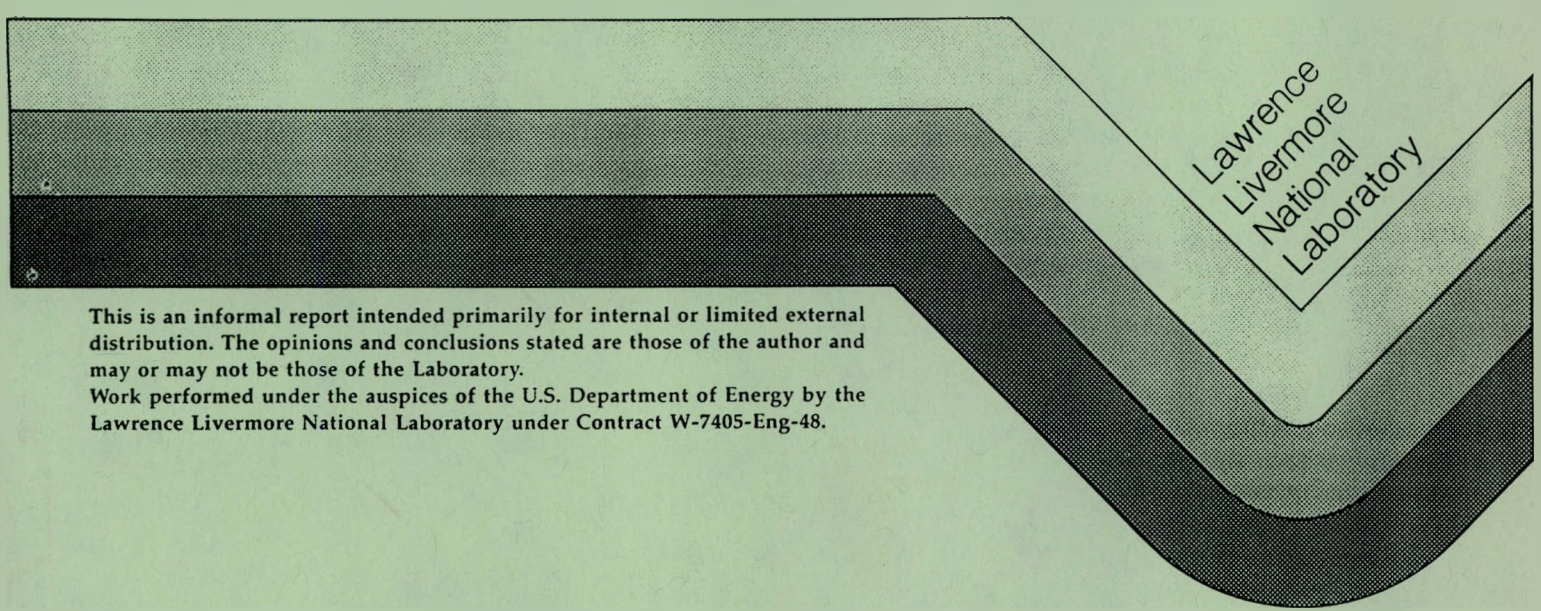
UCID- 21850

Experiences Using INGRES in a Large Battlefield Simulation (ConMod)

Steven D. Rodgers

1 November 1989



Work performed under the auspices of the U.S. Department of Energy by the Lawrence Livermore National Laboratory under Contract W-7405-Eng-48.

This report has been reproduced
directly from the best available copy.

Available to DOE and DOE contractors from the
Office of Scientific and Technical Information
P.O. Box 62, Oak Ridge, TN 37831
Prices available from (615) 576-8401, FTS 626-8401.

Available to the public from the
National Technical Information Service
U.S. Department of Commerce
5285 Port Royal Rd.,
Springfield, VA 22161

Price Code	Page Range
A01	Microfiche
Papercopy Prices	
A02	001-050
A03	051-100
A04	101-200
A05	201-300
A06	301-400
A07	401-500
A08	501-600
A09	601

Experiences Using INGRES in a Large Battlefield Simulation (ConMod)

UCID--21850

DE90 005321

Steven D. Rodgers
Lawrence Livermore National Laboratory

Abstract

This paper describes the experiences of using INGRES[1] in a large battlefield simulation. The paper includes a project overview, a section on INGRES components, and conclusions. The project overview describes the ConMod project and its objectives. This section also discusses our needs for the project with respects to a data storage system. The section on INGRES components briefly describes what the components are, how we used them in the ConMod project, and their advantages and disadvantages. The last section concludes with some general comments about INGRES and its appropriateness for particular projects.

1. Introduction

The Conflict Model (ConMod) is a large scale, high resolution, air/land conflict simulation featuring automated command and control [1]. Its scope includes the modeling of corps level combat combined with selected assets of echelons-above-corps. The model was jointly sponsored by the U.S. Army and the Lawrence Livermore National Laboratory (LLNL). The model was primarily being developed for use as an analysis tool for addressing the operational concepts of air and ground forces engaged in deep, rear, and close battle. It covers a geographical extent of hundreds of kilometers and was to simulate combat covering a time period of days. Other pertinent aspects are that it is a discrete event, stochastic simulation which uses item system (i.e., a tank, plane, etc.) resolution and features high resolution graphics to support rapid set-up, interactive monitoring, and rapid post-simulation analysis. The project was one of significant complexity. In early 1989, after a little over three years of work and the building of a working prototype, it was decided that further development of ConMod would be discontinued.

During the initial design of ConMod, it was decided that a database management system would be used to store data. This is a departure from other projects at LLNL's Conflict Simulation Laboratory which use binary files to store data. The ConMod project has two types of stored data; initial scenario data and post processing data. Initial scenario data is information that is used to start up a simulation run. Most of the initial scenario data is called characteristic data which is static throughout the simulation run. Post processing data is information recorded during a simulation run. The ConMod team

[1] INGRES is a trademark of Relational Texhnologies Inc.

realized that a relational database system would closely match the information models [3] developed for the project. The relational database system could also store data for several scenarios which allows for data analysis between scenarios. After careful consideration, the INGRES Relational Database System by Relational Technology Inc. was selected.

2. INGRES Components

This section describes the INGRES components used in the ConMod project and our experiences in using these components. The ConMod project used INGRES VAX[2] Release 5.0/06 for the VMS 5.0 operating system.

INGRES is a relational database management and application development system from Relational Technology Inc. Its capabilities help users to enter, query, update and analyze data [2]. INGRES also provides a host of tools that allow a developer to create customized user interfaces.

2.1 Tables

INGRES tables are the storage facility for data. A table is made up of one or more fields. Each field has a field type associated with it (i.e. integer, float, string). INGRES being a relational database, tables usually mapped directly to our information model objects. This direct mapping allowed us to quickly develop the database once the information models were completed.

2.2 Quel

Quel is a language used to query INGRES databases. Most Quel operations are also included in Equel, which is discussed in the next section. Quel allows a user to make ad hoc queries on tables to extract data. We found that Quel was a useful tool for the initiated, although, it was too cumbersome for an end user. However, Quel was useful in testing code that was eventually to be included in Equel/Ada. Quel also allowed us to easily load data from other INGRES databases.

INGRES provides two query languages; Quel and SQL. Recently, SQL has received a lot of attention as the emerging standard for all database languages. It is unlikely that INGRES SQL will be the final version of the SQL standard, although it will probably be closer to the standard than Quel. However, we chose Quel for several reasons. At the time, Quel was a more powerful query language than SQL. Quel was also the query language of choice at Lawrence Livermore National Laboratory and many army facilities. After the INGRES license for Quel and its embedded package Equel were purchased, the ANSII standard committee for Ada[3] officially started planning for a SQL interface in Ada. The committee noted that many of the SQL systems had extensions which Ada would not necessarily implement. The standards for including an SQL interface in Ada are still under consideration.

---

[2] VAX and VMS are trademarks of Digital Equipment Corporation.

[3] Ada is a registered trademark of the U.S. Government, Ada Joint Program Office.

2.3 Equel/Ada

Equel/Ada is a hybrid language provided by Relational Technologies Inc (RTI) that allows access to an INGRES database from Ada. The hybrid language is a combination of Ada code with embedded Quel (hence Equel). The Equel/Ada source code is preprocessed into Ada code, replacing Equel code with calls to Ada routines supplied by RTI. From this point, the code is considered pure Ada and is compiled by the Ada compiler.

INGRES provides general purpose user interfaces to databases, however, these interfaces cannot be customized or enhanced to aid the user. Therefore, we used Equel/Ada to develop a customized, user friendly, database interface. As a first step, we successfully duplicated the functionality of the INGRES-provided user interface. After this was accomplished, we started adding customized functions. The end product was a user interface to the ConMod database that was far superior to the INGRES-provided user interface. The Equel/Ada language provided the flexibility to produce a user interface that was customized to our needs.

Although RTI's Equel/Ada language is probably one of the most robust database interface languages around today, there were several problems. Currently, Equel/Ada does not fully use the capability of Ada. Equel/Ada has no concept of Ada generics and has no real concept of Ada packages. Equel/Ada simulates knowledge of packages by using "include" statements similar to Pascal or C. However, an Equel/Ada "include" statement translates to an Ada "with" clause and an Ada "use" clause, thus there is no way to dereference packages.

Another problem with Equel/Ada is the declaration of variables. Any Ada variable that is used in an Equel/Ada statement must be known to the Equel/Ada preprocessor. A double pound sign (##) is used at the beginning of a line to alert the Equel/Ada preprocessor of an impending Equel/Ada statement. Thus, the declarations of all variables used in Equel/Ada statements must be preceded by a ## at the beginning of the line. For example :

```
## include "Another_Package"

## procedure Test is

# #   A_Variable : Integer;
# #   B_Variable : B_Variable_Type;        -- Note the variable is
NOT dereferenced.

##  begin

# #   retrieve (   A_Variable = INGRES_TABLE.A_FIELD,
# #                B_Variable = INGRES_TABLE.B_FIELD )

## end Test;
```

In this example, the type **B_Variable_Type** originates in **Another_Package** but since an "include" statement is translated into an Ada "with" and "use" clause the

declaration of **B_Variable** cannot be dereferenced. Also, **Another_Package** would have to be preprocessed by Equel/Ada. As the program continues to grow, this could eventually involve several packages and becomes a problem when a package needs to be modified. The basic rule is to extract the Ada source code of the package to be modified from the ACS (Ada Compilation System) library. However, to modify Equel/Ada code, you need the original, unpreprocessed Equel/Ada source code, not the Ada source code that resides in the ACS library. Thus, two copies of the code must be kept and the benefits of ACS are lost on the Equel/Ada code.

Another problem related to declaration of variables occurs in procedures that contain Equel/Ada code. Procedures that contain Equel/Ada code must include ## at the beginning of each new line of the parameter heading. This means that all variable types in the parameter list must be known to the Equel/Ada preprocessor even if they are not used in any Equel/Ada statements. For example :

```
## include "Another_Package"

## procedure Test( Passed_Variable : out Passed_Variable_Type )
is

# #   A_Variable : Integer;
# #   B_Variable : B_Variable_Type;        -- Note the variable is
NOT dereferenced.

##  begin

      Passed_Variable := 4;

# #   retrieve (   A_Variable = INGRES_TABLE.A_FIELD,
# #              B_Variable = INGRES_TABLE.B_FIELD )

## end Test;
```

In the preceding example, **Passed_Variable** is never used in an Equel/Ada statement, however, Equel/Ada must know about it since the variable is declared in the package header. Again, this leads to many unnecessary Equel/Ada packages and an added level of complexity. An alternative to this problem is to declare nested procedures. The outer procedure, which does not contain any Equel/Ada code, declares the parameter list as a regular Ada procedure would. A local Equel/Ada procedure is created with a parameter list of only those variables used in Equel/Ada statements. Non Equel/Ada variables needed by non Equel/Ada statements in the local procedure can be used globally from the outer procedure. For example :

```
## include "Another_Package"

procedure Test( Passed_Variable : out Passed_Variable_Type ) is

## procedure Local_Test is

# #   A_Variable : Integer;
```

```
    # #   B_Variable : B_Variable_Type;        -- Note the variable is
NOT dereferenced.

    ##  begin

        Passed_Variable := 4;

    # #   retrieve (   A_Variable = INGRES_TABLE.A_FIELD,
    # #                B_Variable = INGRES_TABLE.B_FIELD )

    ## end Local_Test;

    begin

        Local_Test;

    end Test;
```

This is admittedly poor programming but necessary given the Equel/Ada limitations.

Due to Equel/Ada's lack of knowledge about Ada generics, we needed to develop a separate set of routines for every table in the database. The routines for each table were similar (i.e. Append, Retrieve, Update, etc.) however, due to the differing data structures, each package had to be developed individually. With an increasing number of tables in the ConMod database, the database code grew rapidly. It was determined that we could develop a code generator to write the Equel/Ada packages needed since the data structures were the only differentiating factor. Thus, a code generator was created that took a data structure as input and created the necessary Equel/Ada packages for the database interfaces. This worked extremely well since our data structures tended to change and increase in number rapidly. However, as the number of data structures increased, the amount of code also increased. The increased code generation expended valuable disk space and increased compile and link times.

2.4 Forms

The user interface to the ConMod database was accomplished through the use of INGRES forms. Forms are displayed on a terminal and can be thought of as a window into the database. The forms can display data and accept input from the keyboard. Forms can be displayed in two formats : block and table. Block forms display one row of data from the database with each field on a separate line. Block forms are convenient when the user is entering new data. Table forms display several rows from the database in a spreadsheet style. Table forms are useful to examine and update data.

Equel/Ada includes statements that can manipulate INGRES forms in a variety of ways. These form manipulation statements became an integral part of the user interfaces. INGRES also provides a utility called VIFRED (VIsual FoRms EDitor) to edit forms and create logical groupings of data input.

2.5 Reports

INGRES also provides a Report Writer that allows reports to be created from data that currently exists in the database. One of the main disappointments with the Report Writer was its inability to neatly handle reports in a tabular style. A tabular format consists of one database row per line of output with fields spanning across the page. Due to the length of the rows, it is unlikely that all the data for a row will fit on a 132 characters per line page, thus reports must span pages. The Report Writer does not successfully handle this problem. Therefore, we found it easier to develop our own custom Report Writer that managed the page and column breaks. We also allowed the flexibility to print selective columns in sorted order, an option that is not available with the INGRES Report Writer.

2.6 Applications

Two main projects were developed using the INGRES development system : 1) the ConMod Initial Database, and 2) the ConMod Post Processor. The ConMod Initial Database contained data entered by the user and extracted by the simulation during simulation initialization. The ConMod Post Processor contained simulation event occurrences recorded during a simulation run.

Excluding comments mentioned in the previous sections, the INGRES development system worked extremely well in creating a flexible, user friendly, data entry system for the ConMod Initial Database. The ConMod Initial Database consisted of approximately 150 individual tables. Specific data for a simulation run was known as a scenario. There were on average 20 scenarios stored in the database at any one time. Scenarios ranged in size from Platoon to Division. Rows of data in each table varied significantly depending on the size of the scenario and the nature of the table. The ConMod Initial Database interface was reasonably responsive to queries of users. We found that the response time was directly related to the data load in a particular table. Although a scenario might contain large amounts of data, the data was distributed among 150 tables and thus, the response time for a query on any particular table was not unreasonable. During simulation initialization, the simulation extracted the needed data from the database. It is difficult to determine INGRES's efficiency during simulation initialization since accessing the database was not isolated. Typically, a simulation object would retrieve some data from the database and then perform other initialization duties before the next simulation object retrieved data from the database. However, it is probably safe to say that retrieving data from the database at initialization time is slower (approximately five times) through INGRES than it would be if we were reading the data from files.

The ConMod Post Processor also used the INGRES database system to store data, however the results were quite different. During a simulation run events occur and the notification of these events are written to an ASCII file. After the simulation has completed, the event file is closed. The ConMod Post Processor reads the event file into defined tables and then allows the user to examine the data.

The first problem with using INGRES with the ConMod Post Processor is that textual information is not sufficient for a user to understand the simulation results. With the enormous amount of activity in a typical scenario, it is hard for an analyst to scan a listing of events and reach some useful conclusions. Therefore, the ConMod Post Processor needed more than just a database. The ConMod Post Processor was to emulate the Analyst Workstation, a post processor for the Janus simulation model also developed

by the Conflict Simulation Laboratory. The Analyst Workstation added a graphic interface to a data management system which allowed the user to graphically interpret large amounts of data easily.

The ConMod Post Processor was extremely slow in comparison to the ConMod Initial Database. As previously mentioned, the amount of data stored in any one table directly relates to the response time of a query on that table. Thus, if the table has a large amount of data stored in it, the response time will degrade. Due to the nature of the ConMod Post Processor, there were only ten individual tables compared to the ConMod Initial Database's 150 tables. Typically, the ConMod Post Processor would also contain more data than the ConMod Initial Database for a particular scenario. Thus, queries on ConMod Post Processor tables were extremely slow. We tried reformatting the database tables to take advantage of INGRES's many file organization schemes. This did improve the performance of data retrieval, however, it took an inordinate amount of time to read the event files into the ConMod Post Processor database. The time delay became unacceptable and the tables were returned to their original file organization.

It became apparent that the ConMod Post Processor didn't use many of the capabilities of INGRES since data retrieval was its main purpose. Therefore, we decided to abandon the use of INGRES's storage structure for the ConMod Post Processor, however, we continued to use its form capability. We developed a file based system that allowed users to query the file in the same manner that the database would be queried. The query was processed by our custom Ada code and did not interact with INGRES. This significantly decreased data retrieval time and completely eliminated the need to preprocess the event files. Although this solution solved our problems for now, it should be stated that due to the lack of INGRES in the ConMod Post Processor, its design is less flexible and future enhancements will require more work.

3. Conclusions

It is clear from the discussion above that INGRES is not suitable for all applications. INGRES worked well with the ConMod Initial Database and we intend to continue to use this preprocessing scheme in the future. However, due to the inherent data load and application of the ConMod Post Processor, INGRES is probably inappropriate. Therefore, the choice of using INGRES should be carefully considered for the application. If the application will have enormous amounts of data concentrated within a few tables and data retrieval is the only function performed on the data, then INGRES can be substituted with a simple home brewed data manager. However, if the application will have data distributed among several tables and functions performed on the data extend beyond simple data retrieval, then INGRES provides a powerful and flexible database management system.

References

1. Chiu, Y., et. al., *The Conflict Model (ConMod) Interim Technical Review,* Lawrence Livermore National Laboratory, Livermore, CA, October 1987.

2. *INGRES Terminal User's Guide,* Relational Technology Inc., Alameda, CA, August, 1986.

3. Shlaer, Sally, Mellor, Stephen J., *Object-Oriented Systems Analysis,* Prentice-Hall, Englewood Cliffs, NJ, 1988.