DESIGN AND DEVELOPMENT OF ONTOLOGY-BASED 3D VIRTUAL GREENHOUSES

By

RAJA APPUSWAMY

A THESIS PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

UNIVERSITY OF FLORIDA

2007

To my mom , dad and friends

ACKNOWLEDGMENTS

I thank Dr.Howard Beck, my advisor and chair, for all the support he has offered me during the course of my studies as a graduate student. His style and approach to problems profoundly influenced me and helped me hone my technical knowledge.

This research would have been impossible without the support offered by Dr.Ray Bucklin. I would like to thank him profusely for guiding me and providing me with the right resources at the right time.

I am indebted to Professor James Oliverio as my experience working with him on ISAS greatly influenced the design of the 3D virtual greenhouse.

I would like to thank my parents for I would have never made it this far without their blessings and last but not the least, I would like to thank all my friends whose constant encouragement and support made everything possible.

TABLE OF CONTENTS

page

ACKNOWLEDGMENTS 4

LIST OF TABLES 7

LIST OF FIGURES 8

ABSTRACT 9

CHAPTER

1 INTRODUCTION 11

2 LITERATURE REVIEW 13

2.1 UMD, UW- Madison Greenhouse Virtual Tour 13
2.2 University Of Florida Virtual Field Day 14
2.3 The Greenhouse Simulator by University of Vermont 14
2.4 Virtual Grower 16
2.5 Visualization of Environment by VRML 17
2.6 Evaluation 18

3 TOOLS AND TECHNIQUES 20

3.1 Java Swing 20
3.2 VRML 21
3.4 Xj3D 24
3.5 Lyra Ontology Management System 25
3.6 Addressing the Limitations 30

4 ONTOLOGY BASED 3D VIRTUAL GREENHOUSE 33

4.1 Domain Ontologies 33
4.2 Database Design for 3D Objects 35
4.3 User Interface 36
4.3.1 Methodology 36
4.4 Mentor Mode 37
4.4.1 Methodology 38
4.5 Inspect Mode 38
4.5.1 Methodology 40
4.6 Searching 42
4.6.1 Methodology 42
4.7 Energy Estimation 44
4.8 Comparison Mode 46
4.9 Simulation mode 47

4.10 Methodology 47

5 CONCLUSIONS AND FUTURE WORK 49

5.1 Ontology based search 49
5.2 Natural language Support 50
5.3 Geo referencing the 3D model 51
5.4 Exporting the 3D model to other formats 52

LIST OF REFERENCES 53

BIOGRAPHICAL SKETCH 55

LIST OF TABLES

Table page

5-1 Uvalues 47

LIST OF FIGURES

Figure page

2-1 Web page showing the 360 degree Quicktime virtual tour....15

2-2 Greenhouse simulation....16

2-3 Images of Virtual Grower....17

2-4 VRML Greenhouse developed at Chiba University....19

3-1 Domain ontology for Computers....30

4-1 Greenhouse domain ontology......33

4-2 Tomato Plant ontology....35

4-3 Swing based UI for setting the properties of the greenhouse.....37

4-4 The 3D greenhouse rendered in Xj3D.....39

4-5 Textual Information Pane containing comments, user defined names and viewpoints.....40

4-6 Image of an electric heater being added to the 3D object.....41

4-7 Aggregating shape nodes under a common user defined name.....43

4-8 Search demonstration.....45

4-9 Simulation with the heater showing the internal temperature in the greenhouse.....48

6-1 Images showing geo-referenced Rinker Hall.....52

Abstract of Thesis Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Master Of Science

DESIGN AND DEVELOPMENT OF ONTOLOGY BASED 3D VIRTUAL GREENHOUSES

By

Raja Appuswamy

December 2007

Chair: Dr. Howard Beck
Major: Agricultural and Biological Engineering

With the growing popularity and widespread appreciation for the design and construction of greenhouses, several attempts have been made over the past few years in developing web based virtual greenhouses. The approach taken towards the design of such greenhouse can be classified into two types. The first type focuses on delivering knowledge to the users. Visualization is either not a part of such a system or is minimal. The second type is centered on developing a visualization of the greenhouse so that the users can get a feel of the real world peer. Such a visualization system stands alone as a separate entity and is not linked with any meta information.

This research focuses on developing a 3D virtual reality based virtual greenhouse. The 3D model is coupled with information sources such as text based descriptions and relevant images by using a single data store (Lyra OMS) to store both the 3D model and meta information. Using Lyra made it possible to link objects in the 3D scene with domain ontologies and thus build a rich 3D virtual environment in which users can create new greenhouses, save and load previously saved greenhouse instances from the database, walk through the 3D greenhouse model, inspect 3D objects and view all relevant information. They are able to tag objects with

names and group multiple 3D objects with the same name. A search functionality that redirects the user to his/her point of interest within the 3D space has also been implemented.

An energy loss estimation engine operates on the user specified properties of the greenhouse and produces statistics of energy loss per structural component. Users can compare the energy loss characteristics of multiple greenhouse configurations and refine their design. An educational simulation that shows the status of an electric heater with the internal temperature has been implemented to illustrate the capability of extending this system to support simulations.

CHAPTER 1
INTRODUCTION

Planning, designing and constructing a greenhouse has become an activity of interest to a wide range of people from hobbyists to horticultural experts. To encourage people to develop greenhouses and to educate people worldwide about developing such structures, the concept of developing a "virtual greenhouse" has become very popular. These virtual greenhouses are web based electronic representations of their real world peers. The purpose of such greenhouses is to provide a repository of information which users can utilize to make quality decisions about constructing their own greenhouse or selecting which plants to grow in what types of greenhouses. Some virtual greenhouses link to diverse multimedia information sources like images, educational videos and Flash animations. The user navigates through the model and follows the links to his/her area of interest.

With the advent of 3D on the Internet, one of the ways to model a real world greenhouse is to construct a model in 3D. Several virtual reality (VR) environments exist that import such 3D models and render them. Using these VR environments, the users are able to walk through the virtual greenhouse and look at plants or other components at a various levels of detail.

Traditionally 3D virtual environments utilize files (VRML, X3D) that describe the geometry of 3D objects but contain very little additional knowledge of the objects in the VR world. There are many reasons to enhance VR environments with better database support. In a VR world, we need to know not only what an object looks like, but also what the object is, what its properties and characteristics are, how it behaves, and how it relates to other objects. We also need to be able to interact with and refer to objects by pointing or by voice commands.

In our approach we utilize an ontology management system, a database management system based on ontologies, to provide semantic descriptions of objects in the VR environment.

In fact the VR environment can be considered a projecting of objects described by the ontology into a 3D visualization environment. The ontology provides a way of describing object properties and behaviors, describing object taxonomies, and, using ontology reasoners, automatically classifying and clustering objects into categories. The ontology acts as a dictionary that defines the meaning of concepts, and provides support for natural language references to objects.

This thesis focuses on studying the practicability of a virtual reality based 3D environment which is augmented with meta information. The purpose of this research is to build a unified VR environment which offers its users the capability to walk through the 3D model and also inspect, add, modify and search for meta information on the fly. The meta information provided by the user is then used to relate objects in the 3D scene with domain ontologies. Tagging 3D shapes with ontologies builds up a taxonomy that can be navigated by the users to find relevant information. In addition, coupling semantic content with the 3D shapes also makes future extensions possible, such as using a reasoner to replace a text based search with semantic search.

CHAPTER 2
LITERATURE REVIEW

University of Minnesota Duluth (UMD), University Of Wisconsin - Madison Greenhouse Virtual Tour

UMD virtual tour [1] was developed to expose the users to the different groups of plants in the UMD greenhouse and to help the users understand the way the plants and benches are organized. The tour was simulated by a collection of web pages, each providing in-depth information about the greenhouses. The web site contains an index page that provided the visitors with general information and contains links to the two UMD greenhouses, the Upper Greenhouse and the Lower Greenhouse. Users could navigate to the greenhouse of their interest by following the link. Each virtual greenhouse is modeled as a web page using HTML. The web page contains an image that shows the floor plan of the greenhouse. The floor plan describes the location of plants, benches and other fixtures within the greenhouse and is image mapped to internal links within the web page. Users can click on individual parts of the floor plan and jump to the content of their interest, or they could simply scroll down and view the entire content. Information about each plant, bench and fixture comes in two forms: a paragraph of textual information that describes the relevant information and photographs pertinent to that component.

The design of the UW- Madison greenhouse is similar to that of UMD. Use of hyperlinks to model the relationship between rooms in a greenhouse was the adopted technique. The index page contained hyperlinks to all rooms within the greenhouse. Each room was modeled as a web page that contained its own set of hyperlinks that linked to individual plants within the greenhouse. Each individual plant had a web page that provided detailed information about the plant and relevant pictures.

University Of Florida Virtual Field Day

The Virtual Field Day project [2] undertaken by the University Of Florida aims at providing the users access to a repository of information about the research greenhouses at Gainesville and Suwannee. The Virtual Field Day project provides its users with an "Interactive Map" which is a Flash object embedded in a web page. The users can view the greenhouses in the map. The map assists the users in navigating around the locations in the Suwannee or Gainesville Virtual Field Day. Selecting a location takes them to that location's multimedia resources. In addition to the text based and image based information, the site's multimedia resources include instructional videos, publications and research literature pertinent to the technologies used in the greenhouse. In addition, each greenhouse has a Quicktime VR tour which enables the users to interact with the 360 degree panoramas. The tour plays in Quicktime player and has tags on specific areas of interest. Navigation through the tour is limited to a stationary 360 degree view and zooms in/out. Clicking on tags takes the user to a web page that contains a listing of related videos, publications and other instructional material. Separate 360 degree views show the users how the greenhouse looks in the pre-plant season and during the main season.

In both the Virtual Field Day project [2] and the greenhouses by UMD [1] and UW, photographic images serve as the primary means by which the users can visualize the greenhouse. All the information is static and the users cannot simulate changes in the greenhouse environment based on properties like climate or structure. User interactivity is limited to the traditional web based clicking and hyper linking.

The Greenhouse Simulator by University of Vermont

The Greenhouse simulator [3] is a part of a world wide greenhouse education initiative [4] undertaken by The University of Vermont, The University of Florida, University of Arizona

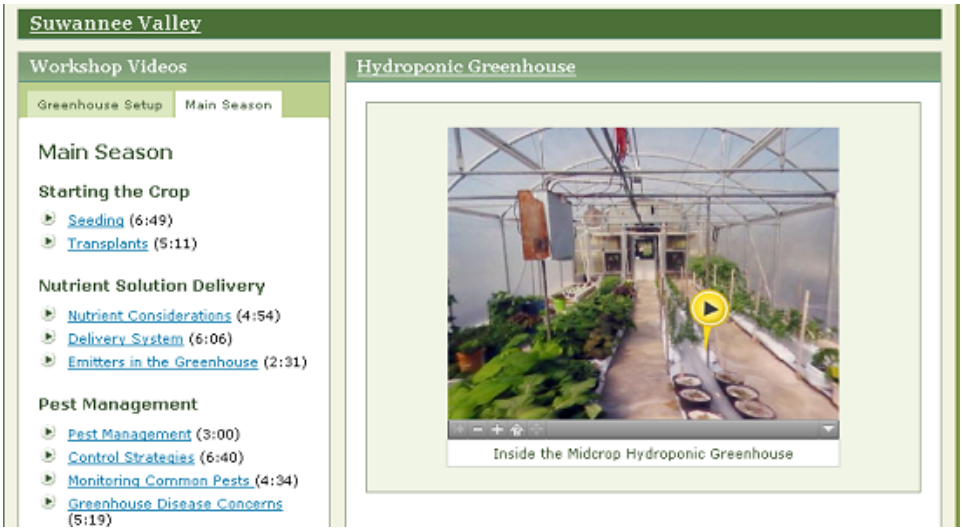


Figure 2-1 Web page showing the 360 degree Quicktime virtual tour on the right and instructional videos related to seeding, transplants etc on the left – Virtual Field Day Project at University Of Florida [2].

and Ohio State. The outcomes of this project include a greenhouse simulator, a digital repository and educational materials. The digital repository contains a searchable collection of images and videos. The simulator is a Flash object that allows users to model the greenhouse environment by making structural, environmental, geographic, and seasonal choices. The simulator incorporates user-selected information from its database of greenhouse designs, operation, and geographic climate conditions, and graphically displays dynamic changes in greenhouse environments, including moist air properties. The animation allows learners to simulate changes in the greenhouse-plant environment based on climate, structure, and environmental control choices.

The interactive greenhouse environment simulator was developed by integrating mathematical models and an animation interface, which was created using Flash MX Pro 2004.The greenhouse mathematical model, which is based on the energy and mass balance of the greenhouse system, utilizes a series of differential equations. The solution provides the dynamic response of the greenhouse climate conditions to the outside climate conditions and for a particular greenhouse design. The design incorporates user-selected inputs for climate, structure,

glazing, and environmental control systems. Each simulation demonstrates the response of a greenhouse system design over a 28-hour period.

The simulator is a stand-alone piece of software and is not linked to any other information sources. The user interface limits the interaction to the user selection of various parameters. Further information about components inside the greenhouse cannot be obtained from the greenhouse model in the simulation.

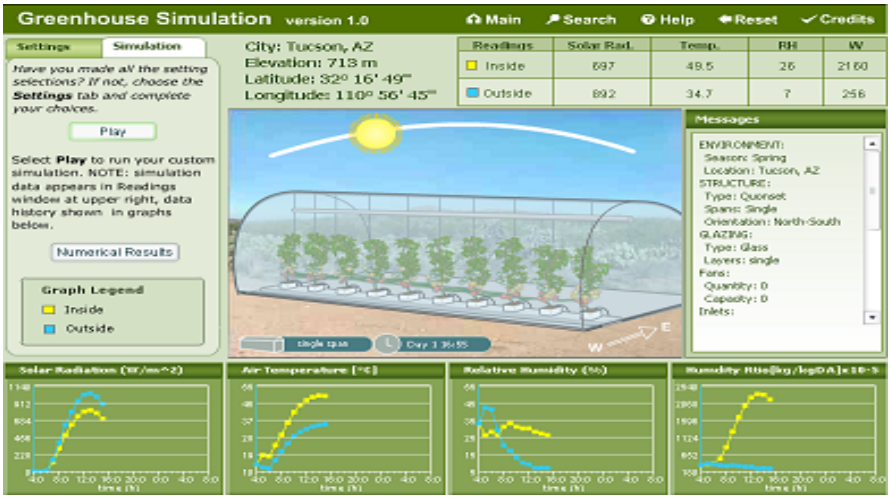
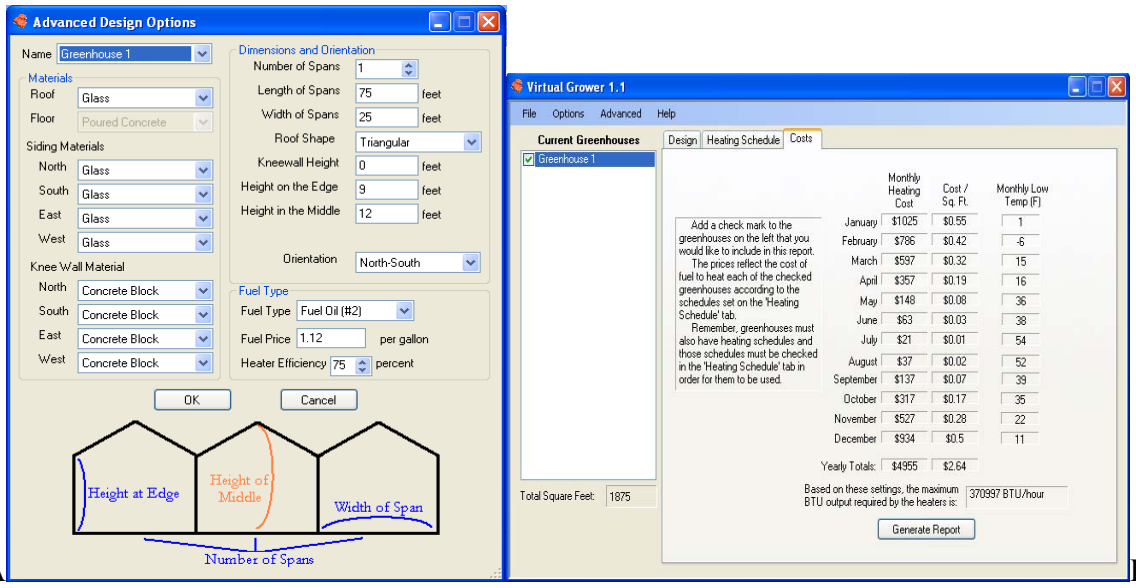


Figure 2-2 Greenhouse simulation shown plotting values of solar radiation, air temperature, humidity etc over a period of 28 hours based on user specified inputs. [3]

Virtual Grower

Virtual Grower [5] is a PC based stand-alone program developed by the ARS Greenhouse Production Research Group at Toledo, Ohio. It is a decision support tool for greenhouse growers that enable the users to estimate the energy requirements of a greenhouse. To estimate energy requirements and costs using Virtual Grower, users input the dimensions of their greenhouse and its construction materials, such as poured-concrete floor, glass sides and roof, or concrete-block walls. They also choose design features, such as roof shape and orientation to the sun. A historic database gives a year's worth of typical weather for the city nearest to the greenhouse location including factors such as temperature, sunlight and cloud cover for each hour of the day.

Growers choose a heating schedule and set the temperatures they want to maintain during day and night, or for each hour. The program then calculates per-square-foot heating costs by the month or year. The users can save their heating schedule, greenhouse configuration or export their results to other formats.



A B

Figure 2-3 Images of Virtual Grower[5] A) User setting various parameters of the greenhouse B) Virtual Grower reporting the estimated heating costs

The visualization of the greenhouse constructed by the users is limited to static line diagrams that depict the structure of the greenhouse as shown above. The primary purpose of this software is cost estimation and hence visualization is not a component. User interaction is also limited to selecting the various parameters.

Visualization of Environment by VRML

Honjo. T and Lim. E [6] designed a visualization system for greenhouse that enables virtual experience in a planned environment was developed using VRML. Their research focuses on walk through simulations as a tool for understanding the design of landscapes and greenhouses. 3D plants and objects with a high polygon count generally take a very long time to render and require expensive, high power hardware for a rich walk through experience. To avoid

this problem, a plant was expressed by using two textured planes instead of thousands of polygons. Computer generated images of plants at several growth stages are made by using AMAP (Atelier de Modelisation de Architecture de Plants) [7], which is the system developed by CIRAD (Center Internationale Recherche Agricultural Development). The plant images then serve as texture for the two planes thereby rendering the plants with minimal overhead. The greenhouse structure itself was made out of simple VRML constructs. The VRML virtual greenhouse enabled walk through. The 3D model was also posted on the web so that users could use a web browser to view the VRML world.

Having the user walk through the 3D model provides a better understanding of various parts of the model since its equivalent to his/her walking through the real world greenhouse. This system focuses only on the visualization aspect. It does not relate 3D models with relevant information or other multimedia sources like images or educational videos. The 3D scene is static since it comes off a VRML file. The user cannot make changes to the 3D scene at runtime or add information to certain components. For instance a user who has an image of a corn plant will not be able to add it to the collection of existing images.

Evaluation

Research literature indicates the need for a system that merges the pros of both 3D based virtual greenhouses and other HTML based models. While the 3D greenhouse provided a better user interaction and navigation capability, it was not associated with multimedia information sources. Compared to this, the traditional web based greenhouses had images, videos and text based information sources. The disadvantage with the HTML based virtual greenhouses was that visualization of the greenhouse was limited to photographs and walking through the greenhouse to foster a better understanding of the real world greenhouse was not possible with plain images.

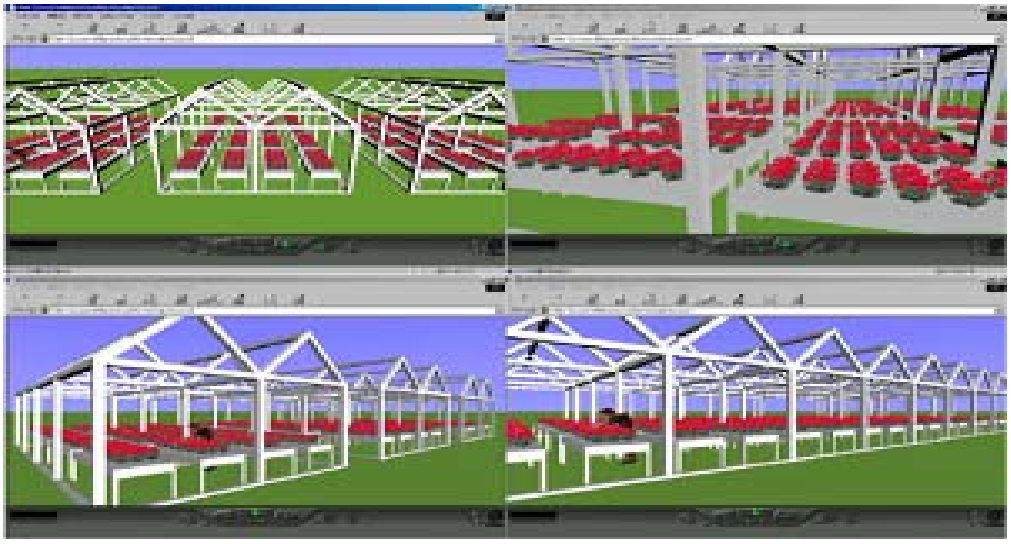


Figure 2-4 VRML Greenhouse developed at Chiba University [6]

CHAPTER 3
TOOLS AND TECHNIQUES

The following design goals were formulated:

- Build a 3D model of a greenhouse through which the users can walk.
- Provide a mechanism by which plants and other components in the 3D model can be annotated with meta-information and also inspected to view existing information sources.
- Store the model and meta-information in an ontology management system that can be accessed remotely.
- Implement a search functionality so that the users can search and be redirected to see the objects of their choice.
- Develop an educational simulation involving thermal characteristics of the greenhouse. The simulation will provide visual cues that depict switching a heater on and off based on heat loss from the greenhouse. This adds dynamic behavior to the 3D greenhouse environment.

To meet all the these goals, the following tools and techniques were employed.

Java Swing

Java Swing [22] is part of the Java Foundation Classes, a group of classes that encompass features for building GUIs and adding rich graphics functionality and interactivity to graphics applications. The other components that form a part of JFC include

- PLAF (Pluggable Look And Feel)
- Accessibility API to enable assistive technologies like screen readers
- Java 2D API to help developers enrich their applications with 2D graphics and text,
- Internationalization to enable the development of programs that can interact with the users in their own local language and cultural conventions
- Drag and Drop.

Prior to Swing, Abstract Window Toolkit (AWT) was used to construct user interfaces. AWT had several problems associated with it. It was not sufficient for complex user interfaces, it was not portable, it heavily relied on the runtime platform to render many of its user interface

components and had a poor event model. Swing was developed on top of AWT to solve many of the problems associated with it. Swing provides pluggable look and feel. All components in Swing are lightweight in the sense that they do not depend on native runtime environment except for a few top level containers. Several new components such as sliders and progress bars are added to the list of existing AWT components. This research uses Java Swing to build the User Interface.

VRML

VRML (the Virtual Reality Modeling Language) [8] is an international standard for describing 3-D shapes and scenery. VRML files describe a 3D world. Visualizing the 3D world can be done by viewing the VRML file in a VRML browser. Since many of the VRML browsers are available as web browser plug-ins, 3D information can be easily published on the web using VRML.

The VRML file contains a node hierarchy which reflects the way in which individual shapes in the 3D scene are grouped together to form composite objects. Grouping nodes in the VRML files serve to group together their children. Transform nodes, in addition to grouping a set of nodes also add a translation, rotation or scale to all of its child nodes. Shape nodes contain information about the appearance and geometry of each shape in the 3D scene and are grouped within grouping nodes. The appearance of the shape node is usually provided by an Appearance node that controls several properties like coloring aspects, textures, transparency and opacity. The geometry of a shape node can be described by making it one of the predefined types like a BOX or by using an IndexedFaceSet node. An IndexedFaceSet node lists the coordinates of the points that form the shape and groups up points that form an individual face of the shape, one group per face, thereby building up the structure. Nodes in a VRML file are named by using the DEF construct. Prefixing a node declaration with a user defined name assigns the user defined

name to the node and makes it possible to refer to that node anywhere in the file using that name. USE construct makes use of such “DEFed” nodes to reuse existing node declarations. For example, in the following node hierarchy given below, an appearance node and a geometry node get DEFed with user defined names and they are reused further down the hierarchy to control the appearance and geometry of another shape.

```
Shape {
        appearance DEF myAppearance Appearance {
        ….
        }
        geometry DEF myGeo geometry {
        …
        }
}
…
Transform {
        translation 0 0 0
        Shape{
                appearance USE myAppearance
                geometry USE myGeometry
        }
}
```

The 3D scene built by the VRML nodes is static. To add animations or to make it interactive based on the user input, the VRML nodes must be connected to each other and messages must be passed between them. For this purpose, most of the nodes have special fields called eventIns and eventOuts. EventIn fields are like receivers, which receive messages called

events from the outside and take them in to be processed. EventOut fields are transmitters, which send events from the node to the outside. Nodes are connected to each other by attaching these fields using ROUTEs. These are like pipes that channel events from an eventOut into an eventIn. Many eventIns can be connected to one eventOut to make one event cause many things to happen. This is called fan-out. Fan-in is also allowed, where two or more eventOuts feed into one eventIn.

VRML also contains some special nodes. These nodes act like sensors in that they sense user interaction with the scene and react to it by throwing events. One such node that provides interactivity in the 3D greenhouse is the Touch Sensor node. This node detects mouse interaction with its sibling geometry. Just like other nodes, TouchSensor also has some eventIn and eventOut fields. isOver, isActive and touchTime are the eventOuts in TouchSensor. When the mouse is moved over the geometry, an “isOver TRUE” event is generated, and when it moves off, an “isOver FALSE” is sent. When any of the mouse buttons is clicked while isOver is TRUE, two events are generated, namely , an “isActive TRUE” event and an the “touchTime <currentTime>” (where currentTime indicates the time when the user clicked on the shape) event is generated. To illustrate the process of creating animations in a VRML scene, consider a scene that has a touch sensor node and a Sound node. The Sound node is used in VRML for generating 3D sounds. When the touchTime eventOut of the TouchSensor is connected to the beginTime eventOut of the Sound node using a ROUTE, the sound file specified in the sound node is played when the user clicks on the shape that the TouchSensor monitors.

VRML files can be hand coded or generated from third party applications. The 3D models for the greenhouse are fairly complex. Hence, a third party software modeling tool SketchUp, was used to build the 3D model for each of 6 different types of greenhouses. The models were

then exported to generate VRML files.

Xj3D

VRML as such provides a language by which modelers can describe a 3D scene. VRML by itself cannot display a 3D scene without the software to render the 3D scene. To render the environment, a VRML browser is required. Xj3D [11] is a project of the Web3D Consortium focused on creating a toolkit for VRML97. X3D content is written completely in Java and provides a VRML browser that can be used to render a 3D scene. We can use Xj3D to render the VRML scene from a VRML file or use its API to build up the scenegraph directly by constructing nodes one at a time and adding them to the scene. Xj3D also has a parser implementation that can be used to parse a VRML file and generate an in memory representation of the VRML node hierarchy.

Interactivity and animations can be added to the VRML file by two mechanisms. The first one is scripting within VRML. Scripting languages like ECMA Script [10], JavaScript can be placed inside special type of nodes called Script Nodes. These scripts process on internal events and change the nodes in the scenegraph. The other way is to use EAI (External Authoring Interface) [11].The EAI enables control of the contents of a VRML browser window embedded in a web page from a Java (tm) applet or such an external source on the same page. The difference between EAI and internal Script node is that EAI offers a generalized method to access nodes and events in the VRML browser's scene graph by connecting to a VRML browser plug-in through a browser plug-in architecture like Live Connect or ActiveX/COM. Internal Script Nodes have node access through definition within a VRML Script Node within a VRML scene graph. The EAI supports events and fields on a dynamic basis, while the Script Nodes' definitions consist of predefined fields and events. Xj3D merges the EAI and scripting paradigm

and provides a single unified interface called SAI (Scene Access Interface). It provides a compatible way to run and modify X3D and VRML scenes for many X3D implementations.

The virtual greenhouse uses the Xj3D API for two purposes. Xj3D Parser API is used to parse the VRML file. The parser operates based on an eventing paradigm. The parser expects the caller to register a content handler with it. The content handler contains a list of functions that are called back by the parser at critical points such as document entry/exit, node entry/exit and field entry/exit. For instance, when the parser hits the beginning of a node, it calls the function startNode on the content handler. The content handler contains a set of stacks that it uses to pass information between calls to the functions. For instance, when the start Node function is called, the content handler creates a new hash map keyed on field names to store the field values for this node and pushes it onto the top of the stack. When field data is encountered by the parser, it calls the fieldValue method which creates a new entry in the hash map. When a node definition ends, the parser calls the endNode method which removes the hashmap from the top of the stack. Thus, using the parser API and a custom content handler implementation, an in memory representation of the VRML file can be generated.

The second place where Xj3D API is used is to provide an environment for 3D visualization. Xj3D provides a VRML browser implementation that is used to host the 3D greenhouse. VRML nodes are constructed based on the shape information present in the database and the nodes are added to the scene graph which renders itself once constructed completely. Thus, using SAI, the application interfaces with the database and updates the scene at run time during the simulation.

Lyra Ontology Management System

In philosophy, ontology is a theory about the nature of existence, of what types of things exists; ontology as a discipline studies such theories. The most typical kind of ontology has

taxonomy and a set of inference rules. The taxonomy defines classes of objects and relations among them. The most basic concepts in a domain form the roots of various taxonomic trees as classes. A concept more specific than the general high level concept is considered to be a subclass. The generic concept becomes the superclass of the sub class. Members of a class are called instances. Instances can be thought of as corresponding to individuals in the real world. Properties assert general facts about the members of classes and specific facts about individuals.

Let us consider the plant ontology given below.

Kingdom: Plantae

Division: Magnoliophyta

Class: Liliopsida

Order: Cyperales

Family: Poaceae

Genus: Zea

Species: Zea mays

Kingdom, Division, Class, Order, Family and Genus are all classes representing concepts in this ontology forming a taxonomic hierarchy for classification of plants. Plantae, Magniliophyta etc are instances of the corresponding classes. The kingdom Plantae indicates that corn is a plant and probably photosynthesizes using chlorophyll. The division Magnoliophyta indicates that it is a flowering plant. The class Liliopsida indicates that corn is a monocot—that is, it has one cotyledon (the first leaf formed on a seedling) and leaves with parallel veins, and its flower parts occur in multiples of three (three or six petals, three or six stamens, and so on). The order Cyperales indicates that the seeds store starch and the flowers lack petals and sepals. The family Poaceae indicates that it has a unique type of inflorescence, round stems, and a distinctive internal anatomy and so on, down to the species level.

An ontology management system (OMS) is a database management system that utilizes formal ontology languages as the basis for modeling and manipulating data. The OMS provides facilities typically available in database management systems including maintenance of database integrity, physical storage management, security, transactions, backup/recovery, and data manipulation. Using the OMS's data definition language, onotologies can be constructed, and using traditional data storage management techniques, the ontology can be stored and retrieved. Physical storage management is particularly important because the VR environment requires management of very large numbers of objects. Dynamic scene generation requires that just the right objects to be loaded into memory at the appropriate time to expand a scene or increase level of detail in object presentation. This research uses Lyra [12], an Ontology Management System as a core database management facility. Lyra stores data in the form of objects. An object in Lyra is variable length block of bytes. Each object can be identified using either an integer Object Identifier (OID) or a string Object Identifier (SOID). Lyra creates hash based indices on both OIDs and SOIDs to enable rapid lookup of objects.

Lyra's data definition language represents all generic concepts as classes and individual members as instances. Attribute is the term used to represent a property in Lyra. Attributes are relationship between objects. They have a domain and a range and they are directional, meaning that they connect the source objects in the domain to the target objects in the range. For instance, an attribute "made from wood" could be used to connect a door in a greenhouse with a type of wood. Thus the domain would be class door and the range would be class wood. Attributes can be used in either classes or in instances. When used with classes, attributes are used to restrict the set of values that the class instances can have for that attribute. Such a restriction imposed on the attribute values is called an attribute restriction. Cardinality is an example of an attribute

restriction. Assigning a minimum and maximum cardinality restriction to an attribute enforces that the attribute appears at least minimum times and no more than maximum times in the class instances. For example, when an attribute restriction is created for the door class that specifies that the cardinality of the attribute “made from wood” to be one, all instances of the door class must definitely specify exactly one value for the attribute “made from wood”.

When an attribute is used in the context of an instance, it is called an attribute instance. attribute instance describe properties of an instance. For instance, the attribute “made from wood” could be used in an instance “door1” to specify that door1 is made from pine. Thus the attribute instance of type “made from wood” relates door1 with the wood instance pine. Lyra provides two ways for storing attribute instances in the database, namely, storing each attribute instance as an object or grouping together a set of attribute instances and embedding the group of values within an object rather than making individual objects. Attribute vector is the term used in Lyra to represent the grouping of attribute instances. Each attribute vector has a name and maintains a hashtable that stores all the attribute instances as a set.

Lyra supports several data types from simple ones like integers and strings to more complex ones like images and videos. The values that attributes take in an instance are stored in the Lyra as “Data” objects. A Data object is actually a generic object type that is used in Lyra to store all types of data in binary format. Values are coerced and type casted to their corresponding data types as and when necessary.

For example, consider the problem of representing a 3D point in Lyra. The coordinate of a point in 3D space can be represented using 3 values: x, y and z. Each of the 3 values is of floating point type. To model this in the form of Lyra objects, three attributes representing the x, y and z properties of a point are created. Let us call the attribute objects aX, aY and aZ and their

OIDs xOID, yOID, zOID. The x coordinate value will be wrapped in a data object. An attribute instance will be created and associated with aX using its oid xOID and the data object that wraps the x value will be added to the attribute instance. Similarly, attribute instances will be created, one for every other coordinate value. An instance is created with the name "Point" and all the three attribute instances are added to this instance.

Extending this approach to model a complex shape is relatively simple. Consider a box shape that has 8 end points. Each point is represented using the same methodology as above with the exception that instead of creating an instance for each point, an attribute vector called "Point" is created and attribute instances for each of x, y and z coordinates is added to it. An instance is then created with the name "Box" and the attribute vector is added to it.

Lyra being an OMS can be used to store domain ontologies. Having both the 3D model and domain ontologies stored in a single data store makes it possible to tag shapes in the 3D model with semantic information. For instance, consider a 3D scene where a box is used to represent a computer. Assume that we have a domain ontology stored in Lyra that provides taxonomy of computer types as shown in Figure 3.1.

If we specify that the box in the 3D scene represents a Desktop Computer instance, then the 3D scene automatically gets enriched with the domain ontology about computers. Since the box is a desktop computer, it can be inferred that it is automatically a digital computer. We could use another ontology that lists the average price of digital computers to get a rough estimate of the price of the desktop computer. Thus, using an OMS helps us combine the 3D visualization aspect with the semantic aspect to produce an ontology based 3D visualization environment.

Connecting to the Lyra database is done using RMI (Remote Method Invocation). RMI provides for remote communication between programs written in Java. RMI allows java objects

running on one system to connect to and invoke methods on an object running in another machine. A typical RMI server creates the remote objects, makes them available for access and waits for clients to invoke methods on the objects. An RMI client obtains references to the remote object and invokes methods on the object. RMI takes care of the communication between the client and the server. An RMI server registers or binds its remote object usually with the simple naming facility provided by RMI called the RMI registry. The client looks up the remote object by its name from the RMI registry in the server and invokes a method on the object. Lyra uses RMI to enable the users to connect to a database from anywhere.

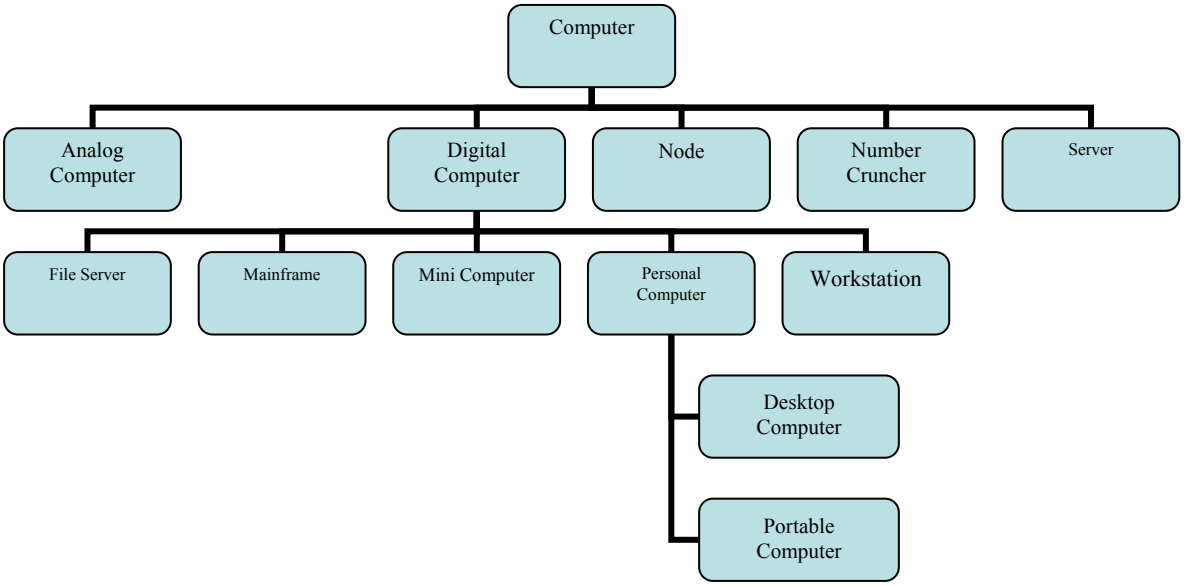


Figure 3-1 Domain ontology for computers

Thus, using Lyra enables us to add structure to the data by organizing the data with well defined domain ontologies and makes remote access possible through RMI.

Addressing the Limitations

In the design of landscapes and greenhouses, walk through simulation in a 3-D space is a useful educational tool, especially for understanding design and for selection of alternative designs. Walk through simulations serve to provide a rich user experience as highlighted in [6].

Storing the entire model in a database and operating off the database provides three advantages. The first advantage is that it gives complete control over building the 3D scene. The 3D scene can be modified as it is loaded or after it is loaded depending on the situation. The entire model or parts of it could be pulled up at runtime depending on the necessity. For instance, we could render the 3D greenhouse without the plants and heaters or with them depending on whether the user is interested in visualizing the structure of the greenhouse or inspecting the components inside the greenhouse. If a static VRML file is used, unless we provide a custom VRML browser implementation , all modifications to a scene can take place only after the scenegraph is constructed and hence partially loading a scene would not be possible.

The second advantage is that it simplifies the process of tracking changes to the 3D model. Changes made to the 3D model can be saved by updating the existing model in the database. Updating a 3D model does not present any problems as it would in the case of a static VRML file. Xj3D does not directly support exporting a 3D scene at runtime to a VRML file and even if we altered the SAI implementation to support this, we would have to write it out to a new VRML file. This presents several problems since the user could make multiple changes and saves over a period of time and each save should either save the entire scene, which impairs user interaction due to disk write delays, or maintain some way of incrementally saving the changes to the new file. As a result, there would be multiple copies of the same VRML file. These problems can be avoided using a database because we can change the 3D model and make updates to the database to reflect the changes upon saving and thus having a single instance of the 3D model. The third and the biggest gain from using an OMS to store both the 3D model and the associated meta information is that the graphics content and the semantics of the scenes are combined into a consistent and cohesive ontological model. Users can connect objects in a 3D scene with classes

in a domain ontology. By doing this, the 3D scene becomes enriched with semantic information. Associating domain ontologies with 3D shapes also generates a taxonomy which can be navigated using a taxonomy browser such as the Web taxonomy Browser [14].Using an ontology management system also opens up several other possibilities as discussed in chapter 5, such as using reasoners to search the semantically augmented 3D greenhouse, positioning the greenhouse on a geo-referenced 2D map and exporting the 3D model to other formats. Sharing models is simplified since anyone can connect to the database remotely.

The text based search functionality is similar to the hyperlink based navigation with the difference that the user viewpoint gets reset within the 3D model to the plant of interest. The user can enter search terms and have their viewpoint reset within the 3D model to the region of interest

CHAPTER 4
ONTOLOGY BASED 3D VIRTUAL GREENHOUSE

Domain Ontologies

A greenhouse ontology is shown below in figure 4.1.

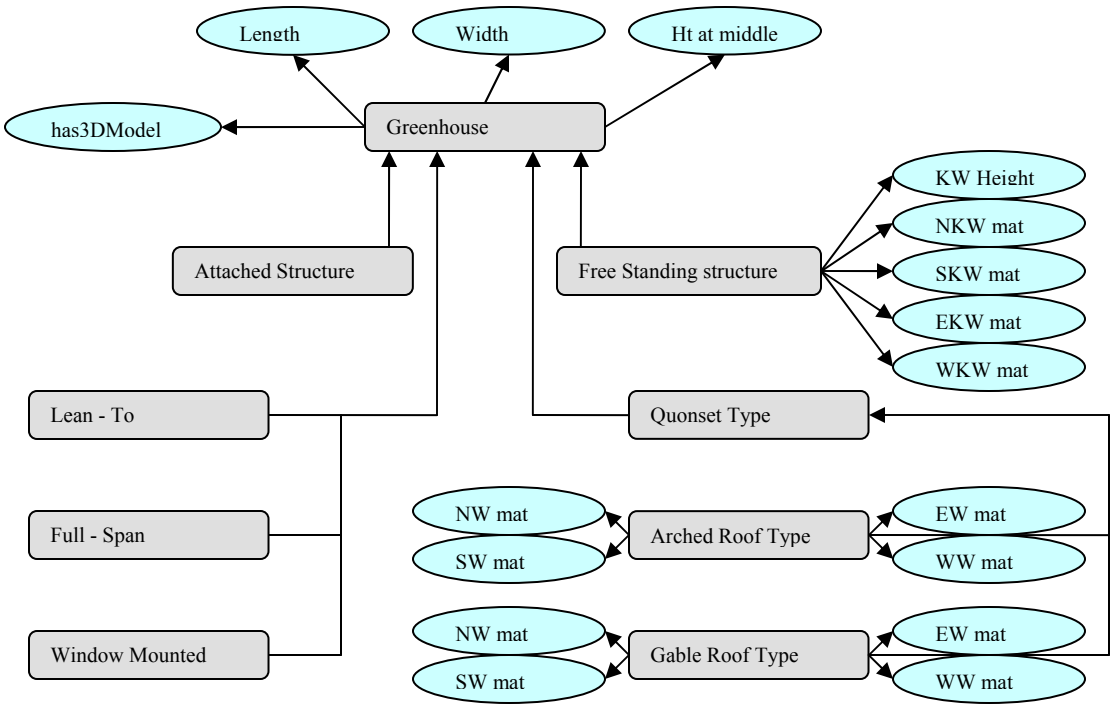


Figure 4-1 Greenhouse domain ontology. Grey shaded boxes represent classes. Arrows point from a subclass to its parent. Blue shaded boxes are properties.

A lean-to greenhouse is a half greenhouse, split along the peak of the roof, or ridge line. Lean-tos are useful where space is limited to a width of approximately seven to twelve feet and they are the least expensive structures. The ridge of the lean-to is attached to a building using one side and an existing doorway, if available. Lean-tos are placed close to available electricity, water and heat. The disadvantages include some limitations on space, sunlight, ventilation, and temperature control. An even-span is a full-size structure that has one gable end attached to another building. It is a costlier option and it provides more space than a lean-to greenhouse. Window mounted structures provide a convenient method for growing a few plants at relatively

low cost. The disadvantage with most of the attached greenhouses is that if not positioned properly, exposure to the sun may be inhibited by the attached structure. There are also problems with snow or water sliding off from the roof of the attached structure onto the greenhouse and with walls of the attached structure retaining the sun's heat while the greenhouse cover loses heat rapidly.

Freestanding greenhouse structures are separate structures. They can be set apart from other buildings to get more sun and can be made as large or small as desired. The Quonset frame style is a simple and efficient construction with the disadvantage that the sidewall height being lower, storage space and headroom is restricted. As far as structural components are concerned, the Quonset has no side walls compared to the arched and gable style greenhouses. All the three freestanding structures may or may not have kneewalls.

The Greenhouse class forms the root of the taxonomic hierarchy. 3 attributes describe its structure, namely, length, width and height at the center. Attribute restrictions on each attribute specify that the minimum and maximum cardinality values to be 1 meaning that every greenhouse instance should have these three attributes exactly once. The greenhouse class also has a restriction on an attribute called "has3dmodel" .This Attribute restriction sets minimum cardinality to 0 for attribute "has3dmodel" and it is used to link a greenhouse instance with the Lyra object that contains the 3D rendering information.

Freestanding structures may or may not have knee walls. If present, there may be different materials for the north, south, east and west knee walls. This is reflected by creating attributes, one for each knee wall and adding Attribute restrictions to the Freestanding structures class to set the minimum cardinality to be 0 and maximum to be 1 for each of these attributes. A domain ontology for tomato diseases was created and stored in Lyra. [14] was used to create the

ontology shown in figure 4.2. A 3D object in the virtual greenhouse will be associated with a class type in the domain ontology. By doing this, the 3D scene gets transformed into an environment which knows about its 3D objects and can be queried using reasoners to infer facts.

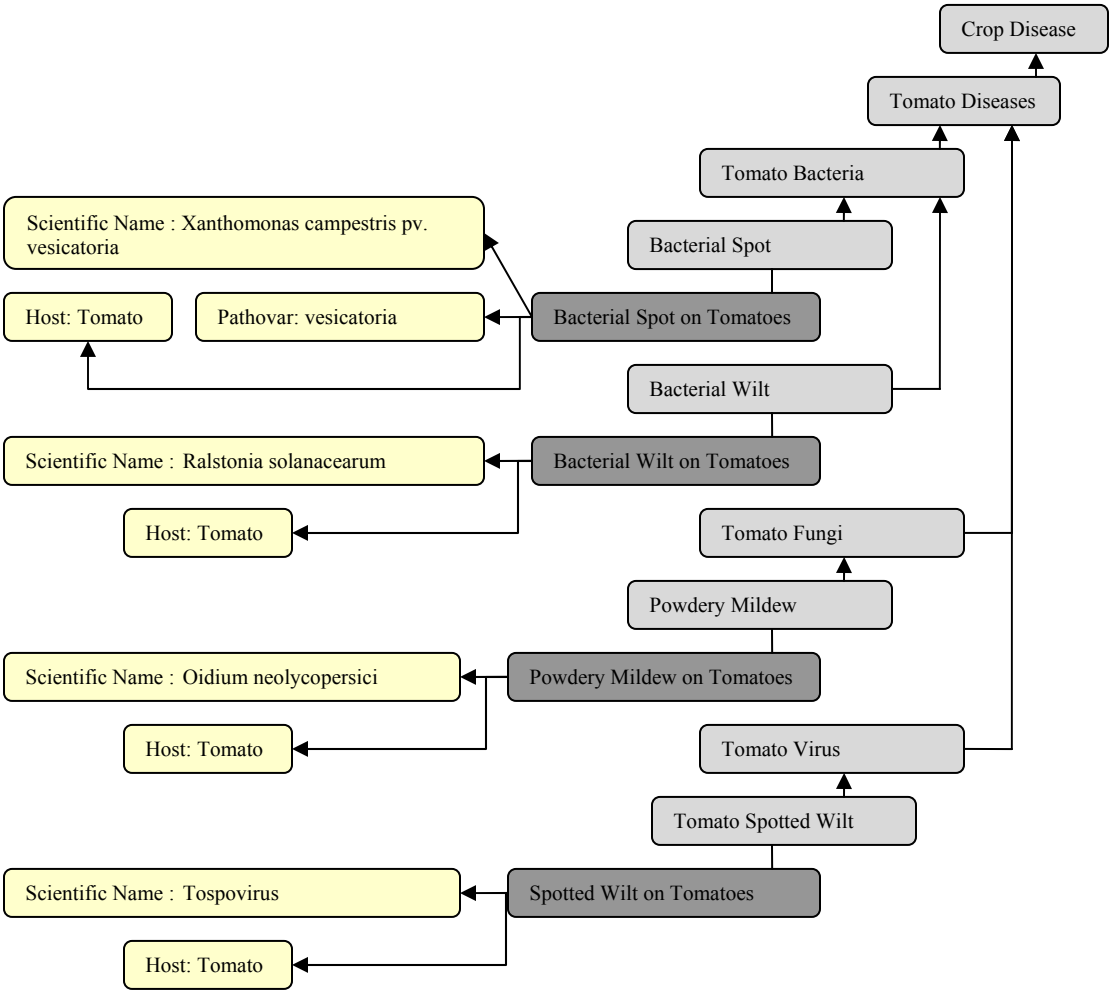


Figure 4-2 Tomato Plant ontology. Light Gray denotes classes, dark grey denotes instances, and light yellow denotes properties.

Database Design for 3D Objects

VRML Shape nodes are the only type of nodes for which physical entities are created in Lyra. The translation, rotation and scale values are stripped off form the parent transform nodes and stored as attributes with the shape node. Attribute vectors are created, one for each shape node present in the VRML file. Each attribute vector records the translation, scale and rotation

values inherited from all parent transform nodes along with the coordinate and texture coordinate values. Each attribute vector is named after the corresponding node's DEF name if the node has one or a random name based on the nodes position in the hierarchy .All the attribute vectors created are then grouped together and added to the greenhouse instance as an attribute instance of type "has3DModel".

User Interface

The user interface shown in Figure 4.3 is used to construct a greenhouse. The user can specify properties such as the length, width, height and the temperature at which the greenhouse must be maintained. By default, the greenhouse is made out of glass. The users can also click on the "expand" button and change the type of structural materials for each part of the greenhouse as shown in the Figure. In addition, the user has to specify a VRML file that can be used to visualize the greenhouse in 3D. The user can save the configuration or load an already serialized configuration from the File menu.

Methodology

For each greenhouse that the user creates, an instance is created in the database with the user specified name. The instance refers to the one of the different class types specified in the domain ontology as its parent depending on the greenhouse type that the user chooses. Attribute instances are created to store the length, width ,height, temperature and other properties of the greenhouse and they are added to the greenhouse instance. The VRML parser is run on the VRML file to create attribute vectors that contain the rendering information .The attribute vectors are linked to the greenhouse instance by adding it as an attribute instance of type "has3DModel" to the greenhouse instances To view an already serialized greenhouse, the user loads the greenhouse using its name. Lyra is searched for an instance with the user specified name. The instance, if present, is obtained and its attribute instances are extracted. Using the data

present, the user interface is then populated. Using the meta information details extracted from the database, a meta information map is created.

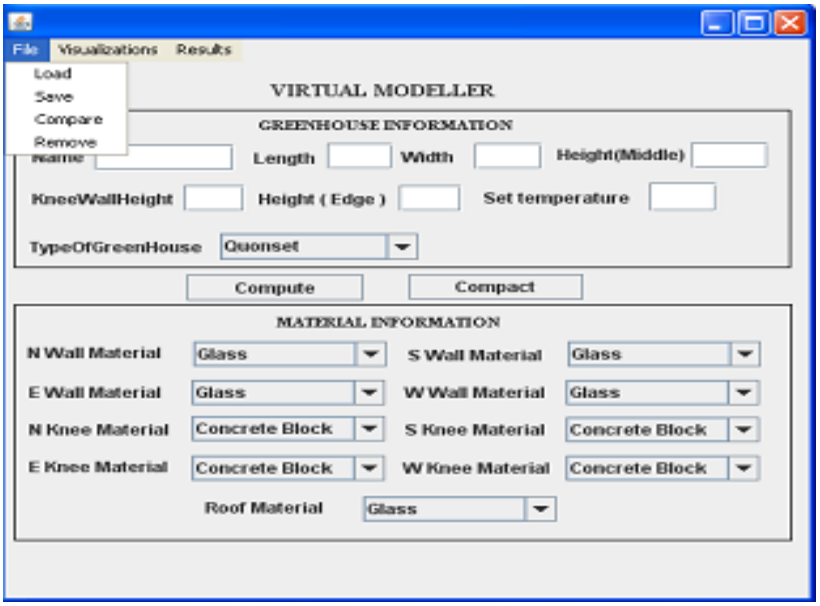


Figure 4-3 Swing based UI for setting the properties of the greenhouse.

The map contains two types of key value pairs. The first type maps a DEF name to a user defined name and the second type maps a user defined name to relevant meta information. This map is kept synchronized with the database and any changes made to the map are immediately updated in the database.

Mentor Mode

Clicking on the "mentor mode" option in the visualize menu brings up the 3D user interface. The 3D model is rendered and displayed as shown in 4.4 depending on the greenhouse type the user selects in the user interface.There are several ways users can navigate through this model. Walking enables collision, so if the user collides on a wall or an object while moving he is stopped. The "Fly" mode turns off collision detection. In addition, users can also pan through the 3D model or tilt the model to adjust their visualization angle. Context clicking on the 3D

model brings up a menu. The users can search the greenhouse or inspect parts of the greenhouse by clicking on the corresponding menu item.

Methodology

When the user requests an entry into the mentor mode, the 3D model corresponding to the greenhouse the user loaded earlier is pulled up from the database. Using Xj3D, shape nodes are constructed from each of the attribute vectors. A touch sensor node and a viewpoint node are created for each and every shape node. A custom event handler is created and associated with the touch sensor node. The event handler remembers the DEF name of the shape node and provides a call back method that gets fired when a user touches this shape node. The call back method creates a dialog that shows meta information about the shape node. A transform node is created, one for each shape node and the translation, scale or rotation attributes are added as fields of the transform node. The shape node and the associated touch sensor and viewpoint nodes are added as children to this transform node. In addition, the sensor node gets added to a vector of sensor nodes and the viewpoint node gets added to a hashtable which is keyed off the DEF name. The transform nodes are then added to the scenegraph.

Inspect Mode

When in mentor mode, if the user wants to inspect objects in the 3D model, he/she enters the “Inspect Mode”. The user can do this by context clicking on the 3D model and selecting the “Inspect Mode” option. By doing this, all the touch sensor nodes , one per shape node, that were added during the time the scene was loaded,are enabled by getting each sensor node from the vector and setting their “isEnabled” eventIn to true. The user can switch out of the inspect mode any time by context clicking again and selecting the “Navigate Mode” option. When the user goes back to the navigate mode, the touch sensors are disabled.

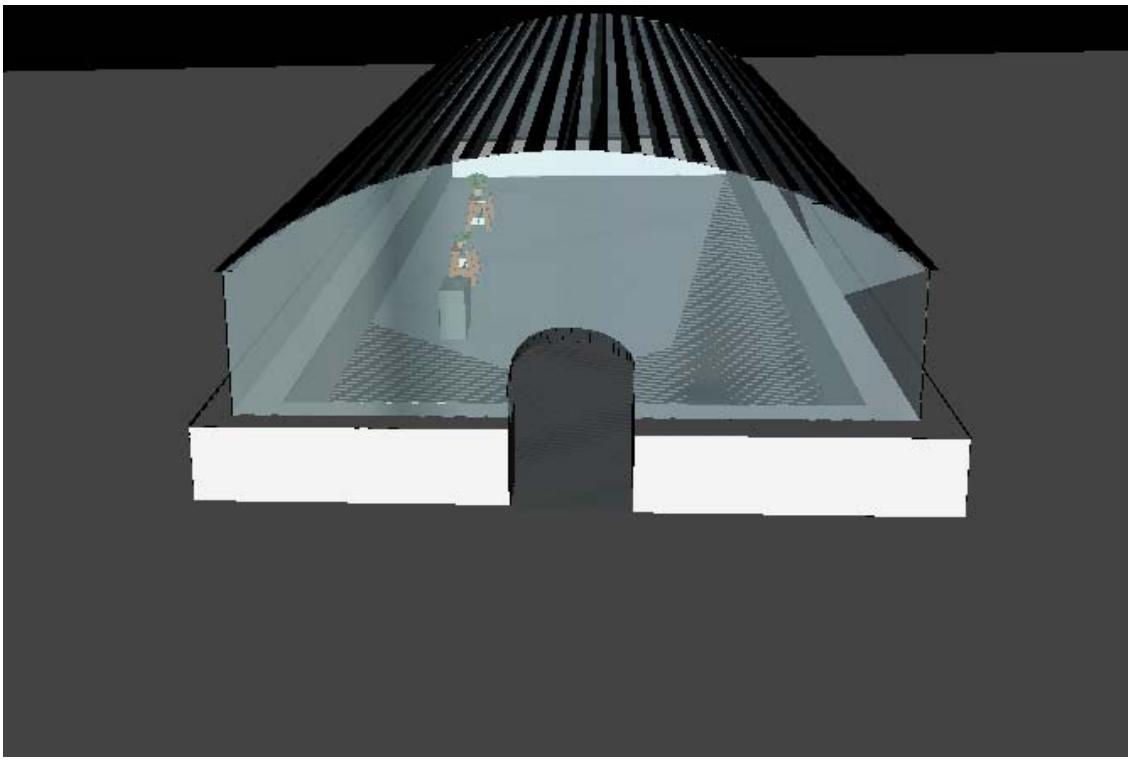


Figure 4-4 The 3D greenhouse rendered in Xj3D.

In the inspect mode, when the user clicks on a component, a modal dialog is displayed. Information in this dialog is divided by a tabbed pane. The user can enter the user defined name for this component, add viewpoint metrics and textual comments describing this component in the first pane as shown in Figure 4.5.Using the second pane, the user can add images relevant to this component. Images that have already been added show up as thumbnail buttons before the "Add a new image" button. Clicking on the thumbnail renders the full size image. Title for the image can also be changed by clicking on the button next to it. New images can be added to the existing ones on the fly by clicking on the "Add a new image" button. Clicking on this button brings up a file dialog which can be used to locate the image. Once the image is selected, a thumbnail of the image is created and added to the existing ones. Clicking on the save button saves the new meta information. Figure 4.6 shows images of different types of heaters being added.

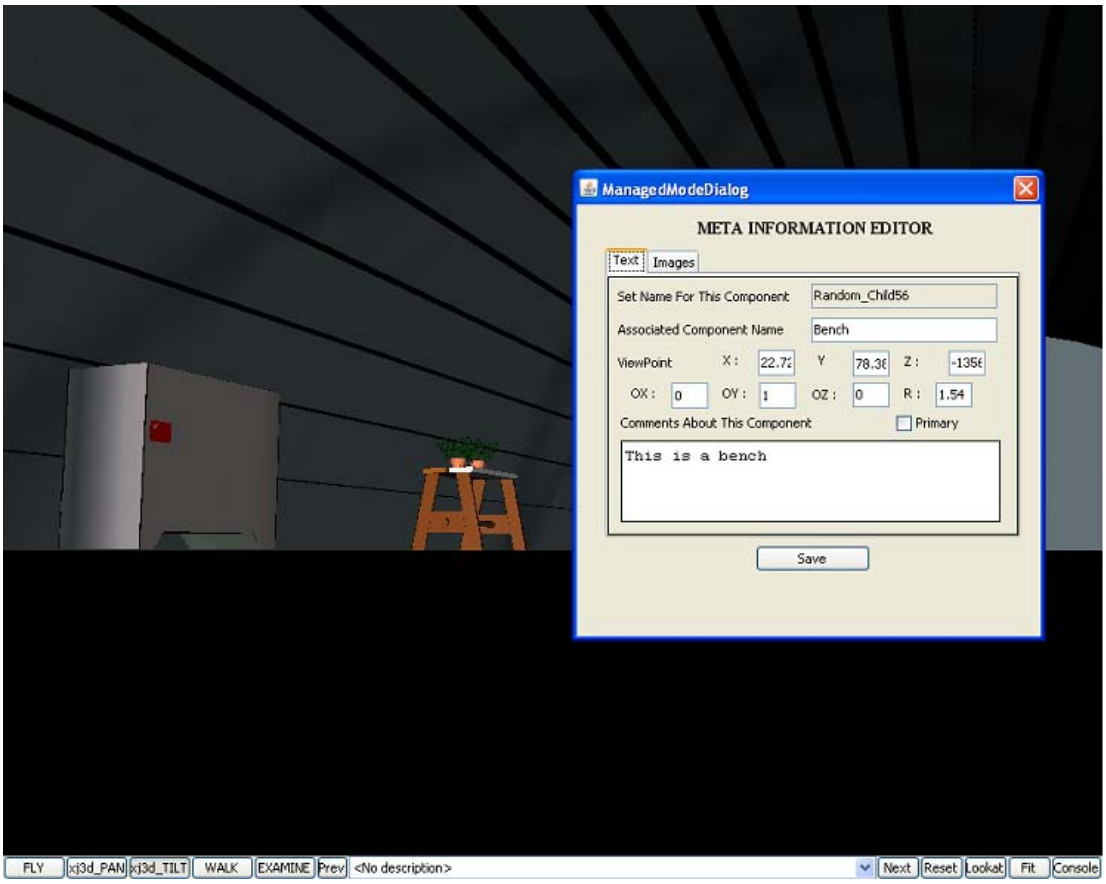


Figure 4-5 Textual information pane containing comments, user defined names and viewpoints.

Methodology

In the inspect mode, when the user clicks on a shape, the meta information map is searched for the shape's DEF name to get the user defined name. The map is searched again based on the user defined name obtained in the previous search. This search returns back all the meta details associated with this shape which is then used to fill the meta information dialog with data.

When the user clicks on save in the meta information dialog, Lyra is searched for classes or instances with the same SOID as the user defined name. Attribute instances are created and user provided meta information like DEF name, comments and images are added as Data values to the corresponding attribute instances. A new attribute called "refersTo" is used to link the instance corresponding to user metadata with the class or instance in the domain ontology. The

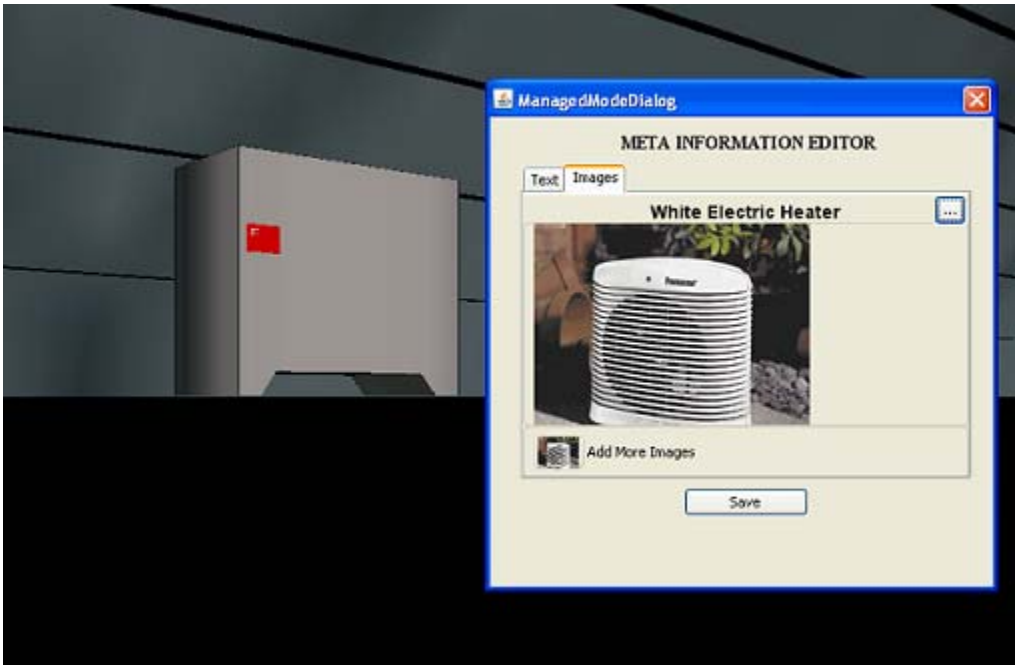


Figure 4-6 Image of an electric heater being added to the 3D object. The image shows the thumbnail and the full sized image.

attribute, as its name implies adds semantic information by asserting that the 3D object that the user clicked on maps to a particular class type or a particular instance in the domain ontology. An attribute instance of type "refersTo" is created and it stores the OID of the class or instance in the domain ontology as its data. All the attribute instances are then grouped together into an attribute vector named after the user defined name and the attribute vector is added to the greenhouse instance. Two entries, one with DEF name and user defined name as the key-value pair and the other with the user defined name and OID of the instance created previously as the key-value pair, are added to the meta information map.

For example, when a user clicks on the 3D object representing a tomato plant, the modal dialog is displayed. The user then tags the object with the user defined name "Bacterial Wilt On Tomatoes", types out a short textual description and adds some images. When the user clicks on save, Lyra is searched for an object with the SOID "Bacterial Wilt On Tomatoes" which is an Instance in our domain ontology. An attribute instance of type "refersTo" is created and the OID

of the “Bacterial Wilt On Tomatoes” instance is added as the data value to the attribute instance. Attribute instances are created to store comments and images. All the attribute instances are then grouped together into an attribute vector and updated back to the current database instance. Two entries are made in the meta information map ,one being the key value pair <DEFNAMEX,”Electric Heater”> and the other being the pair <”Electric Heater”,OID>.

By this mechanism, the 3D scene gets enriched with the semantic information present in the domain ontology. It is possible to use a reasoner in the future, as discussed in the next chapter, to make inferences or to process queries like “Show me all plants in the greenhouse that are diseased”.

The dialog also provides a way by which the user can group different 3D objects into a single entity. For instance, the 3D model for a bench contains several shape nodes one for each face of the bench. This can be verified by clicking on different faces of the bench and observing that each face has a different DEF name. Using this tagging functionality, we can set the user defined name to be the same for all parts of the bench thereby grouping the shape nodes under a common name. Having a common name comes with the additional benefit that meta information added for one shape node of the group shows up when any other member of the group is inspected. Figure 4.7 illustrates this.

Searching

Clicking on the search menu item brings up an input dialog in which the user enters the user defined name of the component being searched. Figure 4.8 depicts searching for corn plant.

Methodology

Searching for a component with its user defined name requires that the component be tagged with that user defined name. In addition, the viewpoint details for the component

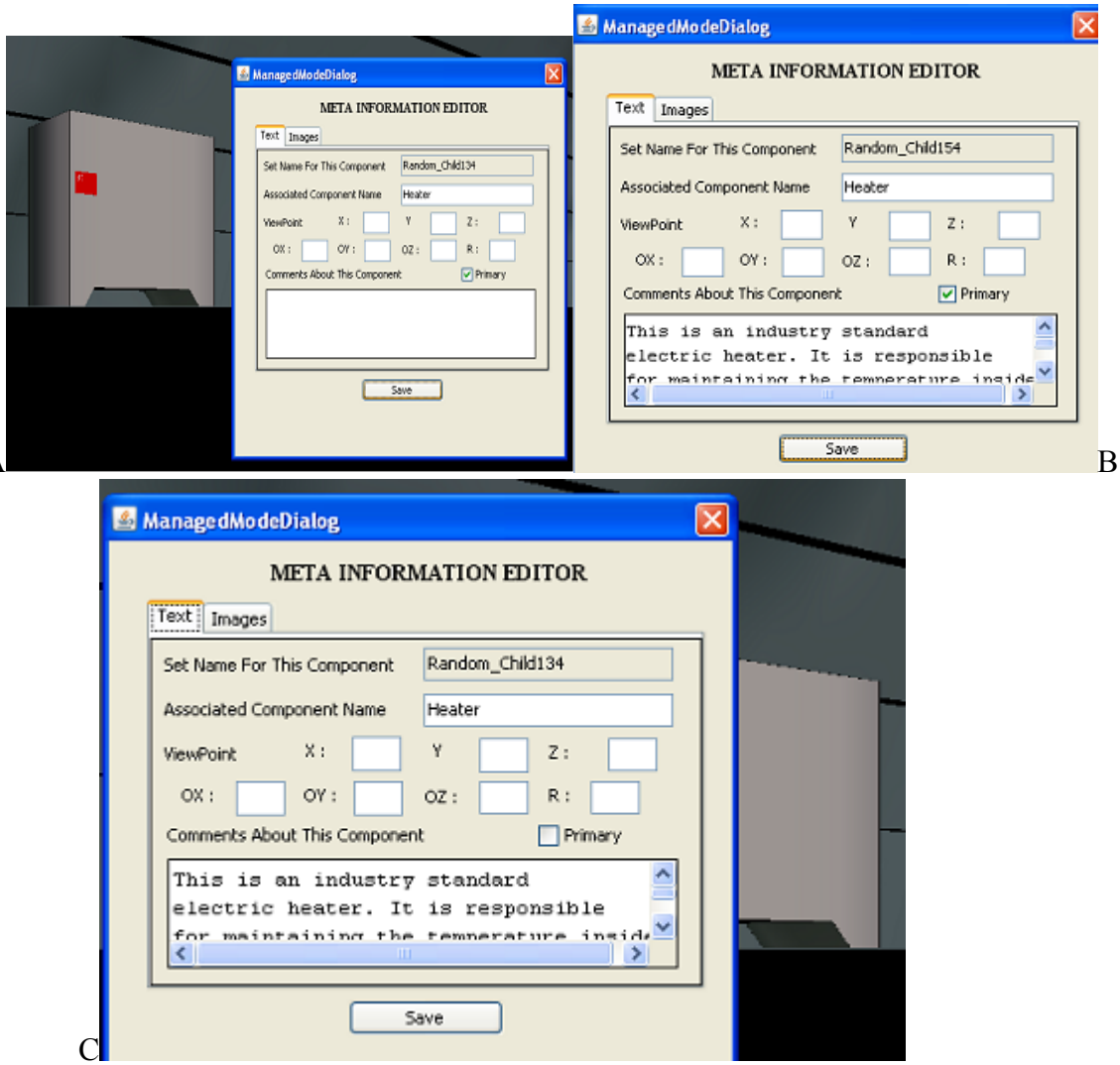


Figure 4-7 Aggregating shape nodes under a common user defined name. A) Shows the front face of the heater carrying the VRML DEF name "Random_child134" and the user name "Heater". B) Additions being made to the comments section of the side face that has a different DEF name (Random_Child154) but the same user name. C) Front face reflecting the changes made to the meta information of the side face since they are tagged with the same user name

must have been added as a part of meta details. When the user enters a user defined name in the search dialog, the meta information map is searched for the name and the viewpoint details for that node are extracted from the data. Then the set name corresponding to this associated name is found from the map. The hashtable of viewpoint nodes which was created during the loading stage is searched using this set name as the key and the node relevant to the searched object is

obtained. The viewpoint position in the viewpoint node is set based on the meta information and its "set_bind" EventIn is set to true. This results in the viewpoint of the shape node being bound to the scenegraph thereby re-positioning the user orientation.

Energy Estimation

There are two basic equations used to calculate energy flow through a greenhouse. One is for conduction, the transfer of heat through the structure of the greenhouse, and the second is for convection, the transfer of heat by mixing warm greenhouse air with cooler outside air. When calculating the heat loss for any greenhouse structure, both of these equations are taken into account, at least to some degree. The particular equations used are from [13], and are outlined below.

To calculate conduction over any greenhouse surface, four values are used; the temperatures on either side of the surface (usually the temperature set point of the greenhouse and the current temperature outside), the area of the surface, and a constant known as the U-value of the surface material. The U-value represents a typical amount of heat energy that will cross a certain area of the material, per degree difference between the inside and outside temperatures.

The amount of heat per hour that is lost to that surface can be found using the equation

$$Q = u * A * \Delta T \qquad \text{Equation 4.1}$$

Table 5.1 shows different uValues for different materials that the model supports. For example, if a small greenhouse was set to 65ºF and the outside temperature was 45ºF, a 100 ft^2 sheet of glass with a U-value of 1.13 BTU/ft^2 -ºF-hour, then the amount of heat lost over the sheet of glass would be: (65ºF - 45ºF) · 100 ft^2 · 1.13 BTU/ft^2 -ºF-hour = 2260 BTU/hour. To calculate convection for any particular greenhouse, four values are used; the temperatures on the inside and outside of the greenhouse (the difference being ΔT), the volume of air within the

greenhouse (V), and a correction factor ($C_{p)}$ representing the relative leakiness of the greenhouse structure. An estimate of heat lost can be obtained using the equation

$$Q = V * C_p * \Delta T \qquad \text{Equation 4.2}$$

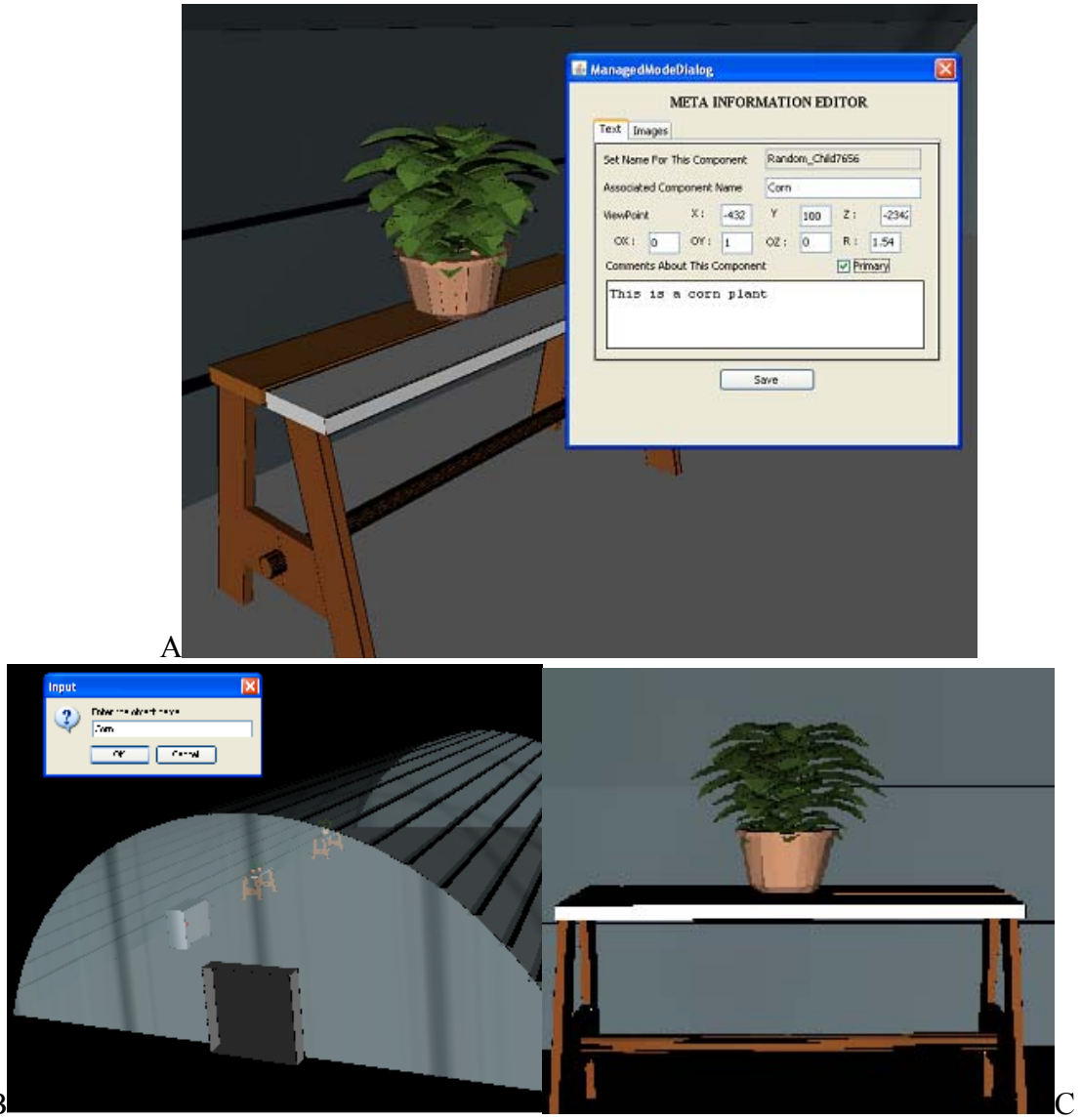


Figure 4-8 Search demonstration. A) The user labels the 3D plant object with the user name "corn". B) Shows the user searching for the corn plant. C) Viewpoint is reset to point to the corn plant

For example, if the temperature outside was 45ºF, the temperature inside was maintained at 65ºF, the volume of the greenhouse was 5000 ft^3 and the greenhouse was assumed to have an

average 'leakiness' (correction factor = 0.0216 BTU/ft^3-ºF-hour), the amount of heat lost per hour due to convection would be (65ºF - 45ºF) · 5000 ft^3 · 0.0216 BTU/ft^3-ºF-hour = 2160 BTU/hour. Calculations assume average leakiness and always use the correction factor value of 0.0216 BTU/ft^3-ºF-hour. To find an estimate of the heat energy lost for the entire structure, conduction is calculated for each wall, each portion of the knee wall, and for the roof, and convection is calculated for the whole structure. Adding the results together gives us the estimate. The values used in the calculation of area/volume of the greenhouse are the structural parameters that the users specify through the interface like length, width and height. A synthetic database of external temperatures, one value per hour over a period of one year, provides the external temperature needed in the calculations. A synthetic database of expected solar radiation was used to get an estimate of how much heat the greenhouse will gain from the sun. The amount of light energy hitting per square foot of ground space was multiplied by the total footprint of the greenhouse to get the estimated energy gain from the sun.

Using the above calculations the net amount of heat energy leaving or entering the greenhouse is calculated. This is simply done by adding the total heat lost to conduction and convection to the heat gained by solar radiation. If the result is negative, i.e., the greenhouse is cooling off, the number of BTUs lost is assumed to be the number of BTUs added to the greenhouse by its heaters, to keep it at its current temperature. If the number is positive, it is assumed that the greenhouse is being cooled somehow and that no fuel is being burned to heat the structure.

Comparison Mode

The user loads an existing greenhouse or specifies the structural properties for a new one. Then when a comparison is run with another greenhouse, cost computation is performed for both the greenhouses. A 3D model corresponding to the greenhouse type is pulled up. When the user

clicks on a component, energy lost form that particular component over each time frame is tabulated for both the greenhouses.

Table 5-1 Uvalues

MATERIAL TYPE	U-VALUE
GLASS	1.13
PLASTIC	1.15
DOUBLE LAYER PLASTIC	0.70
CONCRETE BLOCK	0.51
POURED CONCRETE	0.75
INSULATED CONCRETE	0.13

Simulation mode

Clicking on simulate in the visualization menu loads up a 3D model of the greenhouse. The users can walk into the greenhouse and see the status indicator on the heater change at run time based on several factors such as the external temperature, the internal temperature or the solar radiation. The indicator goes green when the heater gets switched on and goes red when the heater gets switched off. In addition the digital thermometer on top of the heater updates once every few seconds showing the internal temperature in the greenhouse. It can be observed that the heater gets switched on when the internal temperature in the greenhouse falls below the set temperature that was specified as a part of the greenhouse design. The users can also click on any part of greenhouse and get energy loss statistics corresponding to that part.

Methodology

The 3D object representing the light bulb on the heater is tagged with the name “Heater”. The digital text that shows the internal temperature in the scene is a 2D text node. The text node is tagged with the user defined name “Thermometer”.

During the simulation, at every time step, the internal temperature is calculated for the next time step using the equations described in the earlier. The 2D text node with the name

"Thermometer" and the shape node with the name "Heater" are extracted from the scene. The 2D text node is updated with the current internal temperature by calling the "set_text" eventIn of the text node. Then based on the heater's state, the Heater node's color is update by calling the "set_color" eventIn of the shape node. The external temperature data used is from a synthetic dataset that provides external temperatures for every 15 minutes over a period of 24 hours. 2 seconds in the simulation is equivalent to 15 minutes in real world. Figure 4.9 shows the simulation.

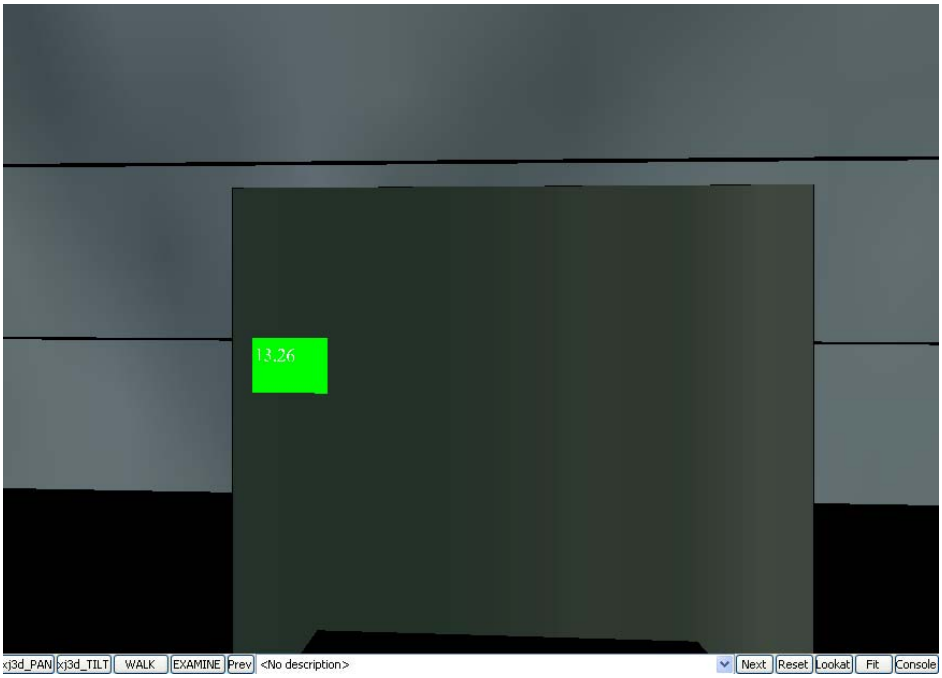


Figure 4-9 Simulation with the heater showing the internal temperature in the greenhouse.

CHAPTER 5
CONCLUSIONS AND FUTURE WORK

Walk through simulations play a useful role in the understanding of design. Previous methodologies adopted in building a virtual greenhouse focused on either providing just a 3D model or on computer based simulations that captured property changes in the greenhouse due to various factors. This research focuses on meeting both the objectives by coupling the 3D model with its information source. Lyra, an ontology management system, was used as the data store. Xj3D was used as the VRML browser and toolkit. Using Lyra helped in organizing domain specific information by using ontologies and these ontologies provided conceptual maps to which the 3D models and other meta information sources were attached. Marrying 3D shapes with domain specific ontologies transforms the static 3D scene which can be used for visualization alone into a semantically rich 3D environment with which multiple users can interact, even from a remote location, by learning from existing information sources or adding more meta information. The 3D visualization also improves the search experience by allowing the user to change viewpoint from the current location to the destination. This creates an experience that is closer to the real world than navigating static web pages using hyperlinks.

A simulation of energy loss from the greenhouse was developed. The 3D visualization enables the user to see the energy loss from each part by clicking on it. Comparison between greenhouses that have the same structure or comparison of a greenhouse configuration with its older version helps isolate areas of energy loss.

Ontology based search

Lyra being an ontology management system supports defining ontologies as we already discussed. Using Lyra to store the meta information and the 3D models opens up several possibilities related to searching the model. The VR environment can be considered as projecting

of objects described by the ontology into a 3D visualization environment. The ontology provides a way of describing object properties and behaviors, describing object taxonomies, and, using ontology reasoners, automatically classifying and cluster objects into categories. The ontology acts as a dictionary that defines the meaning of concepts, and provides support for natural language references to objects.

Ontology reasoners compare the structures of objects to determine how they are the same or different. They can automatically classify new concepts and build categories by grouping together similar objects. This capability leads, among other things, to query processing, as a query can be classified in order to identify objects related to the query (e.g. find all instances that satisfy the constraints of a query). In VR environments, such reasoning processes have many applications. For instance, a user could pose a query such as “Show me all free standing greenhouses in this database”. The reasoner would infer that Quonset, arched and gabled style greenhouses are all free standing structures. It would then get the instances of each class type and display a list showing the names of each instance to the user. The user could then select the greenhouse of his/her choice and open up its configuration. Another example would be a query like “Show me a plant that is diseased”. The reasoner sees that there is an instance which “refersTo” the “powdery mildew on tomatoes” instance in the domain ontology. The reasoner infers that powdery mildew is a type of tomato fungi which in turn is a type of tomato disease and it hence classifies this instance as a diseased tomato variety. It gets the user specified name for this diseased instance and loads it up from the database. It gets the viewpoint data and repositions the camera to point to the 3D shape that represents the diseased tomato plant.

Natural language Support

Support for natural language queries provided by the ontologies enables objects to be named or referenced in many different ways. Multilingual naming and facilities for managing

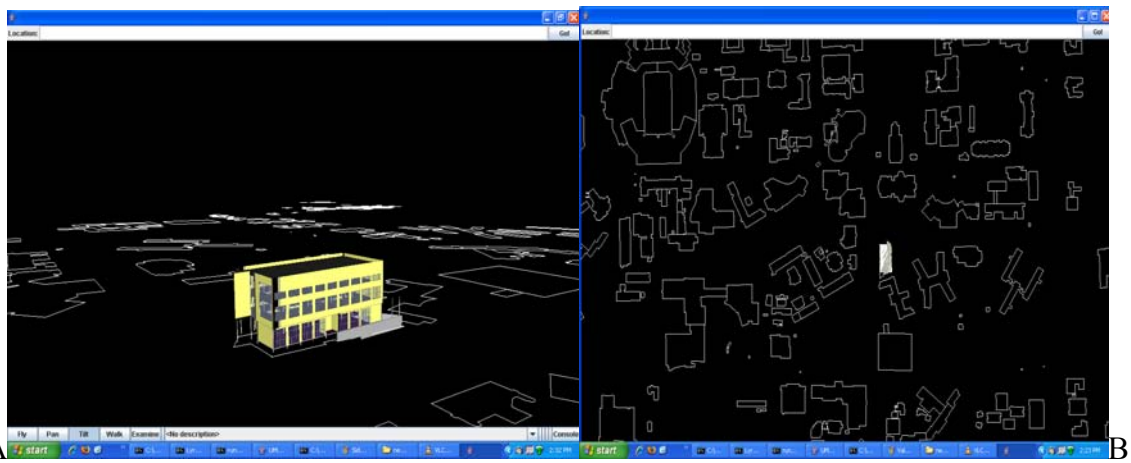
synonyms and homonyms enabled the same object to be named and referenced many different ways. The Lyra OMS includes facilities for creating dictionaries and storing phrase patterns needed to process more complex definite descriptions in natural language. An example of this would be the way the user interface renders the comments and meta information to the user. Based on the user requirements and locality, the UI could leverage the support provided by Lyra to store the textual description in multiple languages.

Geo referencing the 3D model

Most of the virtual tours and virtual greenhouses on the Internet contain information about where the real world peer is physically located. As illustrated in the literature survey, the Virtual Field Day project uses a Flash Object to show the spatial relationship between different greenhouses, ponds and fields. Support for georeferencing a greenhouse can be done relatively easily.

ESRI, a company that specializes in GIS technologies uses Shape files (.shp) to represent geospatially referenced maps [23]. A parser that works off the shape file has already been implemented. The parser is similar to the VRML parser in its functionality. It works on the shape file and stores the geographic information in Lyra. The data can then be used to generate a flat 2D map of the locality. The 3D model can be transformed, scaled and placed at its coordinates. Thus, the user can actually see the georeferenced map and the 3D model placed on the map giving a better viewpoint of the landscape.

An example of this is the ISAS project[18]. A geospatially referenced campus map was imported into the OMS by parsing ESRI Shape Files and a detailed 3D model for one of the buildings at the University Of Florida (Rinker Hall) was positioned at its coordinates. A similar technique could be used for positioning greenhouses in real environments, such as an existing nursery operation.



A B

Figure 6.1 - Images showing geo-referenced Rinker Hall. A) Front view B) Top View shows the entire map

Exporting the 3D model to other formats

Storing the 3D model in Lyra makes it possible to export it into a variety of other formats since Lyra provides a clear separation between content and presentation. For instance, the 3D coordinates in the form of attribute vectors in Lyra can be exported to a VRML file or an X3D file. The exporter would be a software component that operates within Lyra. The target file format is created by the exporter module and this makes it possible to plug-in different exporters to Lyra to output the 3D content in different file formats.

Thus, the virtual greenhouse system developed can be expanded to accommodate several such future extensions very easily.

LIST OF REFERENCES

[1] Stacy , J. & Shubat, D. & Anna, T. (2003).Greenhouse Virtual Tour. University Of Minnesota. http://www.d.umn.edu/biology/greenhouse/VirtualTour/VirtualTourOfUMDGreenhouse.htm

[2] IFAS. (2006). Virtual Field Day. University Of Florida. http://vfd.ifas.ufl.edu/

[3] Tignor, B. et al. (2007). Greenhouse Simulator. University of Arizona. http://ag.arizona.edu/ceac/wge/simulator/greenhouse_content.html

[4] Tignor, B. et al. (2007). Worldwide Greenhouse Education. University of Arizona, University of Florida, University of Vermont, Ohio State. http://www.uvm.edu/wge/index.html

[5] Frantz, J. & Buckingham, L.A. & Locke, J.C. & Krause, C.R. (2006). Virtual grower. http://www.ars.usda.gov/services/software/download.htm?softwareid=108

[6] T. Honjo & E. Lim. (2002). Visualization of Environment by VRML. in Proceedings of the World Congress of Computers in Agriculture and Natural Resources.

[7] Aono, M. & Kunii, T. (1984). Botanical Tree Image Generation. IEEE Computer Graphics & Applications.

[8] VRML. (1995).Virtual Reality Modeling Language. World Wide Web Consortium http://www.w3.org/MarkUp/VRML/

[9] ECMA. (1999). ECMA Scripting Language Specification. ECMA-International. http://www.ecma-international.org/publications/standards/Ecma-262.htm

[10] Marrin , C. (1997). External Authoring Interface Specification. SGI. http://graphcomp.com/info/specs/eai.html

[11] Xj3D (2006). Xj3D VRML Toolkit Specification. http://www.xj3d.org/

[12] Beck, H.W. (2005). Lyra Ontology Management System. University of Florida. http://orb.at.ufl.edu/ObjectEditor

[13] Aldrich & Bartok. (1994). Greenhouse Engineering, NRAES publication 33

[14] Beck, H.W. (2005). Web Taxonomy Browser. University of Florida. http://orb.ifas.ufl.edu/index.html

[15] R. A. Bucklin. (2007). Physical Greenhouse Design Considerations. IFAS Extension. University of Florida. http://edis.ifas.ufl.edu/CV254.

[16] Ross, D.S. (2004). Planning and Building a Greenhouse. University of Maryland Cooperative Extension Service Fact Sheet 645. http://www.wvu.edu/~agexten/hortcult/greenhou/building.htm

[17] R. A. Bucklin. (2002). Florida Greenhouse Design. IFAS Extension. University Of Florida. http://edis.ifas.ufl.edu/AE016.

[18] Oliverio,J. & Masakowski, Y.R. & Beck, H.W. & Appuswamy, R. (2007). ISAS: a human-centric digital media interface to empower real-time decision - making across distributed systems. In the proceedings of 12th international conference on 3D web technology.

[19] Bartok, J.W. (2005). University of Connecticut http://www.umass.edu/umext/floriculture/fact_sheets/greenhouse_management/jb_building_gh.htm

[20] SL. (2007). Second Life. Linden Research. http://secondlife.com/

[21] Smajstrla, A.G. & Zazueta, F.S. (2002). Evaporation Loss During Sprinkler Irrigation. IFAS Extension. University of Florida. http://edis.ifas.ufl.edu/AE048

[22] Java Swing (2007). Sun's Java VM. http://java.sun.com/

[23] SHP File Format (1998). ESRI SHP File Format. www.esri.com/library/whitepapers/pdfs/shape**file**.pdf

BIOGRAPHICAL SKETCH

Raja Appuswamy received his Master of Science in Computer Engineering and Agricultural and Biological Engineering from University of Florida in Fall 2007. He worked with Dr.Howard Beck and professor James Oliverio on ISAS and published a paper titled “ISAS: A Human-Centric Digital Media Interface to Empower Real-Time Decision-Making Across Distributed Systems” in Proceedings of the twelfth international conference on 3D web technology.He pursed research on Virtual Greenhouses for his Masters thesis under Dr.Howard Beck and Dr.Ray Bucklin. Raja received his Bachelor of Engineering degree in Computer Science from Anna University, India in 2005