Modeling Battlefield Sensor Environments with an Object Database Management System

Mark A. Woyna
John H. Christiansen
Christopher W. Hield
Kathy Lee Simunich

Advanced Computer Applications Center
Argonne National Laboratory
9700 S. Cass Ave., Argonne, IL 60439
{woyna, jhc, hield,simunich}@athens.eid.anl.gov
(708) 252-2557

Abstract

The Visual Intelligence and Electronic Warfare Simulation (VIEWS) Workbench software system has been developed by Argonne National Laboratory (ANL) to enable Army intelligence and electronic warfare (IEW) analysts at Unix workstations to conveniently build detailed IEW battlefield scenarios, or "sensor environments", to drive the Army's high-resolution IEW sensor performance models. VIEWS is fully object-oriented, including the underlying database.

1 Introduction

The VIEWS Workbench was designed to allow an intelligence and electronic warfare (IEW) analyst access to a reference database via a convenient graphical user interface in order to build, display, and modify the detailed scenario specifications and ancillary data necessary to exercise a suite of IEW sensor performance models. IEW battlefield scenarios must include all entities in a friendly and/or a hostile force which could be detected by Army sensor systems. Every vehicle, aircraft, fixed site, radio, radar, jammer, and weapon on one or both sides of a simulated conflict must be represented. Once these myriad entities are deployed geographically on a high-resolution terrain grid and transportation network, their movement and behavior must be tracked and updated very frequently (e.g., once per simulation second) to build a realistic sensor environment.

The VIEWS Workbench is a fully object-oriented representation of the battlefield, supported by a Versant object-oriented database [Versant92a]. It includes an embedded Combined Arms Army Movement Model (CAAMM) [PAR86] which simulates vehicle-level movement. The user interface is also an object-oriented windowing system, based around the XView toolkit and OPEN LOOK.

1993 ACM 0-89791-592-5/93/0005/0499

The VIEWS Workbench provides the following general capabilities:

- Interactive construction and maintenance of an IEW reference database;
- Interactive/automatic construction of Tables of Organization and Equipment (TO&E's) for military forces to be simulated;
- Interactive/automatic dynamic scenario simulation, aided by an embedded Combined Arms Army Movement Model;
- Automatic generation of mutually consistent inputs for multiple sensor performance models.

The VIEWS Workbench's operating environment is characterized as follows:

- Hardware Platform: Sun SPARCStation; portable to any UNIX workstation which can support the XView toolkit and the Versant ODBMS.
- Operating System: UNIX.
- Database Management System: Versant Object Database Management System Version 1.7.
- User Interface Standard: Open Look.
- Source Language: C++ 2.1
- Operating Mode: Initial system is single-user, but design supports multi-user operation with each Scenario Library lockable by one user at a time.

2 Data Model

The VIEWS Data Library is the repository for all data generated through the VIEWS user interface. The Data Library is partitioned into a Global Library and a set of user-defined Scenario Libraries. Each Scenario Library contains a set of user-defined Scenario Version Libraries. Each Library contains all the data necessary to support the IEW sensor performance models, including:

- TO&E data sets containing nations, military branches, military echelons, combat postures, military unit types and deployment templates, platform types, payload types, and communication net types.
- Force Rosters containing multiple instances of various unit types, platform types, payload types, and communication net types.
- Sensor Performance Model data sets containing sensor types, sensors, antennas, emitter types, emitters, communication nets, weather data, smoke data, cloud data, and vehicle detection signatures.
- Data sets containing technical specifications for telecommunication equipment.

All data are modeled as objects and are stored persistently in the object database management system. The current schema represents over 250 persistent classes containing over 1000 attributes and 4000 methods.

3 User Interface

The VIEWS Workbench user interface communicates with the user via an object-oriented UNIX graphical user interface toolkit. The User Interface Manager (UIM) and the Graphics and Imagery Manager (GIM) toolkits were written by ANL [Fuj91]. The toolkits are written in C++ and package the C interface of the XView and Xlib libraries used by OPEN LOOK into objects.

Each VIEWS window is itself an object in which toolkit objects are embedded, thereby capturing the functional behavior of the window. The methods of the VIEWS window objects communicate with the rest of the system by sending messages to other inline data objects, including persistent objects stored in the object database.

All actions are driven by user-initiated events such as a mouse click or a key press. The current system (version 1.1) has 250 window objects.

4 Database Management System

Although the VIEWS Workbench itself is fully object-oriented, it was originally designed and partially implemented with a relational database management system as the object store. Persistent classes were to be mapped to the relational database by embedding the necessary SQL code within persistent class methods. This approach allowed the programmer to treat the real world entities as objects within VIEWS while encapsulating the interface to the relational database within the objects. While this approach worked, several problems areas were identified:

Schema Support:

- The database schema language was the not same as the programming language, requiring that the C++ classes be normalized to produce a set of SQL create table statements. This required that someone on the VIEWS development team be proficient in relational database design.
- Complex structures had to be converted to an artificially "flattened" representation in a relational system.
- Programmers had to learn an additional language (SQL) to define the schema.

Additional Input/Output Code:

- The need to maintain 2 representations of data (database and application) was burdensome. Extensive code was needed to keep both representations of the object in agreement.
- Input/output code was needed to translate data from the database to the internal representation. This code was a significant fraction of the overall system code.

Application Language Interface:

- A loose binding existed between the application language (C++) and the database manipulation language (SQL).
- Programmers had to be proficient in 2 languages (i.e., C++ and SQL).
- SQL and C++ are not based on a common data model and data type set.

Performance:

- Reconstituting complex, normalized objects required joining many tables, a relatively expensive operation.
- Since all relational objects were cached on the database server, repeated reference to the same object required repeated fetching of the object from the database server.

An excellent solution to these problems was to convert the system to an Object-Oriented Database Management System (ODBMS). While most ODBMS's are proprietary, the benefits from moving to an object-oriented system outweigh this limitation. These benefits include:

Schema Support

- The database schema "language" is the same as the programming language (C++), and the schema is automatically captured from the application code.
- Object structures do NOT have to be "flattened" into another representation.
- No need for additional Input / Output or synchronization code. Objects are referenced as if they are in memory at all times.

Performance

- Client-side object caching makes accessing the objects efficient (most recently referenced objects remain within the cache).
- Improvements ranging from 10 to 100 times faster than the relational approach.

Unique Features

- Support for design transactions allows transactions which can span multiple database session which can last several days with full commit or rollback capability.
- Support for versioning of objects.

After a review of several commercial and prototype object database management systems, we selected the Versant Object Database Management System, by Versant Object Technology, Menlo Park, California.

Converting to Versant required approximately 10 person-days of effort to convert the approximately 30,000 lines of C++ code completed. The conversion process required the following steps [Versant92b]:

- Derive all persistent classes from the Versant PObject class to inherit persistent behavior.
- Convert standard C pointers to Versant Links, the persistent database equivalent of a transient pointer.
- Add the Versant dirty() method to all methods that update a persistent object.
- Convert our transient version of a linked list to the persistent Versant VIList linked list class.
- Remove the methods which implemented persistence in the persistent classes which were no longer needed.

5 Conclusions

This paper describes the current implementation of the VIEWS Workbench. In developing VIEWS, we have demonstrated the feasibility of using a commercially available object data management system in support of a complex, large-scale application. While there were a few limitations in using the ODBMS, it proved to be a far better approach in managing large, complex data sets than relational systems.

6 Acknowledgment

This work was supported by the United States Army Training and Doctrine Analysis Command (TRADOC), Fort Leavenworth, Kansas, through interagency agreement P86112 with the U.S. Department of Energy.

7 References

[FUJ91] Fuja, R., Widing, M.A. *Application Interface Engine Language and Object Reference Manual*, Argonne National Laboratory Technical Memorandum ANL/EAIS/TM-72, Argonne National Laboratory, Argonne, IL, 1991.

[PAR86] PAR Government Systems Corporation. *Intelligence and Electronic Warfare Movement Simulation*, Vols. 1-10. Technical Report TRANSANA-TR-2-86, U.S. Army TRADOC Systems Analysis Activity, White Sands Missile Range, White Sands, NM, 1986.

[Versant92a] Versant Object Technology, Inc. Versant DBMS, 1992.

[Versant92b] Versant Object Technology, Inc. *C++/Versant Reference Manual, Version 1.7*, Menlo Park, CA, June 1992.