Proceedings of the 1988 Winter Simulation Conference
M. Abrams, P. Haigh, and J. Comfort (eds.)

A space logistics simulation implementation in ADA

Jesus Borrego
Frank Cheng
Dr. Ron Janz
ADVANCED TECHNOLOGY, INC.
222 N. Sepulveda Blvd., Suite 1310
El Segundo, CA 90245

ABSTRACT

This paper describes some of the issues involved in converting a prototype of a large-scale discrete event simulation written in SIMSCRIPT II.5 and FORTRAN into Ada. Specific features provided by SIMSCRIPT that needed to be created in Ada are discussed. Preliminary conclusions regarding the use of both languages for large-scale simulations are then presented.

INTRODUCTION

The Comprehensive Operational Support Evaluation Model for Space (COSEMS) is a discrete event simulation that is being developed by Advanced Technology, Inc., as a subcontractor to General Research Corporation, for the U.S. Air Force Space Division, Director of SDI Logistics. The primary objective of COSEMS is to evaluate alternative logistics support concepts that have been proposed for the Strategic Defense System (SDS). COSEMS does this by computing the resources consumed to implement a given support concept, and the SDS constellation availabilities. The resources consumed are translated into costs by Tecolote Research, Inc. The operational availabilities are used to evaluate SDS constellation effectiveness by initializing force engagement models that have been installed on the National Test Bed along with COSEMS.

COSEMS was written for execution on the VAX family of computers. It evolved from a prototype that was designed primarily to provide an early analysis of various on-orbit support concepts. As refinements in these concepts were implemented, the prototype evolved into a preprocessor, core simulation, and postprocessor. The preprocessor and core simulation consisted of over 7,000 lines of SIMSCRIPT II.5 code. The postprocessor consisted of 6,500 lines of FORTRAN 77 code. This paper describes the SIMSCRIPT portion of the prototype and its translation into Ada. It also describes how the SIMSCRIPT environment facilitated the development of the code, and the tools required for the development of the simulation environment in Ada.

SIMSCRIPT

SIMSCRIPT is a high-order programming language widely used in the simulation community because of its unique simulation environment. It is similar to FORTRAN in its syntax and allows programmers to write simulation programs without concerning themselves with the mechanics of simulation control. The language provides and maintains a system clock. Each process contains an activation time and the environment is responsible for activating the process at the desired time. The environment also provides queue maintenance, so the implementation of the event queue is transparent to the user. Events and processes can be activated by entities. Entities are objects with attributes and are able to hold other entities within themselves; they usually contain information necessary for the activation of processes. Processes can be created, executed, suspended, interrupted, and resumed and are easily maintained by the system.

In the simulation, entities are FILEd in the queue in user-specified priority order, and are available to the program until they are DESTROYed. The language leaves the implementation of the actual data structures to compiler writers. The simulation environment provided by SIMSCRIPT was a major factor in our decision to develop the prototype in SIMSCRIPT.

PROTOTYPE

The preprocessor performs several functions. It reads user options from an ASCII (text) input file. The options include the architecture of the SDS satellite constellations, configurations of any space-based support platforms (SBSPs) and service vehicles associated with the support concept under investigation, and other variables necessary to initialize the simulation. The satellites, SBSPs and service vehicles are created using permanent and temporary entities. The preprocessor then executes scheduling routines to determine when the satellites require support. A temporary entity is created for each service request and FILEd into the SIMSCRIPT internal queue with a given activation time.

After initialization, the core simulation activates processes ordered by increasing time; as processes are activated, time is incremented to reflect simulation time. When a service request is first removed from the queue, the core simulation determines the type of service requested (e.g., corrective maintenance, preventive maintenance, space-based, ground-based, etc.). It then activates a process to handle the request. The processes perform the following services: space-based corrective service, space-based preventive service, ground-based corrective service, ground-based preventive service, space-based satellite replacement service, ground-launched satellite replacement service, and platform resupply service. For more detailed information about COSEMS, see Janz, Cheng, and Borrego (1988).

TRANSLATION ISSUES

As the prototype matured, we decided to translate it into Ada, a high-level language sponsored by the Department of Defense. The Ada language provides generic packages that facilitate software reusability. A generic package is a template for a package. It is declared as a normal package, except for the fact that data items may be declared generically. The actual type of the data is not known until the generic package is instantiated for a given data type (integer, float, record, etc.). For example, if it is desired to write code to sort arrays, most high-level languages require separate procedures for integer arrays, float arrays, and character arrays. In Ada, a package could be written to sort a generic array (any type of array). The instantiation of the package identifies the type of the array, so an instantiation of the package for type integer allows for integer array sorts, while an instantiation of the package for type float allows for float array sorts. Once a package is instantiated, the compiler generates all necessary object code to implement the instantiation. The size of the executable is not reduced by generic packages; it only simplifies code development and maintenance. For more information about generic packages (or the Ada language), refer to Barnes (1984).

The prototype utilized the SIMSCRIPT simulation environment. Ada is a general purpose programming language and does not provide any simulation tools. We considered acquiring commercially available tools to satisfy our software development efforts, but the required licensing agreements were unacceptable for our application. The remaining alternatives were to either develop a complete simulation environment, or to develop only the tools required for our application. Due to time constraints, we chose the latter course. For a more detailed presentation of complete simulation environments, refer to Unger, Lomow, and Birtwistle (1984).

An interesting feature of Ada is the concept of a task. A task is a mechanism provided by the language to allow parallel execution of pieces of code. Tasks may be suspended and resumed. This capability is very attractive for simulation applications, since the events usually overlap, requiring temporary suspension of one event until another one is completed. Unfortunately, tasking in Ada is implemented as a First-In-First-Out (FIFO) queue, unless a priority is assigned to the task. Priorities are assigned at compile time by means of a pragma statement. The difficulty with this approach is that our application determines the reactivation priority at run time, not at compile time.

We feel that tasking in Ada could be improved by adding an attribute to tasks, to uniquely identify that task. The attribute would be treated like a pointer (access type) to refer to a given instance of a task. Our code was written without tasks, thus requiring more logic to support suspension and reactivation of events.

Discrete event simulation also requires a random number generator. Ada, unlike most high level languages used in simulation programs, does not provide a random number generator. Instead of developing our own random number generator, we implemented interfaces with the VAX Run Time Library (RTL).

As part of our simulation tools, we developed generic packages to handle our events and queue control mechanism, using linked lists (access types). For more information about generic packages or access types, see Barnes (1984) or Booch (1987a). Our generic package allows for elements in the lists to be ordered in up to four control fields (keys) and were implemented with generic elements of limited private types. These elements were not defined in the generic package to allow for different instantiation of elements. For example, an instantiation of the package with event information provided the simulation with the Master Event Queue. Another instantiation of the package with text format provided simulation messages for output reporting.

IMPLEMENTATION

To convert the SIMSCRIPT prototype into Ada, the architecture of the program was described using structured analysis and design techniques. While the architecture of the final model was being created, the simulation tools required for the model were being developed. As the architecture of the prototype was mapped, improvements in the design were implemented to take advantage of the features of the Ada language. During this phase of the design, object oriented design (OOD) techniques were used. For more information about OOD, refer to Booch (1987b), or Peterson (1987a, 1987b). Objects created included satellites, SBSPs, orbital replacement units (ORUs), service vehicles, and launch vehicles.

Objects were encapsulated in Ada data structures called packages, together with procedures and functions required to manipulate the data. Access to the data was given only through procedures and functions declared inside the package. The information about the object and the implementation of the procedures and functions were hidden from the rest of the code, to ensure data integrity.

The developed generic packages were used extensively in the implementation, as was a library of reusable simulation components. An example of a reusable component was the linked list generic package previously mentioned. The procedures and functions provided by the package are given in Table 1. The linked list contains pointers to the first element, the current element (last one accessed), and the next element in the list. The position of a new element in the list is determined by a sequential traversal of the list. Each element's control item (key) currently on the list is compared with the new element's control item. The list traversal starts from the current element pointer (last element processed by the scheduler that defines the current simulation time).

The final version of COSEMS may include a more sophisticated search algorithm to speed up seek times if it is determined that queue traversal is too slow.

The linked list generic package was instantiated to manipulate the event queue. Events were ordered in ascending time order. For the events in the event queue, the generic element, called "item," was instantiated as a record structure containing information about the event. Information included object ID, type of event, the source of the event, and IDs of related objects.

In our simulation, a scheduler obtains the events from the event queue and determines what area of the code is responsible for handling the event. The scheduler also obtains the activation time of the next event in the queue. The code handling the event continues execution until the event is completely processed, or until the current time is about ready to exceed the activation time of the next event. At this time, the current event is preempted and filed into the event queue with a new activation time and the next event starts execution. Again, the previous event is not reactivated until its activation time is reached.

Since Ada does not provide a Pascal-like DISPOSE statement, we considered two methods of handling "garbage collection." (Garbage collection is necessary to reclaim storage occupied by data that is no longer needed. Linked list elements are created from available space – "the heap." As elements are created, the heap gets smaller and smaller. If space is not reclaimed after elements are deleted, the available storage space is minimized and therefore wasted.)

The first method consisted of recycling deleted events into a free-space list and then obtaining the next event from this list whenever a new event was desired. The second method consisted of invoking the Unchecked Storage Deallocation ("UNCHECKED_DEALLOCATION") procedure provided by the VAX. This procedure releases storage space previously used by deleted elements. We opted for the invocation of the UNCHECKED_DEALLOCATION procedure.

The items mentioned above, namely the data encapsulation of objects, generic packages for event handling and sorting, garbage

Table 1. Queue Generic Package Software Inventory

Name	Type	Purpose	Remarks
Add	Procedure	Adds elements	Updates pointer
Browse	Procedure	Browse next	No pointer update
Clear	Procedure	Clears queue	Releases storage
Delete	Procedure	Deletes current element	Releases storage
Dispose	Procedure	Deletes first element	Releases storage
Get First	Procedure	Obtains first element	Returns element
Get Next	Procedure	Obtains next element	Returns element
Number of events	Function	No. of elements in queue	Returns integer
Replace	Procedure	Updates element	Updates in place

collection for freed memory, and utility routines such as random number generator, were the key features necessary for the development of our Ada model. Once these items were implemented, the remaining tasks were the rearranging and translation of algorithms and logic into the new structure. With greater control and visibility over individual events, plus flexibility in handling of special cases, the translation and testing were done very smoothly and without major obstacles.

SUMMARY

We believe that SIMSCRIPT language provides an excellent simulation environment and allows for quick development of simulations. It is ideal for prototypes where the concern is to prove a concept before committing the resources required for a full scale simulation. Unfortunately, the language is not sponsored by the Department of Defense and any defense-related simulation implementation has to take into account portability and maintainability. The Ada language is beginning to gain acceptance not only in the defense community, but also in industry and academic communities. Although our software development effort was delayed while we implemented the simulation environment, the time required to develop the tools was not prohibitive, especially since the reusability of the code will make the next simulation effort more attractive in Ada.

REFERENCES

Barnes, J.G.P. (1984). Programming in Ada, Second Edition. Addison-Wesley Publishing Company, Reading Massachusetts.

Booch, G. (1987). Software Components with Ada: Structures, Tools, and Subsystems. Benjamin/Cummings Publishing Company, Menlo Park, California.

Booch, G. (1987). Software Engineering with Ada, Second Edition. Benjamin/Cummings Publishing Company, Menlo Park, California.

Janz, R., Cheng, F., Borrego, J. (1988). COSEMS: A Dynamic Simulation of Space Logistics. AIAA Conference.

Peterson, G.E. (1987). Tutorial: Object Oriented Computing, Volume 1: Concepts. Institute of Electrical and Electronic Engineers (IEEE) Computer Society Press, Piscataway, New Jersey.

Peterson, G.E. (1987). Tutorial: Object Oriented Computing, Volume 2: Implementations. Institute of Electrical and Electronic Engineers (IEEE) Computer Society Press, Piscataway, New Jersey.

Unger, B.W., Lomow, G.A., Birtwistle, G.M. (1984). Simulation Software and Ada. The Society of Computer Simulation, La Jolla, California.

AUTHOR'S BIOGRAPHIES

JESUS BORREGO has 10 years experience in software engineering, 5 of them in simulation and modeling. He is presently a Technical Leader on the COSEMS software development effort. He holds a B.S. in Electrical Engineering from California State University, Fullerton and a B.S. in Computer Science from California State University, Dominguez Hills. He is currently pursing an M.S. degree in Computer Science from the University of Southern California.

Advanced Technology, Inc.
222 N. Sepulveda Blvd., Suite 1310
El Segundo, CA 90245
(213) 640-1050

FRANK CHENG has 3 years experience in scientific simulation and modeling. He is presently a Technical Leader on the COSEMS software development effort. He holds a B.S. in Applied Mathematics from the University of California at Los Angeles and is currently pursuing an M.S. degree in Computer Science from the University of Southern California.

Advanced Technology, Inc.
222 N. Sepulveda Blvd., Suite 1310
El Segundo, CA 90245
(213) 640-1050

DR. RON JANZ has 20 years experience in formulating models/simulations for diversified engineering applications. He is currently Technical Manager of the COSEMS development effort. Prior to joining Advanced Technology, Inc., he was Principal Director of the Mathematics and Programming Subdivision at The Aerospace Corporation. He holds a Ph.D in Applied Mathematics from Northwestern University.

Advanced Technology, Inc.
222 N. Sepulveda Blvd., Suite 1310
El Segundo, CA 90245
(213) 640-1050