   </void>
   <void property="cellSetup">
      <object class="org.jdesktop.lg3d.wonderland.darkstar.common.setup.ModelCellSetup">
         <void property="modelFile">
            <string>models/TestRoomB.j3s.gz</string>
         </void>
      </object>
   </void>
</object>
</java>
```

B.4 TestRoomC-WLC.xml

The following listing comprises the XML code to describe testRoomC in the Wonderland virtual test world from Chapter 4.
B.5 TestRoomD-WLC.xml

The following listing comprises the XML code to describe testRoomD in the Wonderland virtual test world from Chapter 4.

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<java version="1.5.0_13" class="java.beans.XMLDecoder">
<object class="org.jdesktop.lg3d.wonderland.darkstar.server.setup.BasicCellGLOSetup">
  <void property="cellGLOClassName">
    <string>org.jdesktop.lg3d.wonderland.extracells.server.MultiModelCellGLO</string>
  </void>
</object>
</java>
```
<void property="origin">
  <void index="0">
    <double>50.0</double>
  </void>
  <void index="2">
    <double>50.0</double>
  </void>
</void>

<void property="boundsRadius">
  <double>100.0</double>
</void>

<void property="boundsType">
  <string>BOX</string>
</void>

<void property="accessOwner">
  <string>twright</string>
</void>

<void property="accessGroup">
  <string>admin</string>
</void>

<void property="accessGroupPermissions">
  <string>0</string>
</void>

<void property="accessOtherPermissions">
  <string>0</string>
</void>

<void property="cellSetup">
  <object class="org.jdesktop.lg3d.wonderland.extracells.common.MultiModelCellSetup">
    <void property="models">
      <array class="org.jdesktop.lg3d.wonderland.extracells.common.MultiModelCellModel">
        <object class="org.jdesktop.lg3d.wonderland.extracells.common.MultiModelCellModel">
          <void property="modelFile">
            <string>models/TestRoomD.j3s.gz</string>
          </void>
        </object>
      </array>
    </void>
  </object>
</void>

<!-- Cube Wall Decorations -->

<object class="org.jdesktop.lg3d.wonderland.extracells.common.MultiModelCellModel">
  <void property="modelFile">
    <string>models/default_building/teamroom_walldeco_d.j3s.gz</string>
  </void>
</object>

<void property="location">
  <array class="double">
    <double>4.5</double>
  </array>
</void>
<double>5.7</double>
<double>0.0</double>
</array>
</void>
<void property="rotation">
<array class="double">
<double>0.0</double>
<double>1.0</double>
<double>0.0</double>
<double>2.35</double>
</array>
</void>
</object>

<object class="org.jdesktop.lg3d.wonderland.extracells.common.MultiModelCellModel">
<void property="modelFile">
<string>models/default_building/demofloor_desk.j3s.gz</string>
</void>
<void property="location">
<array class="double">
<double>-28.65</double>
<double>5.3</double>
<double>64.13</double>
</array>
</void>
</object>

<object class="org.jdesktop.lg3d.wonderland.extracells.common.MultiModelCellModel">
<void property="modelFile">
<string>models/default_building/demofloor_desk.j3s.gz</string>
</void>
<void property="location">
<array class="double">
<double>-30.65</double>
<double>5.3</double>
<double>64.13</double>
</array>
</void>
</object>
APPENDIX C

WONDERLAND SOURCE CODE EXCERPTS

The excerpts in this appendix include all of the modifications and extensions to the Wonderland 0.3 Java code base. All source code is written in Java 1.5 and documented to point out our changes and additions.

C.1 CellAccessControl

The following excerpt comprises the CellAccessControl class.

```java
package org.jdesktop.lg3d.wonderland.darkstar.server;

import org.jdesktop.lg3d.wonderland.darkstar.server.cell.CellGLO;
import org.jdesktop.lg3d.wonderland.darkstar.server.auth.WonderlandIdentity;
import org.jdesktop.lg3d.wonderland.darkstar.server.DAC;
```
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 * Provides access control support for cells and users
 * Modified by Timothy Wright to enable discretionary access control (DAC)
 * capabilities. See the DAC interface for some details. This code
 * is based upon a stub provided by Paul Byrne.
 * We need to determine whether or not a user has enough privilege
 * to interact with (i.e., view, in this case) a given cell.
 * Do this by comparing their name against that of the cell’s owner.
 * If these aren’t the same, then compare the user’s group against that
 * of the cell. If these aren’t the same, consider the permissions
 * assigned to the universal group “other” for the cell.
 * If the user’s name matches the cell’s owner name, then the user is
 * granted both interact and alter permissions (i.e., full permissions)
 * automatically. If the user’s group matches that of the cell, then
 * the cell’s group permissions are in effect. If the user’s name and
 * group do not match the cell’s owner and group, then the permissions
 * for the “other” group are in effect.
 * There are visual and function consequences when an avatar does not
 * have access to a cell. To ensure avatars don’t materialize in a cell
 * for which they have no permissions, there should be a common area where
 * avatars may safely show up when a user first logs into Wonderland.
 * Moreover, worlds should be architected so as to handle the visual
 * situations of cells appearing for some users and not for others (e.g.,
 * if users X and Y have access to a cell that user Z doesn’t, where X
 * and Y see a cell, Z will see nothing. There are clever ways to
 * handle this situation. To begin with, the floor should always be a
 * separate cell with full permissions for the “other” group (if the
 * floor doesn’t get displayed there are significant issues with avatar
 * movement!).
 * @author twright
 */
public class CellAccessControl implements DAC {
    protected final static Logger logger = Logger.getLogger(CellAccessControl.class.getName());

    private static String userIDNull;
    private static String cellPermsNull;

    private CellAccessControl() {
    }

    /**
     * Returns true if a specific group is found within an
     * array of group names.
     */
private static boolean isGroupListMember(String[] groupList, String group) {
    // Check the groupList array of Strings for the presence of
    // the group String. If it's there, return TRUE; else, return
    // FALSE.
    for (int index = 0; index < groupList.length; index++) {
        if (groupList[index].equalsIgnoreCase(group))
            return true;
    }
    return false;
}

/**
 * Returns true if this user can interact with this cell.
 * @param user a UserGLO object denoting the current transaction's user
 * @param cell a CellGLO object denoting the cell to be accessed
 * @return boolean
 */
public static boolean canView(UserGLO user, CellGLO cell) {
    // It is possible for this method to be called with 'user' NOT
    // containing a NULL WonderlandIdentity and/or cell containing
    // NULL values for cell permissions. If either of these things
    // happens, simply fail open—this isn't as dangerous as it
    // sounds: subsequent calls to this method will have fully
    // populated identity information and cell permissions.
    //
    // This seems to be a side effect of the asynchronous
    // nature of constructing UserGLO objects, user identities,
    // and cell permissions.
    //
    // TW
    if (user.getUserIdentity() != null && cell.getCellAccessOwner() != null) {
        if (user.getUserIdentity().getName().equalsIgnoreCase(cell.getCellAccessOwner())) {
            logger.fine("Access to cell " + cell.getCellName() +
                        " granted based on owner name match for owner name "+
                        user.getUserName() + ".");
            return true;
        }
    } else if (isGroupListMember(user.getUserIdentity().getAccessGroups(),
                                  cell.getCellAccessGroup())) {
        if (cell.getCellAccessGroupPermissions().equals(IAPERMS) ||

cell.getCellAccessGroupPermissions().equals(I_PERMISSIONS)) {
    logger.fine("Access to cell " + cell.getGLOName() + 
        " granted based on group name match for " + 
        user.getUserName() + ";");
    return true;
}
}

else if (cell.getCellAccessOtherPermissions().equals(I_PERMISSIONS) ||
    cell.getCellAccessOtherPermissions().equals(I_PERMISSIONS)) {
    logger.fine("Access to cell " + cell.getGLOName() + 
        " granted based on 'other' group for user " + 
        user.getUserName() + ";");
    return true;
}
else {
    logger.fine("In sufficient interaction permissions for " + 
        user.getUserName() + ", " + 
        cell.getGLOName() + ");
}

else {
    if (logger.isLoggable(Level.FINE)) {
        userIDNull = "";
        cellPermsNull = "";

        if (user.getUserIdentity() == null)
            userIDNull = user.getUserName() + ", Wonderland Identity is null.
        if (cell.getCellAccessOwner() == null)
            cellPermsNull = ", Permissions for cell " + cell.getGLOName() + 
                " are null.");

        logger.fine(userIDNull + cellPermsNull + 
            "Access controls are temporarily failing open.");
        return true;
    }

    return false;
}

/**
 * Returns true if this user can alter this cell.
 * @param user a UserGLO object denoting the current transaction's user
 * @param cell a CellGLO object denoting the cell to be accessed
 * @return boolean
 */
public static boolean canAlter(UserGLO user, CellGLO cell) {
    return canAlter(user.getUserIdentity(), cell);
}
/**
 *  Returns true if this user can alter this cell.
 * @param id the WonderlandIdentity object for the current transaction's user
 * @param cell a CellGLO object denoting the cell to be accessed
 * @return boolean
 */
public static boolean canAlter(WonderlandIdentity id, CellGLO cell) {

    // It is possible for this method to be called with 'user' NOT containing a NULL WonderlandIdentity and/or cell containing NULL values for cell permissions. If either of these things happens, simply fail closed. Unlike the canView() method, a check for the alteration permission cannot happen unless the user clicks on something that exists in their world.
    // Thus, we don't have the problem of synchronization to worry about (the user can't click on something that isn't yet there!).
    // If we should somehow show up here with NULL values, just quietly fail closed.

    if ((id != null) && (cell.getCellAccessOwner() != null)) {
        if (id.getName().equalsIgnoreCase(cell.getCellAccessOwner().getName())) {
            logger.fine("Access to alter cell granted based on owner name match for user" + id.getName() + ".");
            return true;
        }
    }

    else if (isGroupListMember(id.getAccessGroups(), cell.getCellAccessGroup()) ||
             (cell.getCellAccessGroupPermissions().equals(IA_PERMS) ||
              cell.getCellAccessGroupPermissions().equals(A_PERMISSIONS))) {
        logger.fine("Access to alter cell granted based on group name match for user" + id.getName() + ".");
        return true;
    }

    else if (cell.getCellAccessOtherPermissions().equals(IA_PERMS) ||
              cell.getCellAccessOtherPermissions().equals(A_PERMISSIONS)) {
        logger.fine("Access to alter cell granted based on other group for user" + id.getName() + ".");
        return true;
    }

    else {
        logger.fine("Insufficient interaction permissions for" + id.getName() + " to alter cell" + cell.getGLOName() + ".");
    }
}
C.2 ClientIdentityManager

The following excerpt comprises the `ClientIdentityManager` class.

```java
package org.jdesktop.ig3d.wonderland.darkstar.server;

...
/**
 * Provides access control support for cells and users
 *
 * Modified by Timothy Wright to enable discretionary access control (DAC)
 * capabilities. See the DAC interface for some details. This code
 * is based upon a stub provided by Paul Byrne.
 *
 * We need to determine whether or not a user has enough privilege
 * to interact with (i.e., view, in this case) a given cell.
 * Do this by comparing their name against that of the cell's owner.
 * If these aren't the same, then compare the user's group against that
 * of the cell. If these aren't the same, consider the permissions
 * assigned to the universal group "other" for the cell.
 *
 * If the user's name matches the cell's owner name, then the user is
 * granted both interact and alter permissions (i.e., full permissions)
 * automatically. If the user's group matches that of the cell, then
 * the cell's group permissions are in effect. If the user's name and
 * group do not match the cell's owner and group, then the permissions
 * for the "other" group are in effect.
 *
 * There are visual and function consequences when an avatar does not
 * have access to a cell. To ensure avatars don't materialize in a cell
 * for which they have no permissions, there should be a common area where
 * avatars may safely show up when a user first logs into Wonderland.
 * Moreover, worlds should be architected so as to handle the visual
 * situations of cells appearing for some users and not for others (e.g.,
 * if users X and Y have access to a cell that user Z doesn't, where X
 * and Y see a cell, Z will see nothing. There are clever ways to
 * handle this situation. To begin with, the floor should always be a
 * separate cell with full permissions for the "other" group (if the
 * floor doesn't get displayed there are significant issues with avatar
 * movement!).
 *
 * @author twright
 */

public class CellAccessControl implements DAC {
    protected final static Logger logger =
        Logger.getLogger(CellAccessControl.class.getName());

    private static String userIDNull;
    private static String cellPermsNull;

    private CellAccessControl() {
    }

    import org.jdesktop.ig3d.wonderland.darkstar.server.cell.CellGLO;
    import org.jdesktop.ig3d.wonderland.darkstar.server.auth.WonderlandIdentity;
    import org.jdesktop.ig3d.wonderland.darkstar.server.DAC;
    import java.util.logging.Level;
    import java.util.logging.Logger;

/**
 * Returns true if a specific group is found within an array of group names.
 * @param groupList an array of group names to which a participant belongs
 * @param group the name of a group for an object the participant wishes to access
 * @return boolean
 */
private static boolean isGroupListMember(String[] groupList, String group) {
    // Check the groupList array of Strings for the presence of
    // the group String. If it's there, return TRUE; else, return
    // FALSE.
    for (int index = 0; index < groupList.length; index++) {
        if (groupList[index].equalsIgnoreCase(group))
            return true;
    }
    return false;
}

/**
 * Returns true if this user can interact with this cell.
 * @param user a UserGLO object denoting the current transaction's user
 * @param cell a CellGLO object denoting the cell to be accessed
 * @return boolean
 */
public static boolean canView(UserGLO user, CellGLO cell) {

    // It is possible for this method to be called with 'user' NOT
    // containing a NULL WonderlandIdentity and/or cell containing
    // NULL values for cell permissions. If either of these things
    // happens, simply fail open—this isn't as dangerous as it
    // sounds: subsequent calls to this method will have fully
    // populated identity information and cell permissions.
    //
    // This seems to be a side effect of the asynchronous
    // nature of constructing UserGLO objects, user identities,
    // and cell permissions.
    //
    // TW

    if ((user.getUserIdentity() != null) && (cell.getCellAccessOwner() != null)) {
        if (user.getUserIdentity().getName().equalsIgnoreCase(cell.getCellAccessOwner())) {
            logger.fine("Access to cell " + cell.getGLOName() +
                         " granted based on owner name match for " +
                         user.getUserName() + " . ");
            return true;
        }
    }
}
else if (isGroupListMember(user.getUserIdentity(), getAccessGroups(), cell.getCellAccessGroup())) {
    if (cell.getCellAccessGroupPermissions().equals(IA_PERMS) ||
        cell.getCellAccessGroupPermissions().equals(L_PERMS)) {
        logger.fine("Access to cell" + cell.getGLOName() +
                " granted based on group name match for " +
                user.getUserName() + ".");
        return true;
    }
}
else if (cell.getCellAccessOtherPermissions().equals(IA_PERMS) ||
        cell.getCellAccessOtherPermissions().equals(L_PERMS)) {
        logger.fine("Access to cell" + cell.getGLOName() +
                " granted based on 'other' group for user " +
                user.getUserName() + ".");
        return true;
    }
else {
        logger.fine("Insufficient interaction permissions for " +
                user.getUserName() + ". To access cell " +
                cell.getGLOName() + ".");
        return false;
    }
}

/**
 * Returns true if this user can alter this cell.
 * @param user a UserGLO object denoting the current transaction's user
 * @param cell a CellGLO object denoting the cell to be accessed
 * @return boolean
 */
public static boolean canAlter(UserGLO user, CellGLO cell) {
    return canAlter(user.getUserIdentity(), cell);
}

/**
 * Returns true if this user can alter this cell.
 *
 * @param id the WonderlandIdentity object for the current transaction's user
 * @param cell a CellGLO object denoting the cell to be accessed
 * @return boolean
 */
public static boolean canAlter(WonderlandIdentity id, CellGLO cell) {
    // It is possible for this method to be called with 'user' NOT containing a NULL WonderlandIdentity and/or cell containing NULL values for cell permissions. If either of these things happens, simply fail closed. Unlike the canView() method, // a check for the alteration permission cannot happen unless the user clicks on something that exists in their world. // Thus, we don't have the problem of synchronization to worry about (the user can't click on something that isn't yet there!). // If we should somehow show up here with NULL values, just quietly // fail closed.
    //
    // TW

    if ((id != null) && (cell.getCellAccessOwner() != null)) {
        if (id.getName().equalsIgnoreCase(cell.getCellAccessOwner().getName())) {
            logger.fine("Access to alter cell " + cell.getGLOName() +
                    " granted based on owner name match for " +
                    id.getName() + ".");
            return true;
        }
    } else if (isGroupListMember(id.getAccessGroups(), cell.getCellAccessGroup())){
        if (cell.getCellAccessGroupPermissions().equals(IA_PERMS) ||
            cell.getCellAccessGroupPermissions().equals(A_PERMS)) {
            logger.fine("Access to alter cell " + cell.getGLOName() +
                    " granted based on group name match for " +
                    id.getName() + ".");
            return true;
        }
    } else if (cell.getCellAccessOtherPermissions().equals(IA_PERMS) ||
            cell.getCellAccessOtherPermissions().equals(A_PERMS)) {
        logger.fine("Access to alter cell " + cell.getGLOName() +
                    " granted based on 'other' group for user " +
                    id.getName() + ".");
        return true;
    } else {
C.3 ClientIdentityService

The following excerpt comprises the ClientIdentityService class.

```java
package org.jdesktop.lg3d.wonderland.darkstar.server;

import com.sun.sgs.auth.Identity;
import com.sun.sgs.service.DataService;
import com.sun.sgs.service.Service;
import com.sun.sgs.service.TransactionProxy;
import com.sun.sgs.kernel.ComponentRegistry;
import com.sun.sgs.app.ManagedReference;
import com.sun.sgs.app.ClientSession;
import java.util.Properties;

/**
 * The ClientIdentityService class provides access to a persistent
 * ClientIdentity object, which, in turn, permits us to access the
 * WonderlandIdentity object found in a given UserGLO object. In
 * particular, this is necessary for identifying the user group to
 * which a given participant belongs. With such information, we can
 * effectively operate a discretionary access control system.
 */
```
C.4 DAC

The following excerpt comprises the DAC class.

```java
c.4 DAC

The following excerpt comprises the DAC class.

```
The basic discretionary access control system consists of
the following to privileges:

- Interact (I) —> view/use some cell (e.g., enter and walk in a
  room, use a white board); equivalent to read/execute
  in classic UNIX
- Alter (A) —> change/add to a cell (e.g., use a white board,
  create an object in-world); equivalent to write
  in classic UNIX

Just as in classic UNIX, all cells have an owner and a

group assigned to them, where the owner and group are
found in the LDAP database we use to managing all Wonderland
participants. The owner always has full permissions to

t heir cell, while the permissions for a group and 'other'
(i.e., everyone else) must be specified.

Permissions are always wide open for the owner, and default to
interact-only ("2" in the following table) for group/other.

Permissions are indicated by a single character string:

I | A
---|---
1 | 1 —> "3"
1 | 0 —> "2"
0 | 1 —> "1"
0 | 0 —> "0"

This interface provides a convenient means of accessing
the string constants denoting these different access configurations.

@author twright

```java
public interface DAC {
    public final static String IAMPERMS = "3";
    public final static String IUPERMS = "2";
    public final static String APERMS = "1";
    public final static String NOPERMS = "0";
    public final static String DEFAULT_OWNER = "nobody";
    public final static String DEFAULT_GROUP = "nobody";
}
```
C.5 UserGLO

The following excerpt comprises the UserGLO class.

```java
/**
 * Project Looking Glass
 *
 * $RCSfile: UserGLO.java,v $
 *
 * Copyright (c) 2004–2007, Sun Microsystems, Inc., All Rights Reserved
 *
 * Redistributions in source code form must reproduce the above
 * copyright and this condition.
 *
 * The contents of this file are subject to the GNU General Public
 * License, Version 2 (the "License"); you may not use this file
 * except in compliance with the License. A copy of the License is
 *
 * $Revision: 1.19.6.1 $
 * $Date: 2008/01/30 13:43:44 $
 * $State: Exp $
 */

package org.jdesktop.lg3d.wonderland.darkstar.server;

import com.sun.sgs.app.AppContext;
import com.sun.sgs.app.Channel;
import com.sun.sgs.app.ChannelListener;
import com.sun.sgs.app.ClientSession;
import com.sun.sgs.app.ClientSessionId;
import com.sun.sgs.app.DataManager;
import com.sun.sgs.app.ManagedObject;
import com.sun.sgs.app.ManagedReference;
import com.sun.sgs.app.NameNotBoundException;
import java.awt.Color;
import java.io.Serializable;
import java.nio.ByteBuffer;
import java.util.HashMap;
import java.util.HashSet;
import java.util.logging.Logger;
import org.jdesktop.lg3d.wonderland.darkstar.common.AvatarInfo;
import org.jdesktop.lg3d.wonderland.darkstar.server.cell.AvatarCellGLO;
import org.jdesktop.lg3d.wonderland.darkstar.server.cell.MasterCellCacheGLO;
import org.jdesktop.lg3d.wonderland.recorder.RecorderCellGLO;
import org.jdesktop.lg3d.wonderland.darkstar.server.auth.WonderlandIdentity; // TW

//import org.jdesktop.lg3d.wonderland.darkstar.common.messages.Message;

/**
 *
 * public class UserGLO implements ManagedObject, Serializable {
```
/**
 * All GLOs should define a <code>serialVersionUID</code> because
 * they are serialized. This turns off version checking so we can
 * change the class and still load old data that might already be
 * in the ObjectStore.
 */
private static final long serialVersionUID = 1L;

/**
 * The unique name of the user
 */
private String name;

/**
 * Color used to represent user in lists and map
 */
private Color userColor;

/**
 * The Users AvatarCell or RecorderCell
 */
private ManagedReference avatarRef = null;

private ClientSessionId userID;
private ManagedReference thisRef = null;
private ClientSession clientSession = null;
private boolean isRecorder = false;

// N i c e  i m p l e m e n t a t i o n o f g r o u p s
private HashSet<String> groups = null;

// D i r e c t  a c c e s s  t o  t h e  W o n d e r l a n d I d e n t i t y  f o r  t h i s  u s e r .
//
// TW
private WonderlandIdentity wonderlandID = null; // TW

// private GLOReference<UserDataListenerController> userDataListenerController;

/**
 * Creates a user GLO.
 *
 * @param name the screen name of the user
 * @param avatarInfo the name of the avatar
 */
UserGLO(String name, AvatarInfo info, boolean isRecorder) {
    this.name = name;
    this.isRecorder = isRecorder;
}
DataManager dataMgr = AppContext.getDataManager();
dataMgr.setBinding(name, this);

if (isRecorder) {
    avatarRef = dataMgr.createReference(new
        RecorderCellGLO(dataMgr.createReference(this)));
} else {
    avatarRef = dataMgr.createReference(new
        AvatarCellGLO(dataMgr.createReference(this), info));
}

// userDataListenerController =
// SimTask.getCurrent().createGLO(new UserDataListenerController());

public void setAvatarInfo(AvatarInfo info)
{
    if (!isRecorder) {
        avatarRef.getForUpdate(AvatarCellGLO.class).setAvatarInfo(info);
    }
}

public void login(ClientSession clientSession)
{
    this.clientSession = clientSession;
    this.userID = clientSession.getSessionId();
    DataManager dataMgr = AppContext.getDataManager();
    ManagedReference userRef = dataMgr.createReference(this);
    UserManager.getUserManager().addUser(userID, userRef); // Must do this early in login

    if (isRecorder) {
        avatarRef.getForUpdate(RecorderCellGLO.class).login(userID);
    } else {
        avatarRef.getForUpdate(AvatarCellGLO.class).login(userID);
    }

    // Obtain the WonderlandIdentity for this user.
    // TW
    wonderlandID = (WonderlandIdentity)AppContext.
        getManager(ClientIdentityManager.class).getClientID(); // TW
}

/**
 * The user has logged out, cleanup
 */
public void logout()
{
    if (isRecorder) {
        avatarRef.getForUpdate(RecorderCellGLO.class).logout(userID);
    } else {
        avatarRef.getForUpdate(AvatarCellGLO.class).logout(userID);
    }
    MasterCellCacheGLO.getMasterCellCache().removeCell(avatarRef);
DataManager dataMgr = AppContext.getDataManager();
ManagedReference userRef = dataMgr.createReference(this);
UserManager getUserManager(), removeUser(userID);
userID = null;
clientSession = null;
}

/**
 * Find or create and return the user glo
 */
public static ManagedReference getOrCreateUserGLO(
    String userName,
    ClientSessionId userID,
    AvatarInfo avatar, boolean isRecorder) {

    String userObjName = "user"+userName;
    UserGLO userGLO =null;

    DataManager dataMgr = AppContext.getDataManager();
    try {
        userGLO = dataMgr.getBinding(userObjName, UserGLO.class);
    } catch (NameNotBoundException ex) {
        userGLO = new UserGLO(userName, avatar, isRecorder);
        dataMgr.setBinding(userObjName, userGLO);
    }

    /**
     * Check that the userGLO has the appropriate avatarRef
     **/
    if (userGLO.isRecorder != isRecorder) {
        userGLO.isRecorder = isRecorder;
        if (isRecorder) {
            userGLO.avatarRef =
dataMgr.createReference(new RecorderCellGLO(dataMgr.createReference(userGLO)));
        } else {
            userGLO.avatarRef =
dataMgr.createReference(new AvatarCellGLO(dataMgr.createReference(userGLO), avatar));
        }
    }

    ManagedReference userRef = dataMgr.createReference(userGLO);

    userRef.getForUpdate(UserGLO.class).setUserID(userID);
    return userRef;
}

/**
 * Find and return the user glo from sgs datastore.
 *
public static ManagedReference getUserGLORef(String userName) {
    UserGLO userGLO = getUserGLO(userName);
    if (userGLO == null)
        return null;
    return AppContext.getDataManager().createReference(userGLO);
}

public static UserGLO getUserGLO(String userName) {
    String userObjName = "user" + userName;
    UserGLO userGLO = null;
    DataManager dataMgr = AppContext.getDataManager();
    try {
        userGLO = dataMgr.getBinding(userObjName, UserGLO.class);
    } catch (NameNotBoundException ex) {
        return null;
    }
    return userGLO;
}

public ManagedReference getAvatarCellRef() {
    return avatarRef;
}

public boolean isLoggedIn() {
    return (clientSession != null) ? true : false;
}

public ClientSession getClientSession() {
    return clientSession;
}
/**
  * This method returns our users name. This name is unique
  * @return user name
  */
public String getUserName() {
    return name;
}

void setUserID(ClientSessionId userID) {
    this.userID = userID;
}

public ClientSessionId getUserID() {
    return userID;
}

/**
  * Get Color used to represent user in lists and map
  */
public Color getUserColor() {
    if (isRecorder) {
        return Color.BLACK;
    } else {
        return userColor;
    }
}

/**
  * Set Color used to represent user in lists and map
  */
public void setUserColor(Color color) {
    userColor = color;
}

/**
  * Add user to specified group
  */
public void addToGroup(String groupName) {
    groups.add(groupName);
}

/**
  * Remove user from specified group
  */
public void removeFromGroup(String groupName) {
    groups.remove(groupName);
}

/**
  * Check if user is a member of specified group
  */
public boolean isMemberOfGroup(String groupName) {
    return groups.contains(groupName);
}
/**
 * Obtain the WonderlandIdentity object associated with
 * this user.
 */
public WonderlandIdentity getUserIdentity() { // TW
    return wonderlandID;
}

C.6 LDAP Auth

The following excerpt comprises the LDAPAuth class.
import java.util.logging.Logger;
import javax.naming.Context;
import javax.naming.NamingEnumeration;
import javax.naming.NamingException;
import javax.naming.directory.Attribute;
import javax.naming.directory.Attributes;
import javax.naming.directory.DirContext;
import javax.naming.directory.InitialDirContext;
import javax.naming.directory.SearchControls;
import javax.naming.directory.SearchResult;
import javax.security.auth.login.AccountNotFoundException;
import javax.security.auth.login.CredentialException;
import javax.security.auth.login.LoginException;

/**
 * An authenticator that uses LDAP to get a person's information. This
 * authenticator relies on the following properties being set in
 * the Darkstar configuration file:
 * <ul>
 * <li><code>org.jdesktop.lg3d.wonderland.darkstar.server.auth.ldap.directory</code>
 * The LDAP URL to connect to, for example ldaps://directory.java.net
 * </li>
 * <li><code>org.jdesktop.lg3d.wonderland.darkstar.server.auth.ldap.base-dn</code>
 * The base DN to perform lookups under. For example dc=org,dc=jdesktop
 * </li>
 * <li><code>org.jdesktop.lg3d.wonderland.darkstar.server.auth.ldap.search-filter</code>
 * The LDAP filter to search for user names, for example "employeenumber=%s",
 * where %s will be substituted with the user name used for authentication.
 * The default is "uid=%s"
 * </li>
 * <li><code>org.jdesktop.lg3d.wonderland.darkstar.server.auth.ldap.context-factory</code>
 * The directory context factory. Default is "com.sun.jndi.ldap.LdapCtxFactory"
 * </li>
 * <li><code>org.jdesktop.lg3d.wonderland.darkstar.server.auth.ldap.username-attr</code>
 * The attribute in the directory that represents the username. Default is "uid"
 * </li>
 * <li><code>org.jdesktop.lg3d.wonderland.darkstar.server.auth.ldap.fullname-attr</code>
 * The attribute in the directory that represents the full name. Default is "cn"
 * </li>
 * <li><code>org.jdesktop.lg3d.wonderland.darkstar.server.auth.ldap.email-attr</code>
 * The attribute in the directory that represents the email address. Default is "mail"
 * </li>
 * </ul>
 * @author jkaplan
 */
public class LDAPAuth implements IdentityAuthenticator {
    // logger
    private static final Logger logger =


Logger.getLogger(LDAPAuth.class.getName());

private KernelAppcontext context;
private Hashtable ldapEnv;

// properties to use
private static final String PROPBASE = LDAPAuth.class.getPackage().getName() + ".ldap."
private static final String PROPLDAPURL = PROPBASE + "directory";
private static final String PROPBASEDN = PROPBASE + "base-dn";
private static final String PROPSERCHFILTER = PROPBASE + "search-filter";
private static final String PROPCONTEXTFACTORY = PROPBASE + "context-factory";
private static final String PROPUSENAME_ATTR = PROPBASE + "username-attr";
private static final String PROPFULLNAME_ATTR = PROPBASE + "fullname-attr";
private static final String PROPEMAIL_ATTR = PROPBASE + "email-attr";
private static final String PROPACCESSGROUP_ATTR = PROPBASE + "description-attr"; // TW

// default values
private static final String SEARCHFILTER_DEFAULT = "uid=%s";
private static final String CONTEXTFACTORY_DEFAULT = "com.sun.jndi.ldap.LdapCtxFactory";
private static final String USERNAMENAME_DEFAULT = "uid";
private static final String FULLNAMENAME_DEFAULT = "cn";
private static final String EMAIL_ATTR_DEFAULT = "mail";
private static final String ACCESSGROUP_ATTR_DEFAULT = "description"; // TW
private static final String GUESTNAME = "guest"; // TW
private static final String GUESTFULL_NAME = "Guest User"; // TW
private static final String GUESTEMAIL = "guest@nowhere.special"; // TW
private static final String GUESTACCESSGROUP = "guest"; // TW

// properties not in environment
String searchFilter;
String baseDN;
String usernameAttr;
String fullNameAttr;
String emailAttr;
String accessGroupAttr; // TW

// directory context
DirContext dnContext;

public LDAPAuth(Properties prop) {
  // required properties
  String ldapURL = prop.getProperty(PROPLDAPURL);
  if (ldapURL == null) {
    throw new RuntimeException(PROPLDAPURL + " required");
  }
  baseDN = prop.getProperty(PROPBASEDN);
  if (baseDN == null) {
    throw new RuntimeException(PROPBASEDN + " required.");
  }
}
// optional properties
searchFilter = prop.getProperty(PROP_SEARCH_FILTER,
    SEARCH_FILTER_DEFAULT);
usernameAttr = prop.getProperty(PROP_USERNAME_ATTR,
    USERNAME_ATTR_DEFAULT);
fullnameAttr = prop.getProperty(PROP_FULLNAME_ATTR,
    FULLNAME_ATTR_DEFAULT);
emailAttr = prop.getProperty(PROP_EMAIL_ATTR,
    EMAIL_ATTR_DEFAULT);
accessGroupAttr = prop.getProperty(PROP_ACCESS_GROUP_ATTR,
    ACCESS_GROUP_ATTR_DEFAULT); // TW

String contextFactory = prop.getProperty(PROP_CONTEXT_FACTORY,
    CONTEXT_FACTORY_DEFAULT);

ldapEnv = new Hashtable();
ldapEnv.put(Context.INITIAL_CONTEXT_FACTORY, contextFactory);
ldapEnv.put(Context.PROVIDER_URL, ldapURL);

logger.info("Loading LDAP authenticator");

public String[] getSupportedCredentialTypes() {
    return new String[] {"NameAndPasswordCredentials"};
}

public void assignContext(KernelAppContext context) {
    this.context = context;
}

public Identity authenticateIdentity(IdentityCredentials credentials)
    throws LoginException {
    if (!(credentials instanceof NamePasswordCredentials)) {
        throw new CredentialException("Wrong credential\class:\" +
            credentials.getClass().getName());
    }

    NamePasswordCredentials npc = (NamePasswordCredentials) credentials;
    String dn;
    DirContext userContext;

    // map name to an identity
    try {
        userContext = findUser(npc.getName());

        // If our userContext is null, as opposed to
        // bombing out, set up the user with a guest
        // account. This situation can happen if we
        // are using the "run-benchmark" ant target

        dn = "cn=");
    }
}

public String[] getSupportedCredentialTypes() {
    return new String[] {"NameAndPasswordCredentials"};
}

public void assignContext(KernelAppContext context) {
    this.context = context;
}

public Identity authenticateIdentity(IdentityCredentials credentials)
    throws LoginException {
    if (!(credentials instanceof NamePasswordCredentials)) {
        throw new CredentialException("Wrong credential\class:\" +
            credentials.getClass().getName());
    }

    NamePasswordCredentials npc = (NamePasswordCredentials) credentials;
    String dn;
    DirContext userContext;

    // map name to an identity
    try {
        userContext = findUser(npc.getName());

        // If our userContext is null, as opposed to
        // bombing out, set up the user with a guest
        // account. This situation can happen if we
        // are using the "run-benchmark" ant target

        dn = "cn=");
    }
}
// to test with multiple, simultaneous clients
// on one workstation.

// TW
if (userContext == null) { // TW
    String guestName;

    // If we can, use a benchmark name for this client.
    if (npc.getName() != null)
        guestName = npc.getName();
    else
        guestName = GUEST_NAME;

    return new WonderlandIdentity(guestName, GUEST_FULL_NAME,
                                  GUEST_EMAIL, GUEST_ACCESS_GROUP);
} else
    dn = userContext.getNameInNamespace();
} catch (NamingException ne) {
    logger.log(Level.WARNING, "Directory lookup error.", ne);
    throw new LoginException("Directory lookup error.");
}

// make sure we found the user
if (dn == null) {
    throw new AccountNotFoundException("User " + npc.getName() + " not found");
}

// make sure the password is not zero-length
if (npc.getPassword().length == 0) {
    throw new CredentialException("Invalid username or password.");
}

// now try to bind to that dn
Hashtable secureEnv = new Hashtable(ldapEnv);
secureEnv.put(Context_SECURITY_PRINCIPAL, dn);
secureEnv.put(Context_SECURITY_CREDENTIALS, npc.getPassword());

// try connecting. If we succeed, the password was correct.
try {
    DirContext secureContext = new InitialDirContext(secureEnv);
    secureContext.close();
} catch (NamingException ne) {
    throw new CredentialException("Bad password for " + dn);
}

// our connection succeeded, so now return a valid identity
try {
    WonderlandIdentity id = getIdentity(userContext);
    return id;
} catch (NamingException ne) {
logger.log(Level.WARNING, "Identity resolution error.");
throw new LoginException("Error getting identity information");
}
}

/**
 * Map a username to a user's directory entry.
 * @param username the name to lookup
 * @return the DirectoryContext corresponding to the given username
 * @throws NamingException if there is an error finding the given
 *    username
 */
protected DirectoryContext findUser(String username)
throws NamingException
{
    DirContext ret = null;
    // see if we need to instantiate the context
    if (dnContext == null) {
        dnContext = new InitialDirContext(ldapEnv);
    }
    // setup the search controls
    SearchControls sc = new SearchControls();
    sc.setReturningAttributes(new String[] { "cn" });
    sc.setReturningObjFlag(true);
    sc.setCountLimit(1);
    sc.setSearchScope(SearchControls.SUBTREE_SCOPE);
    // do the search
    String filter = searchFilter.replaceAll("%s", username);
    NamingEnumeration ne = dnContext.search(baseDN, filter, sc);
    if (ne.hasMore()) {
        SearchResult res = (SearchResult) ne.next();
        ret = (DirContext) res.getObject();
    }
    // return the name
    return ret;
}

/**
 * Get a WonderlandIdentity from the user's LDAP entry.
 * @param userContext the user's directory information
 * @return a WonderlandIdentity with information about the
 *    given user
 * @throws NamingException if there is an error reading the
 *    directory information
 */
protected WonderlandIdentity getIdentity(DirContext userContext)
throws NamingException
{
}
String username = getAttribute(usernameAttr, userContext);  
String fullname = getAttribute(fullnameAttr, userContext);  
String email = getAttribute(emailAttr, userContext);  
String accessGroup = getAttribute(accessGroupAttr, userContext);  // TW

return new WonderlandIdentity(username, fullname, email, accessGroup);  // TW

/**
 * Get an attribute’s first value
 * @param attrName the attribute to get
 * @param context the context to get the attribute from
 * @return the attribute’s first value, or null if the attribute
 * doesn’t exist
 * @throws NamingException if there is an error
 */
protected String getAttribute(String attrName, DirContext context)
    throws NamingException
{
    Attributes attrs = context.getAttributes("", new String[] { attrName });
    Attribute attr = attrs.get(attrName);
    if (attr != null) {
        return (String) attr.get();
    }
    else {
        return null;
    }
}

C.7 WonderlandIdentity

The following excerpt comprises the WonderlandIdentity class.

/**
 * Project Looking Glass
 * $RCSfile: WonderlandIdentity.java,v $
 * $Revision: 1.2 $
 * $Date: 2007/07/16 20:15:48 $
 * $Creator: rBrowserRouter @project.org $
package org.jdesktop.ig3d.wonderland.darkstar.server.auth;

import com.sun.sgs.auth.Identity;
import java.io.Serializable;

/**
 * An identity for the Wonderland system. Modified by TW on 3/8/08
 * to add access group characteristic for purposes of access control.
 * @author jkaplan
 */
public class WonderlandIdentity implements Identity, Serializable {

    private String username;
    private String fullName;
    private String email;
    private String[] accessGroups; // TW

    public WonderlandIdentity(String username, String fullName, String email,
            String accessGroups) {
        this.username = username;
        this.fullName = fullName;
        this.email = email;
        this.accessGroups = accessGroups.split("-"); // TW
    }

    public String getName() {
        return username;
    }

    public String getFullName() {
        return fullName;
    }

    public String getEmail() {
        return email;
    }

    public String[] getAccessGroups() { // TW
        return accessGroups;
    }

    public void notifyLoggedIn() {
        // do nothing
    }

    public void notifyLoggedOut() {
        // do nothing
    }
}
C.8 CellGLO

The following excerpt comprises the CellGLO class.

```java
package org.jdesktop.ig3d.wonderland.darkstar.server.cell;

import com.sun.sgs.app.AppContext;
import com.sun.sgs.app.Channel;
import com.sun.sgs.app.ChannelManager;
import com.sun.sgs.app.ClientSessionId;
import com.sun.sgs.app.DataManager;
import com.sun.sgs.app.Delivery;
import com.sun.sgs.app.ManagedObject;
import com.sun.sgs.app.ManagedReference;
import com.sun.sgs.app.NameNotBoundException;
import java.io.Serializable;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Collections;
import java.util.HashSet;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.media.j3d.BoundingSphere;
import javax.vecmath.Point3d;
import javax.vecmath.Matrix4d;
import javax.vecmath.BoundingBox;
import javax.vecmath.BoundingBox;
import javax.vecmath.Transform3D;
import javax.vecmath.Matrix3d;
import javax.vecmath.Vector3d;
import org.jdesktop.ig3d.wonderland.config.common.WonderlandConfig;
import org.jdesktop.ig3d.wonderland.darkstar.common.CellID;
import org.jdesktop.ig3d.wonderland.darkstar.common.CellSetup;
```
import org.jdesktop.lg3d.wonderland.darkstar.common.ChannelInfo;
import org.jdesktop.lg3d.wonderland.darkstar.server.CellIDGenerator;
import org.jdesktop.lg3d.wonderland.darkstar.server.UserPerformanceMonitor;

/**
 * Server representation of a Cell
 * @author paulby
 */
public abstract class CellGLO implements ManagedObject, Serializable {
    protected final static String baseUrl = WonderlandConfig.getBaseURL();
    protected final static Logger logger = Logger.getLogger(CellGLO.class.getName());

    protected final static long serialVersionUID = 1L;

    protected CellID cellID; // Unique cell ID

    private boolean originRelative;

    private long version = 0;

    protected HashSet<ManagedReference> childCells = null;
    protected HashSet<ManagedReference> parentCells =
            new HashSet<ManagedReference>();

    protected Channel cellChannel = null;
    protected String cellChannelName = null;

    private Matrix4d cellOrigin;

    protected ManagedReference thisRef;

    private String cellAccessOwner = null; // TW
    private String cellAccessGroup = null; // TW
    private String cellAccessGroupPermissions = null; // TW
    private String cellAccessOtherPermissions = null; // TW

    /**
     * Creates a new instance of CellGLO
     */
public CellGLO() {
    DataManager dataMgr = AppContext.getDataManager();

    this.cellID = CellIDGenerator.getNextID();
    this.thisRef = dataMgr.createReference(this);

    dataMgr.setBinding(getGLOName(), this);
}

public CellID getCellID() {
    return cellID;
}

/**
 * Returns the name of this object's GLO
 */
public String getGLOName() {
    return getGLOName(getCellID());
}

/**
 * Return the GLO name for the given cell
 */
public static String getGLOName(CellID cellId) {
    return "CELL"+cellId;
}

/**
 * Get the origin of this cell, as a Matrix4d
 */
public Matrix4d getOrigin() {
    return cellOrigin;
}

/**
 * Set the cell's origin. By default, the origin is relative to the cell's
 * parent.
 * @param origin the cell's origin, as a Matrix4d
 */
public void setOrigin(Matrix4d origin) {
    setOrigin(origin, true);
}

/**
 * Set the cell's origin. This method lets you define if the cell's
 * origin is relative to its parent cell or relative to the world
 * coordinate system.
 * @param origin the cell's origin, as a Matrix4d
 * @param originRelative if true, the cell's origin is relative to it's
 * parent, if false, it is relative to the world coordinate system
 */
public void setOrigin(Matrix4d origin, boolean originRelative) {
    this.cellOrigin = origin;
    this.originRelative = originRelative;
}

/**
* Returns true if the cell's origin is relative to its parent, or false
* if the cell's origin is in world coordinates
* @return True for relative coordinates, false for world coordinates
*/
public boolean isOriginRelative() {
    return this.originRelative;
}

/**
* Get the origin of this cell in world coordinates.
* @param origin the origin value to modify
*/
protected void getOriginWorld(Matrix4d origin) {
    // Walk up the tree to find the initial parent.
    // We can stop walking as soon as we find a non-relative origin
    if (isOriginRelative() && !parentCells.isEmpty()) {
        // take the first parent. Presumably cells with relative origins
        // should only have a single parent
        ManagedReference parentRef = parentCells.iterator().next();
        parentRef.get(CellGLO.class).getOriginWorld(origin);

        // multiply the origin by our transform
        origin.mul(cellOrigin);
    } else {
        // the origin is absolute or this is the topmost cell, so
        // just set the origin to our current value
        origin.set(cellOrigin);
    }
}
public String getCellAccessOwner() { // TW
    return cellAccessOwner;
}

public void setCellAccessOwner(String cellAccessOwner) { // TW
    this.cellAccessOwner = cellAccessOwner;
}

public String getCellAccessGroup() { // TW
    return cellAccessGroup;
}

public void setCellAccessGroup(String cellAccessGroup) { // TW
    this.cellAccessGroup = cellAccessGroup;
}

public String getCellAccessGroupPermissions() { // TW
    return cellAccessGroupPermissions;
}

public void setCellAccessGroupPermissions(String cellAccessGroupPermissions) { // TW
    this.cellAccessGroupPermissions = cellAccessGroupPermissions;
}

public String getCellAccessOtherPermissions() { // TW
    return cellAccessOtherPermissions;
}

public void setCellAccessOtherPermissions(String cellAccessOtherPermissions) { // TW
    this.cellAccessOtherPermissions = cellAccessOtherPermissions;
}

/**
 * Returns a list of Cells that intersect with the supplied bounds. The
 * bounds are referenced in world coordinates; it is assumed that the cell
 * upon which this method is called also has its bounds given in terms of
 * world coordinates. Returns a list of references to visible cells.
 * @param bounds The viewing bounds, in world coordinates
 * @param monitor The performance monitor
 * @return A list of visible cells
 */
public ArrayList<ManagedReference> getVisibleCells(Bounds bounds,
            UserPerformanceMonitor monitor) {
    return getVisibleCells(new ArrayList<ManagedReference>(), bounds,
                           null, monitor);
}

/**
 * Returns a list of Cells that intersect with the supplied bounds. This
*° method adds to the list of visible cells passed to it and returned the
*° new list. The bounds are referenced in world coordinates; if a cell’s
*° bounds are relative to its parent, then its bounds are translated by its
*° parent’s origin (assuming that if a cell’s origin is given relative to
*° its parents, so are its bounds). The parentOrigin argument gives the
*° translation necessary to place this cell’s bounds in world coordinates,
*° if null, then no translation is assumed. The parentOrigin argument is
*° modified by this routine.
*°
° @param list The list of visible cells
° @param bounds The viewing bounds, in world coordinates
° @param parentOrigin The origin of the parent cell, in world coordinates
° @param monitor The performance monitor
° @return A list of visible cells
*/
ArrayList<ManagedReference> getVisibleCells(ArrayList<ManagedReference> list,
    Bounds bounds, Matrix4d parentOrigin, UserPerformanceMonitor monitor) {
    long t = System.nanoTime();
    
    /°
    *° We first must check whether a the cell’s origin is relative to its
    *° parent. If so, then we translate the bounds of this cell by the
    *° origin of the parent. If the parent’s bounds is given as null, then
    *° we do not need to translate the bounds at all either.
    */
    Bounds worldBounds = (Bounds)this.getBounds().clone();
    if (this.isOriginRelative() == true & & parentOrigin != null) {
        /*
        *° We just simply apply the 4x4 origin matrix to the bounds. In
        *° reality, the matrix often includes rotations as well — we’ll
        *° just assume we wish to carry along these transformations as well.
        */
        worldBounds.transform(new Transform3D(parentOrigin));
    }
    
    /°
    *° We now intersect with the bounds of the cell translated into world
    *° coordinates.
    */
    boolean intersect = bounds.intersect(worldBounds);
    monitor.incRevalidateCalcTime(System.nanoTime()−t);
    monitor.incRevalidateCellCount(getClass());

    if (intersect) {
        if (logger.isLoggable(Level.FINEST)) {
            logger.fine("\nIntersect " + getGLOName() + " " + worldBounds);
        }
        list.add(thisRef);
        if (childCells!=null) {
            /*
            *° Recursively call getVisibleCells() to find the visible
            */
        }
Matrix4d childOrigin;
if (parentOrigin == null || this.isOriginRelative() == false) {
    childOrigin = new Matrix4d(this.getOrigin());
} else {
    childOrigin = new Matrix4d(parentOrigin);
    childOrigin.mul(this.getOrigin());
}

for (ManagedReference v : childCells) {
    t = System.nanoTime();
    CellGLO c = v.getClass();
    monitor.incRevalidateCellGetTime(c.getClass(),
                                       System.nanoTime()-t);
    c.getVisibleCells(list, bounds, childOrigin, monitor);
}

return list;

/**
 * Returns the fully qualified name of the class that represents
 * this cell on the client
 */
public abstract String getClientCellClassName();

/**
 * Return the name of this cells channel
 *
 * TODO this should be abstract
 */
public String getCellChannelName() {
    return cellChannelName;
}

/**
 * Open the channel for this cell using the supplied channelName
 */
protected void openCellChannel(String channelName) {
    cellChannelName = channelName;
    ChannelManager chanMgr = AppContext.getChannelManager();
try {
    cellChannel = chanMgr.getChannel(channelName);
} catch (NameNotBoundException e) {
    cellChannel = chanMgr.createChannel(channelName, null,
            Delivery.RELIABLE);
}

/**
 * Open the default channel for this cell. The client cell class must
 * implement ClientChannelListener
 */
protected void openDefaultChannel() {
    openCellChannel(getCellID() + ChannelInfo.CELL_CHANNEL);
}

/**
 * Open the cell channel, TODO make abstract
 */
protected void openChannel() {
}

/**
 * Returns the cells channelID
 */
protected Channel getCellChannel() {
    return cellChannel;
}

/**
 * Add the user to this cells channel
 */
public void addUserToCellChannel(ClientSessionId userID) {
    if (cellChannel!=null)
        cellChannel.join(userID.getClientSession(), null);
}

/**
 * Remove the user from this cells channel
 */
public void removeUserFromCellChannel(ClientSessionId userID) {
    if (cellChannel!=null && userID.getClientSession()!=null)
        cellChannel.leave(userID.getClientSession());
}

/**
 * Add the specified cell as a childRef of this cell. Also adds this cell
 * as a part of the childRef
 */
public void addChildCell(ManagedReference childRef) {
    if (childCells==null)
        childCells = new HashSet<ManagedReference>();
childCells.add(childRef);

CellGLO child = childRef.getForUpdate(CellGLO.class);

child.addParentCell(thisRef);
}

/**
 * Add the supplied cell as a parent of this cell, called by addChildCell
 */
void addParentCell(ManagedReference parent) {
    if (parentCells==null)
        parentCells = new HashSet<ManagedReference>();
    parentCells.add(parent);
}

/**
 * Remove the child from this cell
 */
public void removeChildCell(ManagedReference childRef) {
    CellGLO child = childRef.getForUpdate(CellGLO.class);
    child.removeParentCell(thisRef);
    boolean o = childCells.remove(childRef);
}

/**
 * Removed the supplied parent cell from the list of this cell's parents
 */
void removeParentCell(ManagedReference parent) {
    parentCells.remove(parent);
}

/**
 * Detaches a cell from the hierarchy by removing all of the references to
 * any its parents and any references its parents may have of this cell.
 * This also removes any references any of its children may have to it. This
 * method assumes it is called using a write-able version of the class.
 */
public void detach() {
    /* First remove all of the parent's references to this child */
    for (ManagedReference parent: this.parentCells) {
        CellGLO parentGLO = parent.getForUpdate(CellGLO.class);
        parentGLO.removeChildCell(this.thisRef);
    }
    this.parentCells.clear();

    /* Next remove all of the child's references to this parent */
    if (this.childCells != null) {
        for (ManagedReference child: this.childCells) {
            CellGLO childGLO = child.getForUpdate(CellGLO.class);
            childGLO.removeParentCell(this.thisRef);
        }
    }

279
public abstract Bounds getBounds();

/**
 * Returns the bounds of this cell in world coordinates.
 * This requires a call to getOriginWorld(), so may be expensive.
 * @return the bounds of this cell transformed to world coordinates
 */
public Bounds getBoundsWorld() {
    // see if the bounds need to calculated relative to the parent
    if (isOriginRelative() && !parentCells.isEmpty()) {
        // calculate the bounds relative to the parent’s world origin
        // take the first parent. Presumably cells with relative origins
        // should only have a single parent
        ManagedReference parentRef = parentCells.iterator().next();
        Matrix4d parentOrigin = parentRef.getCellGLClass().getOriginWorld();

        Bounds out = getBounds();
        out.transform(new Transform3D(parentOrigin));
        return out;
    } else {
        // absolute origin, just return it
        return getBounds();
    }
}

/**
 * Returns the cell id of each parent cell
 */
public CellID[] getParentCellIDs() {
    CellID[] ret = new CellID[parentCells.size()];

    int i = 0;
    for (ManagedReference p : parentCells)
        ret[i++] = p.getCellGLClass().getCellID();

    return ret;
}

/**
 * Return the CellSetup object for this cell
 */
// public abstract CellSetup getSetupData();
/∗ Return the children of this cell, or an empty collection if there are no children ∗/ @SuppressWarnings("unchecked") public Collection<ManagedReference> getChildren() { if (childCells==null) return Collections.emptyList(); return (Collection<ManagedReference>) childCells.clone(); } /** * Get the setup data for this cell */ public abstract CellSetup getSetupData(); /** * Creates a bounding box with the specified center and size. */ public static BoundingBox createBoundingBox(Vector3d center, float size) { BoundingBox cellBounds = new BoundingBox(new Point3d(center.x-size/2f, center.y-size/2f, center.z-size/2f), new Point3d(center.x+size/2f, center.y+size/2f, center.z+size/2f)); return cellBounds; } /** * Creates a bounding box with the specified center and dimensions. */ public static BoundingBox createBoundingBox(Vector3d center, float xDim, float yDim, float zDim) { BoundingBox cellBounds = new BoundingBox(new Point3d(center.x-xDim/2f, center.y-yDim/2f, center.z-zDim/2f), new Point3d(center.x+xDim/2f, center.y+yDim/2f, center.z+zDim/2f)); return cellBounds; } /** * Creates a bounding sphere with the specified center and size. */ public static BoundingSphere createBoundingSphere(Vector3d center, float radius) { return new BoundingSphere(new Point3d(center), radius); } } /** * Creates a Matrix4d with a translation to center */ public static Matrix4d createOriginM4d(Vector3d center) { Matrix3d rot = new Matrix3d();
rot.setIdentity();
return new Matrix4d(rot, center, 1);
}
/**
 * Create a Matrix4D with a translation to center, a rotation in the y axis and a scale
 */
public static Matrix4d createOriginM4d(Vector3d center, double angle, double scale) {
    Matrix3d rot = new Matrix3d();
    rot.rotY(angle);
    return new Matrix4d(rot, center, scale);
}
/**
 * Get the current version number for this CellGLO. This is used by the UserCellCacheGLO to determine if a user’s copy of this cell is up to date. If the version is higher than the user’s version, the UserCellCacheGLO will send a reconfigure message to the associated client.
 * @return this cell's version number
 */
public long getVersion() {
    return version;
}
/**
 * Increment the cell’s version number
 * @return the new version number
 */
public long incrementVersion() {
    return version++;
}
@Override
public boolean equals(Object o) {
    return getCellID().equals(o);
}
@Override
public int hashCode() {
    return getCellID().hashCode();
}
}

### C.9 StationaryCellGLO

The following excerpt comprises the *StationaryCellGLO* class.
package org.jdesktop.lg3d.wonderland.darkstar.server.cell;

import com.sun.sgs.app.ManagedReference;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import javax.media.j3d.Bounds;
import javax.media.j3d.Transform3D;
import javax.vecmath.Matrix4d;
import org.jdesktop.lg3d.wonderland.darkstar.common.SerializationHelper;
import org.jdesktop.lg3d.wonderland.darkstar.server.setup.BasicCellGLOHelper;
import org.jdesktop.lg3d.wonderland.darkstar.server.setup.BasicCellGLOSetup;

/**
 * A cell that can not be moved (ie it's bounds are static)
 *
 * @author paulby
 */

public abstract class StationaryCellGLO extends CellGLO {

    /**
     * Bounds is not serializable so declare as transient and
     * serialize in read/writeObject
     */
    private transient Bounds cellBounds;

    /**
     * The parent origin, in world coordinates. This value will only be valid
     * if this cell is relatively positioned. We keep this in the child to
     * allow an efficient implementation of getOriginWorld() and
     * getBoundsWorld()
     */
    private Matrix4d parentOrigin;
}
/**
 * Create a new stationary cell GLO with default bounds and origin
 */
public StationaryCellGLO() {
    this (null, null);
}

/**
 * Creates a new instance of StationaryCellGLO
 * @param bounds the bounds of this cell
 * @param origin the origin of this cell
 */
public StationaryCellGLO(Bounds bounds, Matrix4d origin) {
    this.cellBounds = bounds;
    if (origin == null) {
        origin = new Matrix4d();
        origin.setIdMethod();
    }
    // By default, a stationary cell’s origin is relative to the
    // world coordinate system. It can be reset later to be relative
    // to its parents coordinates instead.
    setOrigin(origin, false);
}

/**
 * Get this cell’s bounds
 * @return the cell’s bounds
 */
public Bounds getBounds() {
    return (Bounds) cellBounds.clone();
}

/**
 * Set this cell’s bounds
 * @param bounds the bounds for this cell
 */
public void setBounds(Bounds bounds) {
    this.cellBounds = (Bounds) bounds.clone();
}

/**
 * Set up the cell from the given properties
 * @param properties the properties to setup with
 */
public void setupCell(BasicCellGLOSetup<?>) setup) {
    // when loading from a properties file, the origin is relative to
    // the parent
    setOrigin(BasicCellGLOHelper.getCellOrigin(setup), true);
    setBounds(BasicCellGLOHelper.getCellBounds(setup));
}
/**
 * @param properties the properties to setup with
 */
public void reconfigureCell(BasicCellGLOSetup<?> setup) {
    // just call setupCell, since there is nothing to do differently
    // if this is a change
    setupCell(setup);
}

/**
 * @param parentOrigin
 */
protected void setParentOrigin(Matrix4d parentOrigin) {
    this.parentOrigin = parentOrigin;
}

/**
 * @return the bounds of this cell, in world coordinates
 */
@Override
public Bounds getBoundsWorld() {
    if (isOriginRelative() && getParentOrigin() != null) {
        return (Matrix4d) parentOrigin.clone();
    }
    return null;
}
// shortcut -- use the parent origin rather than walking the tree
Bounds out = getBounds();
out.transform(new Transform3D(getParentOrigin()));
return out;
} else {
    return super.getBoundsWorld();
}
*/

@Override
public Matrix4d getOriginWorld() {
    if (isOriginRelative() && getParentOrigin() != null) {
        // shortcut -- use the parent origin rather than walking the tree
        Matrix4d out = getParentOrigin();
        out.mul(getOrigin());
        return out;
    } else {
        return super.getOriginWorld();
    }
}

@Override
void addParentCell(ManagedReference parentRef) {
    super.addParentCell(parentRef);

    // when adding a parent, also copy the cell's origin into our
    // parent origin
    // XXX assumes a single parent
    CellGLO parent = parentRef.get(CellGLO.class);
    setParentOrigin(parent.getOriginWorld());
}

@Override
public void setOrigin(Matrix4d origin, boolean originRelative) {
    super.setOrigin(origin, originRelative);

    // notify all children of the new parent origin
    for (ManagedReference childRef : getChildren()) {
        CellGLO childCell = childRef.get(CellGLO.class);
        if (childCell instanceof StationaryCellGLO) {
            // notify the child cell of our position in world coordinates
            ((StationaryCellGLO) childCell).setParentOrigin(getOriginWorld());
        }
    }
}

/**
private void writeObject(ObjectOutputStream out) throws IOException {
    out.defaultWriteObject();
    SerializationHelper.writeObject(getBounds(), out);
}

private void readObject(ObjectInputStream in) throws IOException, ClassNotFoundException {
    in.defaultReadObject();
    setBounds(SerializationHelper.readBoundsObject(in));
}

C.10 BasicCellGLOHelper

The following excerpt comprises the BasicCellGLOHelper class.

package org.jdesktop.lg3d.wonderland.darkstar.server.setup;

import javax.media.j3d.BoundingBox;
import javax.media.j3d.BoundingSphere;
import javax.media.j3d.Bounds;
import javax.vecmath.AxisAngle4d;
import javax.vecmath.Matrix3d;
public class BasicCellGLOHelper {

    public static Bounds getCellBounds(BasicCellGLOSetup setup) {
        if (setup.getBoundsType() == null) {
            return null;
        } else if (setup.getBoundsType().equals("SPHERE")) {
            return new BoundingSphere(new Point3d(setup.getOrigin()),
                    setup.getBoundsRadius());
        } else if (setup.getBoundsType().equals("BOX")) {
            return CellGLO.createCellBoundingBox(new Vector3d(setup.getOrigin()),
                    (float) setup.getBoundsRadius());
        }
        return null;
    }

    public static Matrix4d getCellOrigin(BasicCellGLOSetup setup) {
        Matrix3d rot = new Matrix3d();
        rot.set(new AxisAngle4d(setup.getRotation()));
        Vector3d trans = new Vector3d(setup.getOrigin());
        double scale = setup.getScale();
        Matrix4d out = new Matrix4d();
        out.set(rot, trans, scale);
        return out;
    }

    public static String getCellAccessOwner(BasicCellGLOSetup setup) { // TW
        return setup.getAccessOwner();
    }

    public static String getCellAccessGroup(BasicCellGLOSetup setup) { // TW
        return setup.getAccessGroup();
    }

    public static String getCellAccessGroupPermissions(BasicCellGLOSetup setup) { // TW
        return setup.getAccessGroupPermissions();
    }

    public static String getCellAccessOtherPermissions(BasicCellGLOSetup setup) { // TW
        return setup.getAccessOtherPermissions();
    }
```

public static String getBoundsType(Bounds bounds) {
    if (bounds instanceof BoundingSphere) {
        return "SPHERE";
    } else if (bounds instanceof BoundingBox) {
        return "BOX";
    }

    return null;
}

public static double getBoundsRadius(Bounds bounds) {
    if (bounds instanceof BoundingSphere) {
        return ((BoundingSphere) bounds).getRadius();
    } else if (bounds instanceof BoundingBox) {
        BoundingBox bb = (BoundingBox) bounds;

        Point3d upper = new Point3d();
        Point3d lower = new Point3d();
        bb.getUpper(upper);
        bb.getLower(lower);

        upper.sub(lower);
        return Math.abs(upper.x);
    }

    return 0;
}

public static double[] getTranslation(Matrix4d origin) {
    Vector3d trans = new Vector3d();
    origin.get(trans);

    return new double[] { trans.x, trans.y, trans.z };
}

public static double[] getRotation(Matrix4d origin) {
    // create an angle from the origin
    AxisAngle4d rot = new AxisAngle4d();
    rot.set(origin);

    return new double[] { rot.x, rot.y, rot.z, rot.angle };
}
```

C.11 BasicCellGLOSetup

The following excerpt comprises the `BasicCellGLOSetup` class.
package org.jdesktop.lg3d.wonderland.darkstar.server.setup;

import javax.media.j3d.Bounds;
import javax.vecmath.Matrix4d;
import org.jdesktop.lg3d.wonderland.darkstar.common.CellSetup;
import org.jdesktop.lg3d.wonderland.darkstar.server.DAC;

/**
 * @author jkaplan
 */
public class BasicCellGLOSetup<T extends CellSetup>
    implements CellGLOSetup, CellLocation, DAC {
{
    /** the setup object to send to the cell */
    private T cellSetup;

    /** the class of cell GLO */
    private String cellGLOClassName;

    /** the location of the cell */
    private double[] origin = new double[] { 0.0, 0.0, 0.0 };
    private double[] rotation = new double[] { 0.0, 1.0, 0.0, 0.0 };
    private double scale = 1.0;

    /** The basic discretionary access control system consists of
     * the following:
     * 
     * + Interact (I) → view/use some cell (e.g., enter and walk in a
     *   room, use a white board); equivalent to read/execute
     * + Alter (A) ---> change/add to a cell (e.g., use a white board,

create an object in-world; equivalent to write

Just as in classic UNIX, all cells have an owner and a group assigned to them, where the owner and group are found in the LDAP database we use to managing all Wonderland participants. The owner always has full permissions to their cell, while the permissions for a group and 'other' (i.e., everyone else) must be specified.

Permissions are always wide open for the owner, and default to interact-only ("2" in the following table) for group/other.

Permissions are indicated by a single character string:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>I</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>I</td>
<td>1</td>
<td>0</td>
<td>I</td>
</tr>
</tbody>
</table>

See the DAC interface for string constants that denote these different access configurations.

TW

private String accessOwner = DEFAULT_OWNER;  // TW
private String accessGroup = DEFAULT_GROUP;  // TW
private String accessGroupPermissions = I_PERMS;  // TW
private String accessOtherPermissions = I_PERMS;  // TW

/* the bounds of the cell */
private String boundsType = "SPHERE";
private double boundsRadius = 4.0;

public BasicCellGLOSetup() {
    this (null, null, null, null);
}

public BasicCellGLOSetup(Bounds bounds, Matrix4d origin, String cellGLOClassName, T cellSetup) {
    setCellGLOClassName (cellGLOClassName);
    setCellSetup (cellSetup);

    if (bounds != null) {
        setBoundsType (BasicCellGLOHelper.getBoundsType(bounds));
        setBoundsRadius (BasicCellGLOHelper.getBoundsRadius(bounds));
    }

    if (origin != null) {
        setOrigin (BasicCellGLOHelper.getTranslation (origin));
    }
public T getCellSetup() {
    return cellSetup;
}

public void setCellSetup(T cellSetup) {
    this.cellSetup = cellSetup;
}

public String getCellGLOClassName() {
    return cellGLOClassName;
}

public void setCellGLOClassName(String cellGLOClassName) {
    this.cellGLOClassName = cellGLOClassName;
}

public double[] getOrigin() {
    return origin;
}

public void setOrigin(double[] origin) {
    this.origin = origin;
}

public double[] getRotation() {
    return rotation;
}

public void setRotation(double[] rotation) {
    this.rotation = rotation;
}

public double getScale() {
    return scale;
}

public void setScale(double scale) {
    this.scale = scale;
}

public String getAccessOwner() { // TW
    return accessOwner;
}

public void setAccessOwner(String accessOwner) { // TW
    if (accessOwner != null)
        this.accessOwner = accessOwner;
}
```java
public String getAccessGroup() { // TW
    return accessGroup;
}

public void setAccessGroup(String accessGroup) { // TW
    if (accessGroup != null)
        this.accessGroup = accessGroup;
}

public String getAccessGroupPermissions() { // TW
    return accessGroupPermissions;
}

public void setAccessGroupPermissions(String accessGroupPermissions) { // TW
    if (accessGroupPermissions != null)
        this.accessGroupPermissions = accessGroupPermissions;
}

public String getAccessOtherPermissions() { // TW
    return accessOtherPermissions;
}

public void setAccessOtherPermissions(String accessOtherPermissions) { // TW
    if (accessOtherPermissions != null)
        this.accessOtherPermissions = accessOtherPermissions;
}

public String getBoundsType() {
    return boundsType;
}

public void setBoundsType(String boundsType) {
    this.boundsType = boundsType;
}

public double getBoundsRadius() {
    return boundsRadius;
}

public void setBoundsRadius(double boundsRadius) {
    this.boundsRadius = boundsRadius;
}

public void validate() throws InvalidCellGLOSetupException {
    // do nothing
}
```
The following excerpt comprises the WhiteBoardCellGLO class.

```java
package org.jdesktop.lg3d.wonderland.whiteboard.server.cell;

import com.sun.sgs.app.ClientSession;
import java.util.HashSet;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.Set;
import java.util.logging.Logger;
import javax.media.j3d.BoundingSphere;
import javax.media.j3d.Bounds;
import javax.vecmath.Matrix4d;
import javax.vecmath.Matrix3d;
import javax.vecmath.Matrix4f;
import javax.vecmath.Vector3d;
import javax.vecmath.AxisAngle4d;
import org.jdesktop.lg3d.wonderland.darkstar.server.auth.WonderlandIdentity; // TW
import com.sun.sgs.app.AppContext; // TW
import org.jdesktop.lg3d.wonderland.darkstar.server.ClientIdentityManager; // TW
import org.jdesktop.lg3d.wonderland.darkstar.server.CellAccessControl; // TW
import org.jdesktop.lg3d.wonderland.darkstar.common.messages.CellMessage;
import org.jdesktop.lg3d.wonderland.darkstar.server.CellMessageListener;
import org.jdesktop.lg3d.wonderland.darkstar.server.cell.SharedApp2DImageCellGLO;
import org.jdesktop.lg3d.wonderland.darkstar.server.setup.BeanSetupGLO;
import org.jdesktop.lg3d.wonderland.darkstar.server.setup.BasicCellGLOSetup;
import org.jdesktop.lg3d.wonderland.darkstar.server.setup.BasicCellGLOHelper;
import org.jdesktop.lg3d.wonderland.darkstar.server.setup.CellGLOSetup;
import org.jdesktop.lg3d.wonderland.whiteboard.common.CompoundWhiteboardCellMessage;
import org.jdesktop.lg3d.wonderland.whiteboard.common.WhiteboardAction.Action;
import org.jdesktop.lg3d.wonderland.whiteboard.common.WhiteboardCellSetup;
```
import org.jdesktop.lg3d.wonderland.whiteboard.common.WhiteboardCommand.Command;

/**
 * A server cell associated with a whiteboard
 * @author nsimpson
 */
public class WhiteboardCellGLO extends SharedApp2DImageCellGLO
    implements BeanSetupGLO, CellMessageListener {

private static final Logger logger =
    Logger.getLogger(WhiteboardCellGLO.class.getName());

// The messages list contains the current state of the whiteboard.
// It’s updated every time a client makes a change to the whiteboard
// so that when new clients join, they receive the current state
private static LinkedList<CompoundWhiteboardCellMessage> messages;
private static CompoundWhiteboardCellMessage lastMessage;

private BasicCellGLOSetup<WhiteboardCellSetup> setup;

public WhiteboardCellGLO() {
    this(null, null, null, null);
}

public WhiteboardCellGLO(Bounds bounds, String appName, Matrix4d cellOrigin,
    Matrix4f viewRectMat) {
    super(bounds, appName, cellOrigin, viewRectMat,
        WhiteboardCellGLO.class.getName());
    messages = new LinkedList<CompoundWhiteboardCellMessage>();
}

/**
 * Returns the fully qualified name of the class that represents
 * this cell on the client
 * @return the class name of the corresponding client cell
 */
public String getClientCellClassName() {
    return "org.jdesktop.lg3d.wonderland.whiteboard.client.cell.WhiteboardCell";
}

/**
 * Get the setup data for this cell
 * @return the cell setup data
 */
public WhiteboardCellSetup getSetupData() {
    return setup.getCellSetup();
}

/**
 * Set up the properties of this cell GLO from a JavaBean. After calling
 * this method, the state of the cell GLO should contain all the information
 * represented in the given cell properties file.
 */
public void setupCell(CellGLOSetup setupData) {
    setup = (BasicCellGLOSetup<WhiteboardCellSetup>) setupData;

    AxisAngle4d aa = new AxisAngle4d(setup.getRotation());
    Matrix3d rot = new Matrix3d();
    rot.set(aa);
    Vector3d origin = new Vector3d(setup.getOrigin());

    Matrix4d o = new Matrix4d(rot, origin, setup.setScale());
    setOrigin(o);

    if (setup.getBoundsType().equals("SPHERE") { 
        setBounds(createBoundingSphere(origin, (float) setup.getBoundsRadius())); 
    } else { 
        throw new RuntimeException("Unimplemented bounds type"); 
    }

    // Handle configuring this cell's discretionary access controls
    // TW
    setCellAccessOwner(BasicCellGLOHelper.getCellAccessOwner(setup)); // TW
    setCellAccessGroup(BasicCellGLOHelper.getCellAccessGroup(setup)); // TW
    setCellAccessGroupPermissions(BasicCellGLOHelper.
        getCellAccessGroupPermissions(setup)); // TW
    setCellAccessOtherPermissions(BasicCellGLOHelper.
        getCellAccessOtherPermissions(setup)); // TW
}

/**
 * Called when the properties of a cell have changed.
 *
 * @param setup a Java bean with updated properties
 */
public void reconfigureCell(CellGLOSetup setupData) {
    setupCell(setupData);
}

/**
 * Write the cell's current state to a JavaBean.
 *
 * @return a JavaBean representing the current state
 */
public CellGLOSetup getCellGLOSetup() {
    return new BasicCellGLOSetup<WhiteboardCellSetup>(getBounds(),
        getOrigin(), getClass().getName(),
        getSetupData());
}

/**
 * Open the cell channel
public void openChannel() {
    this.openDefaultChannel();
}

/**
 * Handle message
 */
@override
public void receivedMessage(ClientSession client, CellMessage message) {

    // Does the user have permissions to alter the whiteboard?  If not,
    // ignore all messages coming from them.
    //
    // TW
    if (CellAccessControl.canAlter(WonderlandIdentity)
            AppContext.getManager(ClientIdentityManager.class).getClientID(this)) {  // TW
        CompoundWhiteboardCellMessage cmsg =
            (CompoundWhiteboardCellMessage) message;
            logger.fine("received whiteboard message : " + cmsg);

        if (cmsg.getAction() == Action.REQUESTSYNC) {
            logger.fine("sending " + messages.size() + " whiteboard sync messages");
            Iterator<CompoundWhiteboardCellMessage> iter = messages.iterator();

            while (iter.hasNext()) {
                CompoundWhiteboardCellMessage msg = iter.next();
                getCellChannel().send(client, msg.getBytes());
            }
        } else {
            // record the message in setup data (move events are not recorded)
            if (cmsg.getAction() == Action.EXECUTECOMMAND) {
                if (cmsg.getCommand() == Command.ERASE) {
                    // clear the action history
                    logger.fine("clearing message history");
                    messages.clear();
                }
            } else {
                if (cmsg.getAction() != Action.MOVE_TO) {
                    if ((lastMessage != null) &&
                            lastMessage.getAction() == Action.MOVE_TO) {
                        messages.add(lastMessage);
                    }
                    messages.add(cmsg);
                }
            }
            lastMessage = cmsg;
        }
    } else {
        // notify all clients except the client that sent the message
        CompoundWhiteboardCellMessage msg =
            new CompoundWhiteboardCellMessage(cmsg.getAction());
    }
}
switch (cmsg.getAction()) {
    case SET_TOOL:
        // tool
        msg.setTool(cmsg.getTool());
        break;
    case SET_COLOR:
        // color
        msg.setColor(cmsg.getColor());
        break;
    case MOVE_TO:
    case DRAG_TO:
        // position
        msg.setPositions(cmsg.getPositions());
        break;
    case REQUEST_SYNC:
        break;
    case EXECUTE_COMMAND:
        // command
        msg.setCommand(cmsg.getCommand());
        break;
}

Set<ClientSession> sessions =
    new HashSet<ClientSession>(getCellChannel().getSessions());
sessions.remove(client);
logger.fine("distributing whiteboard message: "+ msg);
getCellChannel().send(sessions, msg.getBytes());
}
APPENDIX D

WONDERDAC DEMONSTRATION SURVEY, GANTT CHART, AND HSIRB PROTOCOL REVIEW

The 10 question survey administered to participants of the demonstration follows. Each survey question used one of two five point rating scales, depending on the context of the question. Figure D.1 comprises the Gantt chart for the WonderDAC demonstration outlined in Chapter 6. Figure D.2 is a scanned copy of the Notre Dame Human Subjects Institutional Review Board (HSIRB) Protocol Review for the demonstration.

D.1 Demonstration Questionnaire

D.1.1 WonderDAC Basics

Q1) Discretionary access control based on roles (owner, group, and other) and only two permissions (interact and alter) seems straightforward and intuitive.

With this statement, you:

1. Disagree

2. Somewhat disagree

3. Have no opinion

4. Somewhat agree
D.1.2 Spatial Restrictions

Q2) How effective was access control for entering, seeing, and hearing into spaces (e.g., the team lofts and the screening room)?

You feel spatial access control was:

1. Ineffective
2. Somewhat ineffective
3. Neutral
4. Somewhat effective
5. Effective

Q3) Spaces for which one has no interact permission are completely removed from one’s perspective of the virtual world. In practice, this metaphor worked well.

With this statement, you:

1. Disagree
2. Somewhat disagree
3. Have no opinion
4. Somewhat agree
5. Agree
D.1.3 Non-spatial Object Restrictions

Q4) How effective was access control for viewing and using non-spatial objects (e.g., X Window applications, whiteboards, PDF viewers, video viewers, 3D models)?

You feel that non-spatial access control was:

1. Ineffective
2. Somewhat ineffective
3. Neutral
4. Somewhat effective
5. Effective

Q5) Non-spatial objects for which one has no interact permission are completely removed from one’s perspective of the virtual world. In practice, this metaphor worked well.

With this statement, you:

1. Disagree
2. Somewhat disagree
3. Have no opinion
4. Somewhat agree
5. Agree
D.1.4 Audio Chat Restrictions

Q6) How effective was access control for restricting audio conversation in the team room (during presentations) and in the screening room (during the wrap-up phase)?

You feel that audio chat access control was:

1. Ineffective
2. Somewhat ineffective
3. Neutral
4. Somewhat effective
5. Effective

D.1.5 Avatar Cloaking

Q7) The avatar cloak may be used to restrict the ability of participants to see and hear one’s avatar. Avatar’s for which one has no interact permission are completely removed from one’s perspective of the virtual world. In practice, this metaphor worked well.

With this statement, you:

1. Disagree
2. Somewhat disagree
3. Have no opinion
4. Somewhat agree
5. Agree
D.1.6 Graphical User Interface

Q8) By right-clicking on the walls of a space, on a 3D non-spatial object, or on the DAC button of a 2D window, one can activate the WonderDAC Object Configuration dialog box. This interface was sensible and concise.

With this statement, you:

1. Disagree
2. Somewhat disagree
3. Have no opinion
4. Somewhat agree
5. Agree

Q9) By selecting Avatar Cloak from the Wonderland client Tools menu, one can activate the Avatar Cloak Properties dialog box. This interface was sensible and concise.

With this statement, you:

1. Disagree
2. Somewhat disagree
3. Have no opinion
4. Somewhat agree
5. Agree
Q10) By selecting Ad Hoc Groups from the Wonderland client Tools menu, one can activate the Ad Hoc Group Builder/Maintainer dialog box. This interface was sensible and concise.

With this statement, you:

1. Disagree

2. Somewhat disagree

3. Have no opinion

4. Somewhat agree

5. Agree

D.2 Demonstration Gantt Chart and HSIRB Protocol Review
<table>
<thead>
<tr>
<th>Phase</th>
<th>Step</th>
<th>Description</th>
<th>Notes</th>
<th>Start Time</th>
<th>End Time</th>
<th>Complete</th>
</tr>
</thead>
<tbody>
<tr>
<td>Environment</td>
<td>1</td>
<td>Create participant accounts</td>
<td></td>
<td>-</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>Preparations</td>
<td>2</td>
<td>Create TeamA and TeamB groups</td>
<td>N/A - used test monitor account to create groups</td>
<td>-</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>Add test monitor to team groups</td>
<td></td>
<td>-</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>Assign groups to lofts</td>
<td></td>
<td>-</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>Enable X Window apps in lofts</td>
<td></td>
<td>T-10</td>
<td>T-5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>Assign groups to loft X Window apps</td>
<td></td>
<td>T-5</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>Login</td>
<td>7</td>
<td>Test monitor logs into Wonderland</td>
<td>Prepare screen/audio capture software</td>
<td>T</td>
<td>T+2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>Guest observers log into Wonderland</td>
<td>Guests enable cloaking</td>
<td>T+2</td>
<td>T+3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>Participants log into Wonderland</td>
<td>Participants proceed to their respective team’s loft</td>
<td>T+3</td>
<td>T+5</td>
<td></td>
</tr>
<tr>
<td>Research</td>
<td>10</td>
<td>Participants engage in research; test monitor and guests observe</td>
<td>Teams use web browser, presentation builder, and whiteboard to research and create presentations</td>
<td>T+5</td>
<td>T+35</td>
<td></td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>Test monitor recursively changes groups/perms for lofts to allow full access by all participants</td>
<td></td>
<td>T+33</td>
<td>T+35</td>
<td></td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>Test monitor configures team room group and perms</td>
<td>TeamA can speak, TeamB remains silent (execute in Research phase)</td>
<td>T+10</td>
<td>T+11</td>
<td></td>
</tr>
<tr>
<td></td>
<td>13</td>
<td>Test monitor configures team room PDF viewer</td>
<td>TeamA can use PDF viewer, TeamB cannot (execute in Research phase)</td>
<td>T+11</td>
<td>T+12</td>
<td></td>
</tr>
<tr>
<td></td>
<td>14</td>
<td>Test monitor configures team room mic</td>
<td>TeamA can use mic, TeamB cannot (execute in Research phase)</td>
<td>T+12</td>
<td>T+13</td>
<td></td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>Test monitor activates virtual audio recorder</td>
<td></td>
<td>T+35</td>
<td>T+36</td>
<td></td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>TeamA presents</td>
<td>Each team member presents for 2 mins</td>
<td>T+35</td>
<td>T+41</td>
<td></td>
</tr>
<tr>
<td></td>
<td>17</td>
<td>Guest observers de-cloak</td>
<td></td>
<td>T+41</td>
<td>T+42</td>
<td></td>
</tr>
<tr>
<td></td>
<td>18</td>
<td>TeamA Q&amp;A</td>
<td></td>
<td>T+41</td>
<td>T+45</td>
<td></td>
</tr>
<tr>
<td>Presentation</td>
<td>19</td>
<td>Test monitor deactivates virtual audio recorder</td>
<td>Save audio recording</td>
<td>T+45</td>
<td>T+46</td>
<td></td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>Test monitor configures team room group and perms</td>
<td>TeamA can speak, TeamA remains silent</td>
<td>T+46</td>
<td>T+47</td>
<td></td>
</tr>
<tr>
<td></td>
<td>21</td>
<td>Test monitor configures team room PDF viewer</td>
<td>TeamB can use PDF viewer, TeamA cannot</td>
<td>T+47</td>
<td>T+48</td>
<td></td>
</tr>
<tr>
<td></td>
<td>22</td>
<td>Test monitor configures team room mic</td>
<td>TeamB can use mic, TeamA cannot</td>
<td>T+48</td>
<td>T+49</td>
<td></td>
</tr>
<tr>
<td></td>
<td>23</td>
<td>Test monitor activates virtual audio recorder</td>
<td></td>
<td>T+49</td>
<td>T+50</td>
<td></td>
</tr>
<tr>
<td></td>
<td>24</td>
<td>Teams switch places</td>
<td>TeamA becomes audience, TeamB becomes presenters</td>
<td>T+45</td>
<td>T+50</td>
<td></td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>TeamB presents</td>
<td>Each team member presents for 2 mins</td>
<td>T+50</td>
<td>T+56</td>
<td></td>
</tr>
<tr>
<td></td>
<td>26</td>
<td>TeamB Q&amp;A</td>
<td></td>
<td>T+56</td>
<td>T+60</td>
<td></td>
</tr>
<tr>
<td>Wrap-up</td>
<td>27</td>
<td>Test monitor deactivates virtual audio recorder</td>
<td>Save audio recording</td>
<td>T+60</td>
<td>T+61</td>
<td></td>
</tr>
<tr>
<td></td>
<td>28</td>
<td>Test monitor recursively changes groups/perms for team room, screening room, and demo room to allow full participant access</td>
<td></td>
<td>T+61</td>
<td>T+63</td>
<td></td>
</tr>
<tr>
<td></td>
<td>29</td>
<td>Test monitor changes open/closed sign on screening room to open</td>
<td></td>
<td>T+63</td>
<td>T+64</td>
<td></td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>All users may socialize, move about, and try out Wonderland features</td>
<td></td>
<td>T+64</td>
<td>T+75</td>
<td></td>
</tr>
</tbody>
</table>

Figure D.1. The WonderDAC demonstration Gantt chart.
Notre Dame Human Subjects
Institutional Review Board
PROTOCOL REVIEW

REVIEW DATE:  10-24-2006
Protocol No:  09-004

Full Committee Expedited Review

Principal Investigator: Timothy E. Wright
Gregory R. Mady
Department: Computer Science & Eng

Protocol Title: Privacy and Integrity in Collaborative Virtual Environments

Recommendation: [X] Approved Effective until: 10-24-2009
[ ] Exempt 45CFR 46.101(b)
[ ] Deferred for additional information. See comments:
[ ] Not approved. See comments:

COMMENTS:
Expedited no risks to participants

Please note the following requirements stipulated in the University’s procedures on the use of human subjects in research:
A) The IRB committee’s approval is only for the project protocol named above as submitted for their review. If any changes are to be made to the protocol, the changes are subject to review and approval by the IRB committee prior to implementation.
B) If you intend to continue the project beyond the expiration date listed above, you must submit a request for continuation accompanied by a complete protocol for review and approval by the committee in advance. Human participants may not be used in research unless the project has a currently active protocol.
C) The University of Notre Dame has an approved Federal Wide Assurance on file with the Office for Human Research Protections. Assurance number 1 000002485, IRB number IRB00000325.

Tracey L. Poston, Ph.D.
Director Research Compliance, Office of Research, and Administrator of the NDHSIRB

10-28-2008
Date

Copy to: Gregory R. Mady

Figure D.2. The Notre Dame HSIRB Protocol Review Form.


308


50. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series. Addison-Wesley, Reading, Massachusetts, 1995.


