

Old Dominion University

## ODU Digital Commons

---

Electrical & Computer Engineering Theses & Dissertations

Electrical & Computer Engineering

---

Summer 2004

### Interactive Land Use VRML Application Using Servlet Assist

Suresh Chitithoti  
*Old Dominion University*

Follow this and additional works at: [https://digitalcommons.odu.edu/ece\\_etds](https://digitalcommons.odu.edu/ece_etds)



Part of the [Computational Engineering Commons](#), [Digital Communications and Networking Commons](#), and the [Programming Languages and Compilers Commons](#)

---

#### Recommended Citation

Chitithoti, Suresh. "Interactive Land Use VRML Application Using Servlet Assist" (2004). Master of Science (MS), Thesis, Electrical & Computer Engineering, Old Dominion University, DOI: 10.25777/6wdr-pd16 [https://digitalcommons.odu.edu/ece\\_etds/317](https://digitalcommons.odu.edu/ece_etds/317)

This Thesis is brought to you for free and open access by the Electrical & Computer Engineering at ODU Digital Commons. It has been accepted for inclusion in Electrical & Computer Engineering Theses & Dissertations by an authorized administrator of ODU Digital Commons. For more information, please contact [digitalcommons@odu.edu](mailto:digitalcommons@odu.edu).

**INTERACTIVE LAND USE VRML APPLICATION USING  
SERVLET ASSIST**

by

Suresh Chitithoti  
B.Tech. (ECE) June 1998,  
J.N.T.U. College of Engineering, India

A Thesis Submitted to the Faculty of  
Old Dominion University in Partial Fulfillment of the  
Requirement for the Degree of

**MASTER OF SCIENCES**

**ELECTRICAL ENGINEERING**

**OLD DOMINION UNIVERSITY**  
August 2004

Approved by:

\_\_\_\_\_  
Lee A. Belfore W (Director)

\_\_\_\_\_  
Stephen A. Zahorian (Member)

\_\_\_\_\_  
Roland R. Mielke (Member)

## **ABSTRACT**

### **INTERACTIVE LAND USE VRML APPLICATION USING SERVLET ASSIST**

Suresh Chitithoti  
Old Dominion University, 2004  
Director: Dr. Lee A. Belfore II

The integration of VRML and Servlet technologies has the potential to revolutionize web-based simulation visualization. The Interactive Land Use VRML Application (ILUVA) uses Java Servlets enabling one to perform urban planning by taking a virtual land area and add buildings, roadways, landscaping, and other features. The Virtual Reality Modeling Language (VRML) is a web-based technology for specifying and delivering interactive three-dimensional visualizations over a browser. The Java Servlet technology offers several powerful capabilities such as user authentication, session management, database connectivity for maintaining several possibly simultaneous user sessions, and dynamically generated VRML. Save and restore capabilities for any applications are important for persistent session maintenance. Save and restore is about maintaining a series of requests from the same user (that is, requests originating from the same browser) across some period of time. The thesis work proposes a mechanism using Java Servlets to implement save and restore functionality.

## ACKNOWLEDGEMENTS

There are many people who have contributed to the successful completion of this dissertation. I extend my sincere gratitude to Dr. Stephen Zahorian, and Dr. Roland Mielke for graciously agreeing to be on my committee and for sparing their invaluable time.

I am extremely grateful to Dr. Lee Belfore for his untiring, painstaking efforts in guiding me through this thesis. His constant motivation and relentless perseverance deserve special mention. There is no doubt that this work would have been impossible without his cooperation.

Also, I would like to take this ideal opportunity to thank my family (Nanna, Amma, Akka, Baava, Gupi, and Chinna), and friends for their constant support and motivation. I really appreciate Gundu's persistent encouragement in helping me successfully defend my thesis. He made sure I rehearsed well for my thesis defense.

## TABLE OF CONTENTS

	Page
List Of Figures .....	vii
List of Tables .....	ix
Chapter	
1. INTRODUCTION .....	1
1.1. Prior Work .....	2
1.2. Present Work.....	3
1.3. Internet And The World Wide Web (WWW).....	4
1.4. What Is Computer Simulation?.....	5
1.5. Virtual Reality And VRML .....	6
1.6. Thesis Organization .....	7
2. OVERVIEW OF TECHNOLOGIES AND BACKGROUND ON ILUVA CLIENT .....	9
2.1. VRML.....	9
2.1.1. Introduction.....	9
2.1.2. VRML Application- A Hierarchical Scene Graph.....	10
2.1.3. Shape Primitives .....	11
2.1.4. Grouping Nodes .....	11
2.1.5. Sensors and Interpolators .....	12
2.1.5.1.Introduction.....	12
2.1.5.2.Sensors .....	12
2.1.5.3.Interpolators .....	13
2.1.6. Scripting.....	14
2.1.7. Event Routing .....	15
2.1.8. Prototyping: Encapsulation And Reuse .....	15
2.1.9. The VRML Execution Engine .....	16
2.2. ECMAScript And Java .....	19
2.2.1. ECMAScript .....	19
2.2.2. Java .....	21
2.2.2.1.Types Of Java Applications .....	22
2.3. Java Servlets In Detail .....	22
2.3.1. Servlets Versus Common Gateway Interface (CGI).....	23
2.3.2. The Servlet Application Programming Interface (API).....	25
2.3.3. What Servlets Can Do?.....	29
2.4. ILUVA client .....	30
2.4.1. The Menu .....	31
2.4.2. The Resource Manager .....	32

2.4.3. The Arbiter.....	33
2.4.4. The Collision Manager .....	34
2.4.5. Work Zone .....	34
2.4.6. The Simulation Manager.....	34
2.4.7. The Fixed Landscape .....	35
2.4.8. The Object Model .....	35
<b>3. JAVA SERVLETS- A GENERIC SOLUTION FOR SAVING AND RESTORING INFORMATION.....</b>	<b>38</b>
3.1. Discussion Of Problem .....	38
3.2. Overview Of Java Servlets.....	39
3.3. Java Servlets And ILUVA Client .....	39
3.3.1. Database Connectivity.....	41
3.3.2. Logging Files.....	44
3.3.3. Restoring Sessions.....	47
3.3.4. Dynamic VRML Object Generation .....	52
<b>4. INTERACTIVE LAND USE VRML APPLICATION (ILUVA) WITH SERVLET ASSIST .....</b>	<b>53</b>
4.1. An Overview Of Interactive Land Use VRML Application (ILUVA) Client.....	53
4.2. Description Of Servlets Employed .....	54
4.2.1. Login Servlet .....	54
4.2.2. Reg Servlet .....	58
4.2.3. Insert Servlet .....	60
4.2.4. Dataaccess Servlet .....	63
4.2.5. SessionName Servlet.....	66
4.2.6. Scene Servlet .....	67
4.2.7. Reset Servlet.....	70
4.2.8. Record Servlet .....	70
4.2.9. RestoreSession Servlet .....	73
4.2.10. Display Servlet .....	74
4.2.11. Restore Servlet .....	75
4.3. Implementation Of Sample Session Restoration .....	75
<b>5. FUTURE WORK.....</b>	<b>79</b>
5.1. Limitation In Restore Capability .....	79
5.2. Session Tracking .....	80
5.3. Multi User Virtual Environments (MUVE's).....	80
<b>6. SUMMARY AND CONCLUSIONS .....</b>	<b>83</b>

REFERENCES..... 85

## LIST OF FIGURES

Figure	Page
1. Conceptual model of VRML Browser.....	18
2. Illustration of Request processing by CGI-based Server.....	24
3. Servlet handling a request.....	25
4. Simple Servlet.....	28
5. Overview of the ILUVA operation client.....	31
6. Menu Hierarchy.....	32
7. Object architecture.....	37
8. Interactions between Client and Host Machine.....	40
9. JDBC technology – Flow Chart.....	43
10. Servlets (JDBC programs) interaction with Databases.....	44
11. Interaction between ECMAScript and servlet.....	46
12. Restore node scenegraph.....	48
13. Depictions of User Sessions.....	50
14. Login Servlet.....	55
15. Code segments for Login servlet.....	57
16. Reg servlet.....	59
17. Insert servlet.....	60
18. executeQuery method in Insert servlet.....	62



19. Query to check user authenticity.....	64
20. Dataaccess servlet page layout.....	65
21. Query to retrieve session names .....	66
22. sessionName servlet.....	67
23. Scene servlet .....	68
24. Query for session name existence.....	69
25. Query to insert session values.....	69
26. Class for loading a servlet not currently loaded for reuse .....	72
27. Servlet reuse.....	73
28. Broad Picture of the interaction among servlets .....	78

## LIST OF TABLES

Table	Page
1. Browser Methods .....	21
2. Log file record formats .....	47
3. Major events.....	54

# CHAPTER 1

## INTRODUCTION

Computer simulation has proved its worth as an essential tool to implement better solutions in real world situations. Indeed, web technology is having a significant impact on computer simulation. The widespread use of the World Wide Web (WWW) in the day-to-day lives of the general public is a revolution. The success and further explosive growth and exploitation of the WWW have played a major role in the foundation of Virtual Environments (VE). In addition to being an information resource, the World Wide Web (WWW) is being considered as an environment for future modeling and simulation applications. Also, VRML plays a role in supporting these applications. As evidence and in support of these advances, a 3D interactive visualization language, the Virtual Reality Modeling Language (VRML), has been designed to fit into the existing infrastructure of the Internet and the WWW at the time when the thesis began [1]. But, a recent study of Web3D applications shows that XML encoding is being employed for interactive 3D scene graphs – X3D. X3D is a componentized and improved successor to VRML97 for describing and programming Web3D virtual environments [2]. By using the XML encoding of X3D, developers can leverage the entire XML family of technologies and get a suite of tools for data portability as XML is founded on the distinction between content and presentation.

VRML has been used in a variety of application areas such as engineering, scientific visualization, multimedia presentations, entertainment, education, web pages,

and shared virtual worlds [3]. The predecessor to the Interactive Land Use VRML Application (ILUVA) with servlet assist [4] had a significant weakness, it lacked a mechanism to save and restore a session. Integration of Java Servlets with VRML to achieve this functionality is a novel idea that no one had explored when this research was initiated. The idea of integrating servlets with VRML can be extended to different kinds of VRML applications.

Consider an educational VRML application called the Curriculum Web, developed at the Stanford University School of Medicine [5]. The Curriculum Web is an online bank of information for medical students on classes, events, and useful resources within the Medical School. It is a three-dimensional representation of the world as an academic building, with rooms and objects, through which the information can be accessed. In brief, the Curriculum Web represents a virtual School.

But an important feature that is conspicuously missing in this application is the ability of the virtual world to support live updates and to retrieve information dynamically from outside resources. To overcome these problems, Java Servlets can be integrated with the application. This is just one specific instance where the servlet solution that is proposed in this thesis can be applied, but there are numerous other applications where servlet integration is needed.

### **1.1. Prior Work**

The prior work was a virtual reality tool developed using VRML and ECMA Script [4]. This was an urban simulation application developed for the City of Portsmouth that

enables urban planners to complement traditional marketing strategies with interactive computer visualization. It provides a three dimensional visualization over the World Wide Web (WWW). It gives the user the ability to choose a site, and interactively add features such as buildings, roadways, lot lines, lamps and landscaping over the World Wide Web. It also provides the user with the ability to edit, or even delete the objects they created. Furthermore, the visualization enables the user to conduct an interactive walk through of the site.

## **1.2. Present Work**

This thesis is an enhancement to the predecessor to ILUVA [4]. The predecessor to ILUVA has been integrated with Java servlets to utilize some of its unique capabilities. The major contribution of the thesis is the development of a collection of Java Servlets that interact with the client application [6]. Servlets are capable of performing a wide variety of functions. But, more specifically, the servlets have been employed in the project to provide four important capabilities. First, servlet calls make possible the logging of information generated by a VRML application. Second, servlets can restore the work from a prior session. Third, servlets can be used to generate models specified parametrically. Fourth, servlets can establish a connection to the database to store the user information for session maintenance. The database used in the application is MS-Access. Save and restore functionality is necessary to maintain a persistent record of a session and allow the user to resume the session later, a capability lacking when this thesis began. The thesis employs Java Servlets to implement the save/restore

functionality. This required the development of a collection of Servlets that interact with the VRML application.

The Interactive Land Use VRML Application (ILUVA) provides a user with the ability to take a virtual area and populate it with buildings, roadways, and etc. Also, this application, in integration with servlets allows the user to maintain and restore sessions.

The research required venturing into diverse domains such as Internet and the World Wide Web (WWW), Computer simulations, Virtual Reality Modeling Language (VRML), and technologies such as Java Servlets. These concepts will further be discussed in subsequent sections.

### **1.3. Internet and the World Wide Web (WWW)**

Internet could be defined as a network of networks that allows computers (sometimes called hosts) to communicate seamlessly - usually through Internet Protocol (IP), and services. These computers range from PCs and Macs to supercomputers, but they all use a set of rules called TCP/IP to exchange information. Driven by the popularity of services like electronic mail, file transfer, news groups, and the World Wide Web, the Internet's growth rate has been astonishing. Originally funded by the Department of Defense (DOD) and the National Science Foundation (NSF), the Internet is now paid for and operated by the thousands of institutions that use it [7]. A committee known as the Internet Engineering Task Force defines technical standards.

A major reason for the accelerated growth of the Internet in the last few years is the World Wide Web, a simple yet ingenious system that allows users to interact with

documents stored on computers across the Internet as if they were part of a single hypertext. These documents may contain text, images, sound, movie clips, and other content.

The computers that store web documents are called web servers. These servers run special software that allows client users to connect to the web server and view the documents on them. The web server that has been used in this project is the Apache Web Server [8]. A mechanism is provided to store the log files containing the information pertaining to the objects and users on the web server. These log files are later used to reconstruct the scene and maintain user sessions.

WWW is another example of client/server computing. Each time a link is selected, the client requests a document (or graphic or sound file) from a server (also called a Web server) that is part of the World Wide Web that provides the document. The server uses a protocol called Hyper Text Transport Protocol (HTTP) [9]. The standard for creating hypertext documents for the WWW is Hyper Text Markup Language (HTML). But recently, XML is gradually gaining prominence. HTML essentially codes plain text documents so they can be viewed on the Web.

#### **1.4. What is Computer Simulation?**

Computer simulation is the use of computer algorithms and processes to model real world systems of interest, such as the day-to-day operation of a bank, the value of a stock

portfolio over a time period, the running of an assembly line in a factory, the staff assignment of a hospital or a security company [10].

Computer simulation is the discipline of designing a computer-based model of a real-time system and what are believed to be the rules governing the behavior of the real-time system are captured in this computer-based model [11]. The behavior of this model is used to infer how the system being modeled might itself operate. A simulation approach is sometimes used to understand how an existing system operates or how a proposed system might operate. A simulation approach can be a useful support to decision making. In short, Computer simulation can be described as a technique representing the real world by a computer program to assist in decision making.

### **1.5. Virtual Reality and VRML**

Virtual Reality (VR) has received an enormous amount of attention over the past few years because of its increasing dynamic, interactive, and experiential characteristics, and its ability to simulate real environments with various degrees of realism.

Virtual Reality (VR) is a computer-generated environment with and within which people can interact [9]. The advantage of VR is that it can immerse people in an environment that would normally be unavailable due to cost, safety, or perception restrictions. A successful VR environment offers users immersion, navigation, and manipulation. VR encompasses a range of interactive computer environments, from text-oriented on-line forums and multi-player games to complex simulations that combine audio, video, animation, or three-dimensional graphics.



In short, Virtual Reality (VR) provides the experience of perception and interaction through the use of sensors and effectors in a simulated environment. There are numerous applications in the domains of health care, education and lifelong learning, manufacturing, and other areas where this technology shows great promise for improving productivity.

In order to create these virtual three-dimensional worlds on the Web, VRML is preferred and is widely available. Of late, X3D is preferred to VRML to create these virtual three-dimensional worlds. X3D is next-generation successor to VRML which, like XML, moves beyond just specifying a file format or a language like VRML or HTML [2]. X3D is expressed in XML. Nodes are XML tags, fields are attributes of the tag, and children nodes are the content of the tags. X3D is specified in layers called profiles. Profiles are specific sets of functionality designed to address different application domains from simple geometry interchange or interaction for mobile devices and thin clients to the more full-blown capabilities of graphical workstations and immersive computing platforms. Different profiles offer differing capabilities and are intended for different applications. The full VRML functionality is available with the VRML X3D profile. This profile contains all of the functionality of VRML 97 with the same set of nodes as VRML 97.

## **1.6. Thesis Organization**

The thesis is organized into five chapters. Chapter 1 is the introduction. It briefly explains the problem statement, Internet, the World Wide Web, Computer simulation,

Virtual Reality (VR), Virtual Reality Modeling Language (VRML), and prior work. Chapter 2 discusses the background on ILUVA and gives an overview of all the technologies employed to accomplish the objective of the thesis. Chapter 3 discusses Java Servlets, its integration with the VRML application to achieve the save and restore of a session. Chapter 4 describes in detail the implementation of the ILUVA application with servlets. Chapter 5 discusses the summary and the scope for further research. Chapter 6 is a summary and conclusions.

## **CHAPTER 2**

### **OVERVIEW OF TECHNOLOGIES AND BACKGROUND ON ILUVA CLIENT**

This chapter reviews the various technologies employed to achieve the save and restore mechanism of a VRML session in the application and also gives a background on ILUVA client, the prior work [4]. Section 2.1 gives a basic review of all the major features of VRML. Section 2.2 gives an introduction to the two languages: ECMAScript and Java that have been used in the application. Section 2.3 briefly discusses the types of Java applications. Section 2.4 gives a detailed description of Java Servlets. Section 2.5 gives an account of ILUVA client. How the Java Servlets have been integrated with the VRML application and why they have been used to achieve the save/restore mechanism is explained in detail in the next chapter.

#### **2.1. VRML**

##### **2.1.1. Introduction**

VRML (Virtual Reality Modeling Language) is a hierarchical scene description language that defines the geometry and behavior of a 3D scene or "world" and the way in which it is navigated by the user. At its core, VRML is simply a 3D interchange format. It defines most of the commonly used semantics found in today's 3D applications such as hierarchical transformations, light sources, viewpoints, geometry, animation, fog, material properties, and texture mapping. VRML provides the technology that integrates

three dimensions, two dimensions, text, and multimedia into a coherent model. When these media types are combined with scripting languages and Internet capabilities, an entirely new genre of interactive applications are possible. A 3D metaphor presents a natural user experience that supports classic two-dimensional (2D) desktop models as well as extends into broader contexts of space and place.

### **2.1.2. VRML Application- A Hierarchical Scene Graph**

A VRML application can be described as a hierarchical scene graph of geometries, sensors, light sources, script methods, and grouping primitives assembled in a meaningful fashion. Vertices in the scene graph are called nodes [12]. VRML consists of many different node types including geometry primitives, appearance properties, audio and visual properties, and various types of grouping nodes. All the data pertaining to the nodes are stored in fields. Fields override the default values of the properties of a node. Each node type defines a set of fields that can be specified explicitly. Fields have names and take values of specific types. Each node can have several fields to define node parameters and also to define inputs to (VRML `eventIn` fields) and outputs from (`eventOut` fields) the node. A node can define a solid model, such as a box; a capability, such as an interactive touch sensor; or arbitrary behavior, such as can be defined in a `script` node [12].

The VRML scene graph can be thought of as a directed acyclic graph [12]. Nodes can contain other nodes and more specifically a node can be referenced by other nodes, but a node must not contain itself. This scene graph structure makes it easy to create large worlds or complicated objects from subparts.

### 2.1.3. Shape Primitives

Shape primitives can be basic or general. Basic shapes such as boxes, cylinders, cones, and spheres are defined by name with fields to specify the expected dimensions. More general shape geometries include `Extrusion` and `IndexedFaceSet` geometries enabling specification of fairly arbitrary geometries [12]. The `Extrusion` geometry is an analogy with the shape of the material, such as modeling clay, makes after being forced through an opening with a varying cross section. The `IndexedFaceSet` provides a general appearance representing the object with simple faces.

### 2.1.4. Grouping Nodes

A `Grouping Node` can be defined as a node that groups different node types under a single parent node so that the entire child hierarchy can be easily referenced through that root node, can also be collectively associated with one sensor, and probably have some collective behavior. The `Group` and `Transform` nodes are the only grouping nodes used in ILUVA. The `Group` node groups different nodes to achieve the above said mechanism. The `Transform` node provides the same capabilities as that of `Group` node, but in addition it allows arbitrary scaling, translation, and rotations of the child hierarchy with respect to the coordinate axes.

## **2.1.5. Sensors and Interpolators**

### **2.1.5.1. Introduction**

Sensors sense and generate events when the state of an input device (keyboard or mouse) changes, or changes related to the motion of the viewer or objects in the virtual world or time occur.

Interpolators enable the user to incorporate keyframe animation into the scene [13]. The developer has to specify a list of keyframe values and keys, and the VRML browser will automatically interpolate the "in-betweens." For example, to open a door, the user just needs to describe the door only once and then give the door-closed and door-open positions. The use of Interpolator would give the required animation of door opening and closing.

### **2.1.5.2. Sensors**

VRML consists of nine types of sensor nodes and are broadly classified into two categories: Environmental and Pointing-device Sensors [12].

`ProximitySensor`, `TimeSensor`, `VisibilitySensor`, and `Collision` group nodes are the environmental sensors. The `ProximitySensor` detects when the avatar traverses into a specified volume in the virtual world. The `TimeSensor` has no particular geometry or location associated with it; it is a time base used to start and stop time-based nodes such as interpolators. The purpose of the `VisibilitySensor` is to detect when a particular part of the world becomes visible to the user. The `Collision` grouping node detects the collisions of the user with the objects in the virtual world.

Pointing-device Sensors are of five types. They are `Anchor`, `TouchSensor`, `SphereSensor`, `PlaneSensor`, and `CylinderSensor`. The `Anchor` grouping node replaces the current world with a new scene of which the `Anchor` node is a part when the user clicks on some geometry contained within the `Anchor` node's children except when a `parameter` field is specified. The new world is usually retrieved from a URL generally pointing to a valid VRML file. The `TouchSensor` node detects and generates an event when the mouse cursor is over an affected geometry and also the mouse button is depressed. The `SphereSensor` node maps the mouse motion into spherical rotation about the origin of the local coordinate system. The `PlaneSensor` senses mouse drag actions while the mouse button is depressed. The `CylinderSensor` is same as the `SphereSensor` but maps the mouse motion into a rotation on an invisible cylinder aligned along the Y-axis of the local coordinate system.

### 2.1.5.3. Interpolators

In VRML the following interpolators are available:

`ColorInterpolator`, `CoordinateInterpolator`,  
`NormalInterpolator`, `OrientationInterpolator`,  
`PositionInterpolator`, `ScalarInterpolator`.

Interpolators have an `eventIn` `set_fraction`, and an `eventOut` `value_changed`. These two events are wired together, i.e. when the node receives a `set_fraction` event it generates a `value_changed` event. The `set_fraction` event specifies a key value; the `value_changed` event outputs the data, specified in

key value which corresponds to the key value given. Interpolators combined with `TimeSensors` can be used to create animations, color variations, morphing objects, etc.

### 2.1.6. Scripting

`Script` nodes allow arbitrary, user-defined event processing. The script receives an input event, does the required processing, and then generates the necessary output events. An event received by a `Script` node causes the execution of a script method having the ability to generate events through the conventional event-routing mechanism as declared in the `ROUTE` constructs or can bypass this mechanism and send events directly to any node to that the `Script` node can reference [14]. For example, a script can take events from a `PlaneSensor` tracking a mouse drag and convert this into an output event to produce a set of vertices to change geometry. Scripts also have the ability to dynamically add or delete routes, thereby changing the application event-routing structure.

A time stamp is assigned to all the events to determine when events occur, and to control time-dependent nodes. The timestamp serves two purposes. First, it is a conceptual device used to describe the chronological flow of the event mechanism [12]. It ensures that deterministic results can be achieved by real-world implementations that must address processing delays and asynchronous interaction with external devices. But, sometimes results are not deterministic due to networking delays. Second, timestamps are also made available to `Script` nodes to allow events to be processed based on the order of user actions or the elapsed time between events.



### **2.1.7. Event Routing**

Events can be generated and received by most nodes. Event routing gives content a mechanism, separate from the scene graph hierarchy, through which these events can be propagated to effect changes in other nodes [12]. Once generated, events are sent to their routed destinations in time order and processed by the receiving node. This processing can change the state of the node, generate additional events, or change the structure of the scene graph.

An `eventIn` is used to receive input while an `eventOut` outputs information. For example, a `TouchSensor` node generates an event when a mouse click occurs on the target geometry. This event can be received by a script making a wire frame appear around the selected geometry.

`ROUTE` constructs declare the desired connections between the output events of one node to input events of another.

### **2.1.8. Prototyping: Encapsulation and Reuse**

VRML provides a mechanism called Prototyping for encapsulating and reusing scene graph components. Prototyping provides a mechanism for user-defined nodes. It allows the developer to encapsulate any geometry, properties, and animations or behaviors, either separately or together. The prototype definition can be included in the file in which they are used using a `PROTO` declaration, or can be defined externally using an `EXTERNPROTO` declaration.

The `EXTERNPROTO` is similar to the `PROTO` statement but with two exceptions. First, the implementation details of the node type are stored externally, either in a VRML file containing an appropriate `PROTO` statement or using some other implementation-dependent mechanism. Second, default values for fields are not given since the implementation will define appropriate defaults.

### **2.1.9. The VRML Execution Engine**

A VRML world can be viewed schematically as a scene graph where nodes describe geometries, scripts describe behaviors, and routes define information flow. The VRML execution engine is an integral part of VRML browser plug-in that executes the scene graph [12]. Figure 1 illustrates a conceptual model of a VRML browser.

The browser can be described as a presentation application that accepts user input in the form of file selection and user interface gestures such as manipulation and navigation using an input device [12]. The three main components of the browser are: parser, scene graph, and audio/visual presentation. The parser component reads the VRML file and creates a scene graph. The scene graph component consists of a transform hierarchy (the nodes) and a `ROUTE` graph (the connections between nodes). The scene graph also includes an execution engine that processes all events and nodes. The source and target for these events are specified using the `ROUTE` declaration. User input generally affects sensors and navigation, and thus is wired to the `ROUTE` graph component and the audio/visual presentation component. The audio/visual presentation

component performs the graphics and audio rendering of the transform hierarchy that feeds back to the user.

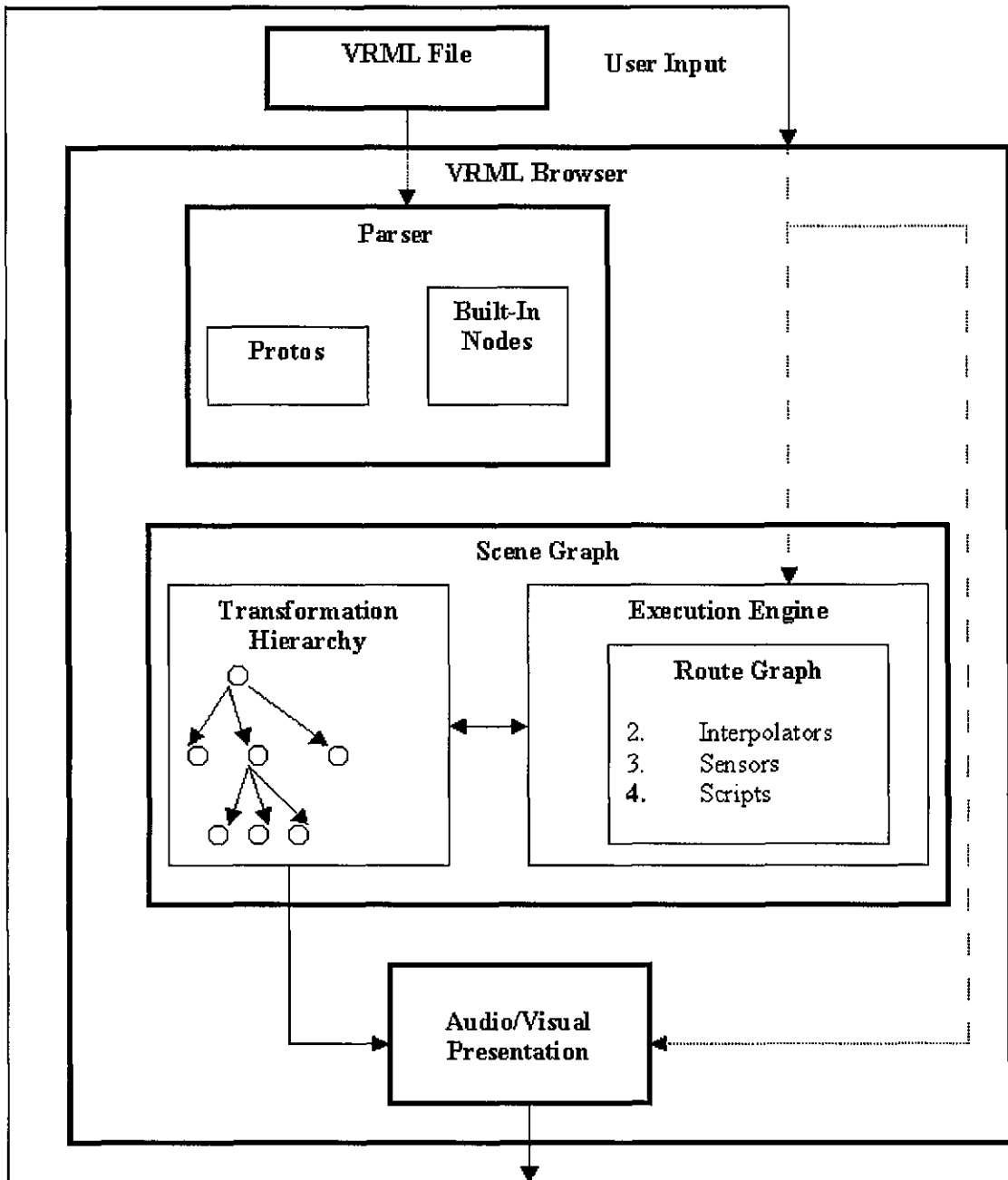


Figure 1. Conceptual model of VRML Browser [12]

## 2.2. ECMAScript and Java

There are two languages that VRML specifications support: ECMAScript and Java [15].

### 2.2.1. ECMAScript

ECMAScript is a general purpose, cross-platform programming language and is a standard based on several originating technologies, the most well known being JavaScript (Netscape), and Jscript (Microsoft). It is an object-oriented programming language within a host environment. ECMAScript was originally designed to provide dynamic capabilities in web pages, and to perform client computation as part of Web-based client-server architecture. But, with the addition of many advanced features it evolved into a powerful language. Some of the methods that have been used in this application are `initialize()`.

The `initialize()` method is invoked before the browser presents the world to the user and before other nodes in the same VRML file as the `Script` node containing this script process any events. Events generated by the `initialize()` method have timestamps earlier than any other events generated by the `Script` node thus allowing the script initialization tasks to be performed prior to the user interacting with the world.

Besides interacting with the containing `Script` node, the scripts can also interact with the user browser using the `Browser` interface class. The `Browser` object is available in the ECMAScript as well as Java [13]. There are some functions defined in the `browser` object that can be used to create VRML content from an URL, or a text

string, load content externally into the VRML application, and to add and delete routes. `CreateVRMLFromURL()`, `CreateVRMLFromString()`, `loadURL()`, `addRoute()`, `deleteRoute()` are some of the Browser's methods used in this application. These methods have been used in an efficient way allowing object creation, manipulation, and user interaction features to be added dynamically at the run-time. The Table 1 below briefly explains the purpose of these methods.

**Table 1. Browser Methods**

<b>Method</b>	<b>Purpose</b>
<code>createVRMLFromURL()</code>	Instructs the browser to load a VRML scene description from the given URL or URLs
<code>createVrmlFromString()</code>	Imports a string consisting of a VRML scene description, parses the nodes contained therein, and returns the root nodes of the corresponding VRML scene
<code>loadURL()</code>	Loads the file specified by the URL. The URL need not necessarily point to a valid VRML file, it can be any file.
<code>addRoute()</code>	Adds a route between the given event names for the given nodes.
<code>deleteRoute()</code>	Deletes a route between the given event names for the given nodes

### 2.2.2. Java

Java is a high level, general purpose, purely object-oriented, and very portable, programming language developed by Sun Microsystems [16]. In the context of web development, Java can be used to create standalone applications and a special type of application, called an applet. Applets are downloaded as separate files to the browser

alongside an HTML document, and provide an infinite variety of added functionality to the Web content.

### **2.2.2.1. Types of Java Applications**

Java is designed to facilitate the creation of three types of programs: Applications, Applets, and Servlets [17]. Applications are stand-alone programs in the traditional sense but are based purely on core object oriented concepts. They are confined to the operating system of a particular machine i.e. they are not portable. Applets can be defined as mini-applications that are specifically designed to be embedded and called from HTML documents. They have to be downloaded from a server in order to be executed on a client. However, they are usually not stored on the client and are generally prevented from accessing files on the client. They are typically used to add animations or user interactivity to a Web page. Servlets on the other hand, can also be defined as mini-programs but, unlike applets, are executed on the server. Instead, Servlet output is a data stream representing the client content. Similar to the way applets run on a browser and extend the browser's capabilities, Servlets run on a Java-enabled Web server and extend the server's capabilities.

## **2.3. Java Servlets in Detail**

Servlets can be thought of as server-side Java. Servlets are to server what applets are to browsers. Unlike applets, however, servlets have no inherent Graphical User Interface (GUI). Just as applets need Java support for the browser to run, in the same way, servlets



require a Java Virtual Machine (JVM) to run a servlet. In addition, the web server has to support the Java Servlet Application Programming Interface (API). An overview of the architecture of the servlet API will be explained later in this section. A servlet can be loaded automatically when the web server is started, or it can be loaded with the first client servlet request. After loading, a servlet continues to run, waiting for additional client requests.

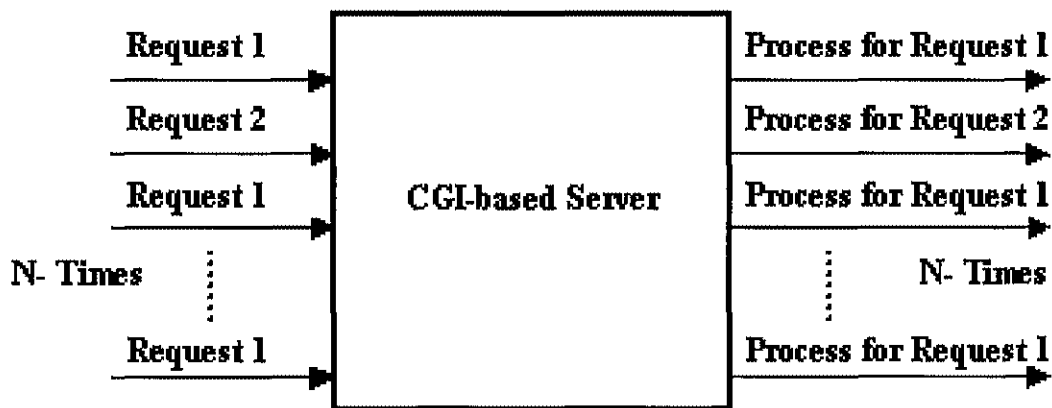
The web server used in this application is the Apache Web Server [8]. Apache is a powerful, flexible, HTTP compliant web server that implements the latest protocols. It is highly configurable and extensible for third-party modules [8]. Apache does not have built-in support for servlets. An add-on servlet engine called Apache JServ was used to provide servlet support. A servlet engine can be defined as a plug-in to an existing server that adds servlet support to the server. Configuration of Apache JServ to run servlets on Apache is described in detail in [18].

### **2.3.1. Servlets versus Common Gateway Interface (CGI)**

Servlets can be embedded in many different servers because the servlet API, used to write servlets, assumes nothing about the server's environment or protocol. Servlets have become most widely used within HTTP servers [19].

Java Servlets are more efficient, easier to use, more powerful, more portable than traditional CGI and than many alternative CGI-like technologies. With traditional CGI, a new process is started for each HTTP request. If the CGI program does a relatively

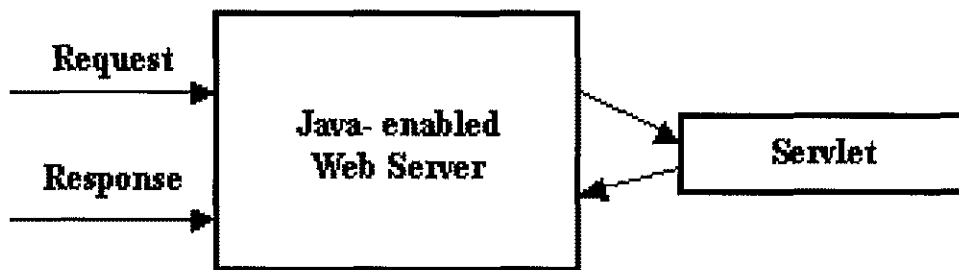
simple operation, the overhead to starting the process can dominate the execution time. With Servlets, the Java Virtual Machine (JVM) continues expecting, and each request is handled by a lightweight Java thread, rather than a heavyweight operating system process. Similarly, in traditional CGI, if  $N$  simultaneous requests are made to the same CGI program, then CGI program is loaded into memory  $N$  number of times. The handling of requests by CGI server is illustrated in Figure 2. With Servlets, however,  $N$  threads are generated, but only a single copy of the servlet class is stored in memory. Servlets also provide more alternatives than do regular CGI programs for optimizations such as caching previous computations, keeping database connections open, and so on.



**Figure 2. Illustration of Request processing by CGI-based Server**

### 2.3.2. The Servlet Application Programming Interface (API)

Servlets extend server capabilities by creating a framework for providing request and response services over the Web. When a client sends a request to the server, the server can send the request information to a servlet and have the servlet construct the response that the server sends back to the client. This request-response paradigm is illustrated in Figure 3.



**Figure 3. Servlet handling a request**

The servlet API consists of two packages: `javax.servlet` and `javax.servlet.http` [19]. Every Servlet must use classes and interfaces from these two packages. The top-level package in the servlet API is `javax` instead of the familiar `java`, which indicates that Servlet API is a standard extension.

The `javax.servlet` package provides interfaces and classes for writing generic, protocol-independent servlets. Some of the protocols that are presently in use are `Ftp`, `Telnet`, `Gopher`, and `Simple Mail Transfer Protocol (SMTP)`. To add specific protocol-specific functionality such as `HTTP`, the classes in the `javax.servlet` package are extended by the classes in the `javax.servlet.http` package. Extending a class is one of the object-oriented methodologies of the Java programming language [20].

The central abstraction in the Servlet API is the `servlet` interface. Every servlet must implement the `javax.servlet.Servlet` [19]. Every servlet implements this interface by extending one of the two special classes: `javax.servlet.GenericServlet` or `javax.servlet.http.HttpServlet`. A protocol-independent servlet extends `GenericServlet`, while an `HTTP` servlet extends `HttpServlet`, which is itself a subclass of `GenericServlet` with added `HTTP`-specific functionality.

Unlike a Java program, and just like an applet, a servlet does not have a `main()` method. Instead, the server in the process of handling of requests invokes certain methods of a servlet. Each time the server dispatches a request to a servlet, it invokes the servlets' `service()` method [19].

A generic servlet should override its `service()` method to handle requests as appropriate for the servlet. When a `service()` method of the servlet accepts a call

from a client, it receives two objects: a request object of type `ServletRequest` and a response object of type `ServletResponse`. A `ServletRequest` encapsulates the communication from the client to the server. A `ServletResponse` encapsulates the communication from the servlet back to the client. `ServletRequest` and `ServletResponse` are interfaces defined by the `javax.servlet` package.

In contrast, an HTTP servlet accepts objects of two types: `HttpServletRequest` and `HttpServletResponse` and usually overrides `doGet()` and `doPost()` to handle GET requests and POST requests respectively instead of overriding the `service()` method. Figure 4 below illustrates all the basic features of the servlets explained above.

```
public class SimpleServlet extends HttpServlet    {

// Handle the HTTP GET method.
public void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException,
IOException {

    PrintWriter out;
    String str = "Simple Servlet Output";
// set content type and other response header fields
first
    response.setContentType("text/html");
// then write the data of the response
    out = response.getWriter();
    out.println("<HTML><HEAD><TITLE>");
    out.println(str);
    out.println("</TITLE></HEAD><BODY>");
    out.println("<H1>" + str + "</H1>");
    out.println("Output from Simple Servlet");
    out.println("</BODY></HTML>");
    out.close();
        }
    }
```

**Figure 4. Simple Servlet**

SimpleServlet extends the HttpServlet class that implements the Servlet interface. SimpleServlet overrides the doGet method in the HttpServlet class. The doGet method is called when a client makes a GET request (the default HTTP request method), and results in the simple HTML page being returned to the client. The user's request is represented by an HttpServletRequest object within the doGet() method. The response to the user is represented by an

`HttpServletResponse` object. Because text data is returned to the client, the reply is sent using the `Writer` object obtained from the `HttpServletResponse` object.

### **2.3.3. What Servlets can do?**

Servlets are capable of performing a wide variety of functions. They are employed primarily for four purposes.

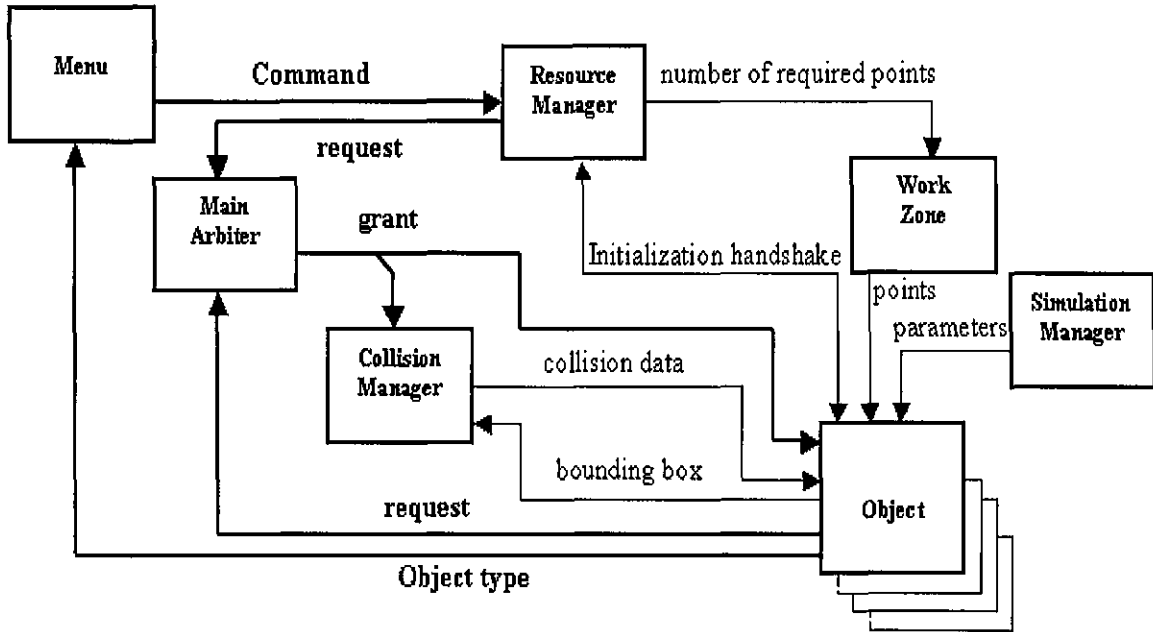
1. Database connectivity. It is explained in detail in Chapter 3.
2. Reading and writing files. The web browser has security limitations in that files on the client machine cannot be arbitrarily read and written, where as, the server can be easily configured to read and write files on the server file system. The mechanism for logging a session requires interaction between servlets, ECMAScripts in the VRML application. A simple method for passing information is for the VRML application to load the universal resource locator (URL) for the servlet. Arguments passed to the servlet URL supply the information to be logged. This is accomplished in the VRML application using the `loadURL` browser method. In this fashion, information from the VRML session is logged on the server. For example, a `Record` servlet was employed in this capacity.
3. Generating VRML to restore a session. Servlets must be able to scan the log file, collect a list of all objects that were saved, and retrieve the most recent set of updates for each. Next, a VRML object is created that encapsulates the collection of restored objects. The object architecture is designed so that by appropriate instantiation, the object is configured so that it appears identical to the prior session.

4. Generate VRML object for a session. The servlets are a powerful mechanism for dynamically generating VRML content. For example, parking lots associated with office buildings are sized according to the building square footage. Per city building codes, parking lots are required to have a particular canopy cover fraction, i.e. trees. The number of trees varies with the parking lot size. In the application, one parking lot model represents all parking lot instances and therefore, the requisite number of trees must be generated dynamically. In previous versions of the visualization, the trees were generated using ECMAScript. While functional, this approach has resulted in instabilities and resource problems. Presently, a servlet has been designed to generate an object containing the appropriate number of trees. This has resulted in faster and more reliable operations.

#### **2.4. ILUVA Client**

Interactive Land Use VRML Application (ILUVA) is a decentralized collection of eight interconnected, concurrently operating modules as shown in Figure 5 [6]. They are 1) The Menu, 2) The Resource Manager, 3) The Arbiter, 4) The Collision Manager, 5) The Work Zone, 6) The Simulation Manager, 7) The Fixed Landscape (not shown in the Figure 5), and 8) The Object models.





**Figure 5. Overview of the ILUVA operation client [6]**

#### 2.4.1. The Menu

The menu is a hierarchical collection of interconnected geometry's as shown in Figure 6 [21]. Edges in the Figure 6 show the flow of information between menus. The output of the menu hierarchy is the `command` event that initiates the object addition and controls object manipulation. When an object is selected, an `ObjectType` input event is sent by the selected object, which the menu hierarchy uses to make the appropriate menu appear.

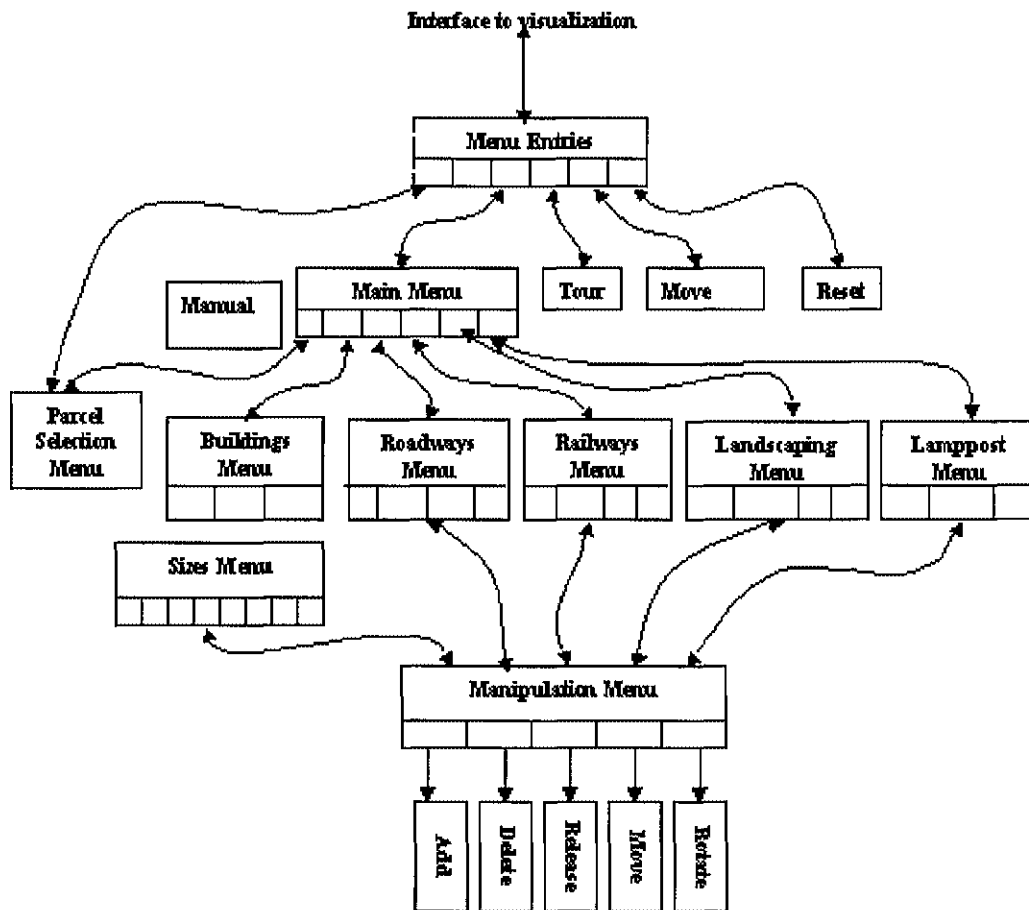


Figure 6. Menu Hierarchy [21]

#### 2.4.2. The Resource Manager

The Resource Manager manages the dynamic instantiation of objects and orchestrates initialization sequencing for new objects [22]. The resource manager has a

dynamic table that maintains records of objects that have been added in the current session. The resource manager monitors the `command` event generated by the menu, if it is a `add` command event, it dynamically creates a new object using `createVrmlFromURL` Browser method and then assigns a unique serial number to the object. The object's corresponding entries are reserved in the dynamic table to keep track of its state. After creation of the object, the resource manager looks into the static table to initiate object initialization sequences and then subsequently creates event routes between the newly added object and the world using the `addRoute` method of Browser interface.

#### **2.4.3. The Arbiter**

The main functionality of the arbiter is to control the access of common resources between a currently active object and the inactive object that has been selected for update or examination [21]. Each and every object has its own local arbiter resulting in as many arbiters as objects. The operation is a simple hand shaking communication of two events `request` and `grant` between the main arbiter and the local arbiters. When an object is selected, it emits the serial number in the `request` event monitored by main arbiter. The global arbiter broadcasts the `grant` event after some delay to all the object arbiters and resource allocator to relinquish the resources thereby deselecting the remaining objects. To simplify the arbitration process, only one object is allowed to share the resources.

#### **2.4.4. The Collision Manager**

Since VRML does not support full collision detection between geometry's in the application, collision detection is explicitly implemented. The collision manager maintains an ordered list of the bounding boxes for all the objects in the virtual world. When an object is edited or moved, its new bounding box is compared to the bounding boxes of all monitored objects and generates a collision detection event when their spatial extents coincide. The collision detection mechanism implemented in the application is very simple that just freezes the ability to drag or modify the object.

#### **2.4.5. Work Zone**

The work zone is designed mainly for three purposes: 1) To outline the working area, 2) To provide a perceivable cue to the user for object insertion, 3) And to process the number of mouse click events to initialize the geometry for each object. When the new object receives all points, it utilizes the points to define its initial geometry and makes itself visible.

#### **2.4.6. The Simulation Manager**

At present, this module has been newly added only for demonstration purposes to the high level architecture to test some simple simulation features [22]. Its function is to store the simulation data and broadcasting it to all data objects. In the application, this has been programmed to produce an oscillating wind and the pine tree responds to give a realistic impression of a tree swaying in the wind.

#### **2.4.7. The Fixed Landscape**

The Fixed Landscape is a solid model that defines the work zone [21]. It consists of geographic features, buildings, ponds, etc. that are expected to be permanent and fixed. This data is obtained from the combination of Global Information Systems (GIS), city personnel of Portsmouth and city architects. It clearly distinguishes the working area from the virtual world. Restoring a previous session requires that these fixed objects be immediately instantiated into the scene. Hence, these are statically instantiated for immediate restoration.

#### **2.4.8. The Object Model**

The Object architecture is designed to accomplish four purposes [21]. First and foremost, the Object architecture is designed to be independent and autonomous, as much as possible from the rest of the visualization. Second, each of the object modules is a self-contained representation of the object. In order to achieve this purpose, each object defined by a unique `PROTO` node declaration is contained in a file with the name being that of the corresponding object. To create an instance of the object in the visualization, another file containing an `EXTERNPROTO` node followed by a single instantiation of the object is created. This coding methodology ensured that the object is self-contained in a single `SFNode` field while the second file is used for dynamic insertion of the object in the visualization. Third, the object must be capable enough to communicate its changes in geometry and appearance to a web server. The monitor script node of the interaction layer in the object architecture acts as an interface between the object and the server. The monitor receives the object updates and communicates the changes by servlet calls. The

monitor accomplishes the call to servlets using the `loadURL` browser method. Fourth, to enable static instantiation of the Object node as is required for restoring a prior session. Specific fields in the node prototype are used to both signal static instantiation and configure the object. All the fields that are necessary to recreate all information from the prior session are included.

To attain this type of generality for all the objects and to make the pure behavior of the object independent of its editing controls and user interface to the visualization the object architecture is divided into three layers [21]. 1. The Interaction layer 2. The Edit layer, and 3. The Model layer, as shown in Figure 7. This well layered object architecture makes the object models independent and autonomous thus simplifying the process of adding new object models into the visualization. The interaction layer manages the interface to the visualization and any user interactions that affect the object as a unit, such as drag and rotation. The edit layer controls the geometric appearance of the object model. The model layer is concerned with only the object's pure behavior and appearance.

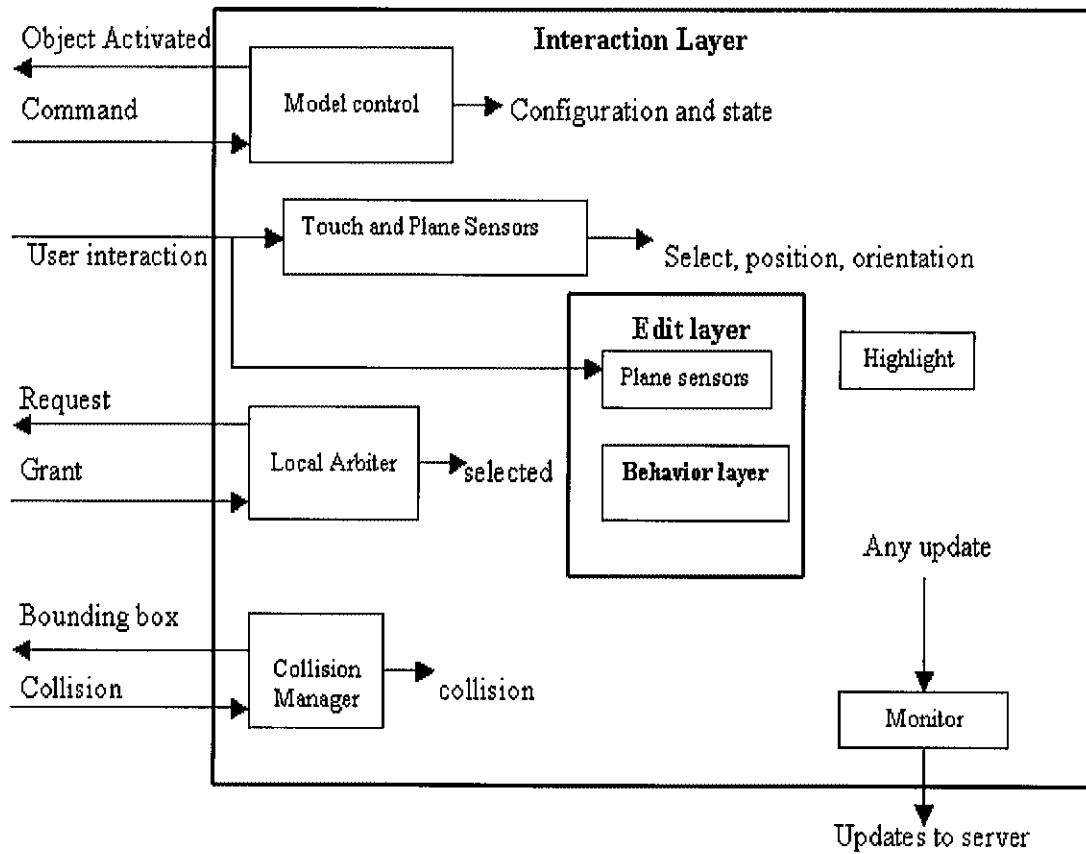


Figure 7. Object architecture [21]

## CHAPTER 3

# JAVA SERVLETS- A GENERIC SOLUTION FOR SAVING AND RESTORING INFORMATION

### 3.1. Discussion of Problem

Save and restore capabilities are essential for maintaining a persistent record of a session and allowing the user to resume that session at a later time. In stand-alone based web applications, security features that the browser imposes on applications that run within a browser complicate the implementation of this task: typically, files cannot be read or written on a host machine with the exception of signed applications [23]. The predecessor [4] to the Interactive Land Use VRML Application (ILUVA) with servlet assist [6] has this conceivable drawback, for it lacked the mechanism to save and restore a session. For example, if the user added objects such as roadways, buildings, etc., and then wanted to save this information, cannot do so because of browser constraints.

In order to have true save/restore functionality without violating the desired security checks or exploiting vulnerabilities, server-based methods for reading and writing files on the server machine are required. In this application, methods for reading and writing files have been implemented using Java Servlets. Integration of Java Servlets with the VRML application to obtain the save/restore capability is explained in detail in later sections. The Interactive Land Use VRML Application (ILUVA) as such, in association with Java Servlets is explained in the next chapter.



### **3.2. Overview of Java Servlets**

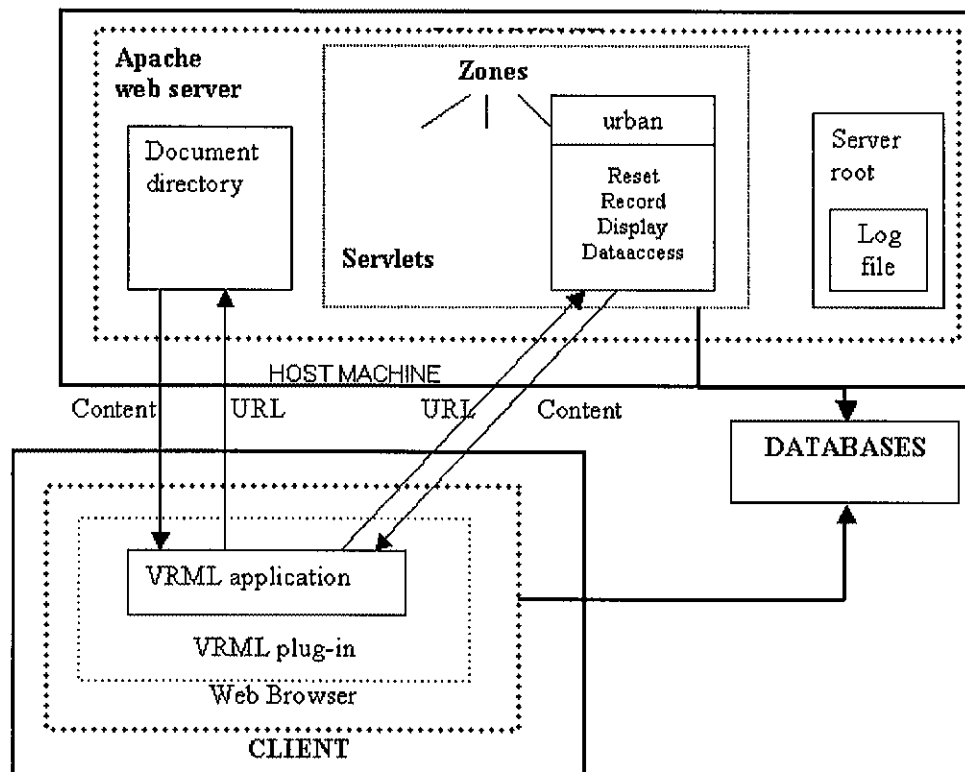
Java is a powerful language for server side development of web-based applications. The same features that make Java a good language for writing client applications, also make it advantageous for developing server-side applications. Java servlets are a key component of Java development on the server side. They provide a general framework for services built using the request-response paradigm. Their primary use is to provide secure web-based access to data, which is presented using HTML web pages, interactively viewing or modifying that data using dynamic web page generation techniques.

A servlet can be defined as a small, pluggable generic extension to a server or a Java class that can be loaded dynamically to enhance the server's functionality. Servlets run inside the Java Virtual Machine (JVM) on the server. Hence, they are secure from the client side, since the programs do not run on the client side and are portable. Since servlets run on the server, the client side support for Java is not necessary [19].

### **3.3. Java Servlets and ILUVA Client**

Java Servlets facilitate the incorporation of Java programs that run on the web server. Servlets have the ability to store and retrieve information from various data files and can, in turn, generate web content of any type. For instance, in our application, servlets have been developed to produce VRML content to save and restore a session. Save and Restore capabilities for the client application have been achieved by employing servlets having four different capacities: 1) to support database connectivity for maintaining

several possibly simultaneous user sessions 2) to provide the ability to read and write files 3) to generate VRML content to restore a session and 4) to generate a VRML object dynamically for a session. The interaction, in the context of Servlets is illustrated in Figure 8. To make use of servlet technologies, the Apache web server version 3.1.12 [8], SUN servlet Development Kit version 2.0 [16], and the Apache servlet engine Apache JServ version 1.2 [18] have been used.



**Figure 8. Interactions between Client and Host Machine**

### 3.3.1. Database Connectivity

Today, few professional web sites lack some sort of database connectivity. The biggest advantage for servlets with respect to database connectivity is that open database connections can be maintained unlike CGI that opens a new connection for every process. Another added advantage servlets provide with JDBC technology is database independency. Servlets developed to access Oracle database can access Sybase database, or any other database with little modifications. JDBC technology is an API provided in Java language to perform queries against a database. The ILUVA developed is a web tool that can be used by many users all over the world. Access control servlets, querying the database were developed, to control the use of this site to users, and to maintain their respective VRML sessions. Executing queries to access database is fundamental for maintaining user sessions. Queries are executed against the database to perform actions such as checking the authenticity of the user logged in, retrieving all the prior session names of a user. The database used in the application is Microsoft Access, the standard ODBC source. The database stores all the information regarding the users like login, password, address, email ID, user-defined session names, and so on. JDBC is relatively straightforward. It consists of three interfaces: Connections, Statements, and Results. A typical operation would establish a connection with a database, execute a query against that database, and return a result set. A typical operation of JDBC technology is depicted as a flowchart in Figure 9. The interaction between Java servlets and the database through the JDBC technology is illustrated in Figure 10.

Establishing a database connection requires loading the database-specific driver into the application's JVM making the driver available later when connections are needed. Once the connection has been established, to really use the database one has to have a way to execute queries, retrieve the information, and display the results.

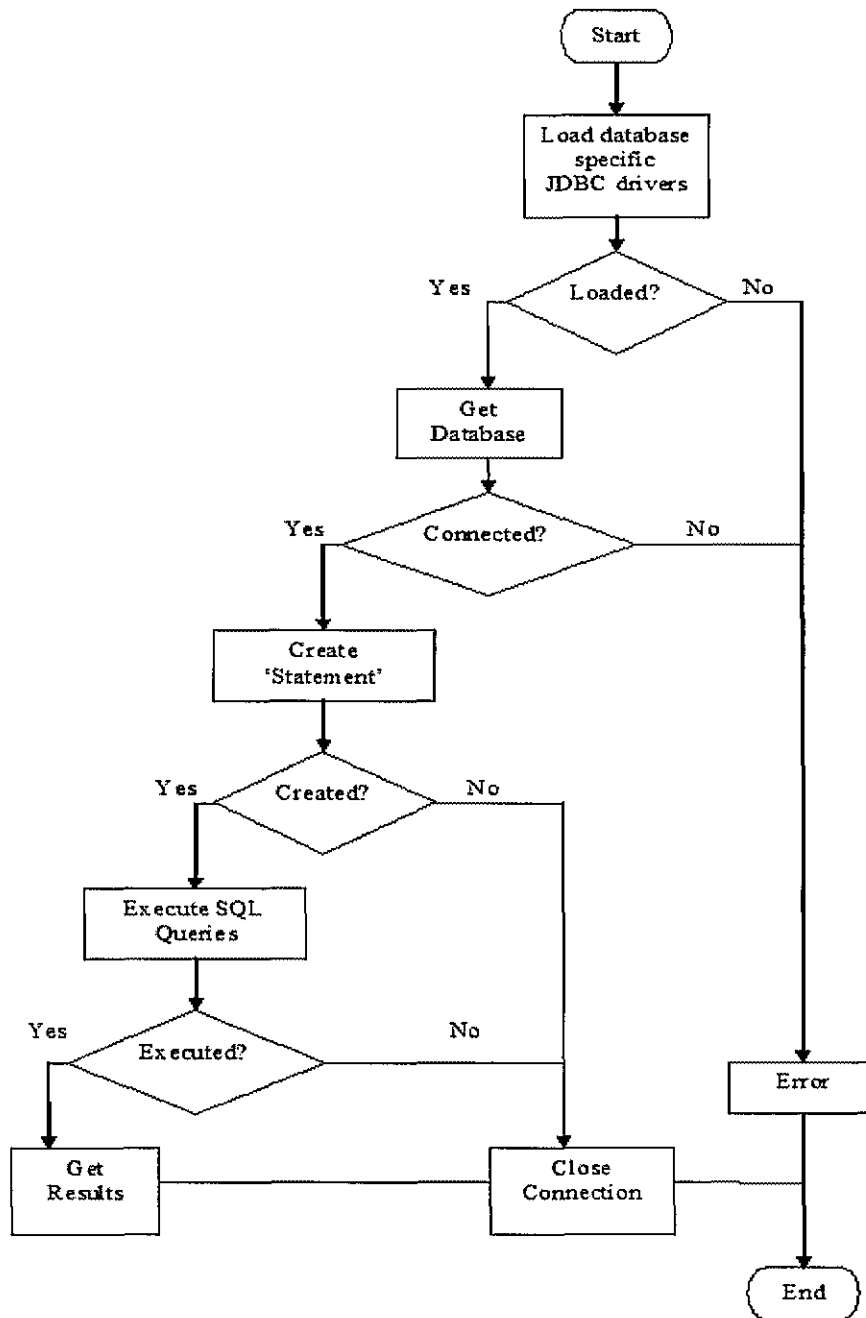
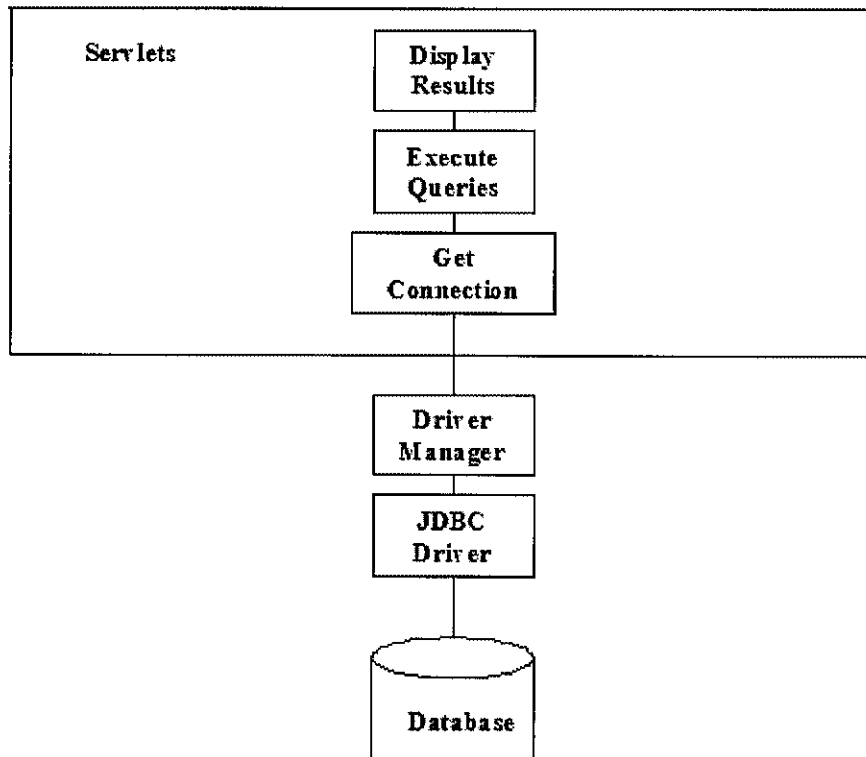


Figure 9. JDBC technology – Flow Chart



**Figure 10. Servlets (JDBC programs) interaction with Databases**

### 3.3.2. Logging Files

One capability noticeably absent in VRML is mechanism for scene persistence. VRML lacks any direct mechanism to save and restore a scene. When a VRML application is initially loaded, the scene is initialized to a certain state. Any subsequent user interactions will change the state of the scene. However, because VRML language provides no mechanism to store those changes, the changes are lost when the user quits

the scene. The scene in initial state is presented when the application is reloaded. The prior work [4] to ILUVA with servlet assist [6] had this drawback. ILUVA enables the user to choose a virtual site, and interactively add features such as buildings, roadways, lot lines, lamps, and landscaping. It also provides the user with ability to edit, or even delete the objects they created. But once the user exits the application, the objects the user created are lost. The user has no provision to continue the same session at a later time. One solution to this drawback is to log the user updates into a file on the server, which can later be scanned, and parsed to reproduce the session.

Therefore, a VRML session is logged on the server so that its information can later be parsed to restore the session. The mechanism for logging a session requires interaction between servlet and the ECMAScript in the VRML application. A simple method for passing information is for the VRML application to load the universal resource locator (URL) for the servlet. The query string that is appended to the servlet URL constitutes the session information to be logged. The `loadURL` Browser method is used to implement the above mechanism in the VRML application. For instance, we have developed a servlet called `Record`. The purpose of this servlet is to store all the information passed from a VRML session to a log file. Figure 11 illustrates how an ECMAScript storing a three-dimensional coordinate in a `SFVec3f` variable point may be passed to a servlet.

```
Urls[0]='http://hostmachine/servletzone/Record?' +point[0]+' ':' +point[1]+' ':' +point[2] ;  
Parameters [0]=' target=updateFrame' ;  
Browser.loadURL(urls, parameters);
```

### Figure 11. Interaction between ECMAScript and servlet

Additions and updates are recorded chronologically, with the recent modifications representing the final appearance. Figure 8 illustrates the interaction between these processes. The server and client machines need not necessarily be different machines. For example, a stand-alone system can be configured to run Apache while at the same time the client browser can request documents from the server. Table 2 lists the different log file record types. The log file format is straightforward and easy to parse. The `id` field is the index for the object and it is a unique identifier that can later be used to associate subsequent updates to that object. As described in earlier chapter, the resource manager generates the log entry when an object is created. The `type` field determines what type of object is selected for update. The object types identify the object geometry. New entries provide this information only. Updates to the geometry require the field updated and the value for that field. Fields are single values, coordinates, or arrays of values. To restore a session, the log file is read and the last value recorded for a field is updated for the restoration. In the event the field is not modified, the defaults values in the object model are used.



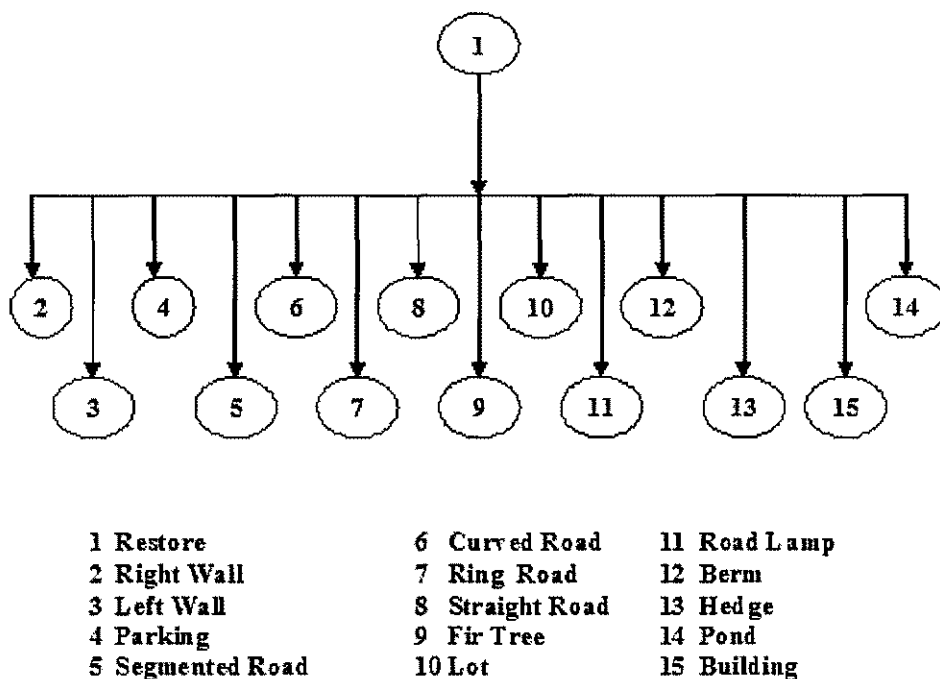
**Table 2. Log file record formats**

<b>Type</b>	<b>Format</b>
New Entry	id:type
Update, simple	id:type: (field) :-:value
Update, x, y	id:type: (field) :-:x-value:y-value
Update, array	id:type:(field) :-:n:value <sub>1</sub> :value <sub>2</sub> :-: ..... :-:value <sub>n</sub>

### 3.3.3. Restoring Sessions

To accomplish the complementary function of restoring prior sessions, a servlet has been developed that scans the log file containing the VRML information to be restored, and retrieves the most recent set of updates for the objects. Using this information, the content of a prior session is represented as a user-defined (prototyping) node called `Restore` that encapsulates all the restored objects. All the other nodes necessary for the objects restoration are passed to the `Restore` node. The object nodes encapsulated in the `Restore` node are so designed that by supplying their configuration, they are so placed that they appear identical to the prior session. All this process is accomplished by the `Display` servlet in the application. But the `Restore` node produced by this servlet consists of non-editable objects. The scene graph depicting the `Restore` node is shown in Figure 12. To bring the recovered objects into the scene graph, the `Restore` node has to be instantiated. In the application, it is the `RestoreSession` servlet that references the restored VRML file. This VRML file

incorporates this restored landscape by instantiating the Restore node thus enabling to load the restored visualization.

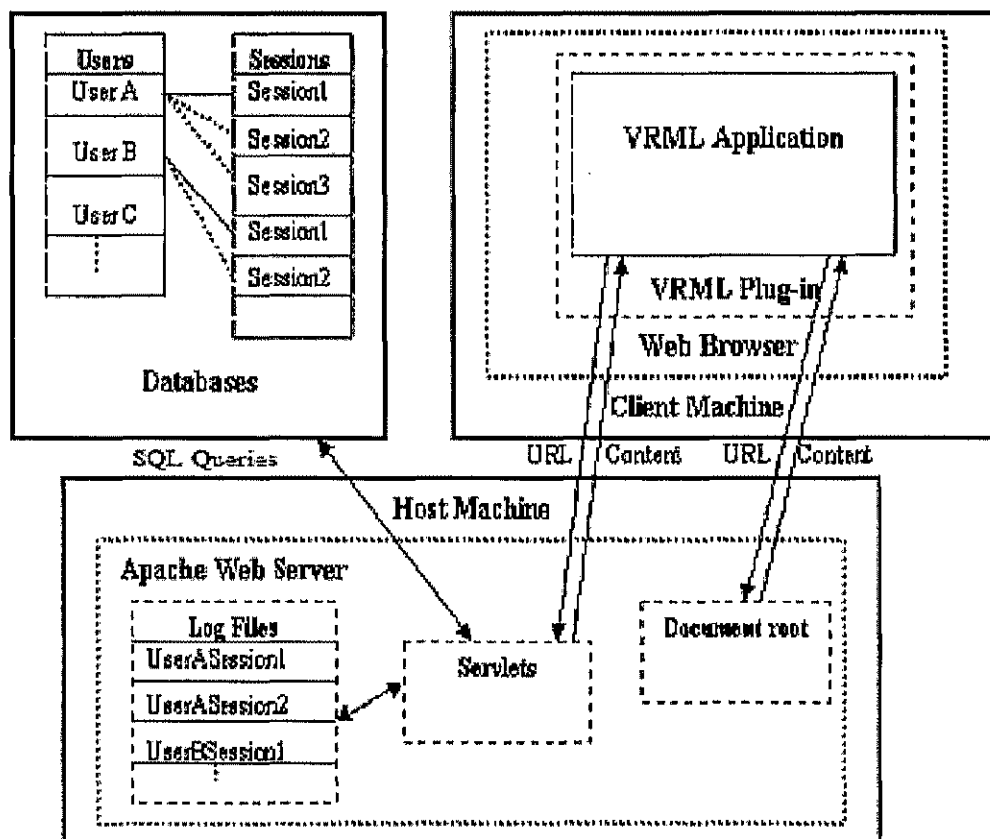


**Figure 12. Restore node scenegraph**

The above discussion illustrated the general mechanism for a session. It can be extended to any number of sessions. The application developed is a web application potentially to be used by many users all over the world. The user may have as many sessions as the number of times he/she enters the site.

In order to maintain the uniqueness of every session and to isolate the sessions from different users, a methodology has been proposed. Each user is given a secured login and a password. All the information pertaining to the user like name, address etc. is stored in the database. Microsoft access is used as the database. However, other databases can be used, as JDK provides access to other databases such as Oracle, Sybase, MySQL etc. Figure 13 shows an instance of simultaneous sessions from different users and different sessions from the same user.

Once logged in, for the first time, the user has to start a new session. The new session is given a name by the user. The session name is stored in the database against the user's name. Both the user name and the session name are passed as parameters to the servlet loading the visualization for the new session. In the present application, it is the `scene` servlet that loads a new VRML session of the visualization for the user.



**Figure 13. Depictions of User Sessions**

As the objects are added in the session, the object updates are written to a log file. The log file name is the concatenation of the user name and the session name parameters passed to the servlet, to assure the uniqueness of the session. Hence, two different users can have a same session name. For example, 'UserA' can have a session name 'Session1', and 'UserB' can have the same session name 'Session1'. The

object updates are written to log files - 'UserASession1', 'UserBSession1' respectively.

### **Creation of Unique Log File for Restoring a Session:**

As described earlier, the Record servlet creates a log file and writes the VRML session into it. Hence, maintaining the uniqueness of the log file is very important so that no information is lost. A novel mechanism has been implemented in that the Servlets communicate with one another to create a unique local log file on the server file system for every new session. Inter-servlet communication is discussed briefly below.

The Servlets that run on the same server have several ways to communicate with each other. But there are three major reasons for servlets to interact. They are

1. Direct servlet manipulation

Direct servlet manipulation can be used when one servlet wants to gain access to the other currently running servlets to perform some tasks on each. For example, one servlet can ask each servlet to periodically write its state to disk to prevent information loss in case of server crashes.

2. Servlet reuse

Servlet reuse is useful when one servlet wants to use the abilities (the public methods) of another to perform various tasks. It is this sub-concept that is used to create a unique log file for each session.

In the application, the Record servlet uses the method of the scene servlet that returns the concatenated string of user name and session name to write information into a log file with the corresponding file name. Similarly, the RestoreSession servlet has

the same method that is later accessed by the `Display` servlet to generate the VRML world by scanning the information from the same log file into which the data has been written. The servlet reuse mechanism is explained in detail in Chapter 4.

### 3. Servlet collaboration

The most common situation involves two or more servlets sharing state information. For instance, a group of servlets managing an online store could share the store's product inventory count. Session tracking is a special case of servlet collaboration.

#### **3.3.4. Dynamic VRML Object Generation**

The servlets are very simple, easy to use, and provide a very powerful mechanism for generating VRML content dynamically. For example, in our application all the buildings are scaled and sized in accordance with the GIS data provided to us by the city personnel of Portsmouth. Also, the parking lots that are associated with the office buildings are sized according to the area of the building. The problem with the parking lots is that the city building codes state that the parking lots are required to have a specific canopy cover fraction, i.e. trees, and the number of trees varies with the parking lot size. In the application, one parking lot model represents all parking lot instances requiring trees to be generated dynamically. In earlier versions of the visualization [4], the trees were generated dynamically using an ECMA script. While this has been functional, the approach resulted in many instabilities and resource problems. This problem has been overcome in the present application by using servlets to generate a VRML object containing the desired number of trees resulting in faster and more reliable operation.

## **CHAPTER 4**

### **INTERACTIVE LAND USE VRML APPLICATION (ILUVA) WITH SERVLET ASSIST**

This chapter discusses Interactive Land Use VRML Application (ILUVA) client, and servlets that demonstrate the implementation of session save and restore mechanism. Also, the implementation of saving and restoring a sample session is explained in detail.

#### **4.1. An Overview of Interactive Land Use VRML Application (ILUVA) Client**

In this section, the underlying operation of Interactive Land Use VRML Application (ILUVA) client with respect to servlets is explained. As explained in Chapter 3, Interactive Land Use VRML Application (ILUVA) is a decentralized collection of eight interconnected, concurrently operating modules [6]. They are 1) The Menu, 2) The Resource manager, 3) The Arbiter, 4) The Collision Manager, 5) The Work Zone, 6) The Simulation Manager, 7) The Fixed landscape and, 8) The Object models. The predecessor to Interactive Land Use VRML Application (ILUVA) had all these modules too [4]. But, the difference is in the server side execution of servlets, the functionality of the objects and resource manager. Object models are self-contained and encapsulate all object capabilities. In VRML, the flow of information is event driven, with the major events being listed in the Table 3. Menu events are encoded into commands that are communicated to all modules and objects. When an object is selected, an `ObjectType` event is sent that the menu hierarchy uses to make the appropriate menu appear. The

request and grant events are used to manage the shared resources. When an object is selected, it sends a request event to the main arbiter that generates a grant event to all objects that each monitors and responds accordingly when the requesting object is recognized [21].

**Table 3. Major events**

<b>Signal</b>	<b>Purpose</b>	<b>Source</b>
Command	Request of the User	Menu
ObjectType	Determines which object is selected, usually occurs when mouse clicked on object	Object
Request	Object request for menu control	Object, Resource Manager
Grant	Permits object menu control	Main Arbiter

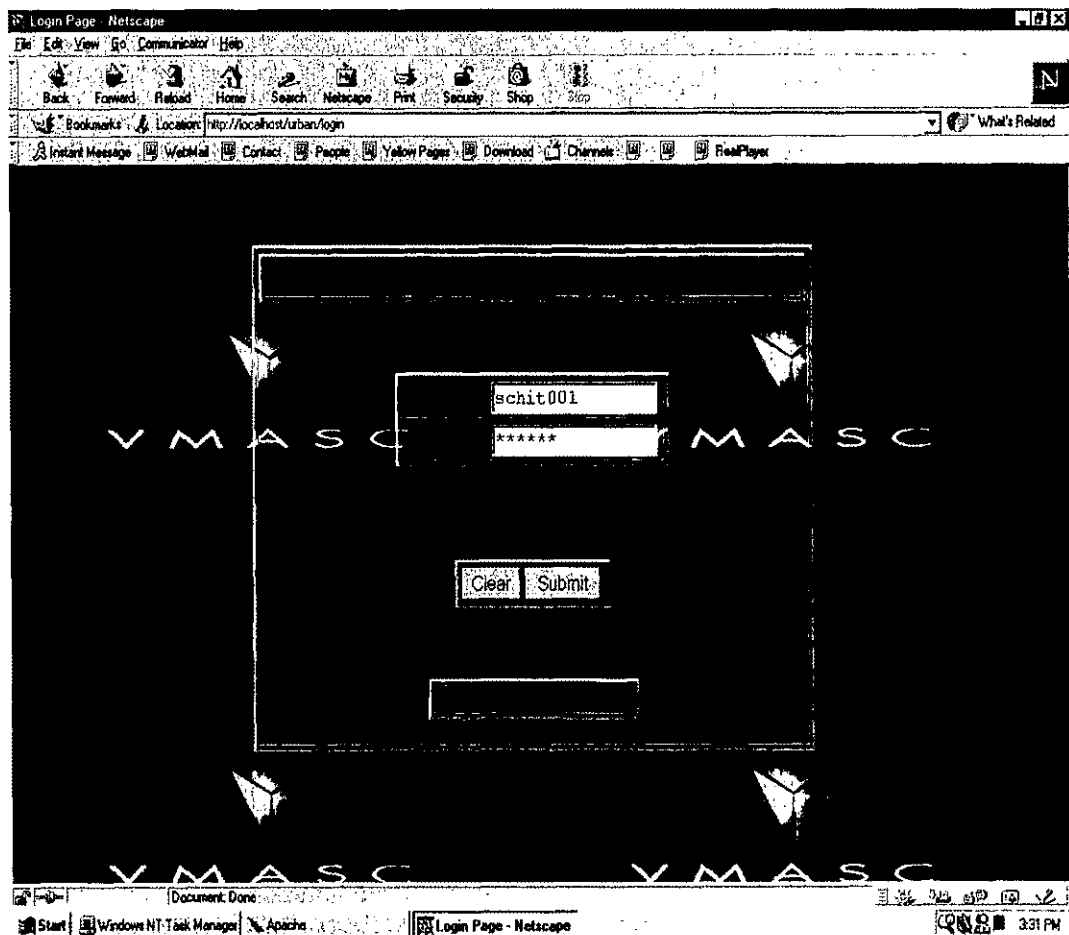
## **4.2. Description of Servlets Employed**

### **4.2.1. Login Servlet**

The visualization developed is a web application potentially to be used by many users all over the world. The user may have any number of desired sessions. Login servlet helps to maintain the uniqueness of every session and to isolate the sessions from different users by providing each user with a secured login and a password.

Login servlet is a straight-forward HTML form that prompts the user to enter the login and password as shown in Figure 14.





**Figure 14. Login Servlet**

If the user is registered, he/she is allowed to create a new session or to continue with his/her prior sessions. If the user is new, the user is directed to a registration form to gain access to the visualization.

Client-side parameter validation is performed using JavaScript. Client-side parameter validation consists of three functions: `isLogin()`, `isPass()`, and `Reset()`. `isLogin()` function checks for a blank user name. `isPass()` function checks for a blank password. Both the functions return a warning to the user if the user name or the password is blank. `Reset()` function clears the form fields: user name, and password. The primary advantage of client-side parameter validation is speed. Form fields can be checked on the fly both during the user's changes and when the user submits, and in either case the user can be immediately notified of a problem before the data is returned to the server. The code segments of `isLogin()` and `isPass()` functions are as shown in Figure 15.

```
function isLogin() {  
    var str = document.forms[0].elements[0].value;  
    // Return false if name field is blank.  
    if (str == "") {  
        alert("UserName is blank.");  
        document.forms[0].elements[0].focus();  
        return false;  
    }  
    return true;  
}  
  
function isPass() {  
    var str = document.forms[0].elements[1].value;  
    // Return false if name field is blank.  
    if (str == "" ) {  
        alert("Password field is blank");  
        document.forms[0].elements[1].focus();  
        return false;  
    }  
    return true;  
}
```

**Figure 15.** Code segments for Login servlet

### 4.2.2. Reg Servlet

Reg servlet provides the new user with a data-entry form to register and there by allowing to access the application. The user is asked to provide information such as name, e-mail, address, login, password, confirmation password, user comments. All the fields, except for user comments are mandatory in the form. Client-side parameter validation for these fields is performed using JavaScript to prevent the user from entering invalid information. The servlet consists of the following JavaScript functions: `isName()`, `isEmail()`, `isAddr()`, `isLogin()`, `isPass()`, `isComment()`, `Reset()`.

`isName ()` function does not allow the user to leave the name field blank. It makes sure the user enters valid data. It returns a warning if the user enters characters other than letters & spaces.

`isEmail ()` function checks for a valid email address. It checks for the characters '@' and '.', as these are the necessary characters in a valid email address. It also checks if the email field is left blank.

`isAddr ()` checks for a blank address field. It alerts the user to enter address, if blank.

`isLogin()` function checks for a blank login, and alerts the user to enter login name, if blank.

`isPass()` function checks for a blank password as well as for a blank re-confirmation password, and alerts the user, if either of them is blank. This function also checks if both the passwords (password and confirmation password) match.

isComment () function alerts the user if the comment field is blank. Comment field is not mandatory. Hence this function asks the user for confirmation if the user really wants to leave the field blank.

Reset () function clears all the form data entered by the user. The page layout of Reg servlet is as shown in Figure 16.

http://localhost/urban/reg

Registration Form

### REGISTRATION FORM

Enter your name	Enter your e-mail address
<input type="text"/>	<input type="text"/>
Enter your Address	Create your LoginID
<input type="text"/>	<input type="text"/>
Enter Password	Confirm Password
<input type="text"/>	<input type="text"/>

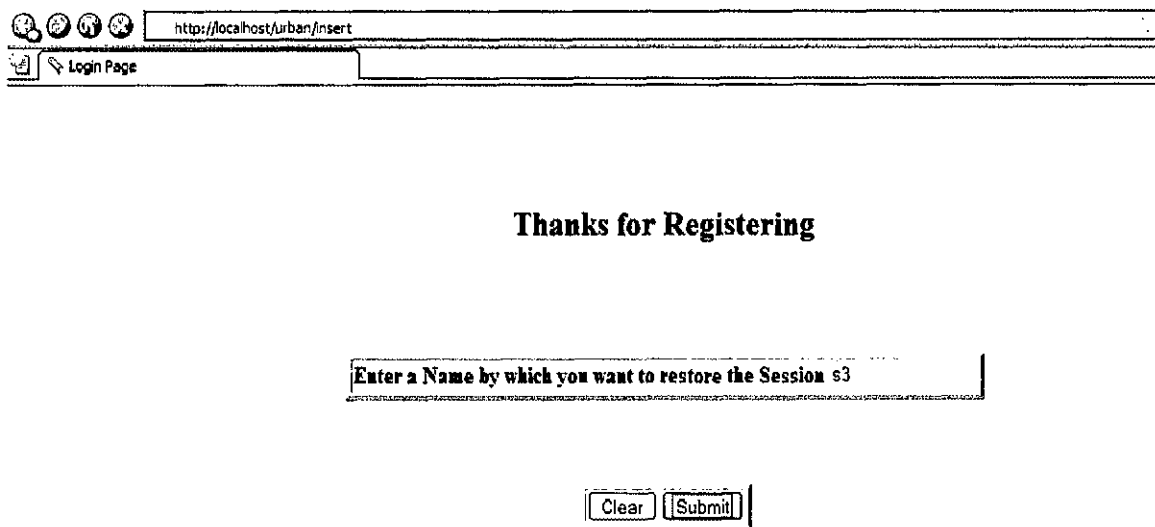
Purpose for Registering

Submit Clear All Info

Figure 16. Reg servlet

### 4.2.3. Insert Servlet

The purpose of `Insert` servlet is to insert the new user registration information into the database. Before inserting the user information into the database, the servlet queries the database to see if login already exists. It throws a 'Sorry' page to the user if login already exists, other wise, the registration information is stored in the database. Once the user is registered, the user is provided with a form to enter a session name to restore the new session as depicted in Figure 17.



The screenshot shows a web browser window with the address bar containing `http://localhost/urban/insert`. The browser title is "Login Page". The main content area displays the text "Thanks for Registering" in a bold font. Below this text is a text input field with the placeholder text "Enter a Name by which you want to restore the Session s3". At the bottom of the form are two buttons: "Clear" and "Submit".

Figure 17. Insert servlet

As explained in Chapter 3, Java Database Connectivity (JDBC) technology is used to access the database. The database used in the application is MS-Access. The database is very simple consisting of two tables: `Users`, and `Sessions`. All the user information such as user name, password, address, e-mail, etc. is stored in the `Users` table. The session names and the login names of every user are stored in the `Sessions` table. To access any individual database system i.e. to establish a connection with any database, a specific JDBC driver has to be used. Drivers exist for all popular database systems, and a few of them are available for free. SUN provides a free JDBC-ODBC bridge driver, used in this application with the JDK versions to access the standard ODBC sources like Microsoft access. The first step in establishing a connection requires loading the specific driver class into the application's JVM. An easy way to load the driver is to use `Class.forName()` method. For instance, 'Insert' servlet loads the SUN JDBC-ODBC bridge driver into memory through the statement `Class.forName("sun.jdbc.odbc.JdbcOdbcDriver")`. Once loaded, it registers itself with the `java.sql.DriverManager` class as an available database driver. The next step is to open a connection to a given database by making a request through `DriverManager.getConnection()` method. The method returns a class that implements the `java.sql.Connection` interface. For instance, 'Insert' servlet makes a connection with the following call.

```
Connection=DriverManager.getConnection("jdbc:odbc:test1");
```

Once the connection has been established, to really use the database, Insert servlet has to have some way to execute queries and retrieve the information. The `java.sql.Statement` class executes queries. The `Statement` objects are never instantiated directly; instead, a program calls the `createStatement()` method of the `Connection` object to obtain an instance of the `Statement` object.

To execute a query that returns data, Insert servlet uses `executeQuery()` method of the `Statement` object. This method executes the statement and returns a `java.sql.ResultSet` that holds the retrieved data as in Figure 18.

```
Statement stmt=connection.createStatement();
rs=stmt.executeQuery("SELECT COUNT(*) AS row_count
                      FROM Users WHERE
                      login='"+req.getParameter("puser")+"'");
```

**Figure 18. executeQuery method in Insert servlet**

The `ResultSet` object could be thought as a representation of the query result returned one row at a time. To move from row to row the `next()` method of `ResultSet` is used.



The `PreparedStatement` object in the application has been used in much same fashion as the `Statement` object. The important difference between the two is that the SQL statements in a `PreparedStatement` object are precompiled by the database for faster execution. Once a `PreparedStatement` has been compiled, predefined parameters subsequently are customized. `PreparedStatement` objects are useful in applications where the same general SQL statement is re-run several times.

To create a `PreparedStatement` object, `PreparedStatement(String)` method of `Connection` object is used. It is used in conjunction with the servlets in this application to escape the SQL control characters (such as “, or ‘) when inserting the user submitted data into the database. For instance, `Insert` uses a `PreparedStatement` to escape the single quotes in the user information.

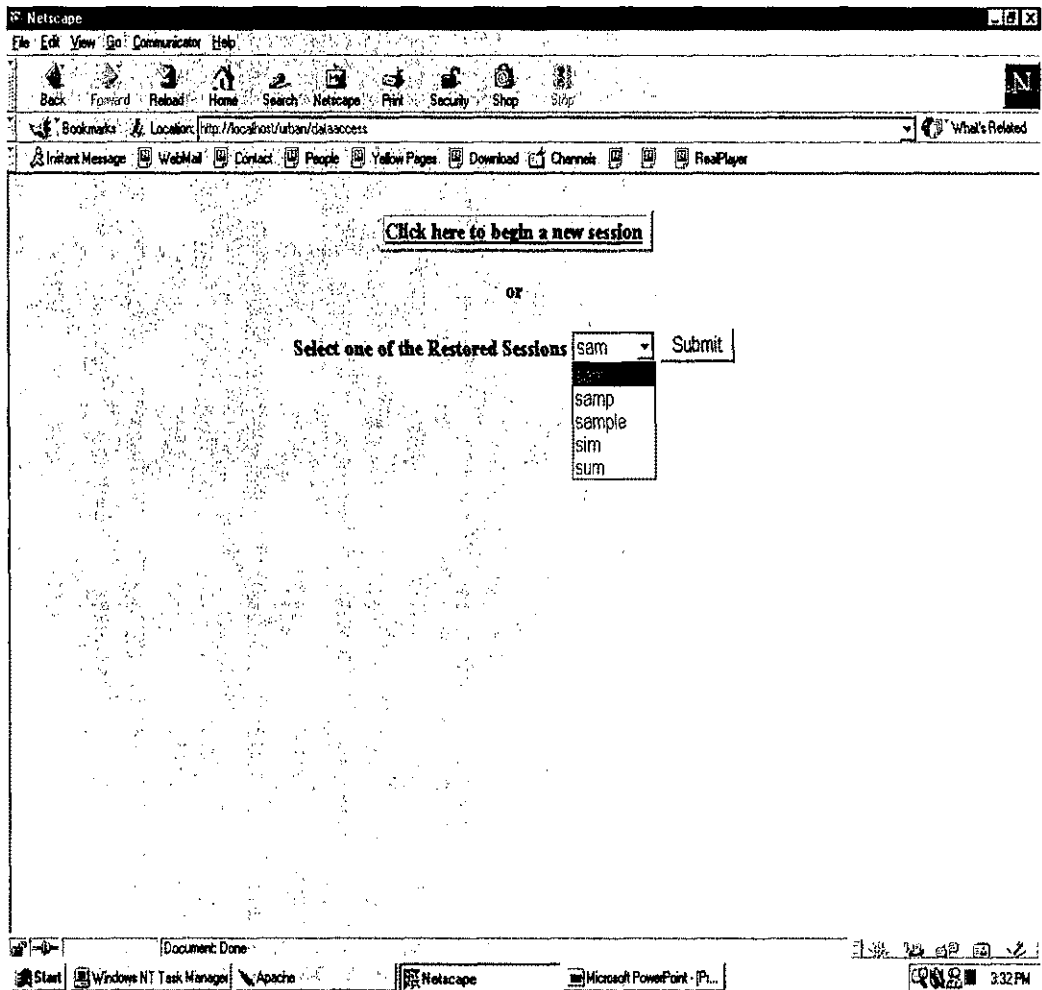
#### **4.2.4. Dataaccess Servlet**

`Dataaccess` servlet checks the authenticity of the user logged in. The values of the login and password are passed to this servlet as request parameters in a query string from the `login` servlet. Query string is text data appended after the “?” in the URL. The authenticity of the user is checked by querying the `Users` table in the database with the query shown in Figure 19.

```
ResultSet rs=stmt.executeQuery("SELECT COUNT(*) AS row_count  
                                FROM Users WHERE  
                                login='"+req.getParameter("P_USER")+"' AND  
                                password1='"+req.getParameter("P_PASS")+"'  
                                ");
```

**Figure 19. Query to check user authenticity**

If the login exists in the database, the servlet provides the user with the choice of resuming any of his/her prior sessions or to begin a new session as illustrated in Figure 20.



**Figure 20. Dataaccess servlet page layout**

If the user is invalid, the user is re-directed to the login page with an error message. All the prior sessions of the user are retrieved by querying the `Sessions` table with the following statement as illustrated in Figure 21.

```
ResultSet sess=inst.executeQuery("SELECT session FROM sessions  
WHERE login='"+req.getParameter("P_USER")+"'");
```

**Figure 21. Query to retrieve session names**

#### **4.2.5. SessionName Servlet**

`SessionName` servlet provides the user with a form to give a name to store a new session. The user is allowed to use this form only if the user is a registered user, or if the user is interested in starting a new session. Client-side parameter validation is performed to prevent the user from entering a blank or erroneous value for the session name as shown in Figure 22. The JavaScript function `isSession()` prevents the user from form submission without a session name. `Reset()` function clears the form fields.

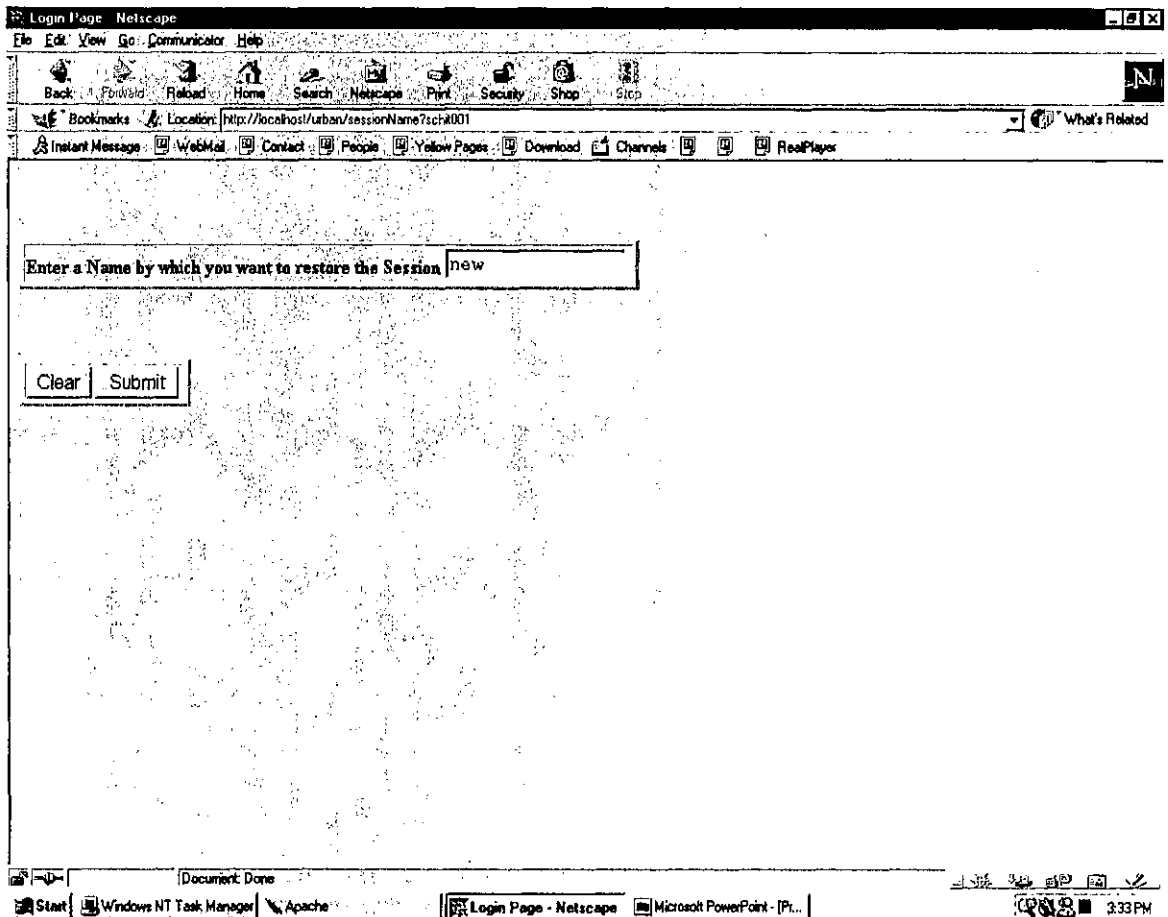
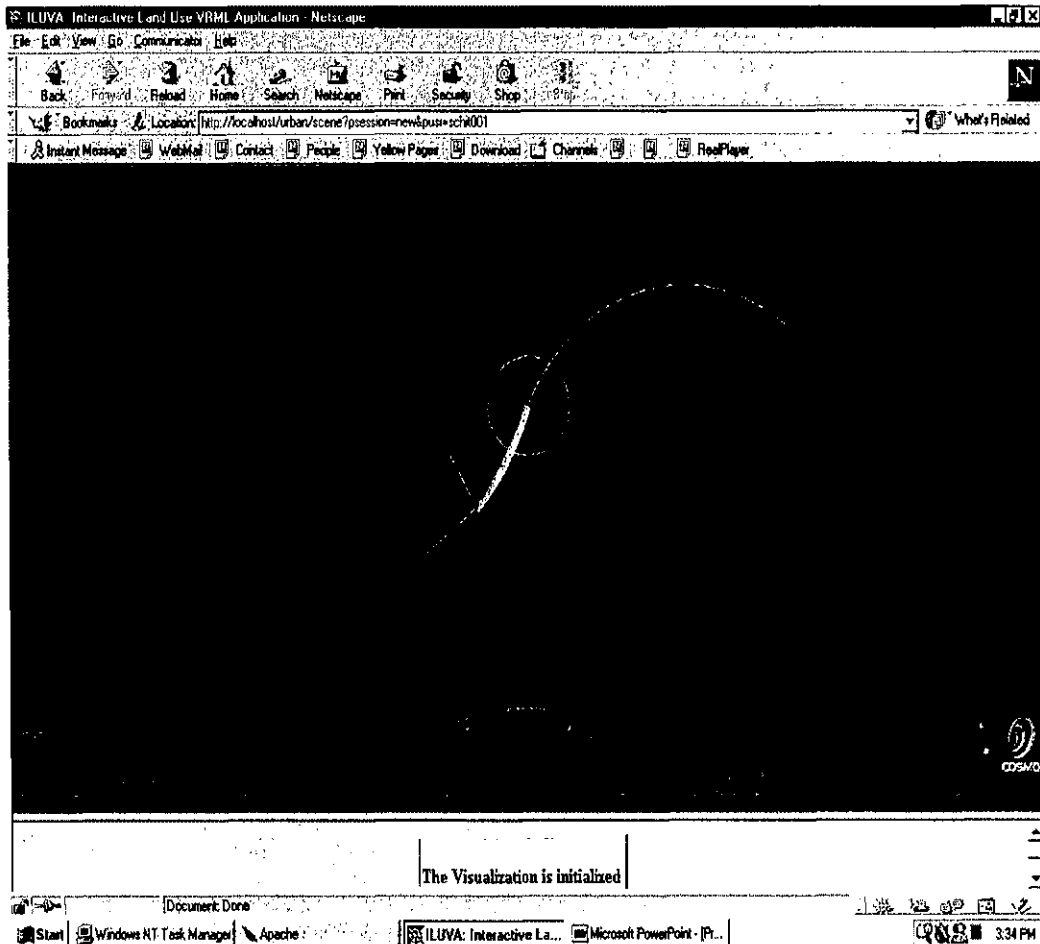


Figure 22. sessionName servlet

#### 4.2.6. Scene Servlet

Scene servlet loads the actual VRML visualization for a new session. The user navigates through Scene servlet only for a new session. This servlet basically outputs HTML content consisting of two frames. The first frame loads the VRML application. The second frame is a place holder for servlet messages. For instance, when the

visualization is loaded for the first time, a servlet message that the visualization is initialized is displayed in frame two as depicted in Figure 23.



**Figure 23. Scene servlet**

Also, when objects are added in a session, the servlet messages consisting of the object updates are displayed in this window. But, before making the visualization

available to the user, the scene servlet checks if user name and session name exist in the database by querying the sessions table with the statement as illustrated in Figure 24.

```
con= DriverManager.getConnection("jdbc:odbc:test1");
Statement stmt=con.createStatement();
ResultSet rs=stmt.executeQuery("SELECT COUNT(*) AS row_count FROM sessions
                                WHERE login='"+usrName+"' AND
                                session='"+sess+"'");
```

**Figure 24. Query for session name existence**

The scene servlet receives the user name and session name as request parameters through a query string. If the session name doesn't exist in the database, it is inserted into the sessions table for later retrieval as shown in Figure 25.

```
PreparedStatement inst = con.prepareStatement("INSERT INTO sessions VALUES (?,?)");
inst.setString(1,usrName);
inst.setString(2, sess);
inst.executeUpdate();
```

**Figure 25. Query to insert session values**

The `scene` servlet also has a public method called `userSessionName()` that stores the concatenated string of username and session name. This method is later re-used by `Record` servlet to create a log file with the same name (concatenation of user name and session name) into which the user updates are written.

#### **4.2.7. Reset Servlet**

The purpose of `Reset` servlet is to just let the user know that the visualization is initialized and can begin working with the visualization tool. It is a simple servlet outputting HTML content with a message “The Visualization is initialized”.

#### **4.2.8. Record Servlet**

`Record` servlet is one of the important servlets developed for restoring sessions. `Record` servlet records the values of the object parameters into a log file. The object parameters are written into a log file under a directory called ‘`recording`’ under the server’s root. To maintain the uniqueness of every session, the naming convention of the log file is decided to be a string concatenation of user name and session name. As user name is unique for the application, and as the same user is not allowed to repeat a same session name, the log file uniqueness is maintained as per the file naming convention. The `scene` servlet has a public method called `userSessionName()` that returns the concatenated string of user name and session name. In order to make sure the `Record` servlet writes into the correct file, the `Record` servlet uses the `userSessionName()` method of the `scene` servlet. But before the `Record` servlet can access the methods of



Scene servlet, it has to obtain a proper instance of Scene servlet when the servlet is not yet loaded into the server. Unfortunately, the servlet API (the servlet packages) lacks methods to control the lifecycle of itself, or other servlets. Also, there are many potential security risks.

One back door solution to this problem is to load the usee (Scene) servlet forcibly to handle the requests through an HTTP connection to the server in which the user (Record) servlet is running [18]. Figure 26 shows how a servlet can be loaded forcibly for reuse. Figure 27 illustrates how 'Record' servlet accesses the public method of scene servlet.

```
class accessServlet
{
public static Servlet getServlet(String name,
                                ServletRequest req, ServletContext context)
{
try(

    Servlet servlet=context.getServlet(name);
    if(servlet!=null)return servlet;

// Opens a new socket for the servlet for connection

    Socket socket = new
    Socket(req.getServerName(),req.getServerPort());

// Opens a new stream

    PrintWriter outs=new
    PrintWriter(socket.getOutputStream(),true);

// Forcibly loads the servlet

    outs.println("GET /servlet/" +name + "HTTP/1.0");
    socket.getInputStream().read();
    outs.close();
    return context.getServlet(name);
    }
}
}
```

**Figure 26. Class for loading a servlet not currently loaded for reuse**

```

// Servlet reuse by Record servlet

String fileName = ""; // File name initialized to NULL
scene sceneInst = null; // Servlet that need to be reused
                        // is initialized and instanced

// For the first time the servlet is null, so load instance
// of the servlet

if (sceneInst == null )
{
    sceneInst = (scene)accessServlet.getServlet( "scene",
        req, getServletContext());
if (sceneInst != null )
{
// Accesses the method of scene servlet for filename

    fileName = sceneInst.userSessionName();
}
}

// Creating a new log file with a file name of concatenated
// string of user name and session name

    RandomAccessFile logfile = new RandomAccessFile (
        fileName+".dat","rw" );

// Code follows to write all the VRML information into the
// log file to be stored in the server root

```

**Figure 27. Servlet reuse**

#### **4.2.9. RestoreSession Servlet**

RestoreSession servlet is same as the scene servlet, but the difference is that it loads the visualization for a restored session. This also outputs HTML content in two frames, with the source of the first frame being the restored visualization, and the source

of the second frame being the servlet messages. The user navigates through this servlet only when working on prior sessions.

#### **4.2.10. Display Servlet**

The essence of this thesis is restoring prior sessions, and `Display` is the servlet that accomplishes this mechanism. The function of this servlet is to scan the log file that contains the VRML information and retrieve the most recent set of updates for the objects. Using this information, the content of a prior session is represented as a `PROTO` node called `Restore` that encapsulates all the restored objects. All the `EXTERNPROTO` nodes necessary for the objects to be restored are loaded in the `Restore` node. The object nodes encapsulated in the `Restore` node are so designed that by supplying their configuration, they are placed so that they appear identically to the prior session. The `Restore` node is included in the static `Landscape` of the visualization as an `EXTERNPROTO` node so that all the prior session objects are recovered. In the application, it is the `RestoreSession` servlet that incorporates this restored landscape thus enabling to load the restored visualization.

But, the noticeable drawback is that the `Restore` node produced by this servlet consists of non-editable objects. In other words, if the user wants to continue a prior session, he/she can only add new objects but cannot manipulate restored objects in any way.

#### **4.2.11. Restore Servlet**

The function of this servlet is same as the `Display` servlet but it overcomes the drawback of the `Display` servlet. It inserts fully editable objects into the landscape. It means the user can easily manipulate the objects that were previously restored. The development of this servlet was not initiated when the thesis was concluded.

#### **4.3. Implementation of Sample Session Restoration**

Consider for example, a new user accesses the application. The `login` servlet is the entry page for the application. As described earlier, the `login` servlet provides the user with the form to enter the login and the password. As the user is new, the user cannot access the application until registered. The login page has a web link for new user registration. This web link links to `Reg` servlet, which provides the new user with a registration form to fill in information such as the name, email, address, login, password, etc.

Consider for example, `schit001` is the login entered by the user. Before the registration information can be saved in the database, the `Insert` servlet makes sure the user doesn't already exist. Once the user is registered, the user is prompted to enter a session name to save the session. The `sessionName` servlet provides the user with a form to input the session name.

Consider for instance, `new` is the session name given by the user. Before, the user is directed to the `scene` servlet that loads the visualization for a new session, the `scene` servlet makes sure the same session name doesn't exist for the same user by querying the

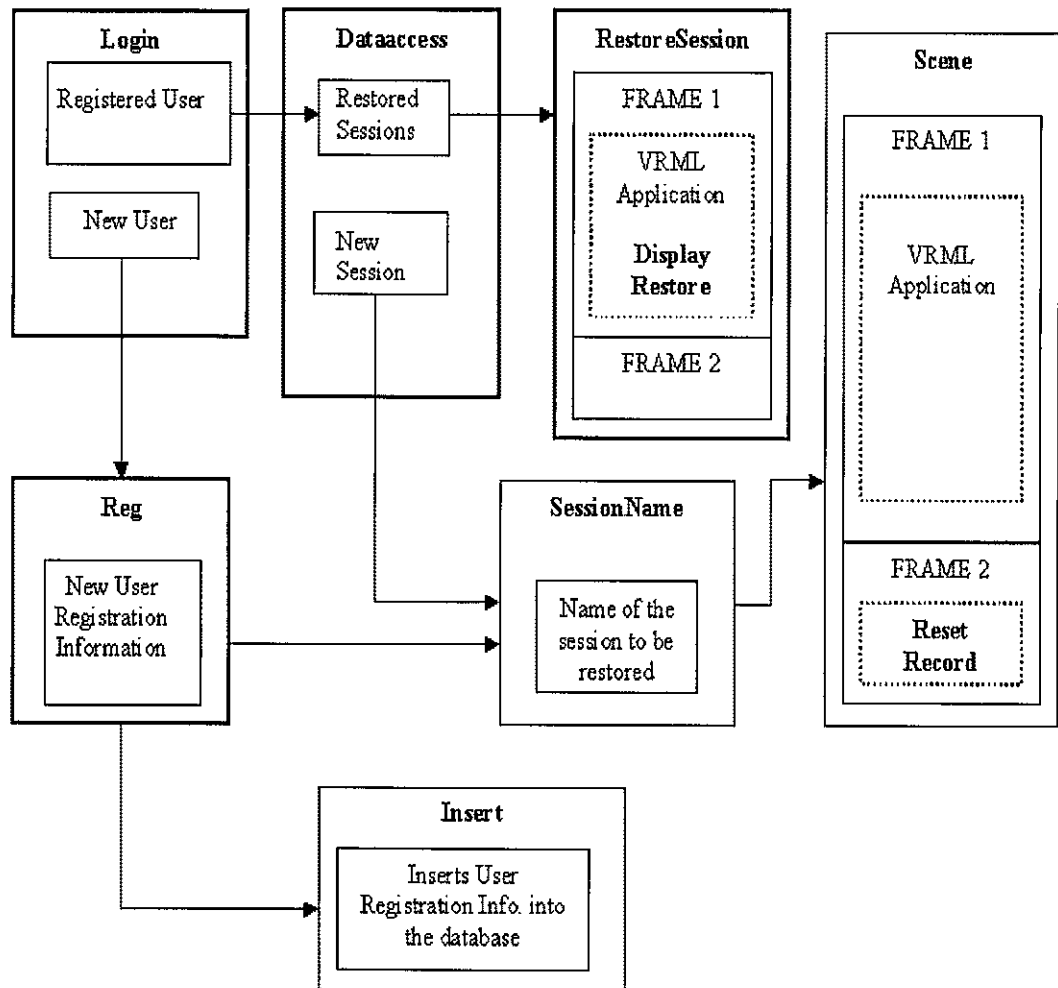
database. The `scene` servlet receives the user name and the session name: `schit001` and `new` respectively as request parameters through a query string. Query string is the text data following the "?" in a URL. The passing of these parameters is necessary for maintaining the uniqueness of the sessions. The `scene` servlet has a `public` method that stores the concatenated string of username and session name. This method is later used by the `Record` servlet to create a unique log file as depicted in Figure 27. When objects are added in the session, the object updates are passed to a log file. As explained earlier, `Record` servlet creates a log file and writes the VRML information into it. The `Record` servlet accomplishes the task of writing the updates of the objects into a log file under the directory called 'recording' under the server's root with a file name being the concatenation string of the user name and the session name i.e. 'schit001new.dat' in this case, and '.dat' is the file extension assumed. The format of the log file is rather straightforward and easy to parse. Table 2 illustrates the standard log file formats. The `id` field is the index for the object and it is unique which can be later be used to associate subsequent updates to that object. The `type` field is also unique which determines what type of object is selected for update.

Consider for example, the user added a ring road and wanted to save the session. The sample format of the log file that 'Record' servlet writes into is as follows:

**1:42:(reform):-:915.25525:585.8434234**

'1' is the index of the object, '42' is type field for the ring road, and the remaining values are the coordinates where the ring road is added. The above discussion explains the interaction among servlets from a new user perspective.

Consider now that the same user `schit001` wants to continue the prior session `new`. `Dataaccess` servlet checks the validity of the user name (`schit001`) by querying the database. Once the user is proved valid, the servlet provides the user with the choice of resuming any of his/her prior sessions or to begin a new session. The servlet queries the `sessions` table to retrieve the session name `new` for the user `schit001`. As explained earlier, `Display` servlet scans and parses the log file `'schit001new.dat'` containing the information that a ring road has been added. This information is represented as a `PROTO` node called `Restore` that encapsulates all the restored objects. All the `EXTERNPROTO` nodes necessary for the objects to be restored are loaded in the `Restore` node. The object nodes encapsulated in the `Restore` node are so designed that by supplying their configuration, they are placed so that they appear identically to the prior session. The `Restore` node is included in the static `Landscape` of the visualization so that all the prior session objects are recovered. `RestoreSession` servlet sources the restored landscape visualization. The broad interaction among the servlets is depicted in Figure 28.



**Figure 28. Broad Picture of the interaction among servlets**



## **CHAPTER 5**

### **FUTURE WORK**

The architecture of Interactive Land Use VRML Application (ILUVA) enables restoring work from a prior session. Save and restore capabilities are important for maintaining a permanent record of a session and allowing the user to resume the session later. Save and restore capabilities for the application has been achieved by employing servlets.

#### **5.1. Limitation in Restore Capability**

A limitation in the restore capability is that the restored objects cannot be edited. Interfacing with to the servlet required that each geometry or configuration update be communicated to the server through the VRML `loadURL` [12] method. The session feature required that all objects in the application have an instantiation mode that faithfully recreates the object appearance from a prior session.

The architecture of ILUVA allows for enhancement in several directions for the future work. First, the application architecture can be enhanced to allow restored objects to be fully editable. As explained earlier, the `Record` servlet is responsible for writing all the updates of the objects to be restored to a file. One of the updates written to a file is a unique index generated by the Resource Manager, which is used to associate subsequent updates to that object. One conspicuous problem in the present application is that when the user resumes an existing session and adds an object the unique index starts

again from '1' instead of continuing from the last index. This is one of the reasons for the restored session to be editable. Also, the object architecture has to be modified to make the restored objects editable. The servlets provide unlimited capabilities in terms of generating custom VRML modules like retrieving data from databases by user request, or synthetically. The framework followed in the thesis allows any such geometry to be an integral part of the application.

## **5.2. Session Tracking**

In the current application, the session tracking mechanism is implemented using servlet reuse methodology. This is not an efficient way of saving and restoring sessions. The servlet API provides several methods and classes specially designed to handle session tracking on behalf of servlets that can be used. Specifically, `javax.servlet.http.HttpSession` object can be utilized. There are many online applications such as shopping carts that employ this mechanism.

## **5.3. Multi User Virtual Environments (MUVE's)**

Virtual Environments (VEs) can be defined as interactive computer simulations that immerse users in an alternate, yet believable reality. MUVE applications incorporate computer graphics, sound simulation, and networks to simulate the experience of real-time interaction between multiple users in a shared three-dimensional virtual world [24]. Each user runs an interactive interface program on a client computer connected to a wide-area network. The interface program simulates the experience of immersion in a virtual environment by rendering images, sounds, etc. of the environment as perceived from the

user's simulated viewpoint. Each user is represented in the shared virtual environment by an entity rendered on every other user's computer. Matching user actions to entity in the shared virtual environment supports multi-user interaction.

The existed ILUVA was capable of saving and restoring a session but, what if many users want to work on the same scene at the same time and what if a user desires to export objects that from another user. One of the solutions to this problem could be to extend the capability of the ILUVA to a multi-user level. So, the first step in that direction could be a chat application that could embed into client frame of the Interactive Land Use VRML Application (ILUVA). This could provide a basic mode of communication in a multi-user environment. So using the classes of the `java.net` [17] package, a chat client and a chat server can be implemented. The server has to use a multi-threaded architecture with each thread for a user. An applet can be embedded into the application behaving as a chat client. The applet is posted with a servlet that sends all the user information to the chat server that keeps track of all the users. To have the multi-user capability, the applet should have the capability of exporting and importing of sessions. There should be a two-way communication between the users who want to export and import a session. The GUI of the applet can have two buttons with names 'Export' and 'Import'. To export a session one has to select the user from the user's list and click the export button. To import a session the other user has to now click on the import button. When an export request is sent to the chat server, all the objects in the exporting user's database are written to the imported user's database. The way the import/export mechanism works is that when the user clicks on the import button, the

VRML scene gets reloaded. Actually, it does not get reloaded; the `Restore` servlet is called into the frame where the VRML scene is present. A method in `java.lang` [17] package can open a URL in a new window or a particular frame.

## **CHAPTER 6**

### **SUMMARY AND CONCLUSIONS**

The ILUVA builds on prior work [4] by adding server based processing to a client based application. Specifically, save and restore functionality, necessary to resume a session later, was implemented to produce this capability. Servlets are capable of performing a wide variety of functions. But, more specifically, the servlets have been employed in the project for four important capabilities. First, servlet calls make possible the logging of information generated by a VRML application. Second, servlets can restore the work from a prior session. Third, servlets can be used to generate models specified parametrically and, fourth establishing a connection to the database to store the user information for session maintenance. The database used in the application is MS-Access. The Interactive Land Use VRML Application (ILUVA) provides a user with the ability to take a virtual area and populate it with buildings, roadways, etc.

In stand-alone web applications, files cannot be read and written because of the browser constraints [23] but, the server can be easily configured to read and write files on the server file system. Hence, a VRML session has been logged on to the server so that its information can later be parsed. The mechanism for logging a session required interaction between servlet code and the ECMAScript in the VRML application. A simple method for passing information is for the VRML application to load the universal resource locator (URL) for the servlet. The operands that are appended to the servlet URL constitute the session information to be logged. The `loadURL` Browser method is

used to implement the above mechanism in the VRML application [12]. A servlet named `Record` has been developed in this capacity. The purpose of this servlet is to store all the information passed from a VRML session in a log file. All the updates are recorded chronologically, with the most recent modifications representing the final appearance.

To accomplish the objective of restoring prior sessions, the log file is read and the last value recorded for a field is used for the restoration. A servlet was developed that scans the log file containing the VRML information and retrieves the most recent set of updates for the objects. It creates a VRML object encapsulating the restored objects. The object architecture is so designed that by supplying the object configuration, it is placed so that it appears identically to the prior session. This is the servlet that produces this node consisting of non-editable objects.

At the time when this thesis was concluded, research had been on-going to enhance the application architecture to allow the restored objects to be fully editable. Also, the application was being extended to make it a multi-user virtual environment. As Servlets provide plethora of capabilities, they can be fully utilized to enhance the application.

## REFERENCES

1. The Web3D Consortium. 1998. The Virtual Reality Modeling Language. Available as <http://www.web3d.org/x3d/vrml/index.html> [Accessed June 15, 2004]
2. Nicholas, F. P, "Stylesheet transformations for interactive Visualization: towards a Web3D chemistry curricula", in *Proceedings of the Eighth International Conference on 3D Web Technology*, Saint Malo, France, March 2003. Pages #1-2.
3. Dimitris, N. C., Heinrich, S., 1995. *Virtual Reality: Practical Applications in Business and Industry*, New Jersey: Prentice Hall Publishing.
4. Rajesh Vennam. *Dynamic and Interactive 3D Visualization over the World Wide Web*, Master's Thesis, Department of Electrical and Computer Engineering, Old Dominion University, Norfolk, VA, 1999.
5. Parvati, D., David, E., Rory, M., Duncan, H., and Meenakshi, D., "The Collaborative Curriculum Web for Medicine: A Virtual representation of Medical School Resources", in *Proceedings of the 1998 Winter Simulation Conference*, Washington D.C., December 1998. Pages #1-3.
6. Lee A. Belfore II and Suresh Chitithoti, "Interactive Land Use VRML Application (ILUVA) with servlet assist", in *Proceedings of the 2000 Winter Simulation Conference*, Orlando, Florida, December 2000. Pages #1-5.
7. Andrew, S. T., 2003. *Computer Networks*, New Jersey: Prentice Hall Publishing.
8. The Apache Project Software Foundation. 2000. The Apache Server Project. Available as <http://www.apache.org/httpd.html>. [Accessed June 15, 2004]
9. HTTP - Hypertext Transfer Protocol. Available as <http://www.w3.org/Protocols>. [Accessed June 15, 2004]
10. Howard, R., 1992. *Virtual reality: The revolutionary technology of computer-generated artificial worlds- and how it*, New York: Touch Stone Publishing.
11. Michael Pidd, "An introduction to computer simulation", in *Proceedings of the 1994 Winter Simulation Conference*, Orlando, Florida, December, 1994. Page #1.
12. Rick, C., Gavin, B., 1997. *The Annotated VRML 2.0 Reference Manual*, Boston: Addison-Wesley Publishing.

13. Jed, H., L., Josie, W., 1996. *The VRML 2.0 Handbook*, Boston: Addison-Wesley Publishing.
14. Rodger, L., Kouichi, M., and Ken, M. 1996. *Java™ for 3D and VRML Worlds*, Indianapolis: New Riders Publishing
15. The Web 3D Consortium, 2004. VRML97 Functional specification and VRML97 External Authoring Interface (EAI) International Standard ISO/IEC 14772-1:1997 and ISO/IEC 14772-2:2002. Available as [http://www.web3d.org/x3d/specifications/vrml/ISO\\_IEC\\_14772-All/index.html](http://www.web3d.org/x3d/specifications/vrml/ISO_IEC_14772-All/index.html) [Accessed June 17, 2004]
16. Sun Microsystems, Inc. 2000. The Java™ Servlet API. Available as <http://java.sun.com/products/servlet>. [Accessed June 17, 2004]
17. Patrick, N., Herbert, S., 1999. *Java™ 2: The complete reference, Third edition*, New York: Tata McGraw Hill Companies
18. The Java Apache Project. 2000. The Apache JServ Project. Available as <http://java.apache.org/jserv/index.html>. [Accessed June 17, 2004]
19. Jason, H., and William, C., 1998. *Java™ Servlet Programming, California*: O'Reilly & Associates, Inc.
20. Simon, R., Philip, H., and Michael, E., 1998. *Complete Java 2 Certification Study Guide, California*: SYBEX, Inc.
21. Lee A. Belfore II, "An architecture for large VRML worlds", in *Proceedings of the 2000 Winter Simulation Conference*, Phoenix, Arizona, April 2000. Page #15.
22. Lee A. Belfore II and Rajesh Vennam, "A VRML Application for marketing Urban Properties", in *Proceedings of the 1999 Winter Simulation Conference*, San Francisco, California, January 1999. Page #15.
23. Lincoln, D., S., 1997. *How to Set-Up and Maintain a Web Site, 2<sup>nd</sup> Edition*, Boston: Addison-Wesley Publishing.
24. Thomas A. Funkhouser. RING: A Client-Server System for Multi-User Virtual Environments, in *Proceedings of the 1995 Symposium on Interactive 3D Graphics*. Monterey, California, April 1995. Page #1.