Proceedings of the 1995 Winter Simulation Conference
ed. C. Alexopoulos, K. Kang, W. R. Lilegdon, and D. Goldsman

CASE TOOL FOR ODBMS DESIGN OF THE FUNCTIONAL DESCRIPTION OF THE BATTLESPACE

Donna L. Cornwell

The MITRE Corporation
McLean, Virginia 22102, U.S.A.

ABSTRACT

This paper summarizes the role of a commercial CASE (Computer Aided System Engineering) tool in the design of the Functional Description of the Battlespace (FDB). The CASE tool product was selected based on specific requirements of the project. It was used as the medium to contain and document the design in a language independent form. Products created included diagrams, reports, and schema definitions for both Object Oriented Database Management Systems (ODBMS) and Relational Database Mangement Systems (RDBMS). Experience with the CASE tool demonstrated many benefits as well as some drawbacks to using such a tool.

1 INTRODUCTION

The FDB is a database under development for use in Warfighter Simulation 2000 (WARSIM 2000). The purpose of the database is to provide a repository for information needed in the analysis and design phase of developing simulations.

Our focus in this paper is the role of a commercial CASE tool in the design of the FDB. The FDB design was created using Paradigm Plus, a CASE tool for Object Oriented (OO) Analysis and Design. A major advantage of the CASE tool approach is the automated maintenance of class definitions and diagrams in a form independent of specific database packages and independent of specific programming languages as well. Selection of Paradigm Plus was based on a number of factors, including support for the Rumbaugh methodology, OS support, reverse engineering, and code generation capability.

Paradigm Plus was used to create diagrams, define classes and relationships, and generate the database schema in C++ and in SQL. The C++ code was used to create a schema for a prototype in UniSQL. The SQL was used to implement some tables in Oracle for comparison purposes. Standard reports created by the tool from the object repository were used in conjunction with diagrams to document and present the database design.

Paradigm Plus had some unexpected advantages, including number of languages supported and an internal scripting language that allowed extensive customization of the tool. It also had some unexpected problems, due primarily, to the immaturity of the product.

In Section 2, we discuss background on the project and its relation to simulation. Section 3 briefly describes OO methodologies as they apply to OO CASE tools. Section 4 covers the selection process for selecting the CASE tool. In Sections 5 and 6, the process of creating the design in Paradigm Plus and the lessons learned are described.

2 FUNCTIONAL DESCRIPTION OF THE BATTLESPACE

The FDB is a repository for those physical, environmental, and behavioral phenomena required to adequately represent the Army's battlespace operating system components and functions that must be represented to produce credible simulations of those functions. Heretofore, the Army has relied on the winning contractor to conduct a domain analysis, using whatever published material and subject matter experts he happened to have at his disposal, to determine what should be represented to meet the requirements stated in the System Requirements Document (SRD). After deciding what should be represented, the contractor then had to access multiple data sources to obtain the required data. With an FDB, the contractor has a single source of approved Army data, descriptions, and algorithms to support his domain analysis and software engineering activities.

Based upon the quantity of equipment, the number of units, the amount of doctrinal guidance, and the complexity of the descriptions of interactions between units, equipment, and terrain required for the FDB, it was determined that object-oriented analysis and design techniques were desirable. The ability to use objects to describe real life entities, their behaviors, and their interactions further influenced the decision. Finally, the characteristics required of the FDB - comprehensive, extendible, observable and measurable, broadly

applicable, traceable, and accessible - led to the conclusion that an object-oriented approach was necessary to meet the government's requirements for an FDB.

3 OO METHODOLOGIES AND CASE TOOLS

3.1 What OO Methodologies Provide

OO methodologies provide guides for the process of identifying and defining objects or classes and their relationships. In addition, most OO methodologies provide diagramming notations in which to express the object model including not only classes and relationships, but also the dynamic model of object interaction.

3.2 How CASE Tools Implement OO Methodologies

OO CASE tools are primarily diagramming tools with underlying repositories which contain information about the design not apparent in the diagrams. In addition, information defining links between diagrams may be contained in the repositories. Some tools implement the repositories in flat files, others use database packages.

CASE tools enforce the diagramming notation by providing a set of predefined symbols for creating diagrams. They also do some rudimentary validation of the object model. This validation usually consists of consistency checks such as name conflicts and references to non-existent classes.

4 CASE TOOL SELECTION

On the FDB prototype project, the CASE tool was to be the medium for developing, documenting, and storing the object model. To ensure that the information captured in the CASE tool would be complete, flexible, and independent of other software packages, a number of CASE tool characteristics were defined as requirements.

The initial implementation of the FDB was to be a prototype. The final production system might or might not be implemented using the same COTS software as the prototype. Therefore, it was important that the design stand alone in a representation that would be clear, easily understood, and non-proprietary. It was decided to represent the design in a notation that was part of a published formal methodology. The Object Modeling Technique (OMT) methodology developed and published by (Rumbaugh 1991) was selected as the design methodology.

In order to support the prototyping effort, but not tie the object model to any particular software package or language, CASE tool support for multiple languages was desired. The CASE tool was required to be entirely separate from any DBMS package used in the prototype, though support for specific RDBMS and ODBMS packages would be welcome.

Also, some initial design work had been captured in C++ code, so it was desirable that the CASE tool import C++ code. This feature could also be used to support spiral development by reading C++ code generated from other software packages.

Language support was a difficult issue, as there is a mismatch between the requirement to use Ada, and the desire to use COTS whenever possible. C and C++ were the common denominators between COTS packages examined in both the CASE and ODBMS categories. However support for Ada was required for future development and integration of a production system. C++ and SQL were used as translation mechanisms between the COTS packages. Ada code generation was looked for in the CASE tool arena. Support for Ada code did not exist in ODBMS packages at the time of the tool search other than in ITASCA, which is no longer being sold.

To support the above requirements, specific OOAD CASE tool features were identified:

- reverse engineering
- C++ code generation
- Ada code generation
- data dictionary reports
- diagrams

Over a three-week period, a number of products were examined. Since the time frame was short, the goal was not to do an exhaustive study of all CASE tools, but to choose one that was reasonable and sufficient to do the job at hand. The leading contenders examined included: ObjectTeam, Paradigm Plus (Protosoft, Inc. 1994), SES Workbench, Rational Rose, OMT Tool, and Software Through Pictures. Paradigm Plus was selected for the FDB prototyping effort as the only available tool that adequately met all above requirments and was available on both DOS and UNIX platforms.

One feature that was not originally used as a discriminator, but was found to be extremely useful was a scripting language. Paradigm Plus has a scripting language that uses BASIC with extensions and macros to provide access to the object repository. This feature has been invaluable in customizing the tool. Minor changes have been made to the scripts provided with package in order to : export C++ code annotated with UniSQL specific classes; customize the data dictionary report to word wrap to fit on the printout without truncation; and produce ad hoc reports to validate/verify the object repository.

The use of Paradigm Plus supported the sponsor's goal of having the class structure developed for the FDB

be implementation language independent. The final language necessary to develop a complete FDB need only support the fundamental concepts of Object-Oriented Programming -- that data and methods be encapsulated within the objects that manipulate their data. Additional implementation language characteristics include the capability to support multiple inheritance, provide container classes, support polymorphic behaviors, and facilitate the use of references to objects.

5 IMPLEMENTATION

The first step in using Paradigm Plus was to reverse engineer the existing C++ code to create class definitions using the *Import* feature. There were several problems with this approach. The main problem was that the automatic diagram generation produces only attribute diagrams, no relationships are represented. So diagrams had to be manually edited in order to show relationships.

The next step was to create diagrams with the tool, which proved to be simple and straightforward. Classes and attributes added to diagrams are automatically entered into the object repository.

The final stage was to generate code from the Paradigm Plus object model to create the UniSQL database and a sampling of tables in Oracle. The goal was to capture information in Paradigm Plus in a language independent form, taking advantage of all information captured in the tool rather than manually transposing it into a form understood by either DBMS package (or other DBMS packages that might be used to implement the final FDB). This goal was made difficult by the generality of the tool and the syntax specificity of the specific languages and software packages. For a design, data types can be described in general terms. However, for implementation, the code generation utility must map the general description to a specific data structure that is valid syntax in the target language. To generate compilable code, either some language specific syntax must be entered into Paradigm Plus or the user must add some type definition semantics and modify the code generation scripts to interpret the semantics. For example, entering "string" as a type in Paradigm Plus, and modifying the code generation to create string in C++ and DB_String in UniSQL (an ODBMS package).

6 PARADIGM PLUS LESSONS LEARNED

Overall, the MS Windows version of Paradigm Plus was useful for creating and maintaining the object model. Initially both the UNIX and MS Windows versions of Paradigm Plus were purchased. The UNIX version was desirable because it allowed multiple users to share the same object repository. However, it proved slow and awkward to use. (An improved more robust UNIX version was issued by the vendor, but unfortunatly, not within time for the FDB prototype.) As a result, the UNIX software was changed for an equivalent set of MS Windows Paradigm Plus software. The lessons learned below all refer to the MS Windows version.

In the first full load of the database, verification showed a number of problems resulting from the code generation. Classes without attributes generated a warning that no code would be generated. It is a common design technique to create virtual classes with no attributes to group related subclasses. This was used numerous times within the FDB, so a number of classes did not exist in the generated code, and many other classes created errors by inheriting from those classes that were not generated.

The work around for classes with no attributes was to extract warnings from the output script and generate SQL scripts to create classes with those names. This was sufficient to create the prototype. A cleaner solution would be to modify the Paradigm Plus code generation script to produce code for the missing classes.

There were a number of problems with the internal database that houses the object repository. While updates can be made from the browser, matrix, or the diagrams, the changes aren't reflected everywhere. This problem is clearly documented, but is undesirable. It was also found that even where the documentation says the repository will be in sync, it was not always true. The browser and matrix can be up at the same time and show class attribute names that are not in sync. This is a definite bug, as opposed to the above, which is merely awkward.

When an object repository is updated from the browser interface, only diagrams that are currently open are updated. (Protosoft, Inc. 1994) To keep diagrams in sync with the repository, Synchronize Diagrams must be run from the Diagrams menu. This option offers choices when definitions do not agree between the repository and diagrams. The choices are to change repository, change diagram or merge. This process is tedious and does not always produce the expected results.

There is a problem when an object name has been changed in the repository and the name change needs to be reflected in the diagram. The software treats this as an attempt to create a second class with the same name and rejects it. To avoid this, make the name change on all the diagrams where the class appears and then synchronize to update the repository. This is inconvenient and tedious especially where the same change needs to be made on a number of diagrams.

Other difficulties with the repository were inherent from the initial reverse engineering of C++ code. There were classes, attributes, and operations that were dropped from the design and therefore were never put into diagrams. They never appeared until the Object Model and Data Dictionary reports were run. Then a manual review had to be done. The diagram synchronize operation makes sure that everything that is on a diagram is in the repository. There is no inverse operation to verify that everything that is in the repository is on a diagram. This would be much less of a problem for a project where all classes were created directly from diagrams rather than through the reverse engineering feature.

Finally, when Paradigm Plus saves generated C++ code to files, the name of the file defaults to the first eight characters of the class name. This convention causes problems as the first eight characters is often not enough to uniquely identify a class, hence overwrites occur. Control files need to be created to map the longer class names to unique eight character file names.

7 CONCLUSIONS

The Paradigm Plus CASE tool provided sufficient functionality to create and house the object model for the FDB. The stated requirements of representation in formal Rumbaugh/OMT methodology notation, language independence, and export in multiple languages were fully met. Complete independence from the database package was not achieved, as some implementation decisions for data representation had to be made to generate correct definitions for the ODBMS. Nor was the support for spiral development completely realized as reverse engineering succeeded in importing only class definitions, not relationships.

REFERENCES

Protosoft, Inc. 1994. *Paradigm Plus 2.01 User Guide.* Houston, Texas: Protosoft, Inc.

Rumbaugh, J.T. 1991 *Object Oriented Modeling and Design.* Englewood Cliffs, New Jersey: Prentice Hall.

AUTHOR BIOGRAPHY

DONNA L. CORNWELL is a member of the technical staff at the MITRE Corporation in McLean, VA. She received a B.A. degree in Early Childhood Education from the University of North Carolina at Chapel Hill in 1979, and she received a B.S. degree in Computer Science from the University of Maryland in 1986. She will complete an M.S. degree in Computer Science at Johns Hopkins University in 1995.