Experience in the Use of an RDBMS in Multiprocess Interactive Simulation

Carsten Gabrisch and Mike Davies

Information Technology Division
Defence Science and Technology Organisation
Department of Defence
PO Box 1500 Salisbury
South Australia 5108

Abstract A relational database management system (RDBMS) can be a powerful tool in the design and development of computer software. Having to take a database view of the project can lead to better storage, management and access of data by the software. Data is stored in specific formats which makes for easier software development and network configuration makes coordination of team effort easier. Such benefits facilitate rapid prototyping and the data management qualities of an RDBMS allow the developer to concentrate on the more important features of the software, namely the overall functionality. This is particularly true in the case of discrete event simulations. This paper illustrates the benefits in using an RDBMS to assist in the overall design and development of a multiprocess interactive simulation.

1. INTRODUCTION

As stressed in the Australian Defence White Paper of 1994 (AGPS [1994]), effective Command, Control, Communication and Intelligence (C3I) for Australia's forces is fundamental to the successful conduct of Australian Defence Force (ADF) operations, in any conflict or peacetime activity. Having the ability to study the effectiveness of existing and future military C3I systems is essential. Information Technology Division (ITD) of the Defence Science and Technology Organisation (DSTO) is sponsored to develop modelling and simulation tools to enable such studies to be conducted.

A typical C3I system structure for the defence of Australia might involve elements of the three services and accommodate conflict at all levels. The strategic level embraces the higher echelons of the military and political organisations concerned and hence addressing this level requires addressing decision-making at lower (operational and tactical) levels also. The C3I architecture can be pictured as a complex network of nodes and links. The nodes are typically centres of decision making; information processing or filtering; information transfer; or combinations of these. The links are the inter-node communication channels that transmit information of many forms. In a time of conflict, the C3I system might be stimulated by intelligence concerning the detection of potentially hostile enemy activity. This would consequently cause the generation and passage of internal information that might result in changes in readiness and maybe, the deployment of reaction forces. To study the effectiveness of military C3I systems requires analysis of the impact of C3I procedures and technologies on the overall military mission concerned. The term *mission* is used here to describe, for example, an operation, battle or exercise.

The requirement on ITD is not to develop tools specific to any particular military service or level of conflict, but rather to create a general purpose suite of tools, specific instantiations of which could be used to address any particular study at hand. It was decided that the main means of achieving this suite of tools and the associated expertise would be through the process of developing the Distributed Interactive C3I Effectiveness (DICE) simulation (Davies [1993]). Making the most efficient use of minimal resources is particularly important in this project.

2. FEATURES OF THE DICE SIMULATION

A typical scenario to be represented by the DICE simulation can be regarded as configured about a complex set of nodes and links that represents the central C3I network. The players in the DICE simulation generally form the nodes of this central network. Players might represent individual, or groups of, commanders in a C3I system or an aggregated representation of some other C3I system entity. The interactive nature of the DICE simulation will allow the decision-making practices of real commanders to be injected and accommodated. Such human players will need, however, to be complemented by a number of artificial ones (artificial agents) in a manner whereby the two types can communicate.

In real C3I systems, communications between nodes can take many forms including both formatted and unformatted messages; tables of data; graphical displays; and video images. In the simulation, all forms of communication are represented by the passage of formatted textual messages which either bear a direct resemblance to a military message, or accompany or summarise information of a different form.

The central network can be considered to be surrounded by an external environment or node that encompasses any aspects that are not explicitly represented in the central network but which, nevertheless, form important contributions to the scenario being addressed. Communication between the central environment and the external node is again achieved through the use of messages with the external node injecting stimuli into the central network of the scenario. What lies outside the central network and what lies within depends on the depth and breadth of the scenario being simulated. The conceptual external node embraces such aspects as enemy activity; battlefield information; and sensor information. Battle simulations, war games and other simple models are employed, as needed, to address the tactical levels. These are interfaced with the main simulation in order for the represented C3I system to have impact on the military mission concerned and hence allow evaluation of the system's effectiveness.

The overall structure, requirements and developments to date of the DICE simulation are best presented with reference to Figure 1 which is a breakdown of the functional areas of the DICE simulation environment (Davies, Gabrisch [1995]). Indicated by the uppermost row of this figure are the *players* in the simulation, namely the simulation controller or analyst; artificial agents; and human commanders. *Peripheral units* in the DICE environment include any war games and battle simulations that may be employed plus the command support systems (CSS) that may be required by the human players (*Peripheral units* are interfaced via tailored Peripheral Unit Interfaces (PUI)). Players plus peripheral units are the nodes in the overall DICE simulation, ie the central network plus the external environment.

The simulation kernel is the main event-stepping engine of the DICE simulation. The kernel controls time synchronisation and execution rate of the distributed processes and can be configured such that the simulation can run in real or non-real time. The simulation is capable of being paused, advanced and resumed as required by the simulation controller. The current simulation is designed around two main events: *message submission*, involving submission of a message by a node for transmission; and *message reception*, concerning the receipt of a message by a node (Davies [1993]).

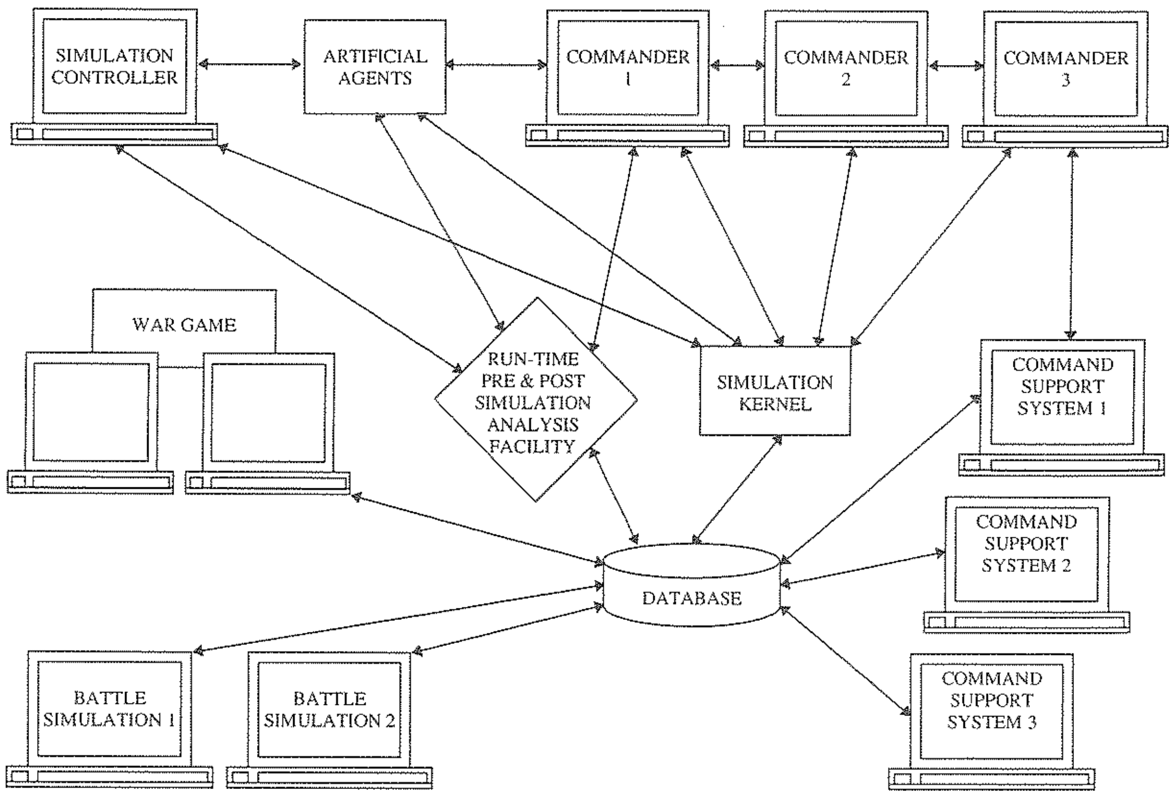

Figure 1: Functional diagram for DICE simulation environment

Artificial agents, human player interfaces, PUI, the simulation kernel and controller facilities are separate processes which may or may not execute on the same processor. The simulation is primarily being developed in the ANSI 'C' programming language using Sun SPARCstations. The Ingres relational database management system (RDBMS) and associated utilities (Ingres Corporation [1991]) have featured strongly in the development of the following capabilities :

- Time and execution rate synchronisation allowing real-time and non real-time execution;
- Access to shared data by multiple processes;
- Ability to pause, resume and advance the simulation;
- Message submission and reception; and
- Run-time monitoring and control.

3. THE USE OF AN RDBMS IN THE DICE SIMULATION

The communication architecture associated with the simulation is illustrated in Figure 2. Each node has an associated mailbox through which it receives incoming messages; outgoing messages are placed directly on the simulation event queue, as *message submission* events, for mailing to the intended recipient. The mailboxes are Ingres database tables; a summary of the main tables used in the DICE simulation is given below :

connectivity: stores the nodes a link connects and transmission time and availability of the link.

entities: stores a description of the nodes and links in the scenario.

event: stores *message submission* and *message reception* events.

mailbox: stores the message number of the incoming messages to a node.

message content: stores the contents of the messages.

message information: stores the message number, node which sent the message, the link the message is sent along and the time the message was transmitted.

message location: stores where a message is located in the simulation and at what time it arrived.

scenario information: stores the name and the time duration of the scenario.

time synchronisation: stores the time of the last time synchronisation, inserted by the simulation kernel, and the execution rate.

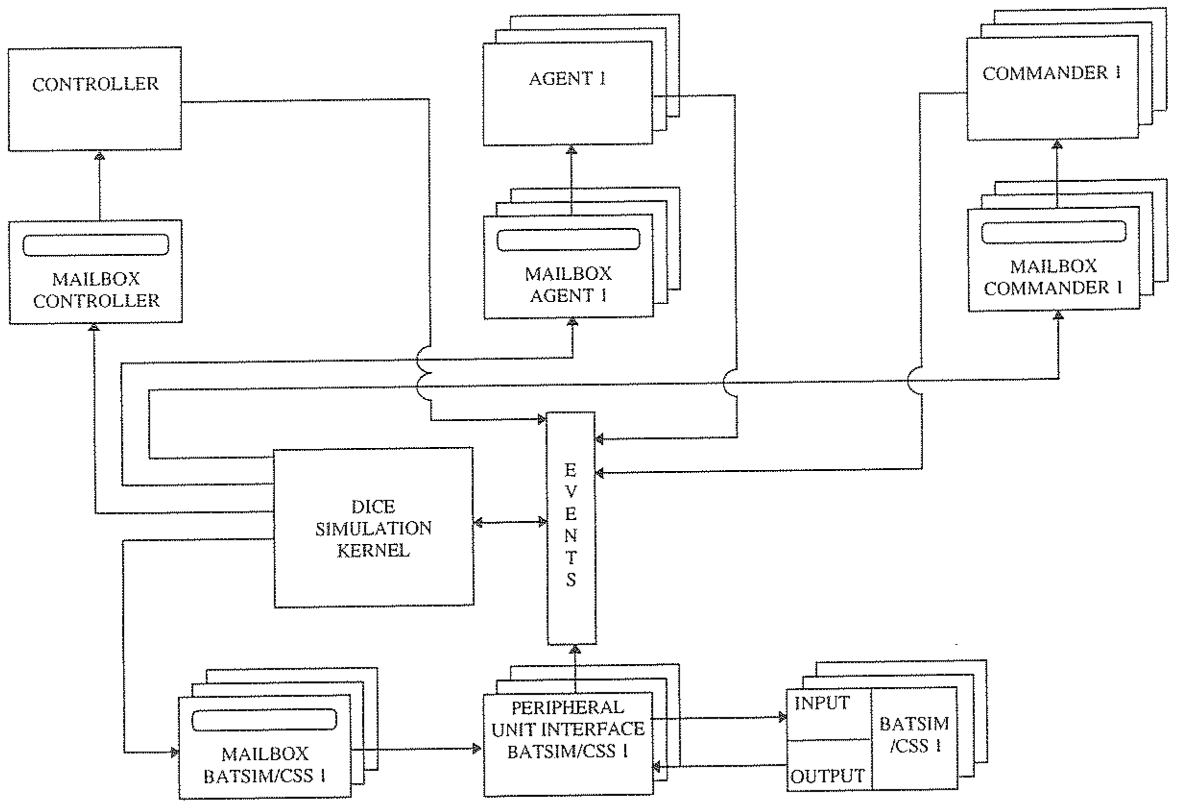

Figure 2: DICE simulation kernel and communication

Database event features that Ingres provides, are also conveniently utilised in the simulation. Database events can be set up to be triggered when certain actions are carried out on specified tables, or created to be resident in the database and then raised by embedded database statements. Processes register for these events and are informed when such events occur, thus giving effectively an event-driven characteristic. Database events that feature in the DICE simulation, along with what processes register for them are discussed below :

Event table event : when a *message submission* event is inserted into the table a database event is raised. The simulation kernel registers for this event.

Mailbox event : when a message number is inserted into a node's mailbox a database event is raised. The process corresponding to the node associated with a mailbox registers for that mailbox event.

Pause simulation event : this database event resides in the DICE database. All processes register for this database event.

Resume simulation event : this database event resides in the DICE database. All processes register for this database event.

Start simulation event : this database event resides in the DICE database. All processes register for this database event.

Stop simulation event : this database event resides in the DICE database. All processes register for this database event.

Time synchronisation event : when values in the *time synchronisation* table are either inserted or updated a database event is raised. All processes (with the exception of the kernel) register for this database event.

The manner in which the database tables and associated events are used is outlined in the following sections that correspond to pre-simulation scenario implementation, run-time execution and post-simulation activities.

3.1 Pre-simulation

The scenario that is to be simulated is loaded into the *entities* and *connectivity* tables. The *entities* table stores a description of the nodes and links that are present in a scenario. Part of this description is whether the node is an artificial agent, human player or a PUI. The *connectivity* table stores which nodes a link connects, the time taken to send a message along that link and whether that link is available for use.

When the simulation kernel is started it connects to the DICE database. The kernel then registers for database events that are raised when the simulation is started, stopped, paused or resumed and when a *message submission* event is inserted into the *event* table. Any information from a previous simulation is deleted from the *event, message information, message location* and *message content* tables.

Once the scenario is loaded a database table is created for each node and acts as the node's *mailbox*.

The simulation controller can program messages to be received by nodes, at given times in the simulation, before the simulation is started. These messages are placed into the appropriate tables and a *message reception* event is placed in the *event* table. Such events are referred to as independent external stimuli since they are pre-scheduled and hence independent of activities that occur whilst the simulation is running (Davies [1993]).

For all nodes in the *entities* table which are either an artificial agent or a PUI their respective processes are started. These processes register for relevant database events. The processes then wait for the first time to be inserted into the *time synchronisation* table.

From the *scenario information* table the time duration of the simulation is read. The system clock is read to obtain the start time, in terms of wall clock time, of the simulation. The simulation time is set to zero (0) or wall clock time. A time factor is also set (by the controller), which determines the rate of execution: eg a value of one (1) for real time and two (2) for twice real time. The simulation time and the time factor is then inserted into the *time synchronisation* table. A database event is raised for which all processes in the simulation are registered. All artificial agents, PUI, human player interfaces and the event handler of the kernel are then set running.

3.2 Run-time

The event handler of the simulation kernel is a continuous loop which deals with raised database events, time synchronisation and time based events. The first object of the loop is to check if any database events have been raised and if so determine the database event type. If a *stop simulation* database event is raised or the simulation time has reached a preset stop time the loop is ended and the kernel attends to its post simulation activities. When an *event table* database event is raised a message submission sub-procedure, detailed later, deals with this event. The details of this sub-procedure are explained later. If the database event is of type *pause simulation* the simulation and all the associated processes are paused. These include all artificial agents; the human commander interface facilities and the PUI which consequently pause the peripheral units if appropriate. The kernel then waits until a *resume simulation* database event is raised. During a pause in the simulation the simulation controller can set the simulation time to some time in the future and can alter the time factor. This new time is entered into the *time synchronisation* table. If the time in the *time synchronisation* table has not been altered then the simulation and all other processes resume at the time they were paused. If the time in this table has been altered processes resume at the new time. Processes will also adopt a new execution rate based on the new time factor if required.

Once any database events have been dealt with the system clock is read and simulation time is updated. If necessary, such updates can be conveyed to other processes, for synchronisation purposes, by updating the *time synchronisation* table with the new simulation time. This raises a database event in all the recipient processes and they in turn update their simulation time accordingly.

If there are any message reception events in the *event* table and their scheduled time is equal to or less than simulation time then a sub-procedure deals with these reception events

3.2.1 Message Submission Sub-procedure

The message submission sub-procedure processes any *message submission* events that are pending, in the *event* table. A database query obtains the message number and the scheduled time of the event from the *event* table, the link the message is to be placed on from the *message information* table and the link availability and transmission time from the *connectivity* table. If the link is available then the *message submission* event in the *event* table is updated to a *message reception* event which is scheduled to occur at a time equal to the scheduled time of the *message submission* event plus the transmission time of the link. All *message reception* events in the *event* table are checked to find the *message reception* event with the minimum scheduled time.

3.2.2 Message Reception Sub-procedure

The message reception sub-procedure processes any *message reception* events in the *event* table where the simulation time has reached their scheduled times. A database query obtains the message number from the *event* table, the link the message is on from the *message information* table and the node that is to receive the message from the *connectivity* table. The message number is then inserted into the *mailbox* table of the node that is to receive the message. The processed *message reception* event is deleted from the *event* table. Once all *message reception* events, with the same scheduled time, have been processed the *event* table is searched to find the *message reception* event with the minimum scheduled time.

3.3 Post simulation

Once the simulation is stopped the *mailbox* tables are deleted from the database as they are no longer necessary. All information of the simulation is stored in the tables of the DICE database and can be retrieved after the simulation for analysis purposes.

Post-simulation analysis includes inspection of C3I system characteristics such as bottlenecks; command and information flow parameters; and effectiveness measures. A history and review function is also needed that provides the ability to select and replay aspects of the simulation, including analysis of the impact of key commands or decisions made by artificial or human players. Configuration of the simulation kernel about an Ingres database will facilitate establishment of this analysis capability.

4. DISCUSSION

It should be noted that the communication architecture outlined in this section is intended for the DICE simulation alone and limited to locally distributed processes. Remote participation by human players and the interfacing to peripheral units that are distributed geographically are expected to be achieved through observation of international Distributed Interactive Simulation (DIS) protocols.

The use of an RDBMS in the design and development of the DICE simulation has resulted in significant benefits. The ability to quickly design and build tables to describe the underlying information that the simulation will manipulate and generate is particularly useful in the early stages. This paper has illustrated the use of such tables in the DICE simulation example, as well as the powerful database event feature that the Ingres RDBMS provides. In a multiprocess environment, where processes can be distributed over a number of processors, the importance of controlling access and updates to shared data is critical. An RDBMS typically has established features to facilitate prevention of undesirables such as 'simultaneous' updates to data and deadlock on information.

Not specifically covered in this paper, is the use of the Windows 4GL programming capability associated with the Ingres RDBMS (Ingres Corporation [1991]). Windows 4GL allows rapid development of GUI for which information needs to be obtained from the relational database. The proximity of this GUI development tool with the database, circumvents or makes easier much software code writing to interact with and manipulate the information in the database. Windows 4GL was used to develop some of the facilities associated with the simulation controller and human players.

The possible disadvantages of using an RDBMS in the manner described, are associated with data manipulation through continual hard disk rather than memory access, and the performance overheads that might result. When executed in an interactive mode, the DICE simulation is required to run in real time; also, to allow credible analysis to be conducted, latency in the processing of message submissions and receptions needs to be kept within tight tolerances. Hard disk access is slower than access of memory, and the effect of this on simulation performance will be continually inspected as applications of the DICE simulation are addressed; no problems have been experienced to date. Although faster, manipulation and storage of simulation data in memory is looked at unfavourably to some extent since a lot of information is likely to be generated during a simulation, which might necessitate some periodic saving to disk. Also in the current configuration, if the simulation was unexpectedly terminated (by power failure, for example) for some reason during execution then it could quite easily be resumed from that point since its state at that instant would have been captured entirely in the database. This would be difficult to achieve by other means.

The advantages of using a commercial RDBMS package, rather than writing one's own, are clear since it is not only the relational database that is useful but also the access control and visualisation and monitoring tools of the management system.

5. CONCLUSIONS

This paper has highlighted the use of an RDBMS in a distributed multiprocess simulation to help represent and manage key features of the simulation. Adoption of the RDBMS and associated utilities resulted in easier establishment of information sharing and control capabilities, and quicker development of user interfaces that allow inspection and manipulation of the data in the relational database. Such benefits allowed resources to be applied to other important development areas of the simulation. Similar benefits could be obtained through the use of an RDBMS in other projects.

6. REFERENCES

Australian Government Publishing Service, *Defending Australia, Defence White Paper*, Canberra, 1994.

Davies,M., Strategic command, control, communication and intelligence (C3I) simulation activities, roceedings of Int. Cong. on Modelling and Simulation, Perth, Australia, 591-596, 6-10 December 1993.

Davies,M and C. Gabrisch, The Distributed Interactive C3I Effectiveness (DICE) Simulation Project: An Overview, proceedings of 5th CGF&BR Conference, Orlando, Florida, 15-20, May 1995.

Ingres Corporation, *Introducing Ingres*, California, 1991.