SIMSAT : AN OBJECT ORIENTED ARCHITECTURE FOR REAL-TIME SATELLITE SIMULATION

Adam P. Williams

Space Division
Cray Systems
Transome House
Victoria Street
Bristol BS1 6AH
United Kingdom

ABSTRACT

Real-time satellite simulators are vital tools in the support of satellite missions. They are used in the testing of ground control systems, the training of operators, the validation of operational procedures and the development of contingency plans.

The simulators must provide high-fidelity modelling of the satellite, which requires detailed system information, much of which is not available until relatively near launch.

The short timescales and resulting high productivity required of such simulator developments culminates in the need for a reusable infrastructure which can be used as a basis for each simulator.

This paper describes a major new simulation infrastructure package, the Software Infrastructure for Modelling Satellites (SIMSAT). It outlines the object oriented design methodology used, describes the resulting design, and discusses the advantages and disadvantages experienced in applying the methodology.

Key Words: Simulation, Object-Oriented, Satellite, Ada

1. INTRODUCTION

SIMSAT is being implemented by the European Space Agency (ESA) at its European Space Operations Centre (ESOC) in Darmstadt, Germany, utilising three software companies. The software development team for the project comprises ten staff, split between the three companies. Cray System's team of 5 software and simulation engineers is responsible for the real-time kernel of the system.

2. ROLE OF SATELLITE SIMULATORS

Simulation plays a vital role in the successful operation of satellites. Simulators are used:

- to test the overall satellite control system, providing a realistic set of responses to the developers of such systems
- to train spacecraft controllers, allowing routine and emergency situations to be realistically portrayed
- to validate operational procedures, ensuring that inappropriate commands are not sent to the satellite, in both nominal and contingency situations

At ESOC the satellite simulators generally need to provide ground control centres with real-time telemetry data, via appropriate telemetry links. This must describe the dynamics, operation and status of the satellite and its subsystems, in the same manner as the real telemetry data which the control centre receives from the satellite. The simulator also needs to model the satellites response to telecommands.

In normal operation, the operations control centre (OCC) communicates with the satellite via a number of ground stations.

Alternatively a link is made to the software simulator. The simulator provides realistic modelling of the ground stations, the satellite orbit and environment, the operation of the satellite subsystems and payloads and their response to telecommands. The satellite also generates housekeeping telemetry. This is illustrated in Figure 1.

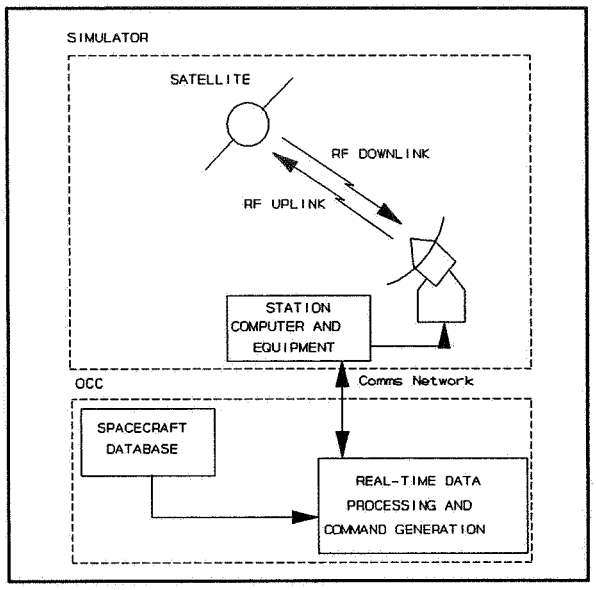

Figure 1 Typical simulator use

The simulator provides various facilities: injection of equipment failures and other events into the simulation (which are manifested to the operator by appropriate changes in telemetry), saving of data sets so that simulations can be started from particular points in the mission and monitoring of information on the health and status of the simulation.

Some simulators also include facilities to allow the on-board flight software to be run as part of the simulation.

As can be seen by the simulator roles described above, a high-fidelity simulation of the satellite and its subsystems is required, in order to give the operators confidence in the telemetry they receive. However the system information required to build a simulator of such fidelity is not available until relatively late in the development life cycle; yet the simulator must be available at least a year before launch, so that it can provide the required support for the necessary planning, testing and training. This means very short timescales for simulator development, hence the need for very high productivity and a sufficiently flexible implementation to accommodate late changes in specifications and requirements.

Since there is significant commonality between different satellite simulators, productivity gains are achieved through providing a common reusable simulator infrastructure which can be used as the basis for each simulator together with a number of general purpose data-driven satellite system models. These include a model of the satellite orbit and the environment within which it operates. Thus the particular satellite simulator developer can make use of generic equipment models tailored by data derived from the latest system documentation, and can reuse (in a realistic manner) previously developed components.

3. OVERVIEW OF SIMSAT

The concept for SIMSAT is to provide those parts of simulators which are application independent as a general purpose system.

Figure 2 shows a typical SIMSAT-based simulator configuration.

3.1 Software Facilities

The operational "kernel" provides general purpose facilities for simulator control, scheduling, error logging and data management. Data Management includes data recording, providing data for display, real-time data access and saving of data sets.

The ground segment simulation includes common equipment models and interfaces to the OCC communications link.

There is a window-based graphics man-machine interface (MMI) used to display simulator information and to control the simulator, and also a reduced-functionality video terminal (VT) display system which provides rudimentary display and control facilities.

The major interfaces between SIMSAT and the satellite-specific code and the position and environment models are the Model Shell and SIMSAT Shell.

All services available to developers are concentrated in the SIMSAT Shell. These include services to schedule models, send telemetry, log events, and obtain key simulator information. These services are implemented using services from other SIMSAT objects, but this istransparent to the developer. This approach insulates the developer from any changes to SIMSAT, and reduces the complexity of the interface with which he has to work.

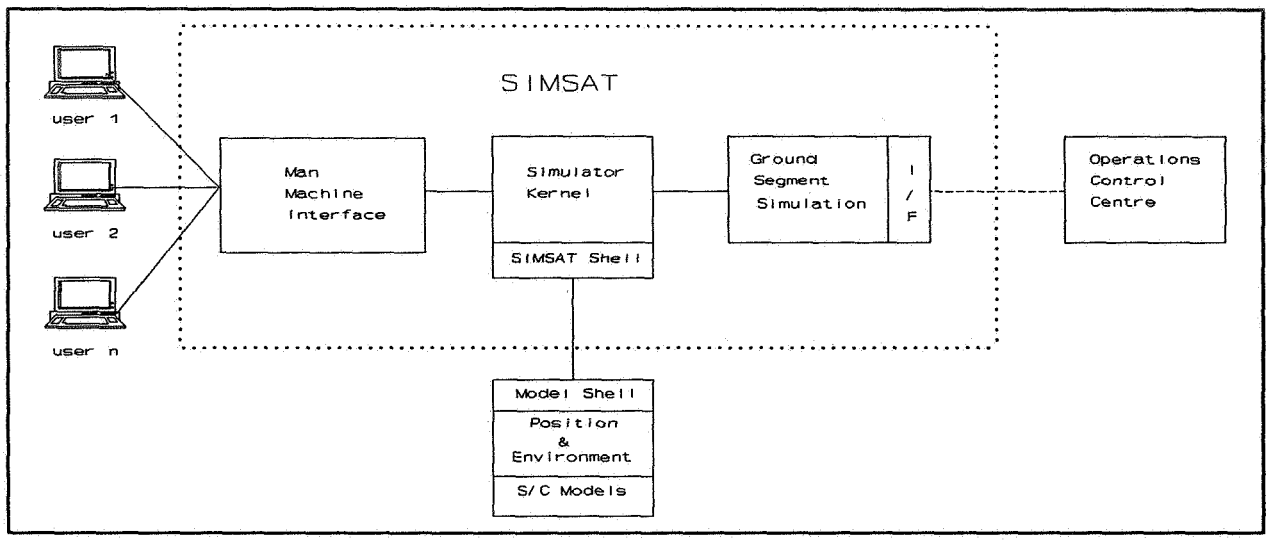

Figure 2 SIMSAT-based Simulator Configuration

The Model Shell is used by SIMSAT to notify the models of the following: change of simulator mode; command received; telecommand event; telemetry event; and model to be executed. The developer then provides an interface layer between this and the satellite-specific models.

SIMSAT will be interfaced to various standard models. These include telemetry encoding and telecommand decoding, power models (solar arrays and batteries) and an orbit and environment model, which includes simulation of the motion of the major celestial bodies (sun, earth, moon) and the perturbations to the satellite orbit caused by these bodies.

In addition there will be tools to assist with modelling of thermal control systems, attitude control systems and power distribution networks.

Simulator developers will then provide satellite-specific models to complete the simulation of their satellite.

3.2 Hardware Configuration

SIMSAT is designed to run on a DEC VAX Cluster. VAXStations host the MMI and/or simulator; a dedicated project computer is used to run the simulator when project resource requirements exceed those provided by a workstation. A file server provides reference versions of SIMSAT and the simulators.

Simulators can be configured in a variety of ways using SIMSAT, varying from running all components on a single workstation, to running the MMI, Ground segment and Kernel/Models on separate nodes of the cluster. Plans are also underway to enable models to be distributed across the cluster.

4. SIMSAT DESIGN RATIONALE

The need for reusability, together with the advantages of data-hiding and encapsulation led to the decision to implement SIMSAT using object-oriented design techniques. The anticipated benefits were as follows:

- o security could be enhanced since object access is only through its operations; access by simulator developers to the internals of the infrastructure parts of the system would not be possible
- o reuse could be enhanced, since the object-oriented approach leads to software modules which are easier to verify.
- o maintainability could be improved, since the well-defined interface specification and well-characterised behaviour of the objects would enable more rapid confirmation that any changes made had not affected the operation of the object or the system.

- implementation of certain objects could be deferred, by hiding the implementation details within the object. The remaining objects could be developed on the basis of the declared interface specification, thus allowing the various parts to be developed independently of each other.

5. SIMSAT DESIGN METHODOLOGY

First the system level data which would form part of the ultimate simulator system was determined from the System Requirements Document. Objects were then identified whose purpose was to manage this data. Access to data was deemed possible only through the operations, termed services, of the object. The details of these services, required both to access the data and to initiate the behaviour of the objects, were then defined. Thus the implementation and structure, of both the data and the behaviour of the object, were not important (or visible) to the clients of the objects.

This approach resulted in 43 system level objects. In order to simplify the management and interaction of these objects, they were subsequently grouped into nine major components, and the interfaces between these components documented in a system level design document and interface specification.

Once the major components had been defined, more specific architectural design was then performed on each, including mapping each object on to the run-time processes. The objects could then be developed relatively independently, the interface specification being used to ensure continued compatibility.

6. OVERALL SIMSAT DESIGN

The resulting SIMSAT design comprises the following major components:

6.1 MMI Display

This is the full functionality display system. It includes objects which provide tabular, graphical and synoptic views of simulator information, which control the configuration of the simulator and which manipulate the log and the event schedule. It also provides commanding facilities.

6.2 VT Display

The reduced functionality display system, with no graphics capability, but able to display data in tabular form, and providing simulator control facilities.

6.3 Display Library Manager

Objects shared between the MMI and VT Displays.

6.4 Ground

Objects which support ground network functions. This includes the **Ground Controller**, the **OCC Simulator** and **Ground Equipment Models.**

6.5 Real Time Nucleus

Objects which provide the real-time core of the simulator: these comprise the **Mode Manager,** which controls the state of the simulator; the **Scheduler**, which together with the **Time Keeper** allows models to be scheduled in real-time; the **Telecommand/Telemetry Streams**, which provide the interface between the ground models and the satellite models; and the **SIMSAT Shell.**

6.6 Command

Objects used to control the simulator: the **Simulator Configuration Manager**, which starts and stops the simulator; the **User Status Manager**, which monitors the status of users connected to the simulator; the **Command Handler**, which allows the user to control the simulator; and the **Command Procedure Interpreter**, which provides facilities for executing groups of commands.

6.7 Data Management

Objects responsible for general management of data: the **Logger**, which records key events in a central history file; the **Public Data Manager**,which provides real-time data access and modification facilities, and can also save sufficient simulator data to allow the simulation to be restarted from the point at which the data was saved; and the **Data Recorder**, which provides recording and playback facilities.

6.8 Models

This component includes the **Model Shell** object.

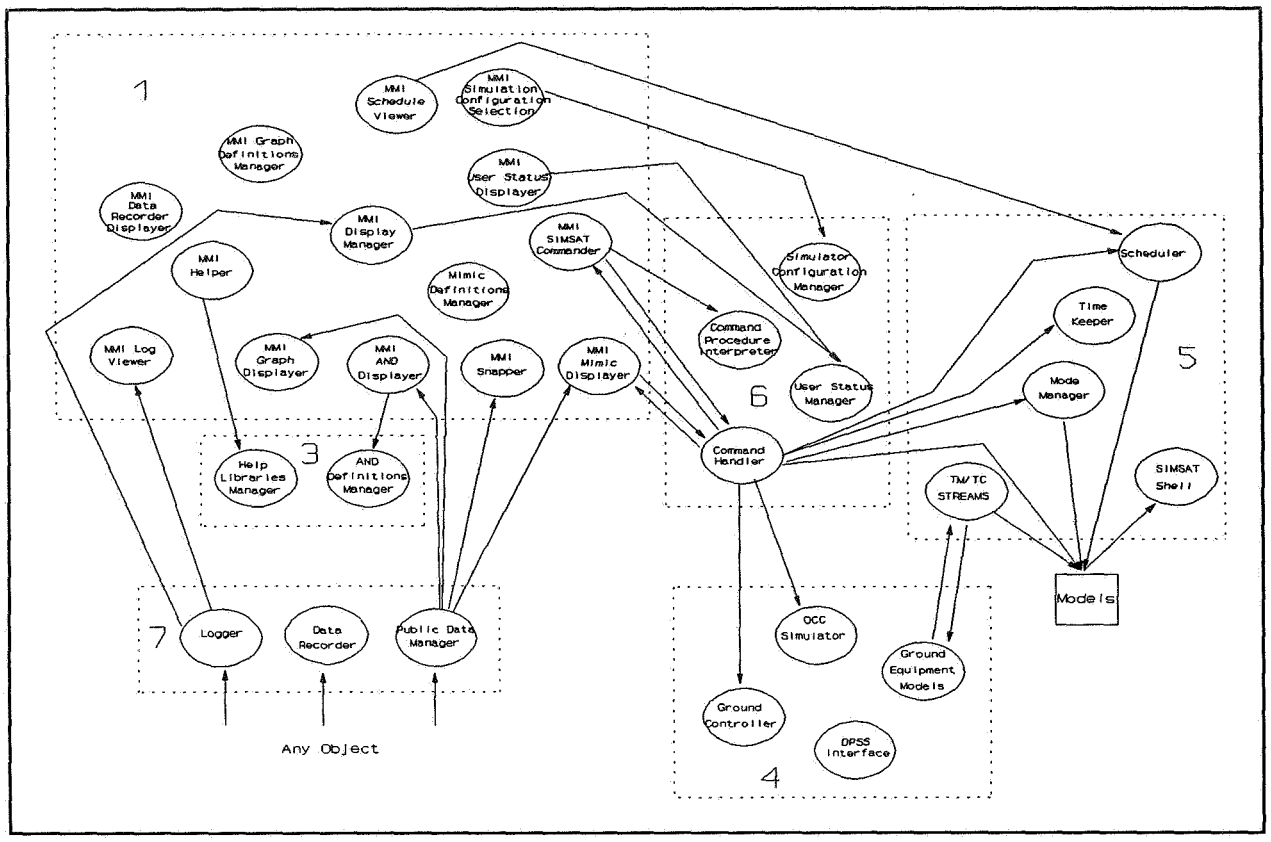

Figure 3 SIMSAT Objects and Service Usage
(Courtesy of ESOC)

6.9 Tools

The component comprises tools which are not internal to any other object. This is principally the **Spacecraft Database Reader**.

6.10 Object Interaction

Figure 3 shows the interactions between the objects, with the major components superimposed. Note that this diagram only shows service usage across major component boundaries. Service usage within major components is regarded as a lower-level effect.

The example shown is for the MMI; the VT case would be similar, since the VT and MMI are simply two examples of the class **User Interface**. Off-line tools are not shown.

Although object services may be used by any other object, there is in fact a relatively low degree of coupling between the major components of the system.

7. CONCLUSIONS

7.1 Benefits of Design Methodology

This section summaries the actual benefits experienced through the use of the object-oriented methodology.

7.1.1 Design Clarity

Specifying objects in terms of their data and operations quickly and clearly revealed any inadequacies in the system design, such as data which was not provided to objects which had need of it, or missing operations which were required. This allowed designers to resolve sooner the issues arising from such problems, when later they might be much more costly or difficult to solve.

7.1.2 Enhanced Security

Since clients did not have access to and were not able to make assumptions about the internal

implementation of an object, the integrity of the object. was maintained. This also insulated system elements from changes within other elements.

7.1.3 Phased Implementation

Since the objects were relatively independent and the work of each team did not need to be tightly coupled to the work of the other teams, a phased implementation of major components of the system was possible.

7.2 Drawbacks of Design Methodology

This section outlines the main drawbacks experienced.

7.2.1 Lack of Dynamic Visualisation

It was difficult to visualise the dynamic behaviour of the system, both in terms of the general interaction of objects and the specific data traffic loads between them.

7.2.2 Ada Constraints

The overheads associated with full data-hiding were not always acceptable. However, this was only of critical importance within the Real-Time Nucleus and certain parts of the Public Data Manager.

The concept of inheritance was not used as part of the implementation. This was because Ada only supports inheritance (and then to only one generation) through the use of types and generics.

There were extra development overheads incurred when extending the visibility of data internal to an object, since specific object services had to be provided.

The above points reflect the fact that Ada could be described as an object-based language, not an object-orientated language.

7.2.3 Documentation

It was necessary to define new documentation standards which suited the object-oriented approach and also met the ESA Software Engineering Standards.

7.3 Summary

SIMSAT is currently still being developed, and therefore it is not possible to report finally on the outcome of using object-oriented design techniques. However, to date, the anticipated benefits of using an object-oriented approach appear to have been confirmed.

An initial version of SIMSAT has been created, and a nominal test simulator defined. This is currently being used to test and verify the system.

The first delivery of SIMSAT is due in the first quarter of 1993, when it will be used in the development of a simulator of the CLUSTER group of satellites, which are being launched in 1995.

8. ACKNOWLEDGEMENTS

The author would like to thank J-J. Gujer and J. Miro for their assistance and general guidance.

Adam Williams is the Project Manager of the SIMSAT Kernel development team, and Cray Systems' Simulations Group Manager at ESOC.