Proceedings of the 1986 Winter Simulation Conference
J. Wilson, J. Henriksen, S. Roberts (eds.)

Object oriented simulation - Ada, C++, Simula

Brian W. Unger
Dept. of Computer Science
University of Calgary
Calgary, Alberta, Canada T2N 1N4

The object oriented design of simulations is based on the concept of abstract data types. An early mechanism for defining abstract types was incorporated in the Simula Language. Constructs were also included in Ada that support this view. C++ is a general purpose programming language in which facilities for defining abstract types have been added to the C language.

An abstract type can be defined as a data structure and a set of operations on instances of that structure. The simplest example is the integer. An integer may be represented by a data structure which consists of a sequence of zeros and ones which are interpreted as in ones complement arithmetic. The operations on instances of type integer, i.e., integers, are the integer :=, +, -, *, and /. Thus, the integer abstract type has an underlying data structure which is used to represent instances of type integer, and it has a set of operations defined for instances of type integer.

Object oriented methods

Instances of an abstract type are called "objects". Thus an integer can be said to be an instance of type integer and the integer operations are defined for objects of this type. The implementation of the underlying data structure and operations are of little interest to the user of integers. Few programmers are interested in whether integers are represented in ones or twos complement form or what algorithms are used to multiply integers in one of these representations. However, the programmer needs a clear idea of how to use integers in a program.

The concept of abstract types extends this idea to enable user defined abstract types. For example, the user may want to define an abstract type "stack" with operations pop and push. Objects of type stack may then be defined and items pushed onto or popped off one of these objects.

The concept of writing programs that define abstract types and then create and manipulate objects of these types is one of the most important programming language developments of the decade. Not only is the structured implementation of programs easier but the design process is directly supported. The integration of design and programming in the development of simulations is an extremely important aspect of this technique.

The world that we are trying to create when building programs can be thought of as consisting of objects which interact. This world of objects can be divided into classes of different kinds of objects. i.e. different abstract types, and then multiple objects of each type. Thus design involves dividing the world of interest into different kinds of objects. Objects within each class or type will be similar but not identical. That is, objects of type stack are all similar in that they contain a linear list of elements, and that elements can be pushed onto this list or popped off. However, multiple objects of this type can also be quite different in that they will have unique contents and relationships with other objects.

The objective of design is to decompose the system into a set of modules with simple interfaces. We now have genuine help in this process. Different types of modules are defined which will have identical functions or operations but may also have important differences. Modularity is strongly supported because the internal implementation of these object types need not concern the user of these objects.

Programming languages as tools

The use of structured programming using a high order language, and abstract typing during both the design and programming phases, hold the greatest promise for dramatically improving productivity in the development of simulations. Both of these objectives are strongly supported within the Simula, Ada, and C++ languages.

An excellent presentation of the history of programming languages before the appearance of Ada is contained in [Wegner, 76]. A few of the major points of this paper are summarized in Figures 1, 2, and 3. Wegner describes 30 milestones in the history of language development. He characterizes the 1950s as the empirical discovery of programming methods with Fortran identified as one of the major milestones of this period. Fortran's contribution was primarily in the representation of arithmetic expressions and the introduction of subroutines. The ability to directly write arbitrary expressions in a familiar form was a large advantage over assembly languages. The subroutine was the first unit of program modularity.

The 1960s are described as a period of theoretical development. Major languages of this decade were Cobol and Simula. Cobol introduced new ways to characterize data, and Simula introduced many concepts and constructs that would take some time to be recognized. The design of a general purpose language with simulation as a target application was an excellent model for later language developments. The world of interest to the modeller consists of multiple concurrently operating objects. Simula introduced classes and class objects to characterize this phenomena. This also happens to be the model of the modern operating systems, such as UNIX, where the support of concurrent processes is a major objective.

Wegner considers the 1970s decade to be one of consolidation where many of the results of language theory were applied to construct new languages and systems. Pascal and the concepts regarding program modularity were major developments of this period. The Simula class was a new unit of program modularity. Twenty years after the subroutine, function, and procedure are introduced we finally see a truly new construct that supports program modularity. This construct forms the basis of abstract types and the methods of object oriented design.

1950's - Empirical (discovery & description)

1960's - Mathematical (elaboration & analysis)

1970's - Engineering (technological & management of complexity)

1980's - ?? Networking, Expert Systems ??

Figure 1. Wegner's First 25 years of Programming

M1:	EDVAC Report	1944
M2-M4:	subroutines, macro-assemblers	
M5:	Fortran	1954-1958
M6:	Algol	1957-1960
M7:	Cobol	1959-1961
M8:	PL/1	1964-1969
M9:	Algol 68	1963-1969
M10:	Simula 67	1965-1967
M11-M13:	IPL, LISP, Snobol	1962-1967
M14-M23:	Language Theory....	1960-1970"
M24:	Pascal	1967-1975
M25:	Apl	1960-1967
M26-M27:	Structured Programming	1969-1979
M28	Software Life Cycle	
M29	Modularity	1955-1975
M30	Data Oriented Languages	

Figure 2. Wegner's 30 Milestones

The impact of the Simula class is illustrated in Figure 3. Many experimental languages appeared in the 1970s that incorporated some form of class construct within a Pascal based syntax. The class also embodied a new approach to the structuring of data and provided the first mechanism that enabled user defined abstract types. An abstract type could now be declared, objects or instances of that type could be created, and a unique set of operations defined for these objects. The modularity concepts that originally appeared in Simula, the new data structuring concepts, and the more attractive syntax of Pascal all contributed to the design of both Ada and C++.

The Ada package, structured types, and generics were all outgrowths of these 1970s language developments. The goals of Ada however, were still more ambitious. Ada's objectives included all encompassing generality, i.e. the support of scientific, financial, real time, and embedded applications. One language was desired that would satisfy the needs of all of these application areas.

M29: Modularity 1955-1975

--Fortran SUBROUTINE
--Simula CLASS

-CLU cluster hierarchical models
-Alphard form environment
-Concurrent Pascal process class, monitor
-Modula module
-Euclid module
-Mesa, Gypsy, ..

M30: Data Oriented Languages 1960-1975

--Cobol
--Simula
--IMS/IDS
--DBMS ...

M??: Ada 1975-1983

Figure 3. From Pascal and Simula to Ada

Portability was also a major goal. Not only program portability, but the ability to easily move software development tools, projects, and programmers from one Ada system to another. This is one of the primary reasons for the reluctance to allow language subsets.

The extremely ambitious goals of Ada have resulted in a very complex language. This complexity significantly effects the value of Ada in improving productivity in the development of simulation. C++, however, incorporates most of the abstract typing facilities that originally appeared in Simula while retaining the simplicity, portability, and efficiency of C.

REFERENCE

Wegner, P. (1976) "Programming Languages - the First 25 years" IEEE Transactions on Computers, 25 (12), December.