A Tutorial on Discrete-System Simulation in Ada

Raymond M. Bryant[1]
IBM Thomas J. Watson Research Center
Yorktown Heights, New York 10598

Summary

Introduction

Ada[2], the new Department of Defense standard language, contains many new features designed to facilitate the rapid construction of software for embedded computer system applications. However, Ada is also suitable for general purpose programming, and the size and backing of the Ada project means that the language will enjoy considerable use and support. It is therefore important that simulation professionals are aware of the features of Ada suitable for the construction of discrete-system simulations. In this tutorial we provide a brief introduction to these features and we outline the implementation of a process-oriented simulation in Ada. We also describe how a SIMULA-like "hold" procedure can be implemented as a package in Ada.

A summary of the presentation is given below; for further details see [4] and the references given there.

Strong-Typing

Like Pascal [6], Ada is strongly-typed. The contemporary view is that strongly-typed languages assist in the construction of clear and readable programs. Additionally, many common programming mistakes can often be detected during compilation of a program written in a strongly-typed language (see, for example, [5]).

The type definition facilities of Ada can be used to define the entities of a simulation. For example, one can use record types to represent permanent entities. Temporary entities can be represented using pointer types; these types are called access types in Ada.

Packages

A package is a "set of facilities provided for the benefit of other packages and procedures" [3]. Packages allow a programmer to create a collection of routines and data structures that can be compiled separately and then included as a unit in other programs. Additionally, the package declaration specifies the interface of these routines to the outside world in a way that allows enforcement of Ada's strong-typing rules.

For example, packages to define a queue of entities can readily be implemented in Ada. The package declaration includes routines to do queue initialization, entity insertion and removal, and a data type that allows the user to declare queues. The package can be constructed so that the queue type is accessible to the user, but so that the internal details of the queue data structure are invisible to the user. This insulates the user from changes in the queue implementation.

A single queue package can be used to implement queues of different types of objects by making the package a generic package; multiple entity types can be placed in a single queue (if so desired) by representing the entitiesusing a record with variants.

Tasks

An Ada task represents an independent execution that runs in parallel (at least conceptually) with all other tasks. However, unlike a process in SIMULA [2], a task cannot be assumed to run to completion before the physical processor is reassigned to another task. SIMULA-like processes can be implemented in Ada through synchronization primitives that only allow one task to run at a time.

[1] This work was performed while the author was with the Computer Science Department, University of Wisconsin-Madison, Madison, Wisconsin 53706, and was supported in part by the Wisconsin Alumni Research Foundation and NSF grant MCS-800-3341.

[2] Ada is a registered trademark of the U. S. Department of Defense.

Proceedings of the 1982
Winter Simulation Conference
Highland * Chao * Madrigal, Editors

82CH1844-0/82/0000-0643 $00.75 © 1982 IEEE

Tasks communicate by calling entries in other tasks. The receiving task must be waiting at a corresponding accept statement when the call occurs; otherwise the calling task is delayed. Similarly, the called task is delayed at the accept statement until some other task executes a call on that entry. When both tasks have reached the corresponding statements (this is called a rendezvous) the body of the accept statement is executed. The calling task is delayed until the body of the accept statement has been completed.

Using task types one can define synchronization objects [&barnes]. Tasks using these objects wait at "wait" statements until another task executes the appropriate "send" call. Synchronization objects can be used to implement SIMULA-like processes in Ada.

A Simulation Package

Combining the ideas of packages, tasks, and synchronization objects, one can define a general, process-oriented simulation package in Ada. This simulation package will have the advantage of portability among Ada implementations, but also will be process-oriented rather than event-oriented. Finally, since it is written in a strongly-typed language, a simulation package in Ada provides the advantages of strong-typing for the simulation programmer. We provide an outline for such a package and illustrate its use in a simple example [4]. Although it has yet to be tested on a production Ada compiler, we feel that standard packages like this will eventually become as portable and well-known as event-oriented packages such as GASP-IV [7].

References

[1] Barnes, J. G. P., "An Overview of Ada," Software -- Practice and Experience 10, pp. 851-887 (1980).

[2] Birtwistle, G. M. et al, SIMULA Begin, Petrocelli/Charter, New York (1973).

[3] Brender, R. F. and I. R. Nassi, "What is Ada?," IEEE Computer 14, 6, pp. 17-24 (June 1981).

[4] Bryant, Raymond M., "Discrete System Simulation in Ada," Simulation, (to appear October 1982).

[5] Bryant, Raymond M., "SIMPAS -- A Simulation Language Based on Pascal," Proceedings of the 1980 Winter Simulation Conference, pp. 25-40 (December 3-5, 1980).

[6] Jensen, K. and N. Wirth, "Pascal: User Manual and Report," Lecture Notes in Computer Science 18, Springer-Verlag Berlin, New York (1974).

[7] Pritsker, A. A. B., The GASP-IV Simulation Language, John Wiley and Sons, Inc., New York (1974).