ABSTRACT

Three-dimensional computer graphics can be extremely expressive. With a correct layout design, people can quickly and easily comprehend large amount of information. This is especially true when a group of people works simultaneously on interpretation of the same information. Both industry and science have already realized the synergy of such a combination and started engaging many resources in order to build environments that stem from the field of Computer Supported Cooperative Work (CSCW). The objective of this thesis is to present one of possible approaches to 3D Collaborative Environments issue. This approach has been used in building **3D GIS** – a system developed at Northeast Parallel Architecture Center (NPAC). The thesis also summarizes requisite capabilities of applications from two different domains of computer science: Geographic Information Systems (GIS) and Distributed Virtual Environments (DVE). It attempts to briefly analyze various features provided by environments, which manifest some degree of similarity to our system. The core of the paper discusses the functionality, architecture and application of our system in generic high-end visualizations with elements of discrete event simulations.

FUNCTIONALITY AND ARCHITECTURES OF 3D COLLABORATIVE ENVIRONMENTS

BY

PAWEL ROMAN

MASTER'S THESIS

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE IN COMPUTER SCIENCE IN THE GRADUATE SCHOOL OF SYRACUSE UNIVERSITY

December 1997

Approved _____
Professor Marek Podgorny

Date _____

UPDATED: 12/21/97

COPYRIGHT 1997 PAWEL ROMAN

ALL RIGHTS RESERVED

ACKNOWLEDGEMENTS

The author would like to thank Remek Trzaska, GIS team co-member, for fruitful cooperation, Lukasz Beca and Tomasz Jurga, TANGO core team, for help during integration phase, and Grzegorz Lewandowski for assistance in pursuing ideal solutions to not-so-ideal problems.

Special thanks to Mr. Marek Podgorny for providing inspiration and supervising the project.

Described system, 3D GIS, was partially supported by the grant of the Rome Laboratory in the Collaboration and Interactive Visualization project.

TABLE OF CONTENTS

I. Introduction ____ *1*

II. Anatomy of GIS ____ *3*

1 What exactly is a GIS? ____ **3**

2 GIS Subsystems ____ **4**

3 DATA models ____ **5**

3.1 Spatial Data ____ 6

3.2 Attribute data ____ 6

III. Distributed Virtual Environments (DVE) ____ *8*

IV. Review Of Existing 3D Environments ____ *11*

4 MWorld ____ **11**

4.1 System Architecture ____ 12

4.2 3D Environment ____ 13

4.2.1 Deformable Objects ____ 14

5 CSpray ____ **16**

5.1 SPRAY ____ 16

5.1.1 Cans ____ 17

5.1.2 Sparts ____ 17

5.2 Collaborative Features ____ 18

5.2.1 Session Management ____ 18

5.2.2 Public Window and Eyecons ____ 18

5.2.3 Private and Public Spray Cans ____ 19

5.2.4 Floor Control ____ 19

5.2.5 Data Sharing ____ 19

6 Virtual GIS ____ **20**

6.1 System Architecture ____ 21

6.1.1 Data Set Pre-Processing Component ____ 21

6.1.2 Visualization Component ____ 21

V. 3D Geographic Information System (3D GIS) ____ *26*

7 Project Background ____ **28**

8 High-level requirements ____ **29**

9 Core Technologies ____ **30**

9.1 Open Inventor ____ 30

9.1.1 Textures ____ 32

9.1.2 Actions 33
9.1.3 Interchange File Format 34
9.1.4 Sensors 35
9.1.5 Engines 35
9.1.6 Draggers and Manipulators 36
9.2 Adaptive Communication Environment 37
9.3 Standard Template Library 40
9.4 Xt Intrinsics And Motif 41
9.5 TANGO 41
9.6 TANGOsim 42

10 Functionality 43
10.1 Viewer 43
10.1.1 Multiple Data Representation 43
10.1.2 Synchronized and Desynchronized Views 44
10.1.3 Operating in the World 45
10.1.4 Hot Spots and Pickable Objects 45
10.1.5 Draggable Objects 46
10.1.6 Database Support 47
10.1.7 Data Downloading 48
10.1.8 Discreet Event Simulation 48
10.1.9 Level of Detail 49
10.1.10 Wavelet Compression 50
10.1.11 Triangulation and Mesh Simplification 50
10.1.12 World Description File Format 51
10.1.13 Dynamic World Description 52
10.1.14 World Extensions 53
10.1.15 Stereoscopic Rendering 53
10.2 Control Center 53
10.2.1 Viewer Management 54
10.2.2 Floor Control 54
10.2.3 Overall World Representation 54

11 Architecture 55
11.1 Viewer 57
11.1.1 Main Class 57
11.1.2 Synchronization 59
11.1.3 Protocol Framework 61
11.1.4 Static and Moving Objects 66
11.1.5 Object Follow-Up Mode 69
11.1.6 World Representation 69
11.1.7 Tag File Format 71
11.1.8 HTTP Communication 74
11.1.9 DATA Caching 75
11.1.10 Database Information Retrieving 76
11.1.11 Inline Nodes 77
11.1.12 Cooperation with TANGOsim 77
11.1.13 Viewer Protocol 79
11.2 Control Center 84
11.2.1 Manager Class 85
11.2.2 TANGO Communication 86
11.2.3 Viewer Communication 87
11.2.4 Control Protocol 87

VI. Conclusions ____ *90*
VII. Bibliography ____ *94*
VIII. Appendix ____ *96*
IX. Vita ____ *102*

LIST OF ILLUSTRATIVE MATERIALS

Figure 1: mWorld architecture 13
Figure 2: mWorld screenshot 15
Figure 3: VGIS architecture 20
Figure 4: Continuous level of detail for four different threshold values 24
Figure 5: The ACE components 38
Figure 6: Data transformation process 51
Figure 7: Overall system architecture 56
Figure 8: Viewer class hierarchy 58
Figure 9: ProtocolInterface class family 62
Figure 10: Interactions within ProtocolInterface class family 66
Figure 11: Moving objects class family 68
Figure 12: Cooperation with discrete event system extension TANGOsim 78
Figure 13: Control Center Architecture 85
Figure 14: Example of CSpray collaboration 96
Figure 15: VGIS. Textures in different level-of-details. 97
Figure 16: VGIS. Battlefield simulation. 98
Figure 17: 3D GIS. Two viewers displaying map and satellite image of Syracuse area 98
Figure 18: 3D GIS. Denver area in closer look with maximum level-of-detail 99
Figure 19: 3D GIS. Discrete event simulation extension 100
Figure 20: 3D GIS. Multiple data representations 101

I. INTRODUCTION

The rapid advances in computer hardware technology in the Nineties have lead to powerful machines with completely new performance capabilities in both computation and graphics domains. Secondly, the growth rate of the Internet exceeded even optimistic predictions and today we can perceive the Internet as an almost household technology. These two factors have established firm foundation for a new kind of software – 3D Collaborative Environments.

Currently there is much effort all over the world engaged in development of such environments. There are several very interesting projects addressing various real-world problems ranging from simultaneous engineering through scientific visualization, remote sensing and military simulations. There is also another trend in that field, which puts emphasis on human interactions and exploits notions like an avatar, subjectivity or 3D socializing. This trend seeks to build Distributed Virtual Environments (DVE) that immerse a user in virtual world and allow him/her to explore it, interact with its entities as well as with other users. Whatever is the focus of the particular environment, they all manifest very similar characteristics, namely, they allow people located geographically apart to share and operate on data among themselves.

This thesis summarizes achievements in development of **3D Geographic Information System** that was initially designed to provide near real-time fly-through terrain visualization

and additional geographic information in the form of thematic layers. However, over the project stretch when new features has been added, the system evolved into a more generic 3D visualization environment and the previous classification is not as valid as it was at the beginning of the project. Today the system possesses some interactivity encountered in distributed virtual environments like draggable objects, it supports collaborative visualization, shared state, discrete event simulation extension and strong database support. 3D GIS has been optimized to efficiently render very complex data sets and to have moderate bandwidth demands. It has been used in visualizations ranging from human organs through electromagnetic airplane profiles. However, the terrain rendering is still the primary objective of our system and therefore we have never changed its name.

It is worth mentioning that our case is not an unusual one in comparison with other projects. Many of them started from relatively small set of requirements, applied evolutionary development process and now they try to generalize their software.

The thesis starts with presentation of components of traditional GIS, then moves towards Distributed Virtual Environments. Next, we attempt to present functionality and architectures of several visualization systems in order to show their approach and how they resolve recurring issues. Eventually, we discuss our system in greater detail and present some design decisions.

II. ANATOMY OF GIS

Since our system manifests many features of traditional Geographic Information System and can be applied wherever a traditional GIS is used, this chapter highlights the nature of such systems and introduces some terminology.

1 WHAT EXACTLY IS A GIS?

The United States Geological Survey (USGS) defines a Geographical Information System (GIS) as:

> *a computer system capable of assembling, storing, manipulating and displaying geographically referenced information, i.e. data identified according to their locations.*

This definition, however precise, does not emphasize complexity of such a system. A modern GIS is a set of heterogeneous technologies usually integrating traditional and digital, Web- and non-Web-based data acquisition, database back-end and high-end visualization hardware and software. This is no longer a single computer loaded with a software handling a proprietary data format and providing static information on a selected area. There are several initiatives struggling over the openness of GIS, like for instance The Open GIS Consortium that tries to involve developers and users of geographic information resources

in collaborative development of interoperable technology specifications. Computation power needed by such a system forces highly distributed solutions scattering the system literally over wide area and as one could expect in the Information Age the involvement of the Internet is significant.

2 GIS SUBSYSTEMS

We can approach GIS from functional perspective and partition it on four subsystems:

- *Data input subsystem* responsible for gathering information from various resources and converting this information to digital format of spatial and thematic data.

- *Data storage and retrieval subsystem* that manages the data. It should provide quick and convenient access to the information and also allow making necessary updates easily and whenever it is needed. This subsystem involves usually use of a database management system (DBMS) for governing the data. Unfortunately, not many database systems support spatial queries, thus encouraging developers to encode geo-referenced data using proprietary formats. There are, however, add-ons like Spatial Data Blade, a module provided by Informix as part of the Illustra Server that handles spatial queries quite neatly.

- *Data manipulation subsystem* that executes spatial and attribute procedures in order to derive more information from the acquired data. This process can be very complex and involve many different fields of computer science, but also some effects can be achieved by filtering or indexing the data according to several criteria. For example, it

can generate business information catalogue from publicly available Web pages that are already stored in the database.

- *Data presentation subsystem* responsible for graphic visualization of gathered information. It should allow the user to conveniently examine the output and to interact with visual objects providing hierarchical access to the detailed information. We could even expand the system to support dynamism in data visualization so that complex processes as weather could be presented.

3 DATA MODELS

GIS uses simple yet powerful concept of thematic layers as the way of storing information about the world. We group the same type of data, put it on a layer and then layers can be superimposed on the world providing significantly more information than the shape of the terrain alone. The real advantage of such an approach is that we can analyze several layers, meaning several different aspects, at the same time.

The GIS data can be broken into two categories:

- *Spatial data* that characterizes geographic features by use of absolute and relative locations.

- *Attribute data,* additional quantitative or qualitative type of data.

The following subsections analyze the nature of these two kinds of data.

3.1 SPATIAL DATA

Spatial data is a result of mapping a real world to its digital artifact. It should represent the world as accurately as possible bearing in mind storage and performance tradeoffs. Spatial data can be stored in either vector format, raster format or image data. Vector data format describes the world in terms of vertices and arcs. It is important to retain spatial relationships of the features, therefore the adjacent information is explicitly joint so that it forms a polygon mesh. Raster data format, on the other hand, uses grid-cell structures that form regular grids of tessellated data. An example of such a structure is a quadtree[3]. Usually, there is no explicit coding of geographic coordinates in these kinds of structures, since they can be easily calculated based on matrix-like nature of a grid. The important issue in using grids is the size of the cell, since it determines accuracy of the represented world and cannot be changed. However, it is possible to create hierarchical structure of nested grids achieving a level-of-detail capability. The data format should be chosen with special care depending on data sources and planned use of the data. For example, the raster model is well suited in handling of continuous data as elevation, while vector model does well in linear data analysis. Third format, image data, can store remotely sensed imagery as satellite images, scanned maps or a certain process and feature visualization like Fe density or rainfall data. Format of this kind of data is usually proprietary, yet widely used, like GIF of JPEG.

3.2 ATTRIBUTE DATA

Attribute data is the essence of GIS and there is no limit on categories of that data. However, it is not a trivial task to provide intuitive user interface so that information retrieval could be effortless. The GIS system itself can store this data internally, though it is

not especially practicable since the storage of voluminous data is not a straightforward piece of work. Therefore, the most common solution is to let a database management system do the work. The choice of DBMS is quite essential and should be done carefully. There are many database systems on the market, the prevalent Relational Databases (RDBMS) does not fully address the needs of GIS, whereas new generation of Object-Oriented Databases (ODBMS) still stumbles against performance issues. Some sort of solution is provided by Object-Relational Database Management Systems (ORDBMS). They support object-oriented features as complex data objects, data behavior, inheritance and overloading along with well known from relational systems flexibility, security, transactions, recovery, SQL parsing and optimizations.

III. DISTRIBUTED VIRTUAL ENVIRONMENTS (DVE)

Even if our system was never intended to be a Distributed Virtual Environment, we find it very interesting to briefly present selected ideas from that field. The two domains, Geographic Information Systems and Distributed Virtual Environments are rather very distant, but our system expanding the notion of GIS could be perceived as an attempt to narrow the gap between them.

Distributed Virtual Environments form a new generation of computer technology. Based on the experience of Virtual Reality systems, they try to immerse users in synthetic worlds so that users can interact with every entity in the world regardless whether it is completely artificial or it has its counterpart in the real world. The user is represented by *an avatar*, three-dimensional icon that either resembles more or less user's physical appearance or it is entirely a product of his/her imagination and creativity. By controlling the avatar the user can move in the world, talk to other participants, modify the world and many more. Since everything is synthetic there is no limit of user's activities.

DVEs encounter and try to resolve many problems that have been often neglected by 'strict' computer science. They care about being an extension of user interactions and providing appealing worlds populated by thousands of avatars, hence these environments are

often called *social software* and as such they have not only to overcome technological limitations, but also to deal with human perception factors.

One such factor concerns communication efficiency in virtual environments, which can dramatically drop because of failing to provide an experience similar to face-to-face encounter. This deficiency is hoped to be alleviated by introducing *collaboration awareness* that eliminates degradation of participants to external agents, whose presence is merely implied. It could be achieved by providing minimal amount of information about other participants: their names, current interest, position and possibly more. There are several approaches to that issue; low-bit rate video stream superimposed on the surface of the avatar, separate still images or video channels to name a few. Whatever is the actual technique it should allow sensing other participants and boosts proximity between them.

Although it is generally desirable that all participants perceive the virtual world in the same way, some researchers argue that there are situations in which allowing each participant to tailor her view, thus providing *subjectivity* in the perception of the world, could augment collaborative visualization. It necessitates division of the visualized world on non-visible (objective) and visible (subjective) features. In such a situation the shared state, that must be coherent among all participants, encompasses solely objective features, whereas subjective features are allowed to vary and to be user-dependent.

There are many more interesting concepts associated with Distributed Virtual Environment they, however, will not be discussed as exceeding the scope of the thesis. What we would like to achieve by introducing this short chapter is the basis for clarification on

why we can call our system 3D Collaborative Environment and not Distributed Virtual Environment despite some degree of similarity. We will attempt a comparison of these two types of environments in chapter VI, after presenting our system in greater detail.

IV. REVIEW OF EXISTING 3D ENVIRONMENTS

This chapter makes an attempt to extract key features from several projects in order to formulate a set of essential capabilities of a 3D environment. We mainly focus on functionality of presented systems since the architectures are usually not sufficiently documented. However, where it was possible we at least outline the components of the described systems in order to signal the underlying design.

4 MWORLD

mWorld[16] has been built by Associação para o Desenvolvimento das Telecomunicações e Técnicas de Informática (ADETTI), a research center aiming at the development of information technologies. The system consists mainly of one distributed application that allows several users to share the same world space, where they can add, modify or delete visual objects as well as navigate in the space. In addition, the system also supports audio conferencing tool.[1]

[1] However, we could not access the paper describing this tool in more details.

mWorld is based on the Joint Editing Service Platform (JESP) also developed by ADETTI, which was designed to provide session control and data communication in cooperative environments. Since the system does not rely on centralized architecture, where application resources are concentrated on server side, but rather replicates these resources at all sites, the point-to-multipoint connections are used. They are, however hidden from the applications and handled directly by the communication services. Session control mechanism is responsible for floor control, admitting new members and exception events handling. Floor control is based on explicit token passing mechanism, so that only the participant that owns the token can send messages, others are limited to listening at that time. Token granularity in this scheme is at the level of the whole world, therefore it is not possible to lock one object and let the other objects to be modified. When a new member wants to join an existing session, he/she has to be explicitly authorized and initial state information must be sent. Owing to the lack of central server other participants have to initiate a new connection upon new user's admission. Since this architecture relies on token passing, session control module must handle exception events. Especially dangerous situations arise that due to failure of one application the token is lost. In such cases session control have to create a new token in order to prevent disabling the whole environment.

4.1 SYSTEM ARCHITECTURE

The collaboration environment provided by JESP is separated from applications and constitutes a separate level. Each participating application on a particular host communicates with it by means of service access point (SAP). This level consists of two layers: Session Manager and Group Communication and it is depicted in Figure 1. The former is

responsible for session control tasks described in previous section and exchanging application events translated into messages, while the latter takes care of multiple connections using network and transport protocols. Access to the system is regulated by authentication list maintained by every host.

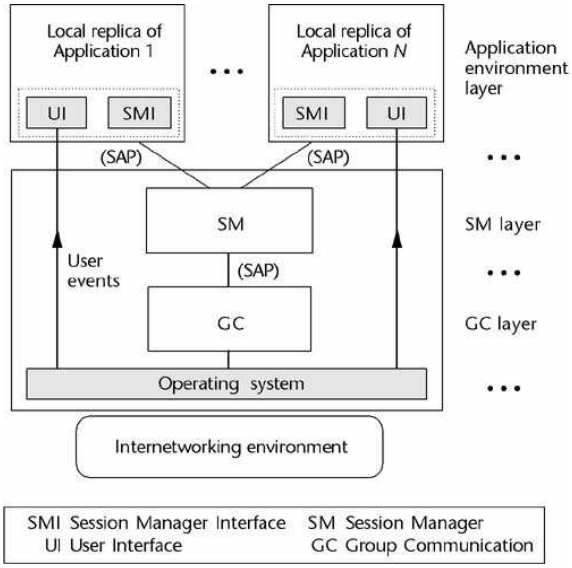

Figure 1: mWorld architecture

4.2 3D ENVIRONMENT

The environment is built on top of the Open Inventor toolkit [for details see sec. 9.1] and allows every user to navigate in the virtual world by changing a camera parameters and also share camera between users, so called *joint navigation*, by sending updates upon parameters' change.[2] In addition, a session initiator is allowed to change the visualized world

[2] Unfortunately, the paper we base on does not explain how users 'jointly' navigate the world. Our understanding is that one user navigates and others just follow her camera.

and all other participants are sent a message with the URL to a new world. The new world is downloaded by initiating another connection without JESP services' help.

A user can modify the 3D world, but only after she has acquired the token, which in this system has to be explicitly granted. World's modifications encompass adding, modifying, picking and deleting objects. Appropriate messages are broadcast to other participants in order to maintain the consistent state of the environment. In addition to changing the content of the world, users are also allowed to alter world's and objects' attributes like light parameters, rotation, scale and translation of the object, its color, texture and so forth.

4.2.1 DEFORMABLE OBJECTS

The system has expanded the Open Inventor class family with *MLBlinker*, a class that supports deformable objects. It is derived from *SoSwitch* node [see sec. 9.1], which enables switching among its children nodes and it is widely used in the simulation of a finite stage movement. In the case of *MLBlinker*, children nodes represent subsequent stages of deformed surface and when switched with appropriate rate, the motion effect is achieved. In order to attain real-time quality these surfaces have to be generated off-line and they are stored as a combination of Open Inventor nodes: *SoIndexedFaceSet* describing topology of simulated surface and *SoCoordinate3* characterizing interconnected particles. The authors report that such a representation is quite memory consuming, but instead it allows eliminating CPU-intensive computations during the rendering phase and therefore decreases hardware requirements for the platform using mWorld environment.

A quite interesting feature related with deformable surfaces is the possibility to delegate the computation of the surface representations to the most powerful host participating in the session so that waiting time for the results is minimized. The token's owner sends a request to other machines for CPU. If the request is granted, the surface, initial parameters and boundary condition are uploaded to the host, which in turn calculates consecutive representation, creates an interim Open Inventor file and broadcast to all participants a message with the URL to that file. Users can then update their world and observe deformations of the surface.

Figure 2: mWorld screenshot. These two flags are examples of deformable objects.

5 CSPRAY

CSpray[17] (Collaborative Spray) developed by University of California at Santa Cruz, is an extension of Spray, a visualization application, into a collaborative environment. The system supports:

- Private and public (shared) virtual workspaces
- Data protection and integrity
- Intuitive user interface
- Extensibility of visualization tools
- Integrated approach to surface, volume and flow visualization.

Before we describe all these features, let us present key ideas associated with Spray, the visualization core of the CSpray. Figure 14 in chapter VIII presents screenshot of CSpray.

5.1 SPRAY

Spray is a framework providing several visualization techniques like surface, volume and flow rendering. It derives its name from metaphor of a spray-paint can and treats datum as an invisible entity that has to be painted during visualization process in order to make it tangible and thus possible to be interpreted. Cans are filled with paint consisting of smart particles also called *sparts*. That paint is spread over the data set to highlight certain features in an incremental manner.

5.1.1 CANS

Cans provide a mean for emitting sparts. There are several parameters by which cans are controlled. They include position, orientation, contents (i.e. spart type), spart density, distribution pattern and nozzle shape. The user has the whole shelf of pre-defined cans and also is allowed to define a new type of a can according to her needs using *Mix & Match* technique.

5.1.2 SPARTS

The notion of smart particles comes from combination of particle systems and behavioral animation. Particle systems have been used in modeling natural phenomena that are hard to be represented by polygons. After setting particle's initial attributes, the particle is emitted into a space where it can evolve over its lifetime by changing the initial values of attributes, moving along a pre-defined trajectory or it can even spawn other particles. On the other hand, behavioral animation allows particles to manifest themselves in terms of interactions with other particles, thus having control over the whole group. For example, we can define the maximum distance between particles and therefore achieve group behavior similar to school of fish.

A spart has a state that consists of position, lifetime, age and trajectory. In addition it has a target and behavior, which are set upon spart's creation. When a spart is emitted, it proceeds according to position updates determined by its trajectory. After each position update it checks if the target has been reached, if so the spart exhibits the pre-defined behavior, usually leaving abstract visualization objects (AVOs) that are displayed in rendering

phase, otherwise it continues to move. A smart particle is destroyed when its age exceeds the spart's life span.

Such a paradigm facilitates implementing many types of visualization algorithms all supported by the same notion of smart particle. For example, we could set a target to be an area 30 meters wide with slope between 15 to 30 degrees, without large objets nearby, like trees and rocks and expect sparts to highlight the best ski trails for a particular terrain.

5.2 COLLABORATIVE FEATURES

5.2.1 SESSION MANAGEMENT

A session in CSpray is a set of connections sharing a virtual workspace. Since this architecture does not provide either content server or directory server, all connections are initiated and maintained by instances directly. This is realized by TCP links between all participants, thus requiring order of n^2 connections in the system. Every instance keeps list of active users and updates it upon their leaving or arriving. CSpray resolves initial state issue by sending to a newly connected participant state of the shared virtual workspace.

5.2.2 PUBLIC WINDOW AND EYECONS

The system supports collaboration awareness by using eye icons also called *eyecons*. An eyecon shows current participant's position, orientation and nickname. These symbols, however, are not displayed in the main user window, but rather a special window called a public window that is devoted for this purpose. It not only informs about other users, but

also provides a mean for synchronization. A user is allowed to select other user in order to see the same part of data set as the selected one.

5.2.3 PRIVATE AND PUBLIC SPRAY CANS

CSpray distinguishes between private and public cans. When a can is private it is only accessible to its creator. Moreover, any results produced by using this can are visible only to the creator of the can. Conversely, all users are allowed to manipulate public cans. In that case, a request to the host of the creator is sent in order to visualize data set accordingly to can's characteristics and broadcast the results to all participants.

5.2.4 FLOOR CONTROL

The system deals with the contention for public cans using floor control mechanism based on traffic lights metaphor. When a public can is ready to use it has green color. If somebody starts using a can, the can turns red for all participants except from the one, who currently owns it. As soon as other participant makes a request to an allocated can, the can changes its color from red to yellow as to signal that other user would like to manipulate it.

5.2.5 DATA SHARING

CSpray provides three levels of data sharing. First level is the least restrictive and it allows sharing 'raw' data set between participants. It yields the fastest visualization process, since only control messages are interchanged and rendering is done locally. However, such degree of access is unacceptable, when the owner of the data wants to protect her research. The second level of data sharing operates on geometric primitives, therefore participants

share visualization effects, which means that other users, not only the owner of the data set, are allowed to manipulate cans and highlight various areas without actual access to the data themselves. Third level base on bitmap interchanging and other users are only granted permission to see visualization effects done by the owner of the data. This level, of course, while the most restrictive is also the slowest in the rendering aspects.

6 VIRTUAL GIS

VGIS has been built in Graphics, Visualization and Usability Center at Georgia Institute of Technology. It provides real-time, highly interactive visualization of spatial data along with raster layers, protruding features, static and moving objects. It is implemented on top of Simple Virtual Environment (SVE) toolkit also developed at GIT, which in turn uses Silicon Graphics OpenGL library. The system has very interesting, modular architecture depicted below that heavily exploits multi-threading as an efficient task separation mechanism.

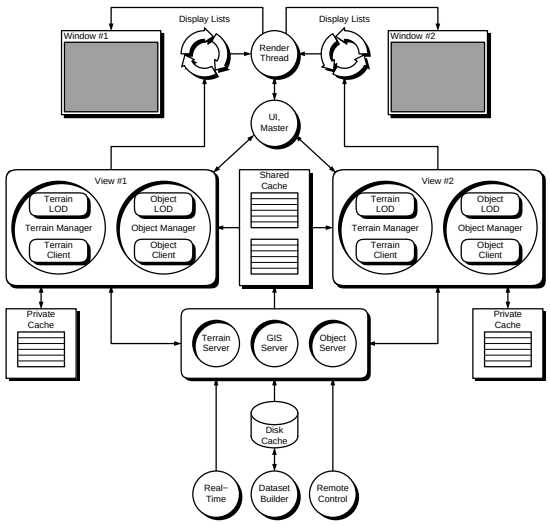

Figure 3: VGIS architecture. Modules are represented by rounded boxes, processes and threads by circles, data structures by rectangular boxes.

6.1 SYSTEM ARCHITECTURE

6.1.1 DATA SET PRE-PROCESSING COMPONENT

There are three levels of data pre-processing in VGIS: off-line transformations from various formats to one, quickly accessible at run-time, intermediate processing performed during data paging process and finally real-time computation required by render thread to display data set.

Since first type of pre-processing is done off-line and only once per data set, it is not a time-critical task and can be carried out arbitrarily long as far as produced output has a convenient format to be rendered very fast later on. This format in the case of VGIS is a quadtree[3]; hierarchical structure, where a top-level node represents the entire data set and each of its four children one-fourth of the parent's area. Leaves in this structure have the highest accuracy of the data set. Such a structure simplifies polygon culling and implementing efficient level-of-detail mechanism described below.

The two remaining levels of pre-processing are done on-line and therefore they have been separated from render thread and run as isolated threads in order not to slow the visualization down.

6.1.2 VISUALIZATION COMPONENT

The run-time part of VGIS is a highly complex multi-threaded application. It has been designed with special care for providing a real-time rendering rate. Therefore, scene updating and scene rendering tasks have been isolated and distributed among two separate threads

with appropriate priorities. For that reason scene manager delivers new update 1-5 times per second while render thread displays 20-30 frames per second. Further tasks allocation was made based on identification of slow pace activities, e.g. I/O intensive paging, and assigning them to lower priority threads.

Servers

Back-end of the component consists of set of specialized servers, which also run asynchronously with respect to the rest of the system. They are responsible mainly for intermediate processing and data paging. Since throughput between them and other tasks is significant, communication with servers relies on shared memory. Typical communication scenario looks as follows. A client module responsible for a view sends a request to a server, the request is put in priority queue and waits for servicing. A server executes the request, which usually encompasses transferring data form a hard disk or over the network and some sort of processing. After a request has been satisfied a server informs view module about completion and hands over reference to the results residing in shared memory.

Caching Subsystem

In order to decrease memory needs of the system, a shared cache mechanism has been implemented. This is especially profitable in independent views, where roughly the same area is displayed using only other camera position and orientation. Every view module has also private cache that is used in storing view-dependent parameters and a structure describing

which parts of the terrain are present in the view and their accuracy represented by level of detail.

Continuous Level of Detail

The goal of high quality real-time visualization can be only accomplished with a proper level of detail algorithm. The VGIS uses regular grid of polygons structured hierarchically in a quadtree and applies two-step simplification algorithm. During the first coarse-grained step discrete blocks are selected and replaced by larger ones depending on the distance from the viewpoint and the angle to the normal of the terrain. Second step of simplification is a recursive triangle substitution based on pre-defined threshold value. Figure 4 presents an area resulting from the application of four different values of threshold to the terrain shape. Such reduced terrain is translated into an OpenGL-like display list and reference to the list is put into a cyclic buffer. Similar simplification affects the terrain textures [see Figure 15 in chapter VIII]. According to the authors such an algorithm allows two orders of magnitude reduction in the number of polygons and a one order of magnitude in the size of textures.

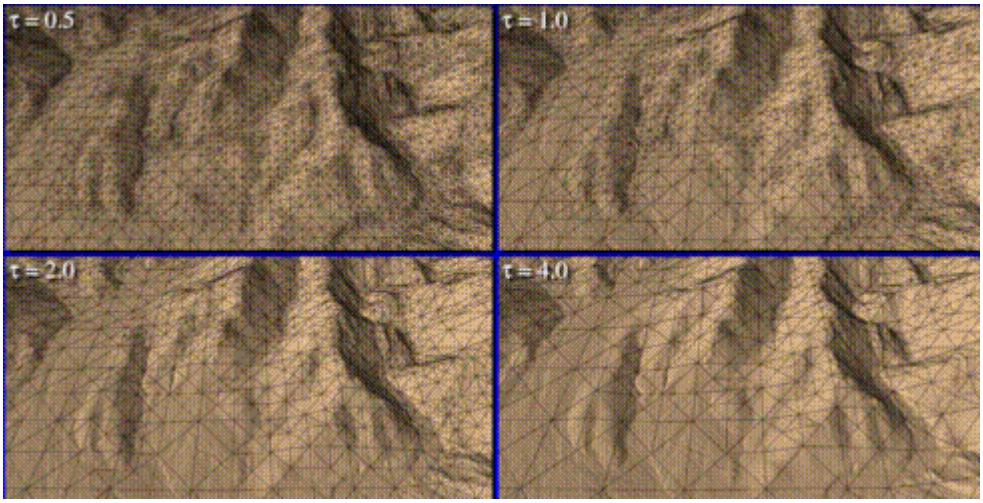

Figure 4: Continuous level of detail for four different threshold values

Visual Object Management and Simulation Capabilities

The system also facilitates adding static and moving objects in order to visualize specific scenarios. Furthermore, such objects can be grouped hierarchically according to the suitable symbology, e.g. military Command Control scheme. Again a separate thread is appointed to handling moving object. It communicates with simulation module by use of Distributed Interactive Simulation (DIS) protocol[4], which reduces amount of data sent over the network by relying on best-effort broadcasting and dead-reckoning for objects. Dead-reckoning enables to extrapolate object's position and transmission is only needed when its position discrepancy exceeds certain threshold. Screenshot of battlefield simulation is presented in Figure 16 in chapter VIII

Navigation

Another thread is responsible for user interaction handling; it ensures certain level of responsiveness to user input. VGIS can be operated either through traditional window-based interface or three-dimensional controller and head-mounted display (HMD). The system supports two modes of navigation:

- *Orbital mode* in which user operates the globe as if she held it in her hand, therefore rotations and zooming is allowed

- *Free flight mode* that permits six degree of movement freedom.

Among other tasks of user interface are resource allocation and thread work coordination.

V. 3D GEOGRAPHIC INFORMATION SYSTEM (3D GIS)

This chapter presents a system developed at Northeast Parallel Architecture Center (NPAC) that was initially designed in order to provide synchronized high-end terrain visualization with near real-time fly-through capability. With the advance of the project it has been, however, expanded to handle multiple kinds of data, not only terrain; discrete event simulation capability, database access, independent views, interactive objects and some more. It no longer resembles a traditional geographic information system but rather manifests some features encountered in distributed virtual environments. Currently the system integrates several technologies and cannot be fully exploited without them. The following list enumerates these technologies and names their role in the environment:

- TANGO: session management and message routing; section 9.5,
- TANGOsim: discrete event simulation; see section 9.6,
- Illustra DBMS: database back-end of the system; see section 10.1.6,

- Viewers: shared state and presentation layer of the system; see sections 10.1 and 11.1,
- Control Center: Viewer management and communication with TANGO; see sections 10.2 and 11.2.

Since the author participated directly in the development of Viewer and Control Center, this thesis will focus mainly on these two components, while others will be described as much as it is necessary in order to present integration issues. Before we highlight the functionality of our system is worthwhile to provide some general information.

There can be many synchronized system instances over the network, each consisting of one Control Center and many connected viewers, participating in many sessions. As it was stated earlier session management and communication between them is maintained by the TANGO collaborative environment. We can distinguish one entity among all system instances participating in the same session, which is called the master; the rest are in the slave mode. Those in the slave mode can either merely follow the actions undertaken by the user operating the master GIS or detach themselves and explore the world on their own. Only the user operating Control Center in master mode has control over the number of the viewers and the content they display. A new participant can freely join an existing session at any time; the issue of the initial state is resolved by exchanging synchronization information between the new participant and the master. The screenshot of one instance of the sytemem is presented in Figure 17 in chatper VIII.

Viewers used in the system are based on Open Inventor technology [see sec. 9.1] and they were written from scratch to meet our needs. This encompasses not only dumb

rendering of information requested by the user, but also processing that information on the client side and more complex interaction with the user than that offered by standard VRML browsers. Thus, it is possible to deploy such effective technologies as adaptive mesh simplification and triangulation of geographic information (DEM and textures) and wavelet compression of data streams to be sent over the network, provide simulation extension or database access.

7 PROJECT BACKGROUND

The system has been developed as a part of Collaboration and Interactive Visualization (CIV) project, grant of the Rome Laboratory. The specific mission of Rome Lab deals with the science and technologies associated with command, control, communications and intelligence (C3I) including surveillance. These basically involve information technology – the acquisition, transfer, processing, storage and display of information. As such, the laboratory's R&D activities cover technology in a wide range of areas including sensors, telecommunications, communication networks, distributed information processing and data bases, software, artificial intelligence, electromagnetics, signal processing, photonics and electronic reliability.

The objective of CIV project is to develop Command Control (C2) application, which is military description of a general real time decision or judgment support environment involving a complex set of people, data sets, and computational resources. A critical characteristic is the need to make the 'best possible', as opposed to 'optimal', choice. In a civilian context, crisis management has essentially similar needs. However, most application

areas have a component that links computers, information, and one or more people to make decisions.

To show the capabilities of C2 prototype we have implemented a simple scenario corresponding to an unmanned aircraft flying into U.S. airspace carrying a bacterial agent. This scenario involves military and civilian radar sensors and personnel tracking aircraft; NORAD as military command in execution stage; federal and state leaders at highest level (President and Governor); weather simulation to assess intercept possibilities, and in response stage, the dispersal of bacterial agents; FEMA as civilian command in response stage; and finally, medical authorities for expertise and treatment.

8 HIGH-LEVEL REQUIREMENTS

The objective was to develop a system providing visualization of multiple kinds of data, including satellite images, maps, weather information, complex models and possibly others. In addition, the system was supposed to possess all the necessary features to be useful in a Command Control application like dynamic layer switching, moving objects, etc. It should also support master-slave collaboration among many users so that the contents of their displays would be fully synchronized, however, independent views of the same data set are also allowed. As high-end visualization system it should provide near real-time speed of rendering without compromising accuracy of the visualized data. Keeping in mind bandwidth constraints, the system was supposed to be very efficient in communication issues, exploiting cutting-edge compression technologies and using compact protocol between its entities. Since the indispensable component of the system, Open Inventor [see

sec. 9.1], is a object-oriented toolkit and to achieve high performance it was decided to implement GIS using C++.

9 CORE TECHNOLOGIES

This section presents main technologies used in development of our system. Perspective from which these technologies are described is highly affected by their role in the system, hence some unused features have not been discussed at all. Reader interested in more comprehensive description should, therefore, follow the references to the bibliography.

9.1 OPEN INVENTOR

Open Inventor, developed by Silicon Graphics Inc. (SGI) can be concisely summarized as an object-oriented toolkit used in creation interactive 3D graphics applications. It is based on OpenGL, also by SGI, hence inheriting all the power of that highly acclaimed library. However, Open Inventor put emphasis on 3D objects rather than on graphics primitives, i.e. points, lines, polygons, etc., and as such, it stores these objets in a *scene database*. Each of these objects encapsulates a set of properties like shape, position, size, color, texture, etc. and methods that can be applied to it. During the rendering phase these objects are projected on 2D plane, but they still exist in the scene database as discrete entities. Therefore, they can be picked, highlighted, printed, searched for, read from and written to a file and many other complex actions can be performed on them.

A basic entity in Open Inventor model is a *node*, which translates into an object in object-oriented paradigm. Nodes store information about 3D world and they can be combined into hierarchical structure similar to a directed acyclic graph called *a scene graph*, which is in turn

stored in the scene database. The top node of scene graph is called *root node*. In general nodes can be classified as belonging either to one of the following categories:

- *Shape nodes,* used to model 3D geometric objects that are drawn on the screen during rendering action. Among shape nodes are cones, cubes, cylinders, spheres, 2D-, 3D-text nodes and more complex ones like quad meshes, triangle strips, face sets.

- *Property nodes,* representing appearance of the scene such as drawing style, geometric transformations, surface material and so forth. Property nodes define translation, rotation, material, draw style, complexity, font style, light model, normals and coordinate system.

- *Group nodes,* used as containers for other objets. Several nodes can be combined together in order to accomplish desired effect. For example, by use of a group node we can couple the transform node with the cylinder node and the material node as to obtain an object that looks like a small copper coin. In addition to a group node we have *separators*, which isolate the effects of nodes in a group, *switch nodes* that swap between their children so that simple motion effects can be achieved and *level-of-detail nodes* that determine which child to traverse relying on its complexity and distance from the camera to the node. That kind of nodes is used when the fastest rendering is required.

There are two other kinds of nodes that are essential in world rendering: lights and cameras. A camera node just generates a picture of the world according to its position in the scene. We can choose between two types of cameras: *perspective* that portrays more distant objects as

smaller thus emulating the human eye and *orthographic camera* that induces no deformity on farther objects. Other requisite node is a light described by type, intensity, color and position. Open Inventor distinguishes between ambient lights, point lights, directional lights and spotlights. We need at least one camera and one light to see the world.

9.1.1 TEXTURES

A texture is a two-dimensional pattern description that can be applied to surface of a 3D shape. Texture coordinates are appointed to polygon's vertices either explicitly by the world's creator or automatically by Open Inventor. Texture mapping is done in a smart manner, i.e. if a texture does not match exactly the polygon then Open Inventor uses some kind of filtering to assign the texture to the surface of the object. There are several types of texture nodes that differ mainly in applied coordinate system and degree of explicit vertex assignment. There are also three textures models that define the way a texture alters original polygon's color:

- *Modulate* that multiplies polygon's color by the texture color.
- *Decal* that completely supersedes polygon's color by the texture color.
- *Blend* that mixes the two colors according to the texture intensity value.

The texture itself can be one of the following types:

- *One-component texture* also called intensity map since it only specifies intensity value.

- *Two-component texture* in addition to intensity it designates an alpha value determining transparency level of the texture.

- *Three-component texture* uses RGB values in the texture description.

- *Four-component texture* besides RGB values it contains alpha value.

Since textures add realism and more importantly auxiliary information to the visualized world, they have been consistently used by our system, for example as a satellite image layer superimposed on the three-dimensional terrain.

9.1.2 ACTIONS

Open Inventor allows performing actions on the world, among them the most basic one – world rendering. When an action is applied, the scene graph or its part is traversed and each encountered node can manifest its own behavior as a response to the action. During traversal Open Inventor database keeps track on *traversal state* – a structure storing transient action parameter. The most common actions apart from rendering are: calculating a bounding box, accumulating a transformation matrix, writing to a file, calling back to the application, searching for a node and picking. In our system the two last actions are especially often carried out, therefore we focus on describing them.

Searching

A node or set of nodes can be found by applying a search action. It is possible to search for a specific node referenced by pointer, for nodes of particular type, for nodes with a given

name or combination of these. In some cases more than one node can exist that meets defined criteria, therefore it is practicable to specify the search scope. We can choose from finding the first node, the last node or all nodes satisfying the criteria. In addition we can decide whether search action should follow the scene graph as it is done during traversal or check all the nodes in the scene graph regardless the order and switch settings.

Picking

Ray-pick action can be used in finding objects along a ray from the world's camera through a specified point. The action returns a shape node (others do not have visual representation) or paths, i.e. parts of the scene graph to all shape nodes that lie along the picking ray. Pick ray can be defined either as a window point and a radius or a point and a direction in world space. In the former case the ray can be promoted to a cone in perspective camera world or cylinder in orthographic camera world. This helps in finding very small objects since the ray does not need to hit them exactly. In a ray-pick action we also can define the search scope narrowing the search to the closest object only or letting the ray to find all of them.

9.1.3 INTERCHANGE FILE FORMAT

Open Inventor is able to read and write a scene graph not only using memory but also a file. For this purpose a file format has been created. It comes in two flavors: binary, for faster I/O operation and human readable ASCII format. The latter is quite famous because VRML format in version 1 originates from it, however original Open Inventor format is far

richer that VRML. Every eligible node can be written to a file and then read from it, even such complex constructs as network of engines. This format is easily extensible and can handle nodes previously non-existent. All it needs is an object file implementing a new node and some support from operating system, i.e. dynamic loading capability.

9.1.4 SENSORS

Sensors provide callback mechanism to user-defined functions. A sensor attaches itself to the database and checks against specified event to occur. We can divide sensors on two categories:

- Data sensors that invoke callback function whenever data in a particular field, in a node or within any of nodes in certain path is modified.

- Time related sensors that can invoke user's function at a specified time, whenever the application is idle or delayed up to a certain amount of time.

9.1.5 ENGINES

Open Inventor defines the notion of *engine* as a node that can be connected to other node or global field (structure that stores a property) in order to animate or constrain the either connected node or some other part of a scene graph. An engine is usually used as a processing unit, this is it receives some input from a field, performs pre-defined actions on it and produces the results, which are sent to the set of fields connected to it. There are about 30 different types of engines, therefore we limit ourselves to barely enumerating main

categories and some of their representatives. Only those engines utilized in our system will be described more thoroughly in the subsequent sections. Engines can be divided as follows:

- *Arithmetic*: as simple as Boolean Engine that performs boolean operation on two input parameters, but also as complex as Calculator Engine that is able to parse arithmetic expressions and execute required computations.

- *Animation*: Elapsed-Time Engine that measures elapsed time; One-Shot Engine that can proceed for a preset amount of time; Time Counter sweeping a range of defined values at given frequency.

- *Triggered*: Gate Engine providing filtering and sampling mechanism between several fields; it is used normally by other types of engines.

- *Other engines*: Concatenate Engine that joins separate fields into a single multiple-value field.

Open Inventor supplies rich set of engines that can be used in off-shelf manner, but developers are not constrained by any means and creating a new engine is quite straightforward. In general, engines furnish very powerful features that can be easily incorporated in any application, thus making the world more dynamic and appealing.

9.1.6 DRAGGERS AND MANIPULATORS

A dragger is a node that responds to user events, specifically as one can expect to the mouse dragging, with a designated behavior. Open Inventor provides the set of 20 various

dragger types, between them as simple as single scaling or translating draggers, but also compound draggers rotating, scaling and translating all together in several directions.

Open Inventor supplies also another type of the node that is related to draggers, namely *manipulators*. Manipulators are not draggers themselves, but nodes that contain draggers. When a user interacts with a manipulator, in fact the contained dragger handles the user events and changes the fields of its manipulator, which in turn changes the fields of other nodes. In other words, manipulators affect other nodes while draggers move only itself. By use of both node types we can introduce even complex interactivity in a visualized world.

9.2 ADAPTIVE COMMUNICATION ENVIRONMENT

The ADAPTIVE Communication Environment (ACE) is an object-oriented framework aiming at development of high-performance and real-time communication services and applications. It provides whole array of reusable components inspired by design pattern concept[5] for common communication software tasks as event demultiplexing and event handler dispatching, service initialization, interprocess communication, shared memory management, message routing, dynamic configuration and reconfiguration of distributed services, concurrent execution, synchronization and so forth. Moreover, the ACE has been ported to many[3] modern operating systems like WindowsNT, Win95, Solaris, SunOS, Irix, Linux, HP-UX and others, therefore all applications founded on the ACE are portable on the level of source code. Figure 5 illustrates the key components in the ACE and their hierarchical relationships.

[3] Current number of supported operating systems reaches about 20.

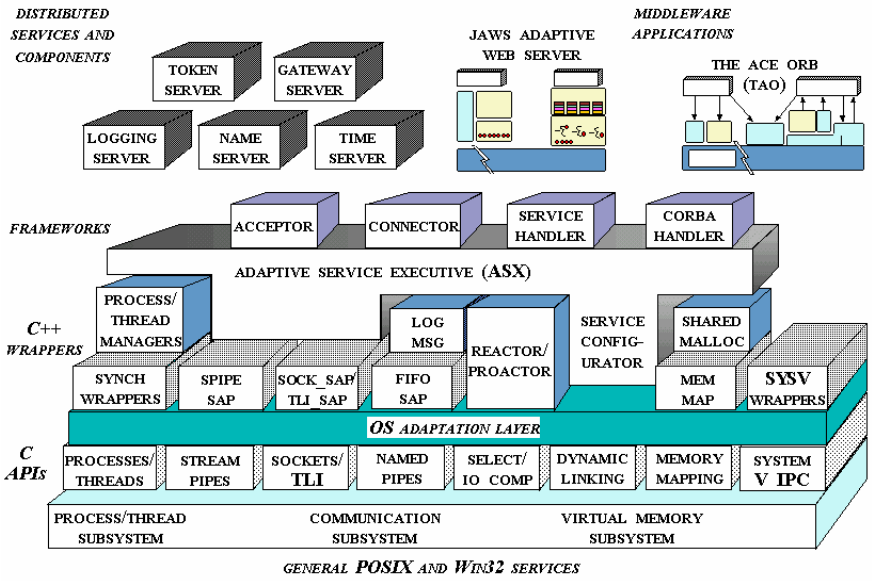

Figure 5: The ACE components

The ACE widely benefits from well-known set of design patterns, as described thoroughly in [5], thus gaining extensibility, modularity and reusability – primary concerns of the design pattern concept, without compromising the performance. This set consists of:

- *Creational patterns*: Abstract Factory, Builder, Factory Method, Prototype, Singleton
- *Structural patterns*: Adapter, Bridge, Composite, Decorator, Façade Flyweight, Proxy
- *Behavioral patterns*: Chain of Responsibility, Command, Interpreter, Iterator, Mediator, Memento, Observer, State.

In addition to the patterns mentioned above, the ACE contributes to development of the community by enhancing design pattern family. Some of them are:

- Reactor, Proactor; handle service requests that are delivered concurrently by multiple clients.

- Acceptor, Connector; initialize communication in a passive and active manner, respectively.

- Asynchronous Completion Token; allows to efficiently associate state with completion of asynchronous operation.

- Active Object; decouples method invocation from method execution in order to simplify synchronized access to shared resource by methods invoked in different threads of control.

- Half-Sync/Half-Async; decouples synchronous I/O from asynchronous I/O to simplify concurrent programming effort without degrading execution efficiency.

- Service Configurator; decouples behavior of the services from the point in time at which service implementation is configured in an application or system.

Currently while still polishing the ACE details, the ACE developers has started releasing middleware components like JAWS, generic, high-performance, adaptive framework for Web servers or TAO, real-time CORBA object request broker (ORB).

In our case, the ACE has been used in implementation both the Viewer and Control Center and proved to be an excellent framework. Since it provides high-level object-oriented

reusable components it boosts development speed and allows to focus on application specific issues, rather then re-implementing well-known pieces of code.

9.3 STANDARD TEMPLATE LIBRARY

The Standard Template Library (STL) provides a set of well-structured generic C++ components that work together in a seamless way. The orthogonal design of the library allows programmers to use library data structures with their own algorithms, and to use library algorithms with their own data structures. The well specified semantic and complexity requirements guarantee that a user component will work with the library efficiently. Every template component in the library has a generic implementation that performs almost as efficiently as a hand-coded routine. The library contains five kinds of components:

- Algorithms (sort, transform, generate, etc.)
- Containers (vector, deque, multimap, etc.)
- Iterator, (input_iterator, random_access_iterator, etc.)
- Function objects (binary_function, greater_equal, etc.)
- Adaptors, (queue, binder, etc.)

The worth of STL has been widely recognized by the C++ community and it will be included in Standard C++ Library.

9.4 XT INTRINSICS AND MOTIF

X Window System, developed by MIT and DEC, is a hardware- and operating system-independent windowing system. A programmer using X Library (Xlib) is able to develop applications with an advanced user interface with complete network transparency and thanks to X Protocol they will run without any modifications on many types of workstations and personal computers. Despite the Xlib is powerful enough to write fully functional X Window applications and it does not need any additional tools, usually in order to simplify development process two other libraries are used, X Toolkit (Xt) and Motif library. The former provides a general mechanism for producing reusable interface components, widgets and gadgets, without imposing any particular look-and-feel whatsoever. Typical widgets include scrollbars, menus, dialog boxes, text-editing areas, drawing areas, etc. Xt Intrinsics, C library forming the core of the toolkit, mask implementation details from a programmer, the widgets and the applications build with them are fully extensible. It provides a consistent interface for writing applications along with some intrinsic mechanisms allowing an access to non-graphical functionality of X Window system. Motif, the leading user interface for the UNIX based operating systems, supplies a complete ready-to-use set of widgets meant to realize the concrete application look-and-feel. It uses Xt and Xlib as underlying libraries.

9.5 TANGO

TANGO is an integration platform that enables building Web-based collaborative environments. The system provides the means for fast integration of Web and non-Web applications into a multi-user collaborative environment. The main functionality provided by the system consists of session management, communication between collaborating

applications, user authentication and authorization and event logging. TANGO supports synchronous and asynchronous, by use of database back-end, collaborations of variety applications. It does not impose any specific implementation language, thus providing programming language independence, as long as socket communication can be used. Therefore, it cooperates with JavaStript, Java applets and applications, C/C++ applications. Recently the family of supported languages has been expanded with a famous functional language – Lisp.

TANGO is an attempt to build a generalized Web based collaborative system capable of creation of diverse sharable information spaces, which in turn could be populated with Distributed Virtual Environments (DVE) in the most generic sense of this notion. TANGO extends Web paradigm to the domain of collaborative computing, which is far beyond the concept of the chat, shared whiteboard, and replicated, identical instances of simple applications.

Currently, when TANGO2 is being developed, TANGO has collected several proofs of its concept. It has been successfully employed in such diverse domains as distant learning programs or a prototype of Command Control application for military, to name a few.

9.6 TANGOSIM

TANGOsim is a discrete-event multithreaded simulation engine developed solely in Java. It implements the notion of virtual time and communication classes of related applications or applets. TANGOsim can be driven by either a rich-syntax scripting language or by a user via Simulation Controller applet. The engine is quite powerful and does not limit

itself only to broadcasting messages at a given time, but also is able to create and manage TANGO sessions, spawn participating applications, set up multimedia streams sending and modify the course of a simulation depending on user input. Its modular architecture and dynamic linking capability allows communicating with great number of different applications and applets employing relatively small-sized stubs that translate simulation events into application specific messages. TANGOsim is a one of components of Command Control application.

10 FUNCTIONALITY

This section describes top level functionality of the system without diving into technical details. We tried to summarize the capabilities of the system in one place so that comparison with other systems on the functional level would be easier. The underlying mechanisms and architecture are explained in section 11.

10.1 VIEWER

10.1.1 MULTIPLE DATA REPRESENTATION

The system allows superimposing several textural layers on the same data set in each viewer window, thus achieving multiple data representation. The feature permits to emphasize different aspects of the visualized data at the same time, thus leveraging user perception of the data. All data representations are exactly synchronized as far as the viewpoint is concerned so that user can focus on the same subset of analyzed data. For example, if we had an area represented in three windows, as a satellite image of terrain, a

population density and weather information, we would be able to reckon the number of casualties in the case of nerve agent spread. Example of multiple data representation is presented in Figure 20 in chapter VIII.

10.1.2 SYNCHRONIZED AND DESYNCHRONIZED VIEWS

Having multiple data representation it is quite easy to submerge in chaos of analyzing different subsets of the data without even being aware of that. To overcome such an effect we introduced synchronization mechanism that constitutes with other some attributes the shared state. Namely, viewers interchange messages informing about their camera updates consisting of the position and the orientation. Each viewer that receives camera update can decide whether to use this information to synchronize its camera parameters entirely, partially or just to ignore it.

In order to use this mechanism effectively, we defined the notion of *viewer communication class* described in greater detail in section 11.1.2. Briefly, master viewers from different communication classes synchronize to some selected parameters of other masters, whereas slaves synchronize to their masters, i.e. master viewers belonging to the same communication class, while ignoring others.

Synchronization mechanism while very useful, might be sometimes perceived as obtrusive. Therefore slave can switch it off or in other words desynchronize from its master. This capability supports independent views. While desynchronized, the slave has total freedom to examine the data by itself, however in order to ensure the data representation

consistency it is not allowed to change superimposed layers.[4] At any time slave can start synchronizing again and follow the camera of the master viewer.

10.1.3 OPERATING IN THE WORLD

The system has been designed to be a very interactive and that was one of the reasons to implement the viewer based only on Open Inventor rather than use an existing VRML browser. Generally there are two modes of operation that the user can switch between: *Picking,* when selecting areas, dragging, picking objects and also accessing the GIS database are possible. All these actions are described in greater detail in next three subsections. Second mode is called *Navigating,* when the user has complete freedom to move in 3D environment. If needed, however, movement of the user can be constrained [see sec. 11.1.2] and then viewer performs collision detection so that the user cannot go under ground and also cannot fly too high. The latter constraint is used in 3D GIS along with a world extension technique [see sec. 10.1.14] in order for the user not to see the world boundaries. Operating in navigating mode can be very impressive, especially in complex worlds with stereoscopic rendering option switched on.

10.1.4 HOT SPOTS AND PICKABLE OBJECTS

A content provider can deliver not only 'raw' terrain shape and textures for layers but also hyperlink entities like *hot spots* and *pickable objects*. The formers are areas defined by geo-

[4] If a user in slave mode wants to explore totally other sets of data, he/she can start another session in which he/she would be a master, thus having complete control over the contents.

referenced coordinates and connected with them URLs to the data repository. When a user clicks on a hot spot, a Web browser is started and the link pointed by the URL is initiated. This solution is very flexible, since the user can have access to additional information about an interesting areas and this information can be of various types: pictures, movies, web pages or even database information accessed and processed by CGI scripts.

While hot spots are limited to geographical information, pickable objects are free from this kind of constraint. Such an object is a part of visualized data set and can be as simple as a single triangle or as complex as a VRML model of the electromagnetic profile of an airplane. The mechanics is similar to the one of hot spots, namely, a URL is connected with each such an object and a Web browser does the actual information retrieval and the presentation of the results.

Pickable objects and hot spots facilitate association of parts of visualized data set with complementary information, moreover this information is accessible to a user by one mouse click.

10.1.5 DRAGGABLE OBJECTS

A next step towards enhanced interactivity is provided by means of *draggable objects*. When we have a comprehensive model representing complex process, it is usually hard to grasp all it at one instant. It seems that the model be better off divided on several parts, which could be examined separately. It is of course possible to visualize each part alone, but then we risk loss of information about how the parts combine the whole. Therefore, we came up with the following solution. Each distinguishable part can be made a draggable object that can be

detached from the whole and examined independently. After a user decides to finish work with a particular part, it can be dragged to the previous position in the model and another part can be detached. There is no limit that only one part can be detached at a time, we can virtually dismantle the whole model and taking advantage of independent views several users can work on individual parts simultaneously. This feature again is accessible to the user with one mouse click and does not need any other actions.

10.1.6 DATABASE SUPPORT

An indispensable feature of the system is an access to large amount of data about the visualized world. A logical solution is to store this data in a database, however it should be possible to query database about any attribute in the form of spatial queries. So that information about certain features in the specified distance from the selected geographical point (latitude and longitude) could be easily retrieved. Therefore, we have used Illustra database armed with the Spatial Data Blade, a module of the Illustra Server provided by Informix that allows to issue spatial queries in the convenient form. In order to facilitate access to the data another module has been used – the Web Data Blade. It serves as a Web interface for the database, so queries can be sent using HTTP protocol. Data for our system have been acquired from various sources like Census Bureau and many public domain organizations.

The user has two-level access to the data. First one is a quick preview based on query about the nearest features like schools, airports, lakes, hospitals and so on, from the selected point. Second level of access is more powerful, thus more complex. A Web browser is

launched and interfaces to several modules like Census Data, Roadfinder are presented. The user can then issue queries about the selected point by herself, narrowing the search and accessing data in other formats than text only, for example 2D outline maps built from TIGER/Line data or 3D VRML models representing many database attributes.

10.1.7 DATA DOWNLOADING

In order for GIS to be efficient in the matter of input data, it was decided not to store local copy of DEMs and textures on each host, but rather use downloading technique. Therefore we have established an HTTP server that stores the terrain data and provided viewer with downloading capability, both synchronous and asynchronous [see sec. 11.1.8]. Since it incurs some latency and decreases data transfer, data caching mechanism has been implemented [see sec. 11.1.9]. Thus we gain storage economy and consistency of the data, while reducing overhead to minimal possible value.

10.1.8 DISCREET EVENT SIMULATION

The system is not limited only to three dimensions as one could expect, but in cooperation with TANGOsim described in section 9.6 a 4th dimension[5] is introduced – time-dependence. Therefore, simulation capability is provided so that we can visualize dynamics in data sets. The simulation engine drives the system by use of control messages that are translated into internal events and by managing the notion of virtual time. The system allows adding two kinds of objects: static and moving. Static objects, although they can be

[5] Should we therefore call the system 4D GIS?

animated, cannot change their position. Moving object, in contrast, can both be animated and move in a virtual world. There are no special requirements as far as object's shape is concerned except that it should be a valid VRML or Open Inventor description. In addition, the TANGOsim is in charge of pausing and resuming the simulation, hiding and showing the added objects and managing the status of the objects. Figure 19 in chapter VIII presents screenshot of the system with activated simulation extension.

Regarding moving objects, a user is often interested in following one of such objects, however it can be a quite hard task to do it manually. Therefore, viewers allow choosing the object to be followed so that the user does not need to chase it. The viewer simply attaches camera to the object and tracks its position updates keeping the constant distance from it [for details see 11.1.5]. For better information current geographical position is also displayed below the object. Whenever the user changes his/her interest he/she can switch to follow other object or disable this feature and explore the world by manual navigation.

10.1.9 LEVEL OF DETAIL

It is wasteful and impractical to render the whole terrain with the best possible accuracy since usually the user examines only a small part of the world at a time and does not need the maximum level of detail for the remainder of the terrain. Therefore, we have employed discrete level-of-detail supported by Open Inventor. Each pre-defined part of the terrain shape has four levels of accuracy and decision on which part is to be rendered is based on camera's proximity to the particular part of the terrain. Use of level-of-detail greatly boosts

performance of fly-through navigation without significant loss of the visualization quality. Figure 18 in chapter VIII presents part of the rendered terrain with maximum level-of-detail.

10.1.10 WAVELET COMPRESSION

By building our own viewer rather than using a VRML browser we could take advantage of doing some processing on the client side. One example of such processing is wavelet compression/decompression technique that we have deployed in order to minimize the amount of data sent over the network. We used Embedded Zerotree Algorithm producing highly compressed data streams of good visual quality. Moreover, such an embedded stream does not have to be transmitted entirely over the network, since even a part of it will resemble the whole. Therefore, we obtain control mechanism over representation fidelity by sending as much of the data as we want. It is even possible to improve quality of rendered data by incremental transmission of the stream and appending received data to the initial part. Needless to say that using such a technique saves bandwidth significantly – accomplished compression ratios are between 25 and 90, depending on complexity and required fidelity of the data.

10.1.11 TRIANGULATION AND MESH SIMPLIFICATION

A further step in bandwidth savings and rendering time improvements could be achieved by altering representation of the data itself. In other words if we were able to find a more compact data representation than the original one it would help in achieving a better compression ratio. For that reason we decided to replace VRML file format, which usually results in huge file size with a compact binary format that can be used to build Open

Inventor scene upon receiving the data by the viewer. Translating process consists of two stages: adaptive mesh simplification and triangulation. It is well known that subjective perception of the data quality is determined strongly by the fidelity of textures. Therefore, complexity of the surfaces can be substantially reduced without visible loss of quality. Usually, we remove 80% to 95% of original number of triangles and the remainder is used to create appropriate Open Inventor nodes. The drawback of this solution is that some computation has to be done by the client. It is however needed only once, when a new data set is loaded and has no effect on the speed of the navigation in the world. Figure 6 depicts whole process of data transformation.

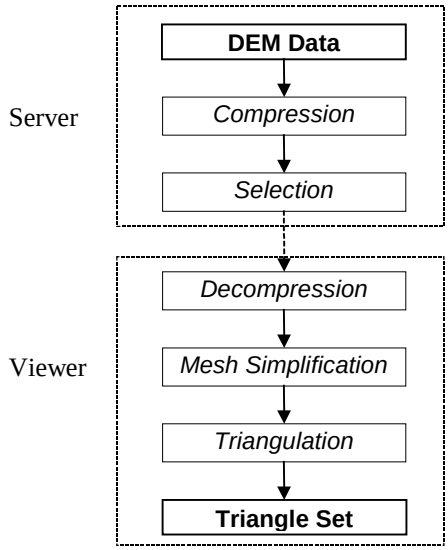

Figure 6: Data transformation process

10.1.12 WORLD DESCRIPTION FILE FORMAT

In order to maintain the system highly flexible in the matter of visualized data sets and their properties, we had to develop a description format that provides the way of expressing

all needed features in a concise manner and at the same time, it still complies with the Open Inventor format. That being the case, the viewer always loads a description file first, which is meta-data about the visualized data. A description file consists of the tags, which are legal Open Inventor nodes. However, the viewer recognizes them as special and performs according actions such as downloading actual data sets, building layer list, creating draggable and pickable objects, inserting hot spots and so forth.

10.1.13 DYNAMIC WORLD DESCRIPTION

We have benefited very much from developing the Description File Format. However it is not practicable to have a description file for every combination of terrain areas that can be selected from a huge terrain repository. For that reason a more convenient way to specify necessary information had to be devised. We have noticed that apart from user defined hot spots or draggable objects other information could be created without user's assistance based on minimal input. If a user specified areas of data sets he/she is interested in, a description file could be generated automatically. Therefore, Dynamic World Description has been implemented. It requires an additional tool that facilitates indication of interesting areas. That among others was the purpose the Control Center has been build for. After receiving the information about the data sets that are to be rendered, a description is created and handed over to the viewer as if it was loaded from description file. Such a mechanism allowed us to develop description format independently from specifying basic information on what to visualize.

10.1.14 WORLD EXTENSIONS

After the first array of system dry runs it turned out that realistic fidelity of rendered data, particularly of geographical type, is somewhat undermined by the fact that a user can fly beyond world boundaries. To alleviate the problem movement constraints have been added. However, it would not have solved it, had the user reached boundary and the edge of the world had been still visible. As a solution, to the visualized data set viewer automatically adds adjacent sets, but using much less detailed textures and surface mesh. In such a prepared world a user can fly over sets selected by her, but never over automatically added extensions. Therefore, without much overhead we have overcome the feeling that world is floating in a void.

10.1.15 STEREOSCOPIC RENDERING

In order to provide a user with 3D experience the system supports stereoscopic rendering. It requires, however, special hardware support. We have tested the system with CrystalEyes equipment that consists of an infrared emiter and shutter glasses. This type of hardware imposes that Open Inventor should be switched in stereoscopic mode, therefore producing two, a little bit shifted representations of the world.

10.2 CONTROL CENTER

Control Center (CC) is responsible for coordination of the work of viewers and communication with TANGO. Its basic function is similar to the one of TANGO; namely it broadcasts messages to all participants but only within one host boundary. The scheme of communication is rather straightforward. Whenever a Control Center receives message

coming from the TANGO side, which means that the message has been sent by other instance of Control Center, it checks if the message is of the control type. If so it processes the message accordingly, otherwise it just relays it to all viewers connected to the Control Center. Below reader can find some highlights of the Control Center.

10.2.1 VIEWER MANAGEMENT

In addition to message broadcasting CC manages the viewers. A user operating in master mode has explicit control over a number of lunched viewers and contents they visualize, whereas CC in slave mode disables the controls and provides only information about visualized data set. In other words, the task of the CC in slave mode is to mirror the system instance in master mode, which is a somewhat similar relation as it is in the case of viewers.

10.2.2 FLOOR CONTROL

Control Center does not administer any form of floor control whatsoever, i.e. messages either from the viewers or from TANGO can arrive at any time and this is responsibility of the viewers to handle them properly. Such a relaxed policy is possible, because viewers operate in master-slave model and even in independent data view mode the resource contention issue can be easily resolved.

10.2.3 OVERALL WORLD REPRESENTATION

Since our system is used mostly in terrain visualization, CC has an option to display overall map covering the whole available area. This area is divided into rectangular chunks of parameterized size so that the user is able to choose data set or sets to be visualized. This

part of user interface was intended to eliminate the need for specifying the URL to the data set by hand. Moreover, the user is able to tailor size of the world to her preferences and more importantly to CPU and graphics power of her machine.

11 ARCHITECTURE

This section discusses some details of the architecture and design decisions that have been made during the system development. It explains how certain functionality described above has been realized and specifies some technical details that often had significant influence on particular solutions. Before we present the details, the over-all architecture needs to be highlighted.

From top-level perspective we can perceive the environment involving Control Center, viewers, GIS database, an HTTP server and TANGO. For illustration purposes we consider three GIS instances that have been started on separate hosts and they are participating in the same session as it is depicted in Figure 7. Each such instance consists of one Control Center and several, it this case three, associated viewers. One of the system instances runs in the master mode, others in the slave mode. As it was stated earlier, master viewers have control over the visualized contents and they also broadcast their camera parameters so that slave viewers would be able to synchronize. In the presented situation we have 4 communication classes denoted by small numbers in the corners. Class 0 belongs always to Control Centers participating in the same session, the remaining classes are created and assigned dynamically to the viewers. The reason behind devoting a separate class for Control Centers is that we need to have means to distinguish between control protocol messages [see sec. 11.2.4],

which are directly handled by CC and viewer protocol messages [see sec. 11.1.13] that are forwarded for servicing to the viewers.

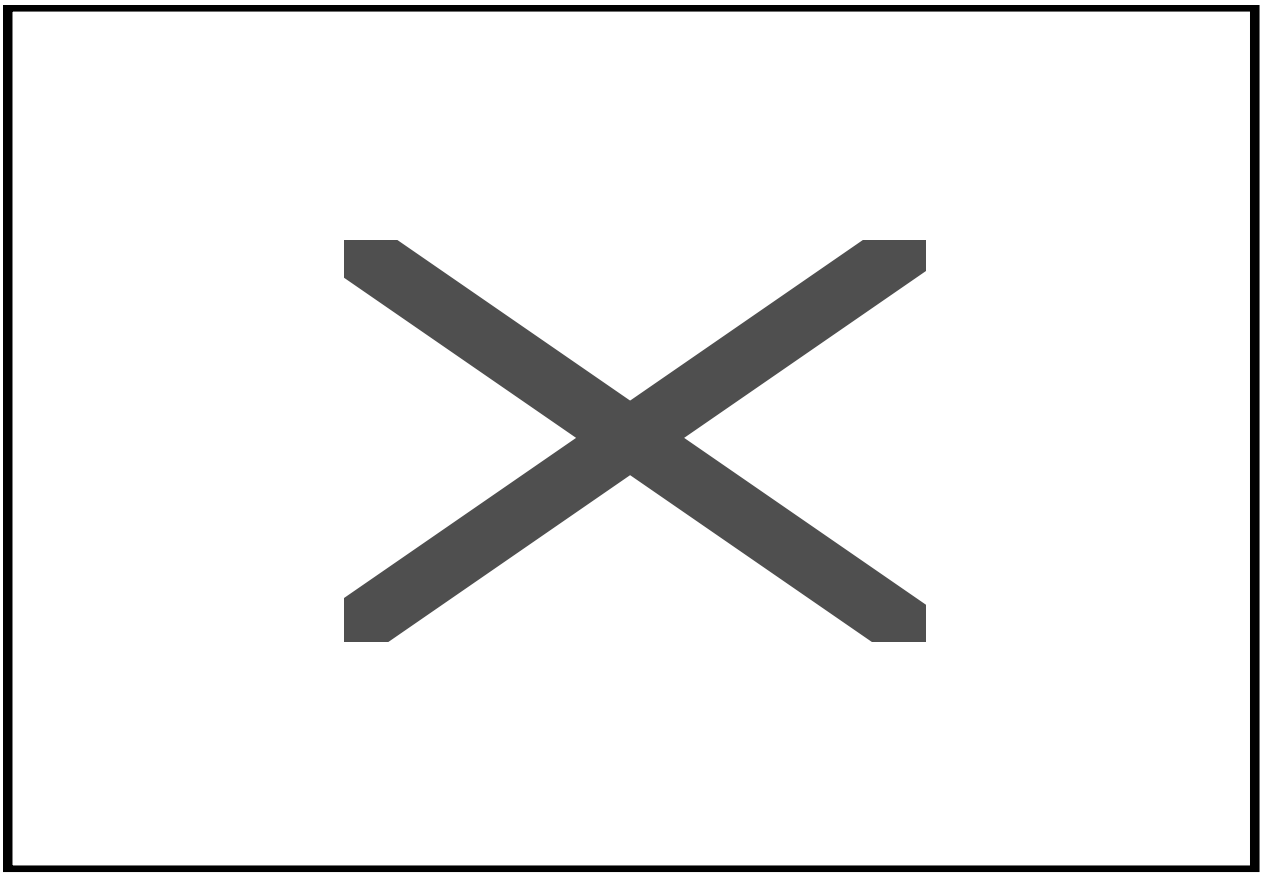

Figure 7: Overall system architecture

From technical perspective, connection with TANGO is managed by Control Centers using INET domain sockets, whereas viewers communicate with Control Center by use of UNIX domain sockets, which significantly improves communication performance. Both kinds of channels use TCP protocol.

Other entities showed in the figure are an HTTP server and GIS database. The HTTP server is used for distributing visualized data, usually in the form of digital elevation model

(DEM) files and textures files. Each viewer regardless of its session status, i.e. master or slave, can request necessary data. However, only masters have enabled the part of user interface that allows changing the visualized world. Slaves download only the data specified by their masters. GIS database, on the other hand, is responsible for maintaining all other kinds of data and handling viewers' queries.

11.1 VIEWER

11.1.1 MAIN CLASS

The main class of the viewer is *SoXtWebViewer* derived from Open Inventor class *SoXtWalkViewer*. We have chosen walk viewer paradigm since it allows user to immerse in the world as if she was walking in it and behavior of the camera can simulate head movement. It also supports easily fly-through capability needed in terrain visualization. Other possible choice was an examiner viewer, which uses virtual trackball paradigm allowing exploration of the world as if it was held in the hand. This type of viewer would be better in showing object models like these in molecular chemistry, but since our primary goal was terrain visualization, we decided that our viewer would inherit from walk viewer. Figure 8 places our viewer in Open Inventor viewer family.

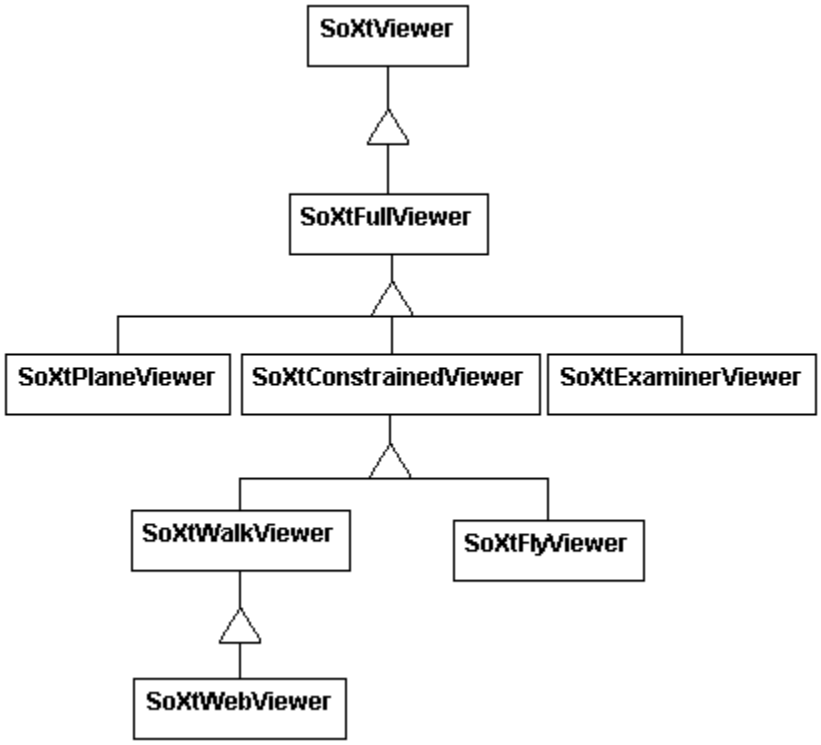

Figure 8: Viewer class hierarchy

The viewer is a quite complex construct and in addition to world rendering it controls, coordinates and provides user interface for:

- Camera movement control
- World loading either from a file or over the network
- Maintaining basic information about the world like world's URL, for terrain visualization: origin and scale of the terrain
- Thematic layers handling

- Building its state snapshot in order to provide the initial state information for other viewers, if needed
- Keeping repositories for static and moving objects
- Managing world extension areas, if needed
- Extracting surface information for placing hyperlinked objects
- Hot spots handling
- Database querying
- Other functions controlled by decoration rim widgets like thumbwheels or buttons.

A viewer can be either in navigation or picking mode. When navigation mode is active all mouse events, except for pressing right button, change the camera parameters and force the viewer to redraw the visualized scene. When a user switches to picking mode it is possible then to query the database [see sec. 11.1.10] by pointing the surface of the world, sending URL to Web browser if picked point belongs to a hot spot or drag some objects.

11.1.2 SYNCHRONIZATION

Since our system supports collaborative decision making, it is necessary that all participants have consistent application state. Therefore, every event triggered in a master viewer should be translated into a compact message and broadcast to all other members. This, of course, requires proper synchronization measures. However, we also support

various visual representations of the same data, which means that the user operating the Control Center in the master mode can launch several viewers and visualize the same data set but then superimpose different layers on it and inspect the set from different perspectives. This situation creates the need for a mechanism that would allow checking if a synchronization event that has been just received is pertinent to a particular instance of the viewer.

To address this issue viewers use the notion of communication class. Every synchronization message conveys in its header communication class of the sender, this message is broadcast by the TANGO to all viewers participating in the same session regardless of their collaboration mode; master or slave. Each viewer in the slave mode updates its state only if the communication class in the message header is equal to the one of the viewer, this means that the message has been sent by its master. Otherwise the message is simply ignored. When a master receives a message it is apparent that the message has been sent by other master and decision whether to ignore it or not is made on the individual message basis. For example messages about layer changing are ignored since master viewers provide various data representations and they are totally independent in that matter from each another. On the other hand, for the data consistency requisite, messages about loading a new data set are performed as if the event was triggered locally. There is even a third case in this mechanism, namely if a master viewer receives camera parameters update it extracts and brings up-to-date only selected parameters while the rest remain unchanged [for details see 11.1.13].

Another issue in synchronization mechanism is efficiency. Therefore special care is taken of processing camera parameters update messages. These messages are sent and received most commonly between the GIS systems, and as a result of TCP/IP encapsulation more than one[6] camera update can be received at one time. Since rendering is a CPU intensive process, it is wasteful to redraw the world when the next message in the queue will change camera parameters again immediately after the previous update. That is why only the last update in the queue really forces rendering process and the other camera parameters' updates have no effect. Of course this technique may cause the motion of the camera in slave to be less smooth than in the master, but it decreases lags in synchronization between hosts of different CPU power, especially when the master runs on a faster machine.

11.1.3 PROTOCOL FRAMEWORK

The system is distributed over the network and bases on the shared state. Therefore, as it has been stated in section 11.1.2, there is a need for synchronization and maintaining the state consistent among all participants. It can be done by exchanging compact messages between instances of the system. For this purpose we need a generic communication mechanism. Taking into account extensibility of the protocol and several examples of design patterns, we have developed reusable component providing such a generic communication mechanism.

[6] Actually up to 30 camera updates in one TCP packet have been observed.

ProtocolInterface family has been designed to express the similarities of any protocol service no matter what messages a particular protocol consists of. Presented architecture resolves several forces, namely it decouples application specific tasks from protocol service issues, it separates protocol administration, like checking whether incoming message is valid in terms of currently handled service and so forth, from sending and receiving data streams itself. Finally it makes possible to have dynamic protocol, properly adjusted to the current state of the application. Overall architecture is depicted in Figure 9. The following subsections provide insights of the architecture.

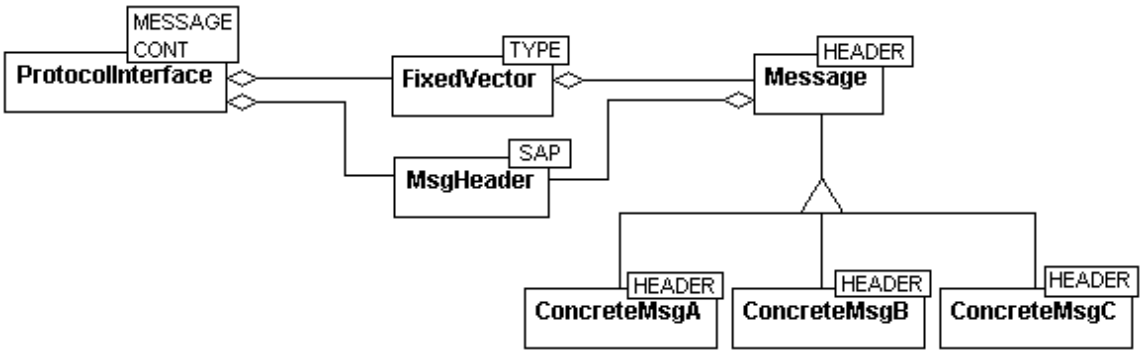

Figure 9: ProtocolInterface class family

ProtocolInterface

The central class of the framework parameterized by *MESSAGE* and *CONT*, which are, respectively, class of the handled messages and container class in which the formers are stored. *ProtocolInterface* is responsible for maintaining the connection with a peer, sending and receiving messages and managing the protocol itself. A connection is realized by a service access point (SAP) abstraction, which in fact is socket wrapper class and provides at least

basic, yet reliable methods like recv, send and so forth. *ProtocolInterface* can be perceived as a implementation of Flyweight design pattern, as described in [5], since the application specific objects during construction register every type of the message the application can understand and handle properly. Therefore we ensure that there exists only one message of a given type and it can be retrieved by use of a key, which in our case is message identification key, stored in the message header. After setting up the protocol, the application registers interest in any activity on the SAP and waits for notification. Upon receiving such a notification, *ProtocolInterface* checks if the message belongs to the current protocol, if so, it passes SAP to the message and let the message read itself and perform appropriate actions.

MsgHeader

MsgHeader encapsulates the protocol specific header. We assume that the message header is able to provide somehow two crucial kinds of information: the length of the message and the message ID. In our system header format complies with TANGO header specification, i.e. 4-byte channel ID and 4-byte message length; it also adds one byte for message identification purpose and one byte denoting viewer communication class. A programmer can freely change the sizes of message identification key and communication class if needed, since this is out of TANGO scope. *MsgHeader* is a template class parameterized by SAP since it happens we need socket communication in UNIX domain as well as in INET domain. The type of SAP propagates to Message class and eventually to *ProtocolInterface*.

Message

An abstract base class providing the interface and some convenient methods for all messages used by application. Each concrete message knows how to serialize itself through SAP, process own piece of information if needed and invoke application methods to prompt certain actions. Moreover, the message header is separated from the message body itself to gain more flexibility in implementing a particular protocol. Therefore, *Message* class is parameterized by HEADER.

ViewerMessage

This abstract class is derived from *Message* class and it is responsible to provide filtering mechanism that compares the viewer communication class identification key stored in the message header against the one obtained by inquiring the viewer. The application specific messages derived from *ViewerMessage* class rely on this information whether to perform shared state update or not.

SimEngMessage

SimEngMessage is an abstract class derived also from *Message* class. Its task is to receive in a uniform way all messages coming from TANGOsim [see sec. 11.1.12] simulation engine. All specific messages that belong to the simulation part of protocol take over the read data and perform necessary actions. *SimEngMessage* provides elegant way of decoupling receiving special format messages from implementing particular simulation events.

FixedVector

This is a simple implementation of message container whose size is set at moment of instantiation and does not grow beyond it. We use *FixedVector* since it provides the fastest way of retrieving certain messages by direct access to the messages, but it can be replaced by more sophisticated containers like STL map or hash table when necessary.

Interactions

During construction phase *SoXtWebViewer* by use of *Xt Toolkit* registers interest in any activity on the socket handled by *ProtocolInterface*. From that moment if any message will be sent to the viewer *SoXtWebViewer* will be notified about a message arrival and this results in the class following interactions [see on Figure 10]. Viewer delegates handling of the messages to *ProtocolInterface*, which in turn read the message header, checks in the message container if the incoming message is valid, if so it let the message read itself. Otherwise it reports protocol violation. If the message was read successfully *ProtocolInterface* administers its execution and the message perform specific actions on the viewer.

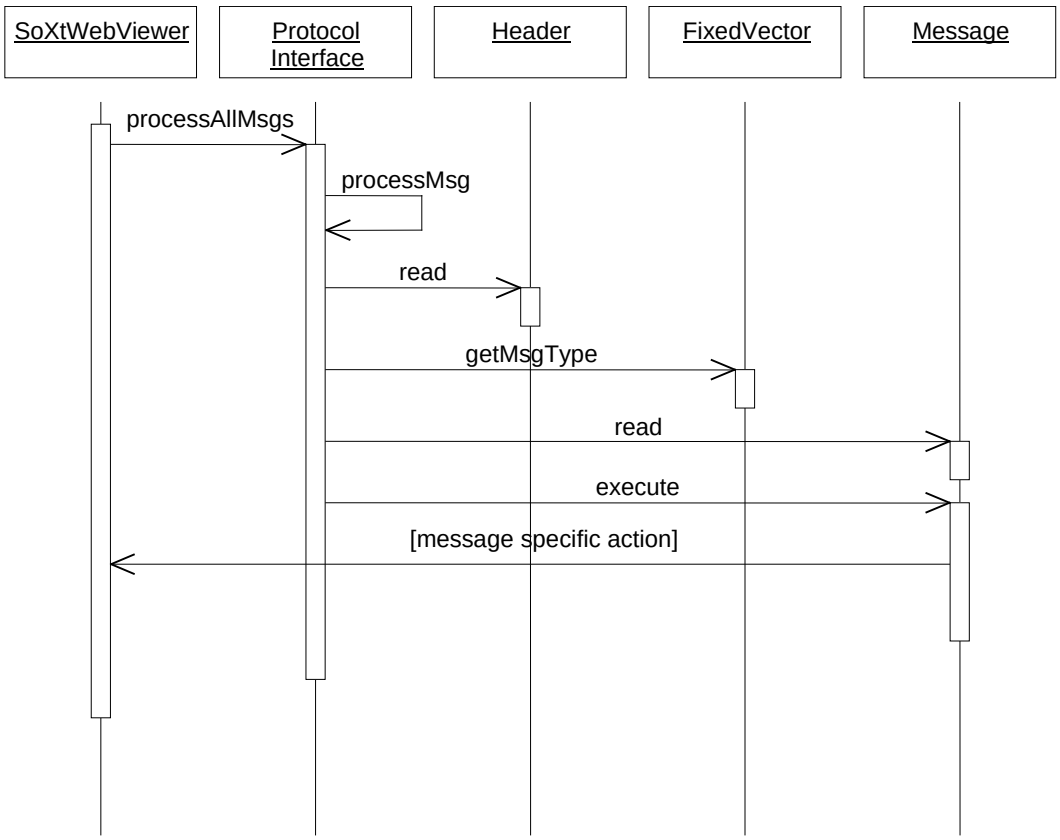

Figure 10: Interactions within ProtocolInterface class family

11.1.4 STATIC AND MOVING OBJECTS

Apart from visualizing strictly geographical information, viewers can also display static and moving objects. In general, such an object is a VRML or Open Inventor model representing a real-world entity like an airplane or radar, and it is added to the current scene. By the term of 'static object', we mean an object whose geographical position is not changed; this allows it to rotate about its own axis if needed, For instance, for Radar Operator console purposes, we had a radar symbol with a spinning part. Such static objects do not need special implementation in Open Inventor, because they can be simply 'forgotten' after having inserted them into the scene.

The case of moving objects is far different; namely, the viewer constantly keeps track of those kinds of objects, altering their movement parameters whenever it gets an update. The updates come from the simulation engine, which is in charge of permanently feeding the viewer with an appropriate data. The simulation engine maintains the notion of virtual time and every updating message has a time stamp. Along with the time stamp, a moving object is sent its next geographical position. The time denotes when the object should reach the specified position. Based on these parameters, the current position and the current time, the object is required to re-compute its actual movement parameters. No additional messages are needed for the object to proceed, and by use of interpolation the next set of coordinates is evaluated. The object stops if it reaches the specified position and new parameters are not set. It is also possible to show and hide such objects, change their status (e.g. from foe to friend) or pause and resume movement of all these objects by sending appropriate messages [see sec. 11.1.13].

By employing interpolation mechanism we achieve substantial bandwidth savings similar to those of dead-reckoning technique proposed by DIS standard[4]. There are, however, some differences; namely, in dead-reckoning a client is sent initial set of movement parameters and allowed to extrapolate next positions of the object. At the same time the *shadow* computation on the server side is necessary in order to decide whether the pre-defined error threshold between the actual object position and its extrapolated position has been exceeded and sending new set of parameters is inevitable. In our case the simulation engine is written entirely in Java and adding computation burden for each moving object is

not desired. Therefore, we take advantage of the knowledge of the 'simulation future' and cut drastically bandwidth utilization.

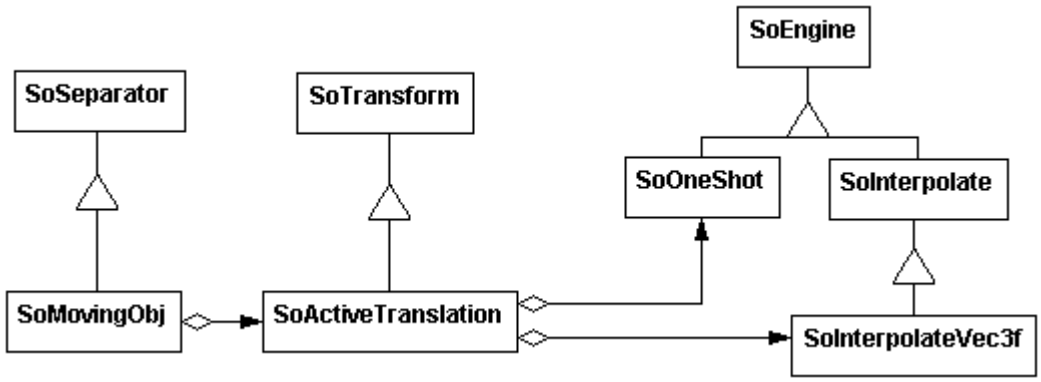

Figure 11: Moving objects class family

Although Open Inventor does not provide an exactly suitable class fulfilling all these requirements, it supports a very helpful notion called *engines* [see sec. 9.1.5]. An engine can be connected to a field of a node and alter its value in a function of time; therefore if the field is responsible for the node's position we achieve the movement effect. Details can be found in [7] and [6]. We took advantage of this feature and encapsulated all above functionality in a class family, with the most important *SoMovingObj*, depicted in Figure 11. In addition to normal features like shape, texture, etc., *SoMovingObj* has *SoActiveTranslation,* which alters the set of its coordinates every time the node is traversed; i.e. it gets updated in the rendering process. *SoActiveTranslation* uses two engine objects: one for interpolation purposes and one for time measurement.

11.1.5 OBJECT FOLLOW-UP MODE

As it was stated in functional description viewers help the user in keeping track of moving objects by supporting automatic follow-up mode. After the user chooses the object to be followed the viewer creates a data sensor, summarized in sec. 9.1.4, attaches it to the object's active translation and registers a callback function responsible for taking an appropriate action. Whenever the value of the translation changes, meaning the object has moved, the sensor notification is triggered and as a result the callback function is invoked. In this case the callback function simply checks current position of the object and moves the camera accordingly. In order to eliminate jitter movement effect we had to precisely coordinate movement of the camera with rendering moment. It has been done by disabling the render action for the period of time between the object's and camera's position update.

11.1.6 WORLD REPRESENTATION

The system is able to understand any data set described by Open Inventor ASCII or binary file and also VRML version 1 format. This representation, however, ubiquitous in the Internet world is not compact enough, particularly ASCII Open Inventor and VRML formats. Since our primary objective was to develop the GIS system supporting near real-time fly-through capability and data remote storage, we had to represent the terrain in the way that terrain can be rendered and sent over the network very efficiently.

The terrain information can be broken into two categories: the terrain shape itself and array of thematic layers that can be superimposed on the shape. As a base for the terrain shape we have used freely available set for the whole New York State of 1-degree Digital

Elevation Models (DEM), a sampled arrays of elevation for a number of ground positions at regularly spaced intervals. However, unprocessed DEM does not fit our purpose either, for the reason of being voluminous. For instance, the area of 70 by 70 miles (120 by 120 km) consists of about 3 millions triangles. Therefore, a concise representation of the terrain shape has been necessary. The solution is to remove redundant elements from the data. Fortunately, terrain shape can be efficiently simplified since many areas of the terrain can be approximated by large polygons without much loss of visual quality. In order to conduct such an approximation the terrain shape has to be represented in one of convenient formats. In our system we chose Triangulated Irregular Network (TIN), which is a set of contiguous non-overlapping triangles whose vertices are located on the surface of the terrain. We can dramatically decrease the size of the terrain shape by eliminating unnecessary points from the full elevation sample matrix in a process called *mesh simplification* and by eventually *triangulating* them, i.e. creating from the remaining points set of triangles.

The second category, thematic layers are usually high-resolution imagery as satellite images or scanned maps, therefore the same issue of the size arises. We cannot, however, apply to these images techniques that are valid for size reduction of the terrain shapes. Instead, we can use wavelet compression and filter out the high-frequency details achieving significant gain with slightly noticeable loss of resolution quality parameters.

Application the above techniques both boosts up the performance of rendering process and decreases the bandwidth needs of the system.

11.1.7 TAG FILE FORMAT

Owing to the system interactivity and versatility it is necessary to provide it with significant amount of meta-data about visualized data set. It should convey the information about all visible and non-visible features that allows to conduct an interaction with a user implemented by our viewer and also properly render the data set. On the other hand, the format in which the meta-data is coded must not violate Open Inventor format. For that reason we decided to use Open Inventor *SoInfo* type in association with capability of giving names for a particular nodes. *SoInfo* nodes are usually placed in world description files and used in giving comments on the world that VRML browsers are free to simply ignore. Our viewer, however, searches the scene graph built from a description file for *SoInfo* nodes with pre-defined names, which we call *tags*. Currently there are 11 tag types handled by the system. Below is the brief description and actions they prompt.

GIS_Root

Usually presented data is not limited to a single set, but it is of significant size and it has inherent structure that breaks down into a great number of files. Therefore, it is stored remotely and distributed by an HTTP server. GIS_Root specifies the URL to the server, which is used in subsequent data retrievals.

GIS_Origin

If visualized data set represents some kind of spatial information, true for terrain, it has to be properly placed in the Open Inventor space. GIS_Origin binds the Open Inventor

origin to the geo-referenced center of the data, so that hot spot feature and database querying work correctly.

GIS_Scale

This is another tag that is essential for database querying and hot spot handling. It provides information that allows translation from Open Inventor space to geographical coordinates.

GIS_Bounds

GIS_Bounds tag specifies the world's boundaries that limit the user's navigation space. The primary goal of this tag is to constrain user's movement in order for her not to go under ground or too high over the world. However, it is even more useful in association with world's extension feature since it does not let a user to reach one of the world's edges preventing the unrealistic impression that the world floats in a void.

GIS_Terrain

There are usually several GIS_Terrain tags in a one description file. They enumerate, using geo-referenced rectangle areas, which part of the globe is to be visualized. The viewer bases on these tags and GIS_Root tag in order to download appropriate chunks of the terrain.

GIS_Texture

Again there are normally several tags of this type. They list all available layer types for the data set. Based on information provided by GIS_Terrain tags a viewer decides which particular texture files should be downloaded.

GIS_RectArea

It defines a hot spot as a geo-referenced rectangular part of the visualized terrain, which is added to the repository of hot spots maintained by the viewer.

GIS_CircleArea

Likewise as the above, it defines, however, a circular area. The areas can overlap, thus covering more complex shaped parts of the terrain built from primitives like rectangles and circles.

GIS_Object

GIS_Object is used to insert in the scene graph an object that apart from having a shape has also a hyperlink connected to it. Such a link is the URL that provide relevant information of whatever type and it is sent to a Web browser when the user clicks on it.

GIS_Slider

It defines a so-called draggable object, this is an object that can be dragged away from its original position by the user. This can be used in complex model visualization, when a user prefers to dismantle the model and examine each part separately.

GIS_V5DFile

The tag inserts dynamic isosurfaces used in weather visualization.

11.1.8 HTTP COMMUNICATION

Since the system needs access to significant amount of visualization data, we have established HTTP server for the data sets distribution, e.g. DEMs and textures. As none of the used technologies provide an off-shelf class capable to communicate with the server, we have developed such a class by ourselves. After investigating our needs we decided to implement generic class able to handle both synchronous and asynchronous communication. Need for asynchronous mode stems from the fact, that one of the requirements compels the system compatibility with Irix 5.3 thus precluding use of thread programming. We need, however, capability of requesting information and gathering it as it comes without waiting for it. This allows to request a new set of textures while a user is still using the other set.

All necessary capability has been encapsulated in one class – *WebNode* so that switching between modes would be straightforward. When an object of the class is constructed without a handling function, the object assumes synchronous mode and waits for data to be

entirely downloaded. It is used, for example, in the world initialization when the whole description file and possibly the shape of the modeled data set are requisite. Accordingly, when a handling function is supplied during the construction of a *WebNode* instance, the object sends a request as previously, but instead of waiting for the data it registers its interests in activity on a particular socket by invoking Xt intrinsic function XtAppAddInput. When the HTTP server or the database web interface process the request and TCP packets start arriving, the internal *WebNode* callback function is called. Its task is to collect all coming packets and combine them into a one continuous stream of data. After the entire information has been received *WebNode* invokes the user handling function, which can process the data properly. The asynchronous mode is usually used in downloading texture layers or in database querying.

11.1.9 DATA CACHING

In order to reduce the time needed to load a new data set, a cache mechanism has been implemented. It works in two cases, when a previously downloaded scene is referred to once again and when one viewer wants to use data cached by another, which is quite common case. For example, whereas textures for a satellite image and map differ, DEM data is the same, because the same area is visualized.

Cache implementation itself is rather simple. Namely, every instance of the viewer before downloading checks in the cache index whether the resource specified by the given URL has been already cached. If so it simply reads the appropriate file, otherwise it downloads the resource, writes it in the cache directory to a uniquely named file and inserts in the cache

index a file entry that maps the URL to the file. Since the cache index is shared by all instances of the viewer launched on the same host, maximum downloading efficiency is achieved. Although simple, the cache has proven to be very useful; the loading time of a medium-size world decreased by a factor of 5 in the case of a local network and up to 30 for distant communication.[7]

11.1.10 DATABASE INFORMATION RETRIEVING

All real Geographic Information Systems are firmly built on top of voluminous data of multiple kind. Our system is not any different in that matter. The difference is rather in the way the data is accessed, namely, the system employs standard web HTTP protocol. This solution has gained some acceptance in the Internet community, it is, however, not so common in large and expensive systems where data is too precious to be publicized all over the world. In order to handle HTTP requests, GIS database has been armed with the Web Blade – a module that provides web interface to the Illustra server. By use of the module we can send requests and receive information in the form of HTTP message.

When a user switches to picking mode and picks a point of the world, the communication with GIS database is engaged. We support two presentation levels of the data. The first is a preview window that displays the results of automatically sent queries about the nearest features from a pre-defined set of features. The set consists of schools, lakes, villages, cities, airports, hospitals and so forth. The second provides the simple yet

[7] These factors are rough estimates not based on precise benchmarks.

powerful interface to the GIS database itself. It uses a Web browser and HTML forms in order for the user to specify a query. The system sends to a Web browser geo-referenced coordinates so that the user have a brief description of the area she clicked and easy access to additional information about this area. The information consists of Census and TIGER/Line data, but also to Roadfinder – application allows the user to determine street intersections and their corresponding coordinates. The user can browse specific tabular attributes, small maps of the cities or even analyze some of the attributes presented in VRML format.

11.1.11 INLINE NODES

Inline nodes enable specifying a URL instead of inserting an actual piece of world into the description file. This is very convenient when we would like to display multiple kinds of data maintained and updated independently by different people or groups. The Open Inventor provides basic functionality by providing the framework class *SoWWWInline*, the task for the programmer is to supply a callback function, which is able to download any resource over the network. Since *WebNode* and *URL* classes have been implemented exactly for such a purpose it was possible simply to reuse these components in the callback function. Inline feature has been added in order to comply with VRML 1.0 specification as described in [14].

11.1.12 COOPERATION WITH TANGOSIM

Cooperation with TANGOsim relies on set of messages sent by one of its modules. The module, called also *a stub*, is a translator that based on simulation events creates and emits

messages understandable by the system. There are several messages in the system that have been exclusively added for simulation purposes. They mainly encompass managing of static and moving objects that are driven by simulation events. The high-level architecture of the system with simulation extension is depicted in Figure 12.

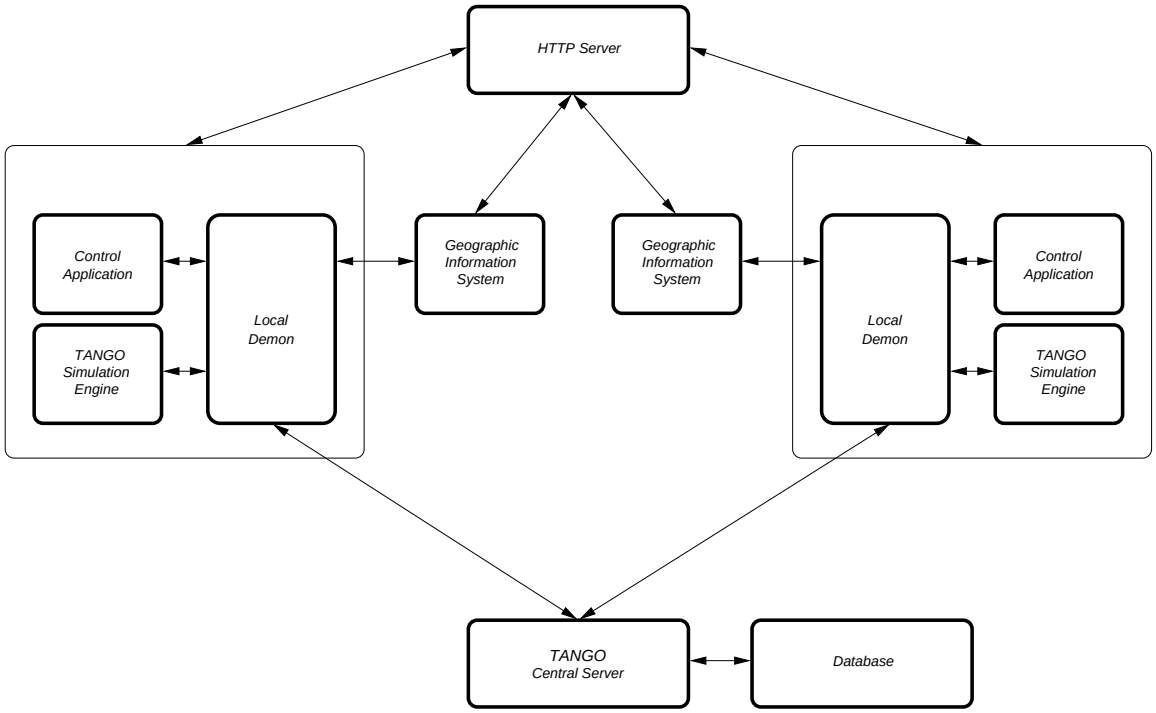

Figure 12: Cooperation with discrete event system extension TANGOsim

The stub itself is a Java class that understands both simulation engine script commands and the system messages. It takes advantage of dynamic linking and can be exchanged by newer version without having to compile the whole simulation engine. Despite that simulation extension has been implemented to be used in a prototype of Command Control application it can easily support any kind of simulation or dynamic behavior.

11.1.13 VIEWER PROTOCOL

Currently implemented viewer protocol consists of 18 messages and it is described briefly below. Please note that part of the protocol used in simulations is dynamically activated only when simulation extension is initiated so that at any time the protocol is appropriately tailored to the application needs. It was possible by flexible design of *ProtocolInterface* framework.

Camera Parameters Update

The message sent by a master as a side effect of the user navigation. It informs all other participants that viewpoint in the master viewer has changed and provides minimal yet sufficient amount of information needed to full view synchronization. It consists of the camera position (X, Y, Z) and camera orientation (rotation vector and the angle). Receiver of the message depending on the communication class of the sender can choose whether to use this information and synchronize to the master or not.

World Downloading

The message informing about changing the visualized data set. The user operating one of master viewers can specify the location of the data set that is to be rendered. Other viewers are broadcast this location in order to display the same data, which means only the same objective and there is no constraints on types of data relevant to the objective, e.g. different thematic layers. The location is in the form of a URL.

World Dynamic Building

Instead of specifying a URL by hand, user can take advantage of dynamic building of world's description file. In this case the data set is selected using a overall map from Control Center, which in turn sends boundary parameters to the viewers. Based on these parameters viewers are able to create necessary world description and download requisite data.

Layers Switching

Whenever the user switches a thematic layer in master viewer this information is sent to other participants. Slave viewers that belong to the same communication class as master in question also have to switch to the same layer. The code name of the layer is enclosed in the message.

Dragger Parameters Update

When the user interacts with a dragger the information about that fact needs to be propagated along with updated parameters of the dragger. Since the current version supports linear draggers, only position of the dragger needs to be sent.

Synchronization Information Request

A slave viewer that has recently joined the session sends synchronization request in order to set its initial state properly. It includes its communication class in order to discern which of master viewers should respond to this request.

Synchronization Information

This is the response to the message described above. It contains either the URL to the current data set or its boundary parameters. Additionally the information about the camera, draggers and the current thematic layer is also sent.

Short Synchronization Information Request

The request sent by the slave that is switching from independent view mode back to synchronized mode. As a response to the request appropriate viewer, depending on the communication class of the slave, sends only its camera parameters so that the slave can synchronize immediately without waiting for the master to change the camera parameters as a result of the user navigation.

Adding Static Object

This and the following messages are sent by TANGOsim simulation engine and activated dynamically when the GIS system is switched into the simulation mode. The message adds a static object specifying its geographic position, code name, description and type. Currently supported types: base, ship, hospital and radar.

Adding Moving Object

The message adds a moving object to the scene graph without specifying its current position. It contains the information about object's code name, description and type. Currently supported types: airplane and helicopter.

Position Parameters Update

The update is usually sent when new moving object is added to the scene, however, we do not want it to start moving immediately. Therefore the position of the object is fixed and does not change until moving parameters update, described below, arrives.

Moving Parameters Update

The message concerns moving objects and contains the current position of specified object, the next position of the object and the time in which the distance between these two positions is to be proceeded. After receiving this type of message the object starts interpolating its position based on sent parameters. Therefore, there is no need to constantly send the new position of the object over the network, thus achieving bandwidth savings similar to those obtained by dead-reckoning technique.

Status Information Update

Anytime during simulation the classification of an object could be changed based on flow of the simulation and decisions of the user. This type of massage updates the

classification and as a result the model of the object is changed. Currently supported object classifications: friend, foe and unknown.

Setting Follow Mode

The message sent by a master to its slaves in order to synchronize their follow-up mode. The message includes object's identification name assigned by the simulation engine.

Hiding Object

Simulation engine message that forces specified object to be hidden. If a particular object was followed, the filed sensor attached to its position is destroyed and follow-up mode is switched off. It is used during the simulation whenever the object vanishes from the radar screens.

Showing Object

Reverse action to the described above. It simply shows the previously hidden object. Even if follow-up was switched off because of object's hiding, the message does not restore it.

Pausing Moving Objects

The message pauses motion of all moving objects included in the scene graph. It is sent by simulation engine whenever the user stops the simulation using Simulation Control applet.

Resuming Moving Objects

The message restores movement of the stopped objects as if they were not paused. That means that object internal timers are also stopped when the pause command is issued.

11.2 CONTROL CENTER

The function of the Control Center can be summarized as the management of the viewers. Our goal was to keep this function as separate from viewer implementation itself as possible so that effortless changing or even complete elimination of Control Center would be possible and viewers still would be able to operate. Therefore, Control Center mainly exchanges messages between viewer instances. Nevertheless there is a small set of operations that are out of viewer's scope, for example creation of a new viewer or negotiation with TANGO about session status. This is the operation field of Control Center. Figure 13 depicts Control Center architecture; below there is description of its main classes.

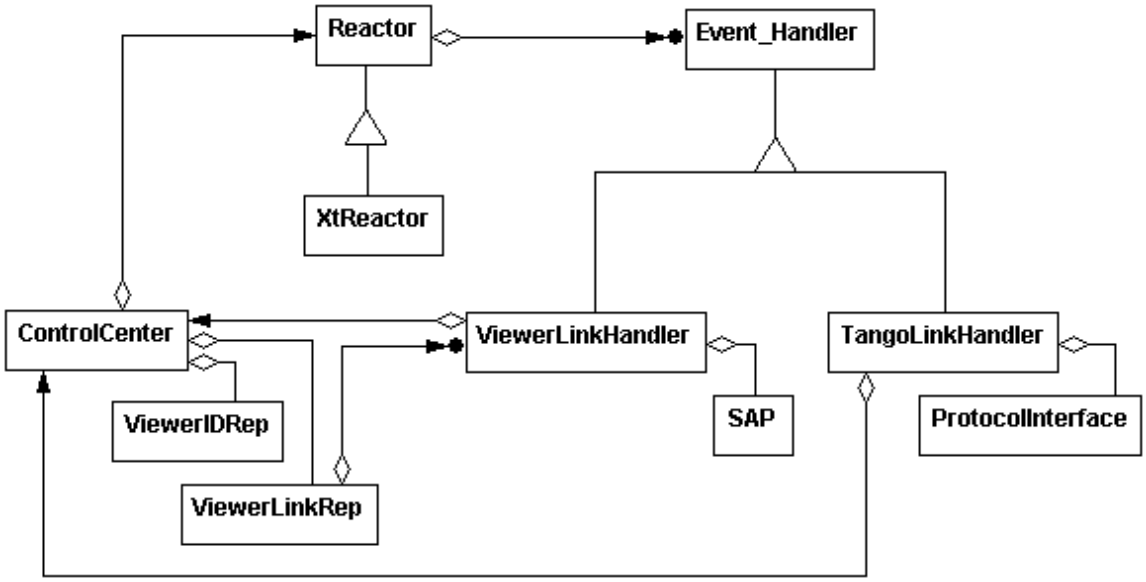

Figure 13: Control Center Architecture

11.2.1 MANAGER CLASS

Manager class called *ControlCenter* is responsible for maintaining communication with TANGO demon and all local viewers. With Control Centers participating in the same session small shared state is associated and the task of keeping it coherent boils down to launching and killing the viewers reflecting the master operation. Therefore, when the master spawns a new viewer, a new communication class is created and this viewer is assigned to it. On each slave side are spawned corresponding viewers and they are assigned to the same class as newly created master viewer. The notion of communication class is quite important it the system since it allows discerning whether an incoming message is meant for a particular viewer. *ControlCenter* class is in charge of initializing the system instance, providing message dispatching mechanism, implementing specific user interface and taking appropriate

actions as a result of incoming messages, whereas the communication burden itself is distributed between two other classes *TangoLinkHandler* and *ViewerLinkHandler*.

11.2.2 TANGO COMMUNICATION

The responsibility for TANGO communication rests on *TangoLinkHandler* class. As it is showed in the above figure the class is derived from *Event_Handler*, a class provided by ACE for, as one can suspect, event handling purposes. During the system initialization stage *ControlCenter* creates *TangoLinkHandler* and registers it in *XtReactor*, also an ACE compoment, specially adopted version of Reactor working along with Xt Toolkit and Motif library. From that moment on, every activity on the *ProtocolInterface* socket will trigger invoking of event handling methods of *TangoLinkHandler*. Please note that thanks to quite sensible design, *ProtocolInterface* can be also reused in this application for handling control protocol between Control Centers.[8] This protocol is summarized in section 11.2.4 below.

Since Control Center application has direct contact with TANGO demon it has to comply with TANGO protocol specification. Moreover, it supports session mode negotiation in order to initialize properly itself and later the viewers. If TANGO Control Application informs Control Center that it is in the slave mode it has to disable all graphical user interface controls, send synchronization request and then merely reflect the master state.

---

[8] *ProtocolInterface* will be probably reused by the author also in implementation of Directory Server for Buena Vista – a multiplatform conferencing system.

11.2.3 VIEWER COMMUNICATION

The scheme of communication concerning viewers handled by set of *ViewerLinkHandler* objects, one corresponding to each viewer, is quite simple in comparison with the previous one and does not need any protocol handling. The handlers mainly act as multipexling and demultiplexing units, i.e. whenever a viewer sends a message, *ViewerLinkHandler* uses *ControlCenter* method to broadcast this message to other local viewers and to forward it to others *ControlCenter* instances through TANGO link. Conversely, if a massage comes from the TANGO side *ViewerLinkHandler* objects relays it to their viewers. These handlers similarly to *TangoLinkHandler* also use sockets in communication with viewers, however, the sockets are in UNIX instead of INET domain, which significantly improve local communication.

11.2.4 CONTROL PROTOCOL

As it was stated before Control Center has been kept as simple as possible and it is reflected by its protocol consisting only of five messages. In order to distinguish between control messages and viewer messages again communication class notion has been applied. We reserve class number 0 only for Control Center use, therefore any other number in the message header classifies the message as to be addressed to the viewers and relays it to them. Below is brief description of the control protocol. Please note that the term master in this context concerns only Control Center mode, not viewers. The difference is important when we realize that there is only one master Control Center in the session, while we can have

many master viewers in the same session. However, these viewers reside on the same machine and the Control Center on the machine must be definitely in master mode.

Synchronization Information Request

If during session mode negotiation turns out that the Control Center will work in the slave mode, then it sends synchronization request. Since there is only one master Control Center at a time it will respond with its current state.

Synchronization Information

The message sent by the master as a response to the synchronization request message, consisting of the information on the current number of launched viewer and, if applicable, the selected area of the terrain. After receiving this information slave Control Center starts appropriate number of viewers and they on their own send request for synchronization to their master viewers.

Spawning A New Viewer

The message is sent upon launching a new viewer so that other systems could synchronize to the master. When slave receives this type of message it simply starts new viewer on its side.

Terminating A Viewer

The message is complementary to the previous one and informs slave Control Center that should break communication link with specified viewer, thus forcing it to quit.

Update

Message used when an overall map is displayed by Control Center. When the user selects an area of the terrain to be rendered this information had to be propagated to other system instances so that they synchronize their state.

VI. CONCLUSIONS

This thesis has attempted to discuss functionality and architectures characteristic to 3D Collaborative Environments. As a basis we have assumed 3D Geographic Information System, a system developed in Northeast Parallel Architecture Center at Syracuse University. Since the system has been primarily designed for terrain visualization, we presented functional components of traditional geographic information systems along with typical data models used in such systems. However, due to enhancements introduced to our system it evolved into a more generic visualization tool and classification it as strictly GIS is no longer adequate. Therefore, we have mentioned some ideas pertaining to broader class of software: Distributed Virtual Environments so that some sort of comparison would be possible. Next, we have presented several existing 3D visualization environments and the approach used by their designers. The main part of the thesis consists of detailed description of 3D GIS. Used technologies, functionality and architecture are subsequently introduced. We depict current version of the system that features complex data rendering, near real-time fly-through, shared state synchronization, multiple data representations, database support, pickable and draggable objects, hot spots connected with URLs, discrete event simulation extension with automatic follow-up mode, HTTP data downloading and efficient use of network bandwidth by use of sophisticated compression techniques.

Our system has been employed in several projects ranging from prototype of the Command Control application for military, more specifically as a part of Radar Operator Console, through the Visible Human project as a 3D extension for visualization of reconstructed human organs or as a system for visual analysis of airplane electromagnetic profiles. We believe that the list of possible applications is still open and the system will have many more chances to prove its usability. Currently the system runs on Silicon Graphics platform and in order to visualize large worlds comfortable it needs Onyx and Reality Engine, however we have done many presentations of moderate sized worlds using O2 workstations. As the computation and graphics power will continue to grow we can expect widespread of similar systems on PC platform really soon.

If we compare our system with the environments presented in chapter IV, we can notice some advantages in favor of our system. Let us enumerate the key features of each system separately.

mWorld, based also on Open Inventor technology, seem to be 'only' a viewer with audio capability and an extension enabling to simulate the motion of generated off-line deformable objects. It, however, possesses quite interesting feature of putting the burden of generating these objects on the most powerful host participating in the session. The decision of distributed architecture without a central server and at the same time use of order of n^2 reliable, meaning TCP, connections maintained on client side is not quite understandable to us.[9]

[9] mWorld uses multicast in transmission of audio streams only.

CSpray has very powerful visualization capabilities. Their set of metaphors, smart particles and cans, is precisely elaborated and may result in real support of cooperative work. That's why the system seems to be the most universal among all presented environments and it was used in great number of projects. It also provides rudimentary collaboration awareness by eyecon metaphor and possibility to look over other participant's shoulder. CSpray, similarly to mWorld, uses order of n^2 TCP connections, which of course eliminates the need for central server, however scalability problems sill remain.[10]

The VGIS is the system that manifests most similarities to the 3D GIS. It clearly emphasizes its main objective, which is real-time rendering capability. Therefore, it deploys such techniques as sophisticated continuous level-of-detail or heavy multi-threaded architecture. It also provides very good simulation capabilities and benefits from DIS protocol. We did not, however, find any information on its collaboration capabilities and therefore we assumed that VGIS do not have such a feature. Also from various articles and presentations on that system one can infer that database support is rather minimal.

Now we can return to Distributed Virtual Environments and address the question whether our system could be classified as a DVE. The 3D GIS has been optimized to render efficiently large worlds therefore it would handle typical DVE worlds without any problems since they usually consist of lesser number of polygons and less richer textures than let's say high-resolution satellite imagery. Moreover, our system employs in world's description compressed binary data streams rather than VRML, which also seems to be a performance

[10] 1000 TCP connections scale almost as badly on client side as on the central server.

advantage. Nonetheless all this does not render our system a DVE at one instant. We do not support avatars, collaboration awareness is limited to participants' list displayed by TANGO and there is definitely no subjectivity, the system shows the world as it was recorded by satellites or scientific measurements. From the architectural perspective, Distributed Virtual Environments stress the scalability issues very much, therefore they most commonly employ decentralized topology and multicast as transport protocol, whereas our system takes advantage of services provided by TANGO and its central server that uses TCP/IP. To summarize, albeit we could enhance the system with some, typical DVE features, for instance implementation of avatars in our system would be straightforward, it still seems that differences in the paradigms are to significant and substantial changes in the architecture of the system would be imminent to bring the 3D GIS into the DVE realm. On the other hand, such a venture is not necessary since our system is very successful in the field of 3D Collaborative Environments.

VII. BIBLIOGRAPHY

[1] Koller, D., et al. Virtual GIS: A Real-Time 3D Geographic Information System. In *Proceedings of Visualization'95*, October 1995, pp. 94-100.

[2] Lindstrom, P., et al. An Integrated Global GIS and Visual Simulation System. GVU Technical Report, July 1997.

[3] Samet, H. The Quadtree and Related Hierarchical Data Structures. *ACM Computing Surveys* 16(2), June 1984, pp. 187-260.

[4] DIS Steering Committee. The DIS Vision: A Map to the Future of Distributed Simulation. Institute for Simulation and Training, Version 1, May 1994.

[5] Gamma, E., et al. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.

[6] Wernecke, J. The Inventor Mentor: Programming Object-Oriented 3D Graphics with Open Inventor, Release 2. Addison-Wesley, 1994.

[7] Wernecke, J. The Inventor Toolmaker: Extending Open Inventor, Release 2. Addison-Wesley, 1994.

[8] Walczak, K., Trzaska, R. and Roman, P. Implementation of Geographical Information System Based on Virtual Reality. In *Proceedings of The Dual-Use Technologies & Applications.* May 1997, pp. 85-90.

[9] Nye, A. and O'Reilly, T. X Toolkit Intrinsics Programming Manual. O'Reilly and Associates, Inc., 1990.

[10] Heller, D. Motif Programming Manual. O'Reilly and Associates, Inc., 1991.

[11] Stevens, W. R. UNIX Networking Programming. Prentice Hall, 1990.

[12] Stepanov, A. and Lee, M. The Standard Template Library. Hewlett-Packard Company, 1995.

[13] Fielding, R. et al. Hypertext Transfer Protocol – HTTP/1.1. Internet Engineering Task Force, July 1996.

[14] Bell, G., Parisi A. and Pesce M. The Virtual Reality Modeling Language. Version 1.0C Specification, January 1996.

[15] Schmidt, D. C. Ractor. An Object Behavioral Pattern for Demultiplexing and Dispatching Handles for Synchronous Events.

[16] Dias, J. M. S., et al. mWorld: A Multiuser 3D Virtual Environment. IEEE Computer Graphics and Applications, March-April 1997, pp. 55-65.

[17] Pang, A. and Wittenbrink, C. Collaborative 3D Visualization with CSpray. IEEE Computer Graphics and Applications, March-April 1997, pp. 32-41

VIII. APPENDIX

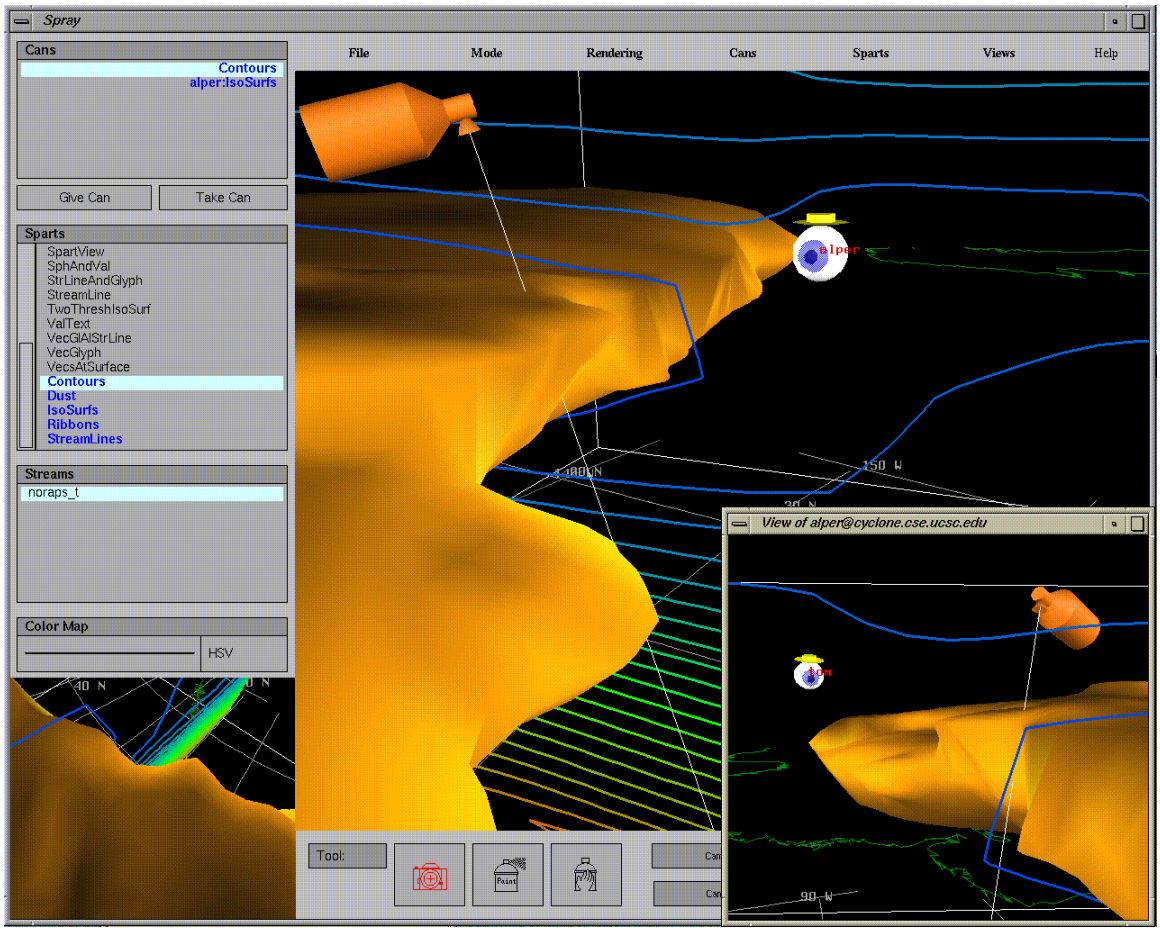

Figure 14: Example of CSpray collaboration

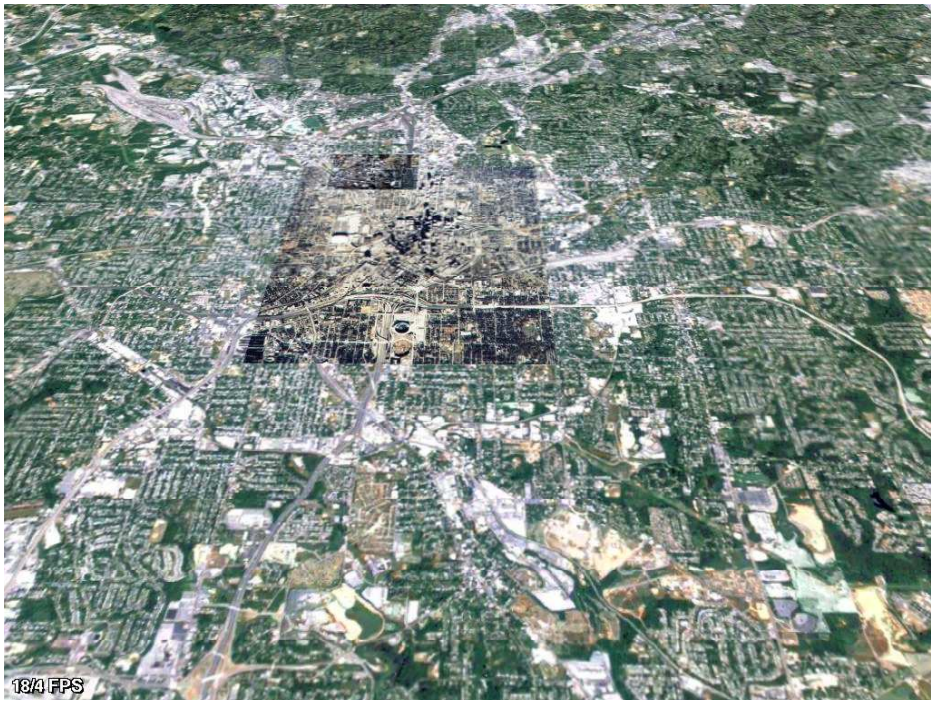

Figure 15: VGIS. Textures in different level-of-details.

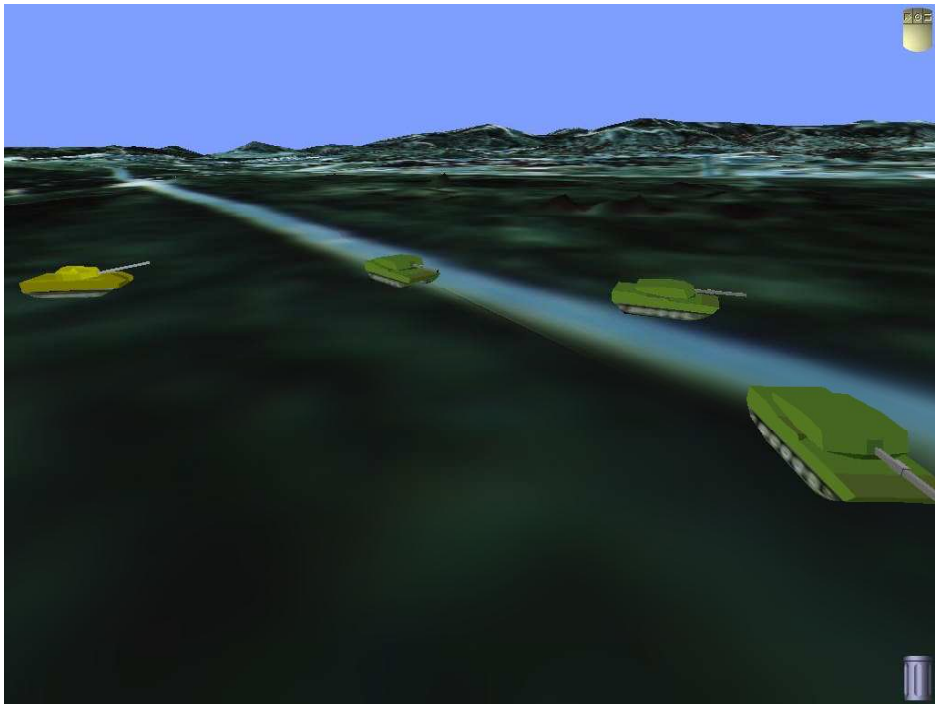

Figure 16: VGIS. Battlefield simulation.

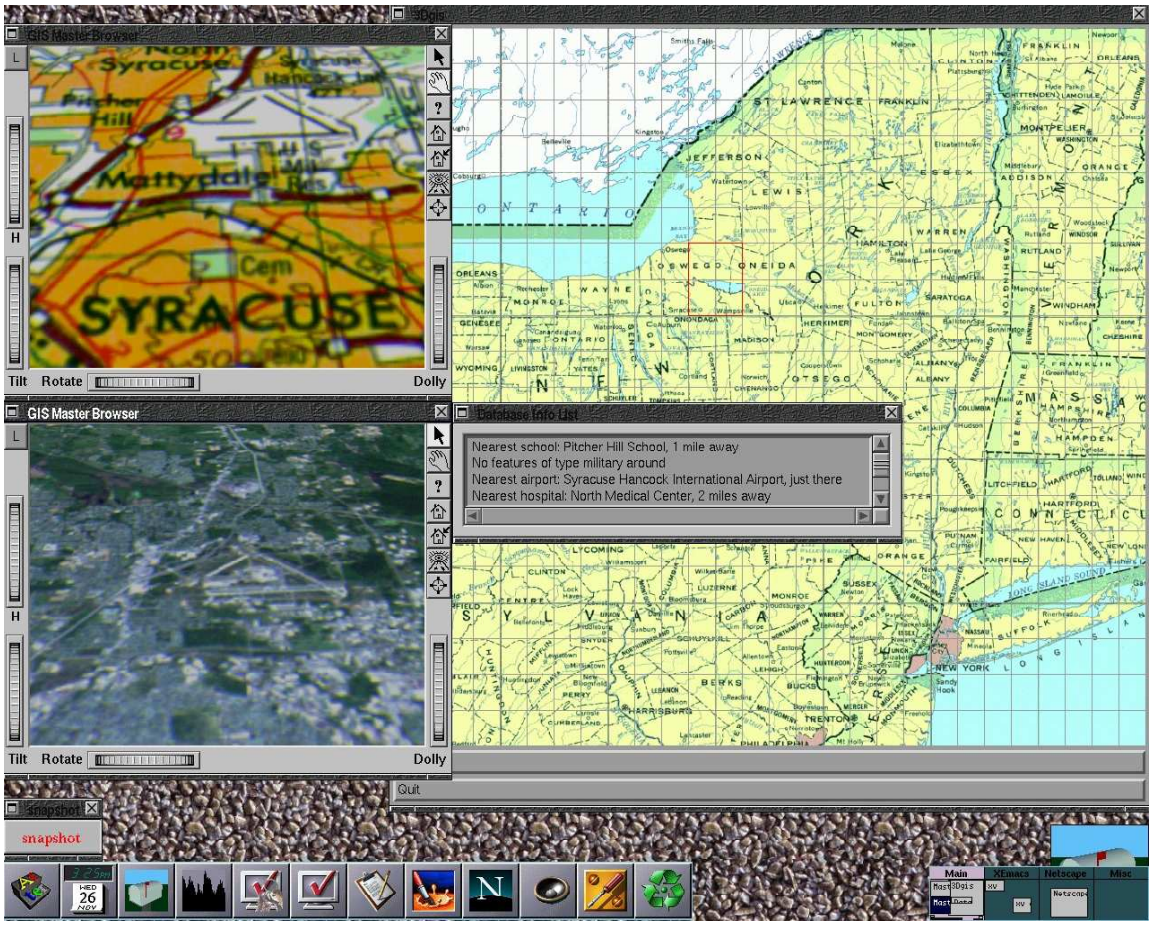

Figure 17: 3D GIS. Two viewers displaying map and satellite image of Syracuse area. Control Center presenting over-all map of New York State. Database query pre-view window in center.

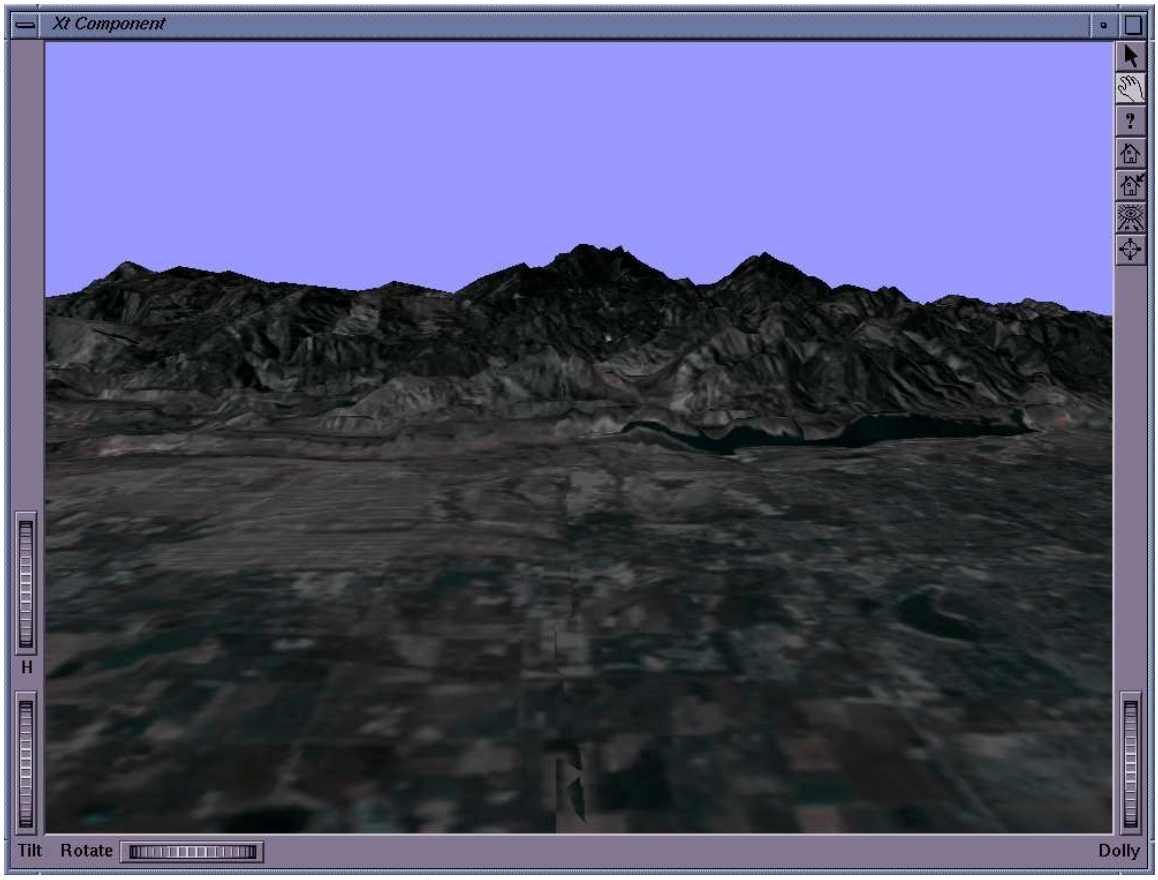

Figure 18: 3D GIS. Denver area in closer look with maximum level-of-detail.

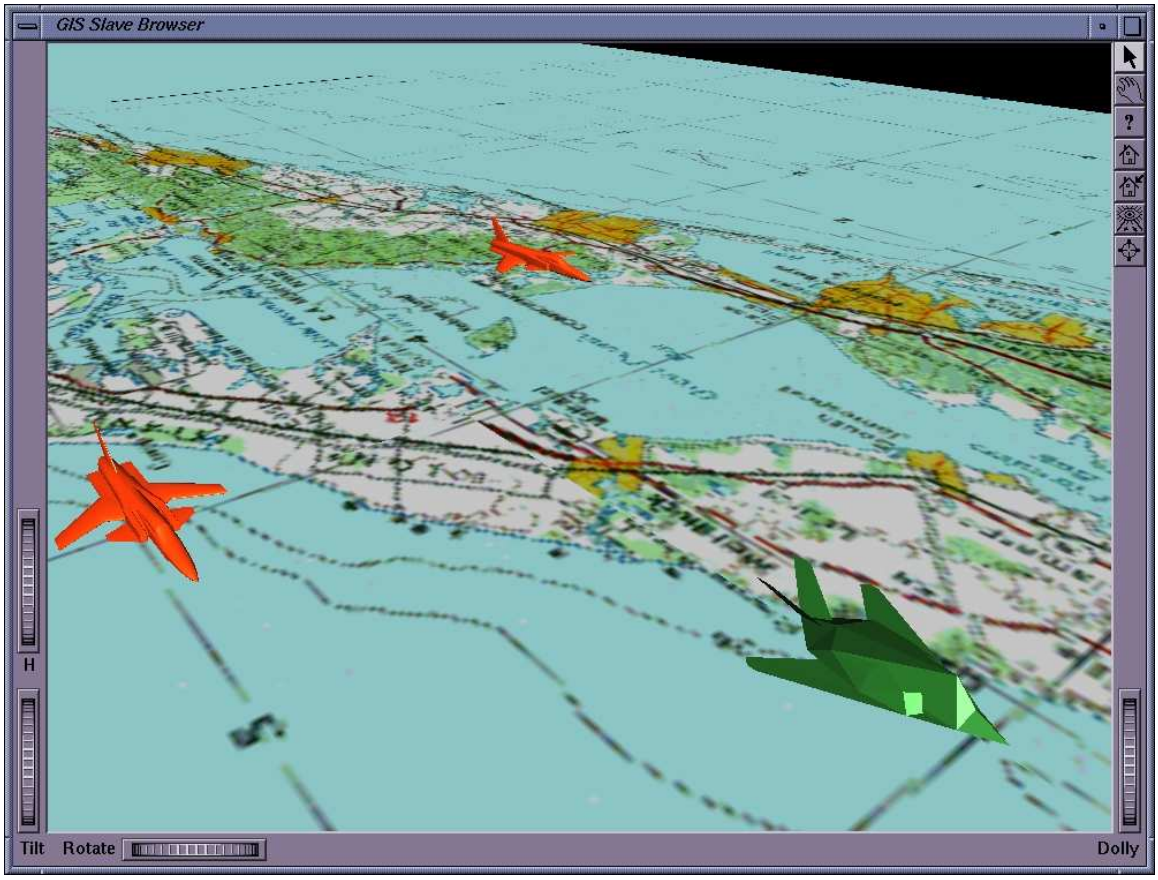

Figure 19: 3D GIS. Discrete event simulation extension. Two interceptors, Seeker and Poker, are pursuing unmanned aircraft with nerve agent on board. Part of Radar Operator Console.

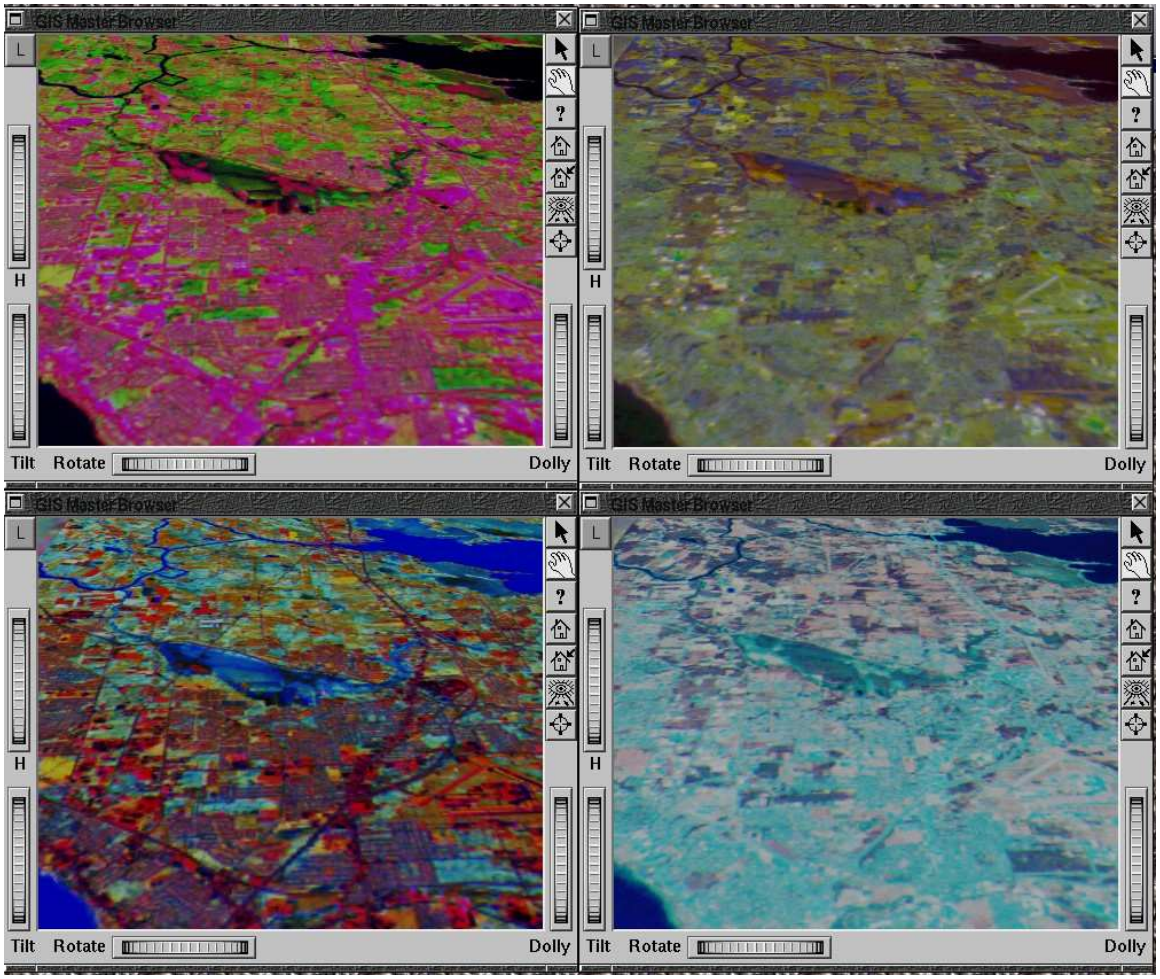

Figure 20: 3D GIS. Multiple data representations. Images acquired by applying 4 different filtering algorithms in ER Mapper.

IX. VITA

NAME OF AUTHOR: Pawel Roman

PLACE OF BIRTH: Augustow, Poland

DATA OF BIRTH: August 25, 1972

GRADUATE AND UNDERGRADUATE SCHOOLS ATTENDED:

Technical University of Gdansk, Poland

Franco-Polish School of New Information and Communication Technologies, Poland

AWARDS AND HONORS:

Supplementary Scholarship from Stefan Batory Foundation.

PROFESSIONAL EXPERIENCE:

Research Scholar, Northeast Parallel Architecture Center, Syracuse University, 1996

Graduate Research Assistant, Northeast Parallel Architecture Center, Syracuse University, 1997