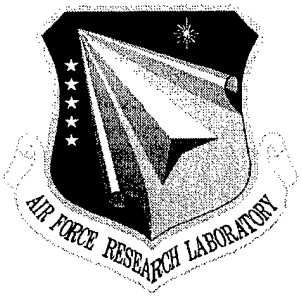
AFRL-SN-RS-TR-1999-77
Final Technical Report
May 1999



DYNAMIC DATABASE FOR SENSOR FUSION

Atlantic Aerospace Electronics Corporation

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. A0-F070

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

AIR FORCE RESEARCH LABORATORY
SENSORS DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

DTIC QUALITY INSPECTED 4

19990607 089

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-SN-RS-TR-1999-77 has been reviewed and is approved for publication.

APPROVED:

ROBERT E. PURPURA
Project Engineer

FOR THE DIRECTOR:

ROBERT G. POLCE, Acting Chief
Rome Operations Office
Sensors Directorate

If your address has changed or if you wish to be removed from the Air Force Research Laboratory Rome Research Site mailing list, or if the addressee is no longer employed by your organization, please notify AFRL/SNRD, 26 Electronic Pky, Rome, NY 13441-4514. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

DYNAMIC DATABASE FOR SENSOR FUSION

Paul Baim

Contractor: Atlantic Aerospace Electronics Corporation
Contract Number: F30602-97-C-0304
Effective Date of Contract: 31 July 1997
Contract Expiration Date: 1 June 1998
Short Title of Work: Dynamic Database for Sensor Fusion
Period of Work Covered: Jul 97 – Jun 98

Principal Investigator: Paul Baim
Phone: (781) 890-4200 x3260
AFRL Project Engineer: Robert E. Purpura
Phone: (315) 330-7685

Approved for public release; distribution unlimited.

This research was supported by the Defense Advanced Research Projects Agency of the Department of Defense and was monitored by Robert E. Purpura, AFRL/SNRD, 26 Electronic Pky, Rome, NY.

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED	
	May 1999	Final	Jul 97 - Jun 98

4. TITLE AND SUBTITLE

DYNAMIC DATABASE FOR SENSOR FUSION

5. FUNDING NUMBERS

C - F30602-97-C-0304
PE - 62301E
PR - D844
TA - 00
WU - P5

6. AUTHOR(S)

Paul Baim

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Atlantic Aerospace Electronics Corporation
470 Totten Pond Rd
Waltham MA 02154

8. PERFORMING ORGANIZATION REPORT NUMBER

DFR-1401

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Defense Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington VA 22203-1714

AFRL/SNRD
26 Electronic Pky
Rome NY 13441-4514

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

AFRL-SN-RS-TR-1999-77

11. SUPPLEMENTARY NOTES

AFRL Project Engineer: Robert E. Purpura/SNRD/(315) 330-7685

12a. DISTRIBUTION AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This effort developed a prototype Dynamic Database front-end using Virtual Reality Modeling Language, Object Design's Persistent Storage Engine, and custom Java code. This approach produced a self-contained web browser based tool for displaying and manipulating complex information typical of anticipated products from the Dynamic Database. The result is visually appealing, compact, and runs on most platforms using readily available low- or no-cost software.

14. SUBJECT TERMS

VRML, Dynamic Database, Browser, Java, Visualization, Object Oriented

15. NUMBER OF PAGES

60

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Standard Form 298 (Rev. 2-89) (EG)
Prescribed by ANSI Std. 239.18
Designed using Perform Pro, WHS/DIOR, Oct 94

Table Of Contents

1 Introduction

One of the most fundamental requirements for any system aimed at comprehensive representation of the battlespace is the capability to accept very high data volumes while maintaining accurate and up-to-date storage and rapid retrieval of information. Conventional database systems achieve efficient retrieval through methods developed to exploit the largely static nature of the data they store. These systems are aimed at archival storage of large volumes of continually increasing data in which data, once entered into the system, rarely changes. In addition, few applications require guaranteed throughput: system response may be suspended for brief intervals while internal maintenance activities are performed. The battlefield situation does not support either of these design criteria: virtually all of the data is continually changing with addition and deletion of transitory information representing a large fraction of the activity, and life-and-death decisions do not allow suspended processing for maintenance activities. Unfortunately, database research has never focused on the problems unique to dynamic database requirements.

Atlantic Aerospace Electronics Corporation has designed, implemented, and fielded an innovative capability for use in DARPA's SAIP ACTD system. This component, Object Level Change Detection (OLCD), reduces the number of false alarms produced by automatic target detection algorithms in wide-area search applications. By maintaining a dynamically updated database of up to 1 million detected objects over an operational area of up to 40,000 square nautical miles, OLCD distinguishes between objects that have moved, appeared, disappeared, or changed their appearance. This information is then used to distinguish static, unchanging objects (probably false alarms) from changing objects (probably true targets), allowing use of continually evolving historical information to dramatically reduce numbers of false alarms while preserving the probability of detecting valid targets with comprehensive coverage of theater-sized areas.

OLCD supports real-time operation with SAR imagery from operational and planned sensors performing a broad variety of mission profiles and producing input pixel rates up to 2.5 million per second. Multiple, interleaved sensor inputs can be accommodated. It runs on six Silicon Graphics R4400 processors in the SAIP van and is currently undergoing evaluation with the U2-ASARS system. It supports sustained object detection rates of more than 300 per second and peak loads of about 1000 objects per second. Both stripmap and spotlight SAR modes are supported. The architecture is specifically oriented to allow straightforward scalability for capacity and throughput.

A key aspect of situation analysis is terrain context. The behavior and objectives for a given deployment of forces can only be discerned against a background of terrain, water, roadways, trafficable and untrafficable ground, line-of-sight, and other factors that influence troop and vehicle movement and weapon effectiveness. Historically, terrain has been viewed as a static element represented by a map. The analyst or planner must overlay current information from sources such as weather reports, battle damage reports, and new construction to arrive at a clearer picture of the true terrain situation. The availability of accurate and timely information of these types can vary widely. Pivotal engagements have been determined by the destruction or creation of a bridge, or the arrival of fog. One goal of real time representation of the battlespace must be to provide up to date terrain information. Intelligence resources currently used for target and force detection must also be used for terrain analysis and new sensor systems may be necessary. Finally, storage and retrieval of this highly dynamic information must be provided.

2 Technical Discussion

2.1 Approach

We originally proposed an effort to develop a prototype object-oriented database to represent, store, and retrieve comprehensive terrain information and to investigate sensor phenomenology as applied to terrain analysis for detecting and interpreting changes of operational significance. The basis of this approach was work, in cooperation with other contractors, as part of a coordinated set of development efforts in which we would use raw materials in the form of data, schemas, and derivative data products from the other efforts, to be hosted and displayed via our database prototype. The central motivation of this combined effort was to establish basic capabilities and investigate key feasibility aspects in advance of the start of the procurement process for the full-blown Dynamic Database Program. This effort would form an abbreviated proof-of-concept of the ideas generated by the two working panels convened earlier. The original approach included:

Architecture and Design – In concert with other component developers, modify and/or augment our OLCD architecture to account for differences in the content and use of terrain-specific information.

Implementation – Based on our architecture and design, implement a database system built on top of the commercial object-oriented database ObjectStore (by Object Design Inc) and support integration of our component as necessary.

Terrain Phenomenology –analyze new and existing data to determine candidate features for terrain change detection and perform experiments to assess the suitability of each feature. High value features would then be implemented in a separate component that generates terrain analyses in a form suitable for storage in the dynamic terrain database.

Unfortunately, technical problems in data collection and schedule and plan difficulties precluded our ability to follow the specific original plan. In concert with the Sponsor, we redirected our effort to achieving the fundamental goals using an alternative approach. Given the absence of experimental data, the essential change was increased effort in prototype development and visualization using the portion of the program funds originally intended for data analysis. The content for the prototype would rely on synthetic data as opposed to experimentally derived data. A more subtle shift was away from a full-blown architectural concept to a concept focussed on delivery of DDB-derived information "packages" as mini-DDBs suitable for visualization and manipulation by an end-user. The goals of this approach were to:

Portray the 4D Situation – Represent the spatial and temporal information for a portion of the battlespace sufficient to capture the entities present and their relationships of military significance.

Support Manipulation "At Will" – Provide a "browsable" representation that does not impose a static picture of the situation but allows the user to understand the evolution of the information in time and space.

Minimize Bandwidth Consumption – Avoid reliance on a high-speed continuous data stream from a central server. Such a connection would not be available to every potential user of this information, and would also limit the widespread dissemination of capability prototypes for the DDB development.

Maximize Infrastructure Reusability – Produce a generalized "template" for rapid development of similar end-user tools.

Minimize Development Cost – Maximize use of off-the-shelf software components to minimize new development cost and risk.

Based on these goals, a functional architecture was defined and candidate technologies were assessed, including visualization tools and database technologies.

2.2 Architecture

A functional architecture for our prototype is shown in Figure 1. A self-contained visualization front-end communicates with the central DDB through a backend server process. This front-end "client" requests information using a standardized query language and receives results as "information packages" containing appropriate information for all objects and attributes specified in the query. A query may specify geographic and temporal extent of interest and may constrain the information of interest using additional qualifiers such as type or quantity.

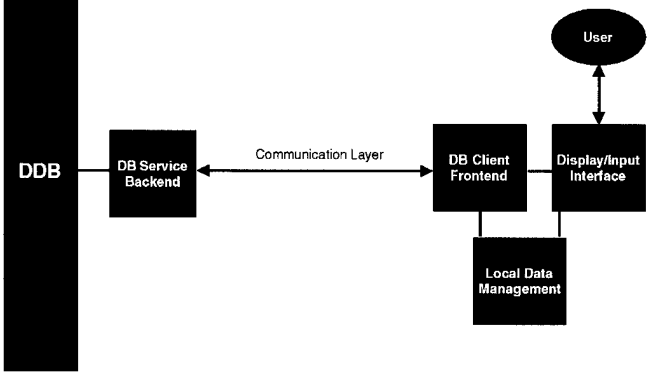


Figure 1 – Functional Architecture

Our visualization tool uses a common "notebook metaphor." The result of any request by a user will be a package of information that encapsulates the fragment of the DDB responsive to the request. A set of map objects with collateral text, statistics, image chips, and changes in these items over a selected time period, would be an example of such a package. We developed the capability to extract, encapsulate, and present such a package to an end-user in a dynamic page-oriented style similar to information-rich presentations common on the World Wide Web. This approach has several benefits:

- Compactness – the information transmitted to the browser is compact object data, not rasterized or heavily formatted binary images
- Selfcontained – the object data is complete in that visualization can be performed on-the-fly without additional information from the server.
- Loose Coupling – most manipulation of data can occur on the browser-side without interaction with the DDB server processes. In fact, notebook "packages" can be saved as

durable products and used for prototypes or archived as engineering results or for other record-keeping purposes.

- High Performance – because a notebook data package represents a small DDB fragment, simple database technology can be used to temporarily store and manipulate the information including query processing and data mining, without the overhead of a massive database.

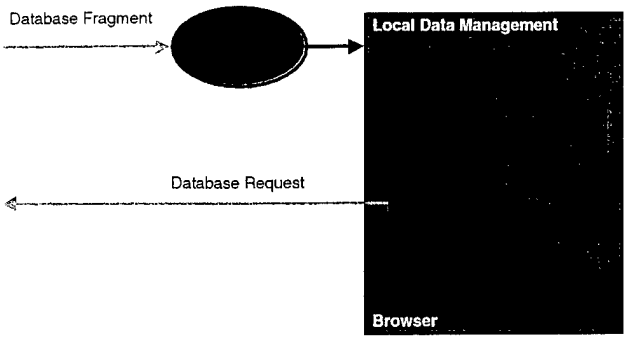


Figure 2. Visualizer Detail

2.3 Enabling Technologies

Three enabling technologies were identified to provide the essential elements of the functional architecture:

Web Browsers – Web browsers provide a flexible infrastructure for retrieval and display of complex information. As cross-platform tools of considerable power available to everyone without charge, browsers are compelling vehicles for delivering complex information content.

VRML – The *Virtual Reality Modeling Language* (VRML) is a data description language, developed to satisfy the demand for 3D-appearing graphics in the low-bandwidth heterogeneous environment of the worldwide web. Using VRML, network servers can supply a description of a scene, but allow the client (browser) to render the scene, thus drastically reducing the computational requirements for the server and the bandwidth requirements of the network. A VRML rendering engine adds the necessary 4D visualization capability to enable use of a conventional web browser as a DDB visualization tool.

PSE – ObjectStore's Persistent Storage Engine (PSE) technology supports creation of an object-oriented database on the user's local machine. This means that data can be downloaded,

displayed, studied and changed without changing the permanent data repository which is located on a remote server. The capture of an information "package" from the central DDB in a locally persistent form is essential to achieving the minimum bandwidth, maximum performance goals of this effort.

A secondary, but significant aspect of these technologies is their ready availability with minimal or no cost on a variety of platforms including PCs and workstations.

A schematic of the use of these technologies to compose a complete visualization tool is shown in Figure 2. For this effort, the connections to the DDB server were not implemented as no DDB server was available in advance of the DDB program start.

Several technologies were assessed for achieving communication with the server when it is developed. These include CORBA, Remote Method Invocation, Common Graphics Interface, UNIX Sockets, File Transfer Protocol, etc. A graphical representation of these alternatives is shown in Figure 3.

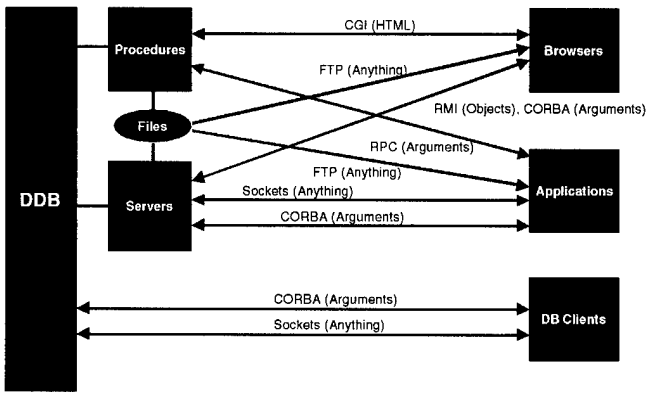


Figure 3. Communication Alternatives For Different Client Types

3 Prototype Design

The prototype consists of two main components: the database creation tool, and the data viewer. Each of these is accessed via hypertext pages viewed with a web browser. A dataflow for the prototype is shown in Figure 4.

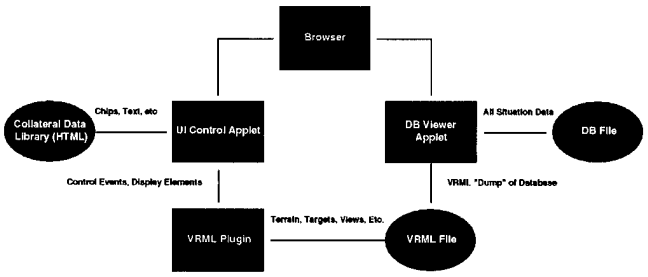


Figure 4. Prototype Dataflow

The *Virtual Reality Modelling Language* (VRML) is a data description language, developed to satisfy the demand for 3D-appearing graphics in the low-bandwidth heterogeneous environment of the worldwide web. Using VRML, network servers can supply a description of a scene, but allow the client (browser) to render the scene, thus drastically reducing the computational requirements for the server and the bandwidth requirements of the network. For more information, see the VRML-FAQ list at `http://vag.vrml.org/ VRML_FAQ.html`.

Java is an object-oriented programming language developed by Sun Microsystems. Modelled after C++, it is relatively simple, and extremely portable. Java mini-applications (*applets*) can execute within many popular browsers, which allows a web developer great flexibility in page design. For more about Java, see `http://java.sun.com`.

The *External Authoring Interface* (EAI) is an extension to VRML. It allows communication between the VRML world and other elements within the browser. A developer using VRML with the EAI can route event signals between VRML worlds, HTML documents, and applets, making real the possibility of powerful, flexible, interactive web pages. As of this writing, the EAI has not been formally accepted as part of the VRML standard. Current information about the EAI can be found at `http://www.vrml.org/WorkingGroups/ vrml-eai/ExternalInterface.html`.

The database creation tool uses ObjectStore's Persistent Storage Engine (PSE) technology to create an object-oriented database on the user's local machine. This means that data can be downloaded, displayed, studied and changed without changing the permanent data repository which is located on a remote server. Object Design, Inc. provides two Persistent Storage Engine (PSE) products for object-oriented database programming (http://www.odi.com/products/pse.html). These PSE products are intended for single-user Java applications and applets requiring persistent object storage. PSE Pro for Java is completely written in Java and has a small footprint enabling it to be quickly downloaded with Java applets. ObjectStore PSE for Java is similar to PSE Pro, but has less functionality and is distributed free of charge. The dynamic database proof-of-concept uses the latter. The portability of Java applications and applets, together with freely available browser applications for many platforms, make the approach of building a Java-based valuable by promoting distribution of data and applications in a timely manner. The user can always obtain the most up-to-date applet with which to view and analyze the data.

3.1 Theory of Operation

Generally speaking, the database creation tool is acting as a simulation of an eventual Dynamic Database. For the purposes of prototype, when changes are committed to the database, an updated VRML document is automatically generated. When the data viewer loads, it gets the latest available page (or an older page, if one is specifically requested), displays it, and allows simple manipulations via a Java applet and the EAI.

3.1.1 Database creation tool

3.1.1.1 Database API

Database functionality is provided through the Java API which primarily provides functions to:

- create, open and close databases
- start and end transactions

- store and retrieve persistent objects
- store collections of objects with indexed lookups

There are five main classes with appropriate methods which provide the programmer's interface to the database functionality of PSE:

- The **ObjectStore** class defines system-level operations not specific to any database. For example, initializing and shutting down PSE is controlled through this class, as well as setting various system level parameters.
- The **Database** class represents databases and provides methods to open, close, and destroy databases. It also provides an interface for creating and retrieving database roots. Roots are used to label persistent objects that an application wants to retrieve by name.
- Databases consist of **Segments** which provide physical clustering of objects. For PSE, there is a migrate method defined on the ObjectStore class that allows you to specify the segment where an object should be placed. In the PSE products there is only one segment and so migrate has no effect.
- The **Transaction** class provides interfaces for beginning, committing, and aborting transactions. PSE supports single-user multi-threaded applications.
- The **Persistent** class is the root of all persistence-capable classes. It defines storage for tracking an object's persistent status (e.g. whether the object has been fetched, dirtied, evicted, etc.). It also defines methods that give you control over fetching and eviction of objects. There is also a check to see if an object is persistent.

3.1.1.1.1 Persistence-Capable Classes

PSE provides an interface for storing and retrieving Java objects. Persistent Java classes, and their fields and methods are defined the same way as transient Java classes - declare classes to be persistence-capable and then use standard Java constructs to create and manipulate both persistent and transient instances.

The Java API uses a postprocessor tool that analyzes the class files of persistent classes. The tool extracts schema information and inserts annotations that automatically fetch objects from the

database when they are first accessed and automatically write objects back to the database when they have been updated.

All persistence-capable classes defined by applications inherit from PSE's abstract base class Persistent. There are four kinds of persistence-capable classes:

- String
- Arrays
- Java primitive wrapper classes:

  Boolean
  Character
  Double
  Float
  Integer
  Long
- Application defined subclasses of Persistent

One can store objects of the Java wrapper classes **Boolean**, **Character**, **Double**, **Float**, **Integer**, and **Long** but in general, you cannot store objects of the standard Java classes other than those listed above. In addition, a library of useful persistence-capable classes is provided that includes analogs of the Java Vector and Hashtable classes.

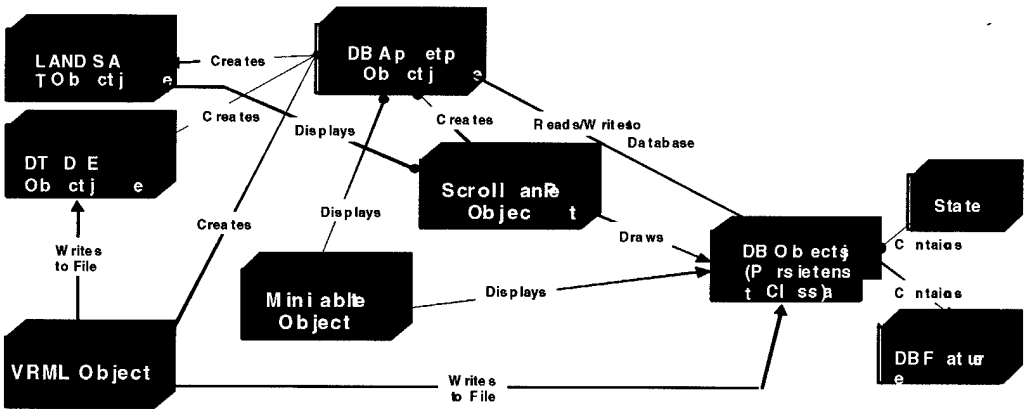


Figure 5. Major Client Applet Classes

3.1.1.2 Applet Architecture

The major classes which comprise the applet are shown in Figure 5.

- *DB Class* – the applet class which initializes all processing. Its primary functions are
- Read input files from web-server.
- Create and initialize local PSE database.
- Create panels for displaying and editing data.
- Accept user commands to manipulate data.
- *DBObject Class* – objects of this class form contain the data of interest. A DBObject contains objects of two other classes:
- *DBFeature Class* – these describe the measured features of the DBObject.
- *State Class* – an variable sized array of State objects forms a vector describing the evolution of position of the DBObject in time. The state of the object is its latitude, longitude, altitude and time of measurement.
- *DTED Class* – an object of the DTED class contains the digital terrain elevation data for the ROI which is inserted into the file ddbgen.wrl. file by the VRML object.
- *VRML Class* – a VRML object encapsulates the data the gets written out to the vrml-file ddbgen.wrl.

- *Landsat Class* – this class encapsulates the Landsat imagery data which is display by the ScrollPane object
- *Minitable Class* – this object displays the state and feature data for any object selected in the geographic ScrollPane and allows spread-sheet-like editing of object attributes.
- *ScrollPane Class* – this object displays the trajectories of database objects overlaying a Landsat terrain image in the Landsat object.
- The standard Java language classes and Java AWT classes.

3.1.1.3 Applet Client/Server Architecture

Figure 6 shows how the applet interacts with the web server. User-supplied information is entered into the appropriate fields of the HTML page. After the "Request" button is pushed, an http-request is made to the database server, which searches the database for data appropriate to the time and region of interest. An HTTP connection returns the data in the form of the three files mentioned above. These files are transient in nature, existing only for the duration of the session. The HTML page also receives the Java applet code and begins execution - reading the input data and creating the local database in the user's local file system. It then puts up the display windows for data manipulation.

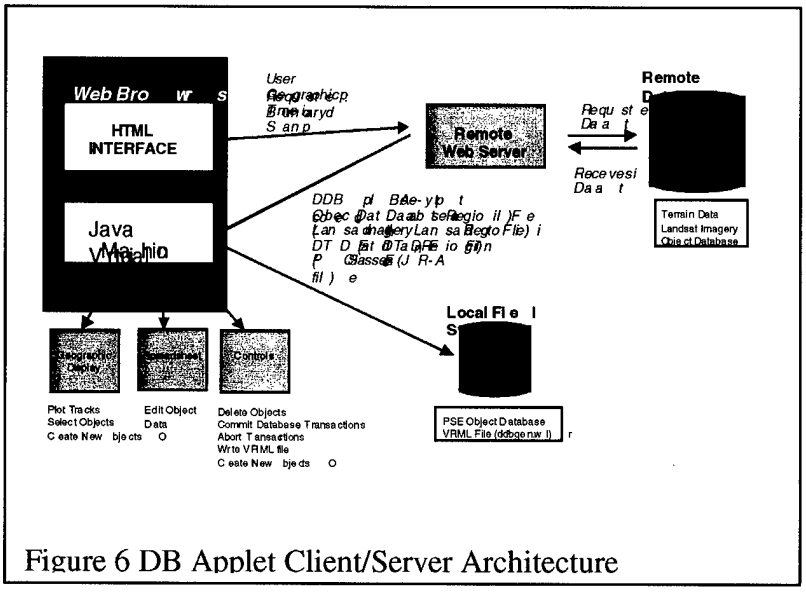


Figure 6 DB Applet Client/Server Architecture

3.1.2 Data viewer

Figure 7 is a diagram of the major elements and communication routes of the data viewer.

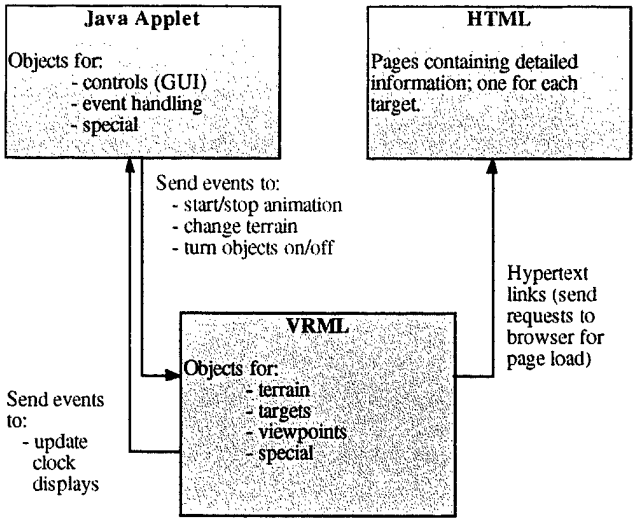


Figure 7. Data Flow

The following table provides a detailed description of nodes and event routes:

VRML			
WorldInfo	Contains basic information about the VRML world		
NavigationInfo	Contains basic information about the VRML world		
Background	Contains basic information about the VRML world		
Viewpoint	Defines a fixed point in space for convenient viewing (file contains several of these nodes)		
clock	Produces timer events	(constantly, whenever enabled)	all motion nodes and the y-interpolator node, updating time
startinggun	Script node; used primarily for communication with the Java control panel	continue animation signal from Java	clock, enabling it and setting start & stop parameters
			all target motion nodes, updating time
		set animation signal from Java	clock, disabling it and setting start & stop parameters
			all target motion nodes, updating time
		showOpForce signal from Java	all OpForce targets, enabling or disabling them

		showBlueForce signal from Java	all BlueForce targets, enabling or disabling them
target	Describes a target (file contains several of these nodes)	selection (e.g., mouse click)	browser, requesting target information be loaded into HTML window
target motion	Describes the location of its attendant target with respect to time (file contains several of these nodes–one for each target node)		
terrain	Describes the terrain in the world	changing from flat to DTED or DTED to flat	y-interpolator node, updating elevation data
y-interpolator	Script node; used to interpolate Y-values of target locations as they move across the terrain	time update signal	all target nodes, inserting new Y value in position vector
Java			
various GUI formatting objects	Create and place GUI objects within the applet panel		
init and start	Initialization; sets up the GUI, the External Authoring Interface, and callbacks for GUI objects		
callback	Catches clock events from VRML	VRML clock change	simulation time display and simulation time scrollbar
action	Catches Java GUI events	button event	VRML startinggun
handleEvent	Catches Java GUI scrollbar events	scrollbar event	VRML startinggun

3.2 Notes on Future Implementations

Some notes on possibilities for future client implementation are:

- The client could have a local version of the database (or a portion thereof), and the user could issue queries. VRML could be generated locally and updated more frequently.
- The client should have the ability to make database entries, which must be routed back to the server.
- The target data window could employ hypertext links to access reference materials about targets (e.g., vehicle capabilities, known availabilities, etc.).
- The viewer should be updated dynamically, if the user is viewing the most current available information. In this case, objects would be created, move, and be destroyed as new data became available.

- The viewer could allow comprehensive content subselection and viewing choices, possibly in concert with or by way of local database queries.
- A quadtree-based implementation of terrain would allow for quicker animation and increased realism. Similarly, levels-of-detail (LOD) nodes could be employed to increase the rendering speed of objects, especially when viewing areas are densely populated.
- Intelligent guided tours could simplify interaction with the 3D data, allowing the user to concentrate more on content and less on navigation through the scene.

4 Prototype Operation

This section provides a description of the system requirements for prototype operation, and some notes on how to install the prototype software.

Generally speaking, the prototype is intended to operate within a common hypertext browser, such as *Netscape* or *Internet Explorer*. Therefore, there are no specific system requirements, other than what the browser requires. However, some of the technologies used for the prototype–such as VRML, VRMLScript, and the EAI–are new enough that they are not yet universally supported by browsers, so finding a browser/plug-in combination that works can be a challenge.

The best advice is to pick a VRML plug-in that will work on your system first, then choose the browser version that the plug-in requires.

4.1 System Requirements

4.1.1 Browser

Generally, any browser that will support Java 1.1 and VRML on your system will suffice, however, the VRML capability must include support for VRMLScript and the EAI (see below). The browser must also support "trusted applets", since the database creation tool is an applet that writes a file to the client machine's file system. Netscape *Navigator* (or *Communicator*) and Microsoft *Internet Explorer* seem to be the most popular, and can be downloaded from `http://home.netscape.com/comprod/mirror` and `http:// www.microsoft.com`, respectively.

4.1.1.1 Netscape 4.0.5 Considerations:

In order to use the application with Netscape, the user must invoke "codebase" principles. This means modifying the Netscape preferences file while Netscape is not running to add the line:

```
user_pref("signed.applets.codebase_principal_support", true);
```

The appropriate file to add this to depends upon the operating system:

UNIX: $HOME/.netscape/preferences.js
Windows: C:\Program Files\Netscape\Users\<User Name>\prefs.js
Macintosh: System Folder:Preferences:Netscape Users:<UserName>:Netscape Preferences

4.1.1.2 Internet Explorer 4.0 Considerations

If an applet is to be downloaded from a remote site, the remote site must be registered as a "Trusted Site". This is done in the Java Security menu for all platforms. If the applet is to be invoked from the local machine, no changes to the security levels need be made.

4.1.2 Java

Both *Navigator* and *Internet Explorer* support Java, but by default this capability is disabled. The capability can be enabled by editing your user preferences. If another browser is utilized, it will need to include a Java virtual machine.

Additionally, special Java class libraries are required for EAI usage. These are normally included with the VRML plug-in (see below).

4.1.3 VRML

If VRML capability is not included with your browser (and as of this writing, it typically is not), you will need a VRML plug-in. A list of these (with links for downloading) can be found at http://www.sdsc.edu/vrml.

Be aware that these plug-ins do not always conform to the VRML specification (which can be found at http://www.vrml.org). A good, comparative reference of conformance can be found at http://www.cs.brown.edu/people/gss/vrml/comparison. Pay special attention to the node support sections, as unsupported nodes will result in objects missing from the display.

The VRML plug-in must support VRMLScript as a scripting language, and it must support the EAI. Inline nodes and TouchSensors are also required. Texture-mapping is required if terrains

are to be correctly rendered; Background nodes improve appearance but are not strictly necessary. Extrusions, Text, PlaneSensors, CylinderSensors, SphereSensors, and colorPerVertex are not required.

4.1.4 EAI

The External Authoring Interface (EAI) is needed in order for the control panel to communicate with the VRML display. As of this writing, it has not formally been accepted as part of the VRML standard. It is close, however, and several plug-ins do support its use. The most broadly-compatible of these is *CosmoPlayer* from Cosmo Software, available at `http://cosmosoftware.com`. Also, for a list of plug-ins, some of which also support the EAI, see `http://www.sdsc.edu/vrml`.

At the time of this writing, the list of browsers that support the EAI includes:

Netscape *Communicator 4.04* for Macintosh, Win32, or IRIX
Netscape *Navigator 3.01* for Win32
Netscape *Navigator 3.01S* for IRIX
Microsoft Internet Explorer 3.0 or 4.0 for Win32

The list of VRML 2.0 plug-ins that support the EAI includes:

Cosmo Software Cosmo Player 2.1 or later for Macintosh
Cosmo Software *Cosmo Player 1.0beta3* or later for Win32
Cosmo Software *Cosmo Player 1.0.2b3* or later for IRIX
Intervista *WorldView 2.0* or later for Win32
VRWave
Blaxxun Interactive *CC3D*

The EAI includes Java class libraries that it needs to operate. Be sure that these are included with your VRML plug-in, and that they are installed in the proper location. Most plug-ins install these files automatically.

4.1.5 ObjectStore PSE

Optionally, a user can permanently install the ObjectStore PSE (free) or PSE Pro distribution in the browser's classpath. This improves the performance by not requiring repeated downloads of the PSE classes to the local machine. For more information, see http://www.odi.com/products/pse.html.

4.1.6 Prototype files

The final step is to locate the prototype files in a directory that may be accessed by your browser. Supplying the appropriate URL for that directory will start up the prototype.

The applet requires several "region" files which contain the data pertinent to the ROI. The applet requests data from the web server which then sends several files to the browser. These are:

DatabaseRegion – An ASCII file of objects for the ROI.
DTEDRegion – DTED Elevation data for the ROI.
LandsatRegion.gif - A LANDSAT gif image for the ROI.

These files contain the data that the applet will process to create the local PSE databases.

The local PSE database itself consists of two files: an .odb and an .odt file. The basenames of these files are set by the user in the DB.html file. If you transfer a database to another machine or another directory of the same machine, both files must be moved.

The user need not have any knowledge of the structure of any of these files.

4.2 User's Guide

The primary interface to the prototype is by way of your browser. Knowledge of the operation of this is assumed.

Similarly, the VRML plug-in you select will have its own interface and control panel. It is suggested that you familiarize yourself with the navigation controls within VRML worlds in order to maximize the utility of the data viewer.

4.2.1 Database creation tool

The DDBJ Applet is a java-based applet that uses ObjectStore's Persistent Storage Engine technology to create a persistent database on the user's local machine useful for on-line/off-line data manipulation, analysis, editing and display. The applet was written using Java 1.1 and uses PSE 1.2.

4.2.1.1 Invocation

The DB applet is invoked from the DB.html web page. The web page has entry fields for the geographic bounding box surrounding the region of interest (ROI) for the user. The page also requests a start and stop time defining the user's times of interest. The user must enter the east and west longitudes and the north and south latitudes of the bounding box in decimal degrees. The user enters the start and stop times of interest in DD-MM-YY HHMMSS format. One other piece of information requested of the user is an ASCII string denoting the name of the local PSE database that the browser will create.

Example:

West Longitude	-116.5 (degrees)
East Longitude	-116.25
North Latitude	35.60
South Latitude	35.33
Start Time	25-Mar-1997 01:20:00
Stop Time	25-Mar-1997 04:50:00
Local Database Name	LocalDB.odb (The other file LocalDB.odt will be constructed automatically)

4.2.1.2 Display

After all files are transferred to the browser, the applet window shows two panes and several buttons:

- A spread-sheet like interface which will show information about selected objects.
- A geographic display of the paths of the objects overlaid on the Landsat image for the region contained within the user selected bounding box.
- Buttons to control various interactions with the database.

4.2.1.3 About the Geographic Display

Each path is color coded according to the force to which it belongs. Right now there are two colors - blue for BLUEFORCE, and red for OPFORCE. Selecting a object by mouse-clicking on its path causes data about that object to be displayed in the spread-sheet. Fields in the spread-sheet can be changed by the user and saved. If the ROI is larger than the applet window, scroll bars enable the user to move around the image.

4.2.1.4 About the Spreadsheet

The spreadsheet allows the user to obtain information about any object which appears on the geographic display. Cells in the spreadsheet are either editable or non-editable. Non-editable cells, displayed in red text, correspond to label fields. Editable cells are free for the user to change. Information that the spreadsheet currently displays are (1) object position in latitude, longitude verses time, (2) object ID, (3) object type. (4) force membership, (5) html location of object chip.

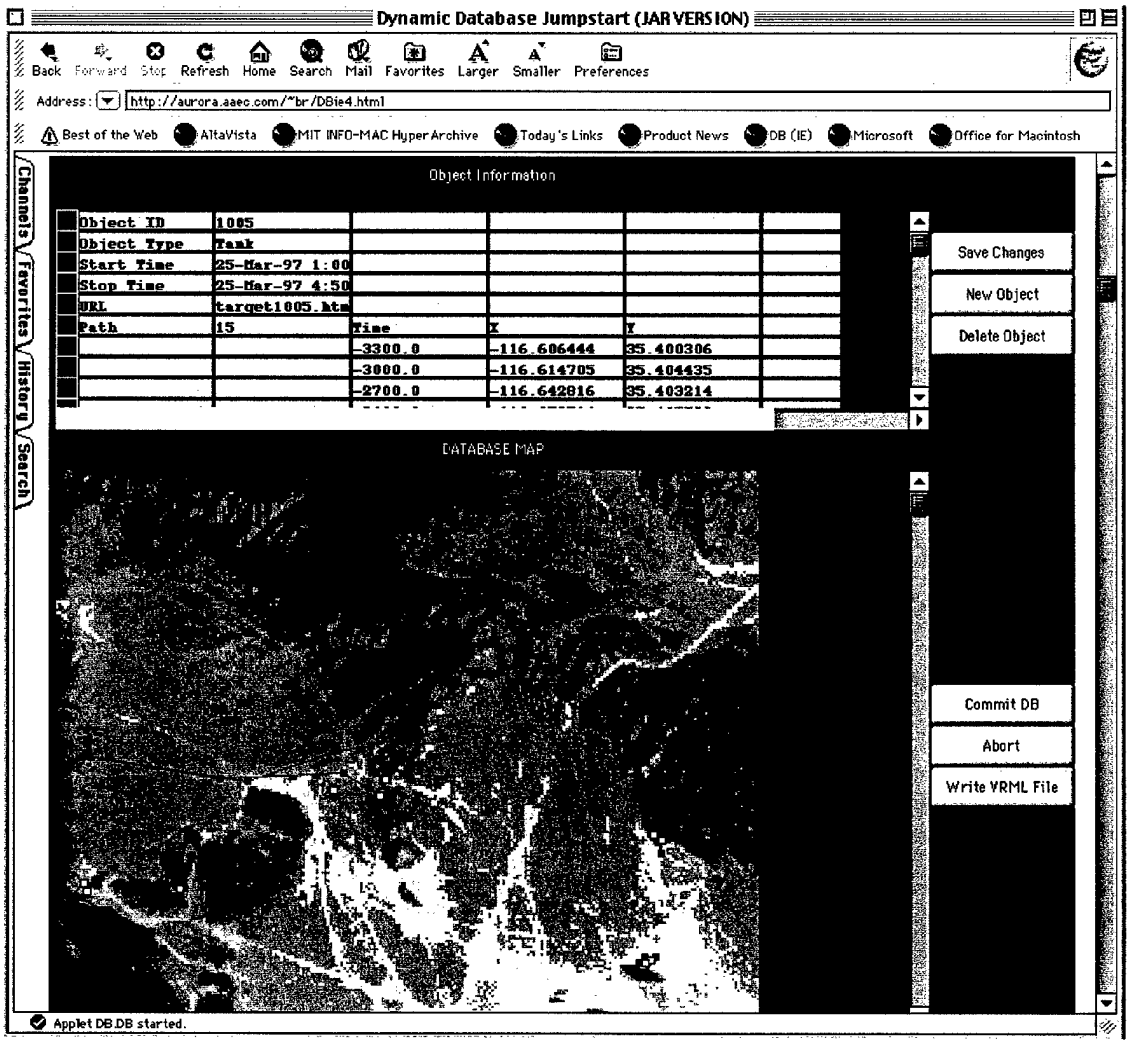


Figure 8. The track of an object is selected.

4.2.1.5 About the control buttons

Using buttons on the right-hand-side of the display, the user can perform various manipulations on objects and the database:

SAVE CHANGES	Saves changes to an individual.
DELETE OBJECT	Removes a selected object from the database.
NEW OBJECT	Creates a new database object
COMMIT DB	Commits all changes made to the database.
ABORT	Erases all changes made since the

	last **commit**.
WRITE VRML FILE	Writes a VRML file to the user's file system describing the dynamics of the data. The file name is ddbgen.wrl.

4.2.1.6 Using the New Button:

Once the NEW OBJECT Button is pressed, the Geographic Display changes modes. Mouse clicks in the Geographic Display cause the creation of a track for the new object. The spreadsheet is also changed to a blank table into which the user can enter data. As the mouse is

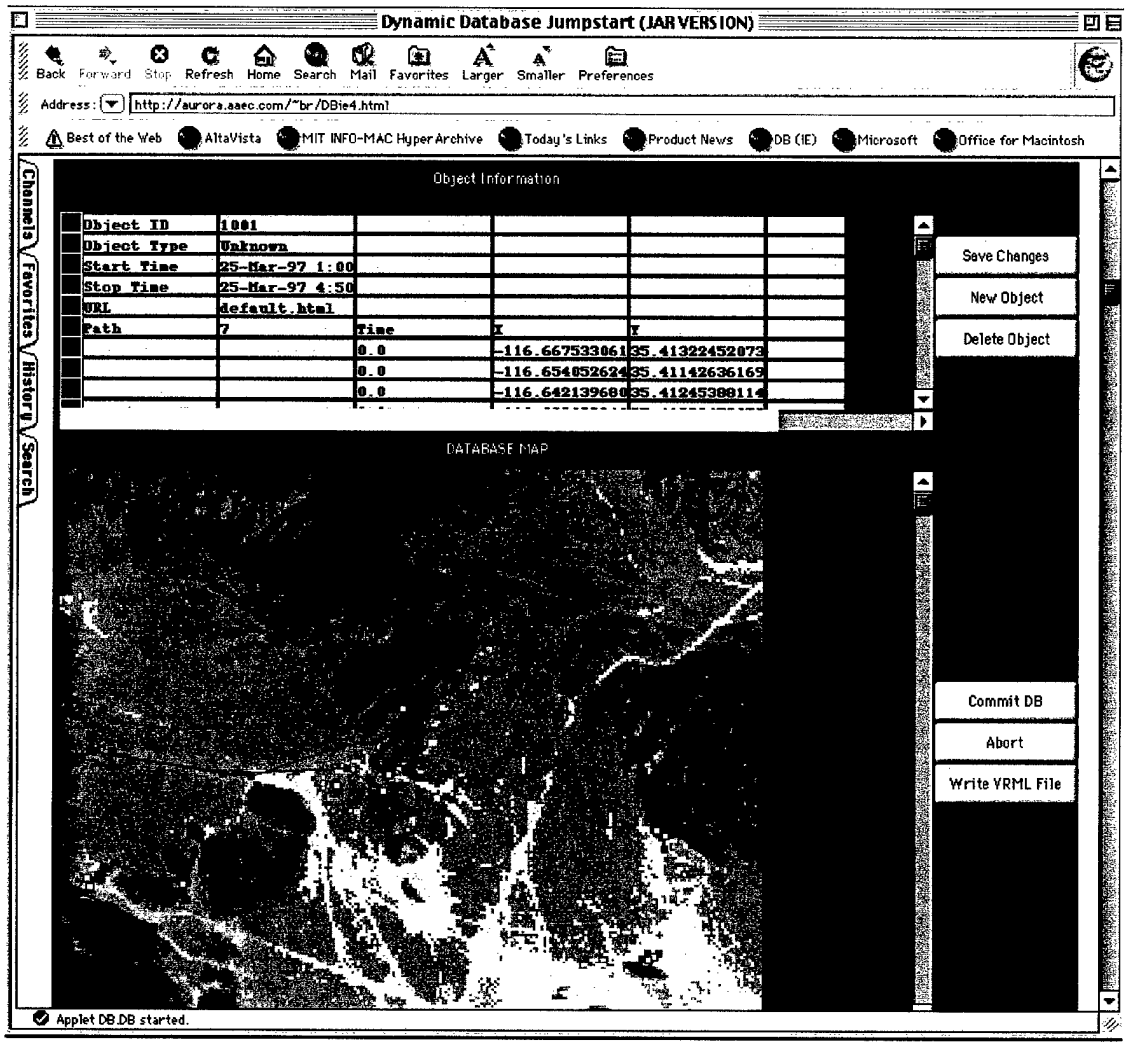


Figure 9. The track of a new object.

clicked on the geographic display, the path of the new track is displayed in yellow and the spreadsheet entries are automatically filled-in. Pressing the SAVE CHANGES button ends this mode and reverts to object-selection mode.

4.2.1.7 Using the COMMIT Button

Changes to an object, even if saved, do not become part of the persistent database until a commit is performed. When a COMMIT DB is performed, the data is changed inside the database.

4.2.2 Data viewer

The viewer allows you to examine the available class, position, and movement data using an animated 3D interface. You may 'move around' within this 'world' to gain perspective on the contents and their relationships. The scene may be set in motion, too, to illustrate the dynamic nature of the vehicles represented therein.

Usage of the data viewer is generally intuitive; the most difficult aspect is in mastery of VRML navigation controls. The layout of the viewer is described below, and each panel is detailed later in this section.

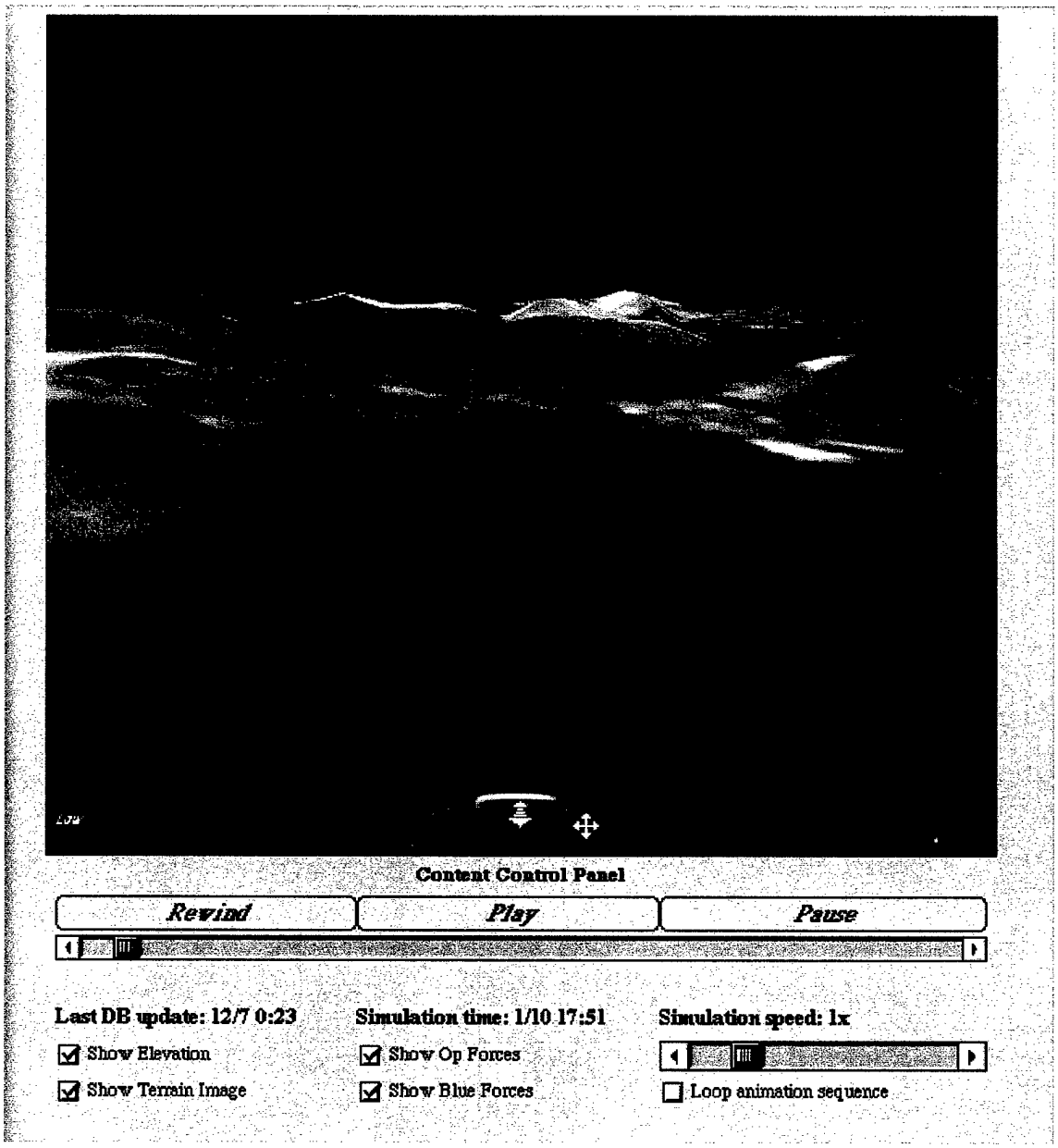


Figure 10. The data viewer.

4.2.2.1 Layout

The database viewer page contains two panels. The upper panel consists of the VRML window, including any 'dashboard' that your VRML plug-in utilizes. The lower panel contains the

associated display control pane. Additionally, specific target information from the database is loaded into a separate pop-up window when requested.

Note that if your browser's default page size is small, the page may be partially obscured. A page height of at least 800 pixels is recommended so that scrolling will not be required.

4.2.2.2 3D view panel

The primary viewer employs the VRML capability of your browser to present an interactive 3D display of the database contents. When the page opens, a "bird's-eye view" of the area is presented. Depending on the settings of your VRML browser, gravity may begin to act upon your avatar, so you may wish to turn this feature off.

Targets will be visible on the terrain below. A few other fixed viewpoints are provided, but primarily, viewing of the data will be via the normal navigation controls your browser (or plug-in) provides. Since the database may cover a very large area, and vehicles are relatively small, the displayed targets are enlarged as your viewpoint moves farther away from them so that they remain visible.

Selecting a target (e.g., clicking on it) while in navigation mode results in a portion of the database enter for that target to be displayed in another window (see below for more information). Note that selecting a target while in examine mode will move your avatar toward the target.

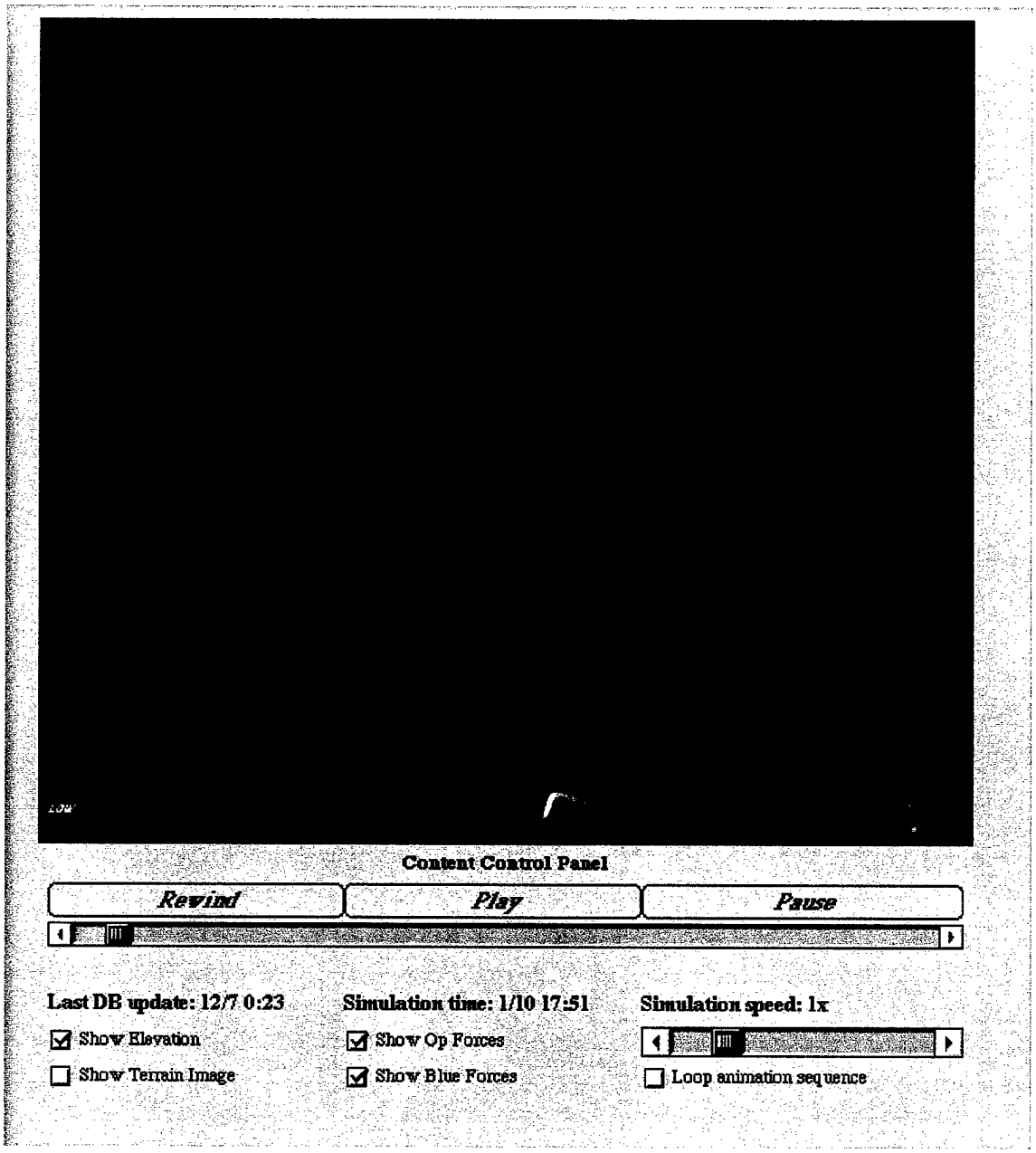


Figure 11. The viewer shown without a terrain image.

4.2.2.3 Control panel

This panel encompasses controls that affect simulation time, appearance of terrain, and the content to be viewed. Each is individually described below.

4.2.2.3.1 Time controls

The major function of the control panel is to affect the time of the view. Since the data show vehicles positions changing over time, the 3D view may be animated to reflect this dynamic aspect.

4.2.2.3.2 Time of DB update

The time of the last database update is displayed as a reminder of the 'age' of the information being viewed. This time is static within the context of a browser page load, as the view information is only updated at that time.

4.2.2.3.3 Simulation time

The time represented on-screen at any given instant is displayed on the panel. This is not a control; rather, it is a clock indicating the time shown in the 3D world.

4.2.2.3.4 Slider

The 3D world time is also displayed as a proportion by a scrollbar-like control. The further to the right side the box is within the bar, the later in time the view is. The slider may be manipulated to set the time of the view, either by dragging the box to the desired time, or by clicking on the bar to the left or right of the box, or by clicking on the arrows at the ends of the bar.

4.2.2.3.5 Play

Clicking on the *Play* button animates the view. The speed of the animation is determined by the speed control (see below). Clicking the *Play* button when the animation is already running has no effect.

4.2.2.3.6 Pause

Clicking on the *Pause* button stops the animation, if it is running.

4.2.2.3.7 Rewind

Clicking the *Rewind* button sets the time displayed in the viewer to its origin–the beginning of the data represented in the database query. This action does not explicitly start or stop the animation.

4.2.2.3.8 Speed control

This scrollbar sets the speed at which the view is animated. Like the *Simulation Time* scrollbar, this may be manipulated by dragging the box, clicking on the bar, or clicking on the arrows; however, the scrollbar has an exponential effect on the animation speed.

4.2.2.3.9 Loop animation

If this option is selected, the animation repeats continuously.

4.2.2.4 Terrain controls

4.2.2.4.1 Show elevation

If this option is selected, DTED elevation data is displayed in the viewer. Otherwise, a flat landscape is shown. Navigation and animation are much smoother with elevation turned off.

4.2.2.4.2 Show terrain image

When this option is selected, satellite imagery of the area is draped onto the elevation information, making a realistic depiction of the landscape. Otherwise, a monochromatic terrain is employed. Again, navigation and animation speed and fluidity are greatly improved when the terrain image feature is turned off.

4.2.2.5 Content controls

The *Show Blue Forces* and *Show Op Forces* controls allow you to select which target vehicles are displayed.

4.2.3 Target Data window

Clicking on a target object in the 3D viewer results in detailed information about the target–if it is available–being displayed in this window. The browser's scrolling capability will allow you to view this information in its entirety.

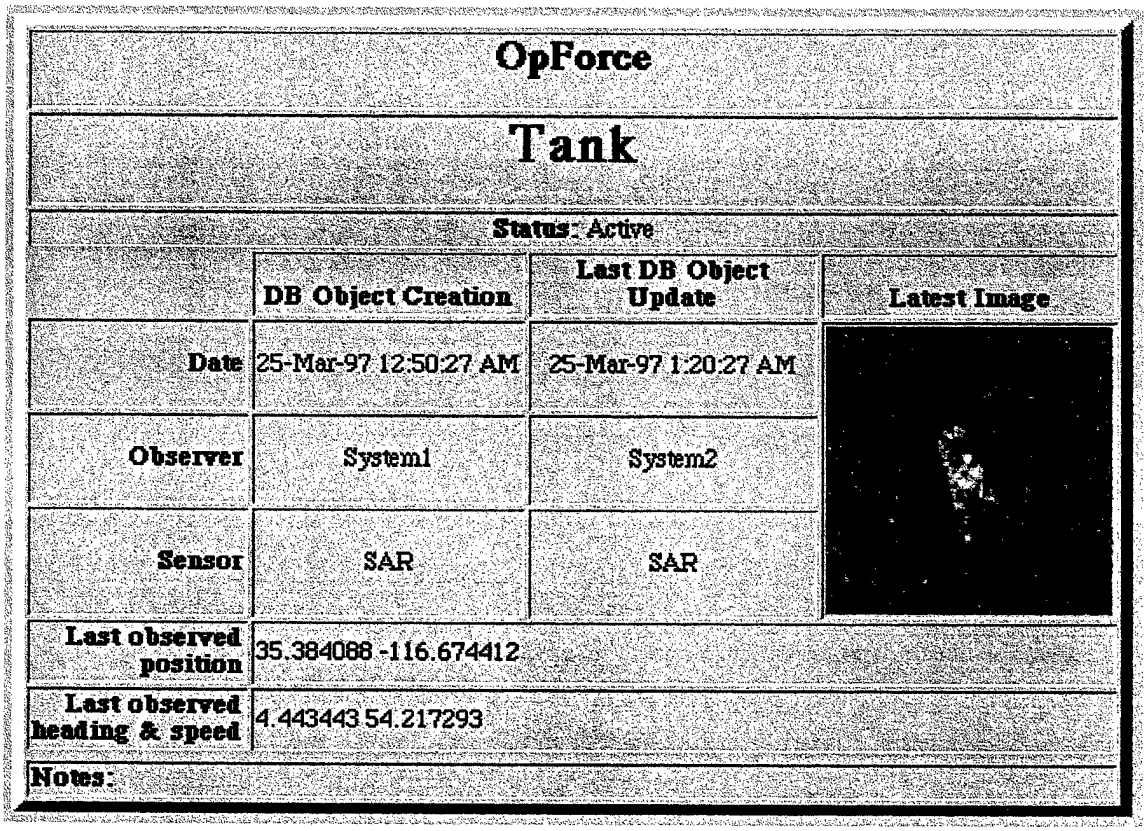


Figure 12. An example Target Data window.

4.3 Known Problems

Database creation tool	
DB.html file	The user cannot interact with the DB.html file yet. Bounding-box files are changed by editing the html file. Future versions will have an interactive home page.
Track editing	We cannot add to or modify the path of an existing track using the geographic display yet; it must be done using the spreadsheet. Future versions will allow mouse interaction.
Object deletion	When an object is deleted, the spreadsheet does not become blank.
Data viewer	

Reload	The EAI, which is used to allow the Java-based control panel to interact with the VRML-based 3D viewer, has a known problem with many browsers which results in the inability to properly load VRML after the EAI has been used. Consequently, the browser may crash if the demo page is loaded a second time in any given session, or if it is resized (most browsers require their VRML plug-ins to reload if the page is resized). This problem has been observed with WorldView, but not with CosmoPlayer.
Timer issues	On some platforms, clicking on the *Play* button for the first time after loading the page results in the animation jumping immediately to its end point. Simply clicking the *Play* button a second time starts the animation properly in this situation. This problem is typically not observed when *Loop Animation* mode is employed.

5 Conclusions

The experience developing this prototype has shown the value of the data package/browser-side approach to visualization of dynamic information. The technologies used are viable, though immature (VRML in particular). With the processing power available in current high-end PCs and workstations, on-the-fly visualization is a practical reality. As these technologies, or their successors, enter the mainstream, improved performance and robustness will allow high performance visualization for complex information structures exemplified by the products expected from the Dynamic Database. The current state of these technologies is certainly adequate to use this approach in developing both demonstrations and working prototypes with minimal development time and effort for delivery to the broadest class of users at minimum cost.

DISTRIBUTION LIST

addresses	number of copies
ROBERT E. PURPURA AFRL/SNRD 26 ELECTRONIC PKY ROME NY 13441-4514	5
ATLANTIC AEROSPACE ELECTRONICS CORP 470 TOTTEN POND RD WALTHAM MA 02451	5
AFRL/IFOIL TECHNICAL LIBRARY 26 ELECTRONIC PKY ROME NY 13441-4514	1
ATTENTION: DTIC-OCC DEFENSE TECHNICAL INFO CENTER 8725 JOHN J. KINGMAN ROAD, STE 0944 FT. BELVOIR, VA 22060-6218	2
DEFENSE ADVANCED RESEARCH PROJECTS AGENCY 3701 NORTH FAIRFAX DRIVE ARLINGTON VA 22203-1714	1
ASC/ENEMS WRIGHT-PATTERSON AFB OH 45433-6503	1