Compiling a simulation language in APL

Manuel Alfonseca, Enrique Alfonseca, Juan de Lara

Acknowledgement: This paper has been sponsored by the Spanish Interdepartmental Commission of Science and Technology (CICYT), project number TIC-96-0723-C02-01.

Abstract

This paper describes the procedure used to build several compilers, written in APL and APL2, to translate two continuous simulation languages into APL and C++. The advantages and disadvantages of using APL to write a compiler are discussed. A compromise had to be found between performance (the model execution speed) and flexibility (the ease to modify parameters and test "what if" situations). The resulting compiler (an APL2 packaged workspace) has been used successfully to generate educational applications and in medical research.

Introduction

System simulation [1] is one of the oldest branches of computer science. It was well advanced in the sixties, and came to maturity in the seventies. Its objective is to build a "model" of a real system, i.e. a system that behaves in a similar way as the real system, but is easier to study and experiment with. If the model is a set of mathematical equations, it is called a "mathematical model". If it is a computer program, we have a "computer model". There are three kinds of computer simulation:

- Analog simulation: time is continuous. Models are built by means of electronic circuits in an "analog computer", invented by Vannevar Bush in the nineteen thirties. Analog computers are now outdated.

- Digital continuous simulation: time is represented by the set of multiples of a fixed (or quasi-fixed) time step (the elementary interval). This is equivalent to representing a continuous function by a set of samples at fixed intervals. The appropriate mathematical tool for continuous simulation is the set of algebraic-differential equations.

- Digital discrete simulation: time is represented by an ordered (growing) set of discrete values. The difference between two successive values is usually random. The appropriate mathematical tool for discrete simulation is the Markov chain.

Continuous simulation is programmed either in a special purpose language, or in general purpose code. Continuous simulation languages may be of different kinds, depending on their syntax:

- Block languages: each instruction represents an "electronic block", similar to those traditionally used in analog computers [2].

- Mathematically oriented languages: the mathematical model may be used almost directly as the source program. CSMP (Continuous System Modelling Program) [3] was broadly used during the seventies and the eighties.

- Graph languages: the mathematical model is represented as a special kind of directed graph (the bond graph) [4], or by a systems dynamics graph, using the terminology and symbols proposed by J. Forrester [5].

APL Continuous System Modelling Program

In the late nineteen seventies, one of the authors of this paper implemented in APL a compiler for a subset of the CSMP continuous simulation language [6]. This compiler was subsequently announced by IBM as three different products running under APL interpreters in the mainframe computers of the time [7].

The CSMP language has the following features:

- Models are directly entered in the form of mathematical equations.
- The language provides a large number (56) of predefined functions and blocks, including a derivator, an integrator, many mathematical functions, wave generators, arbitrary function generators, switches, logical gates, flip-flops, a delay, a solver of implicit functions, and a block displaying the behavior of a hysteresis loop.
- Several integration methods are provided, which may be applied to different modelling situations. Some of the methods provide for automatic adjustment of the time interval to comply with the selected error bounds.

The APL CSMP compiler has the following features and advantages:

- It translates CSMP code into APL programs, generated by means of the function-fix system function.
- Simulation parameters (including those for simulated time control) become ordinary APL variables with the same names, which means that their values may be changed at any point by simple APL assignments. This makes it very easy to run interactive "what if" experiments, one of the main justifications of modelling and simulation.
- The generated APL programs may be invoked and controlled by other APL programs. In this way, automatic parameter adjustment or boundary problems may be easily solved.
- Powerful APL graphic capabilities (such as those provided by the AP206 and AP207 auxiliary processors) may be used by the simulation programs to display data.

The main disadvantage of the APL approach is the execution speed of the simulation models. Their translation into APL implies that they must be executed under an interpreter, with the subsequent performance degradation. On the other hand, continuous simulation models are typical examples of programs that contain loops that cannot be eliminated by means of matrix operations: as simulated time goes by, the values of all the variables in the model are updated as a function of the preceding values.

Listing 1 shows a model written in CSMP, that solves the well-known Volterra equations [8] for a two-species system.

```
TITLE VOLTERRA EQUATIONS
DATA M:=5,N:=.3,P:=.5,Q:=.3
DATA X0:=2 Y0:=20
XP:=(N*X*Y)-M*X
YP:=(P*Y)-Q*X*Y
X:=INTGRL(X0,XP)
Y:=INTGRL(Y0,YP)
TIMER
delta:=0.01,FINTIM:=10,PRdelta:=
0.1
PLOT 20 50 X Y TIME
METHOD ADAMS
```

Listing 1. A model of an ecological system written in CSMP

The OOCSMP language

Object-oriented programming originated in the sixties in a discrete simulation language, SIMULA67 [9], which incorporated many of the ideas later included by Alan Kay in the first general purpose object-oriented language, Smalltalk [10], and by Bjarne Stroustrup in C++ [11]. Object-oriented continuous simulation languages and tools, however, took longer to arrive.

We have used the object-oriented concepts to design a new simulation language (OOCSMP), a pure extension of CSMP, with some features that help to simplify the models of systems with several equivalent interacting components. The simplification obtained may be very significant. The main object-oriented extensions added to the language are:

- Definition of objects and classes.
- Simple inheritance.
- Vectors of objects.
- Definition of attributes and functions (methods) associated to a class of objects.
- A simple way to reference object and object vectors attributes and methods.

Additional extensions allow us to:

- Include previously defined classes in a new model.
- Specify different simulation runs for the same model.

Listing 2 shows an example of a model written in OOCSMP, which makes use of its object-oriented extensions to simulate a multi-level multi-species ecological system.

```
TITLE EXTENDED VOLTERRA EQUATIONS

************************************
* Definition of the Species class  *
************************************

CLASS Species {
 NAME name
 DATA M, X0, start
 X:=STEP(start)*LIMIT(0,1000,XT)
 XT:=INTGRL(X0,XP)
 XP:=M*X
 PLOT X TIME
}

************************************
* Definition of the Plant class    *
************************************
CLASS Plant : Species {
 DATA AHPl
 XP-= AHPl*X*Herbivore.X
}
************************************
* Definition of the Herbivore class*
************************************
CLASS Herbivore : Species {
 DATA APlH, APrH
 XP+= APlH*X*Plant.X
 XP-= APrH*X*Predator.X
}
************************************
* Definition of the Predator class *
************************************
CLASS Predator : Species {
 DATA AHPr
 XP+= AHPr*X*Herbivore.X
}
************************************
* Actual species                   *
************************************
*
* Level 0: plant
Plant     PA ("Plant", 0.4, 100,  0,
      0.02)
* Level 1: herbivore
Herbivore HA ("Herb1",-1.28, 20,  0,
      0.02, 0.36)
Herbivore HB ("Herb2",-1.28,  5, 10,
      0.02, 0.12)
* Level 2: predator
Predator  PrA("Pred1",-7.2,   2,  0,
      0.36)
Predator  PrB("Pred2",-4,     2, 10,
      0.20)
STEP(PA)
STEP(HA)
STEP(HB)
STEP(PrA)
STEP(PrB)
************************************
* Timer and show data
************************************
TIMER delta:=0.005,FINTIM:=40,
      PRdelta:=0.1,PLdelta:=0.1
METHOD ADAMS
```

Listing 2. An extended model of the ecological system written in OOCSMP

The OOCSMP compiler

A new compiler has been written in APL2 to translate OOCSMP code. To overcome the performance problem, it was decided to generate C++ code instead of APL. This means that something had to be done to maintain at least part of the flexibility of the previous simulation environment.

The OOCSMP compiler reuses a large part of the code of the previous CSMP compiler written in APL, including the following sections:

- The whole scanner (lexical analyzer).
- Most of the parser (syntax analyzer), which has been extended to incorporate the new features.
- The implementation of the old CSMP blocks, except for the code generator instructions.

Of course, since the target language is different, the code generator had to be completely replaced. However, the effort needed to build the new compiler (with a total of about 1200 APL2 lines) was surprisingly small: It took one person about three weeks, in spite of the fact that the old compiler (which is made of about 900 APL lines) had been written seventeen years before (by the same person) and was sparingly commented. We think that using APL and APL2 made this programming performance possible. In a typical systems language, the effort required would have been much larger.

To maintain the interactive simulation environment provided by the previous compiler for the APL translated models, the new compiler provides the option to generate a main program written in C++ that makes it possible to:

- Modify the values of the model parameters.
- Modify the attributes of all the objects in the model.
- Adjust the values of the time control parameters.
- Run simulations of the model or continue the execution of the previous simulation.
- Show the results as a graphic plot, fixed or animated.
- Print the results in a text file.

The main program provides a graphic interface built by means of calls to a C++ library designed and written by one of the authors, that implements a DOS message-based window system with multitask, similar to higher level operating subsystems. Figure 1 shows the appearance of the graphic interface during the execution of the ecological system represented by the model in listing 2.

If this option is not used, The APL2 compiler generates only the simulation routines, which may be invoked from any hand-written main program. In this way, automatic parameter adjustment or boundary problems may be solved, although in a slightly less simple way that that provided by the old APL compiler, as the main program must be written in C++.

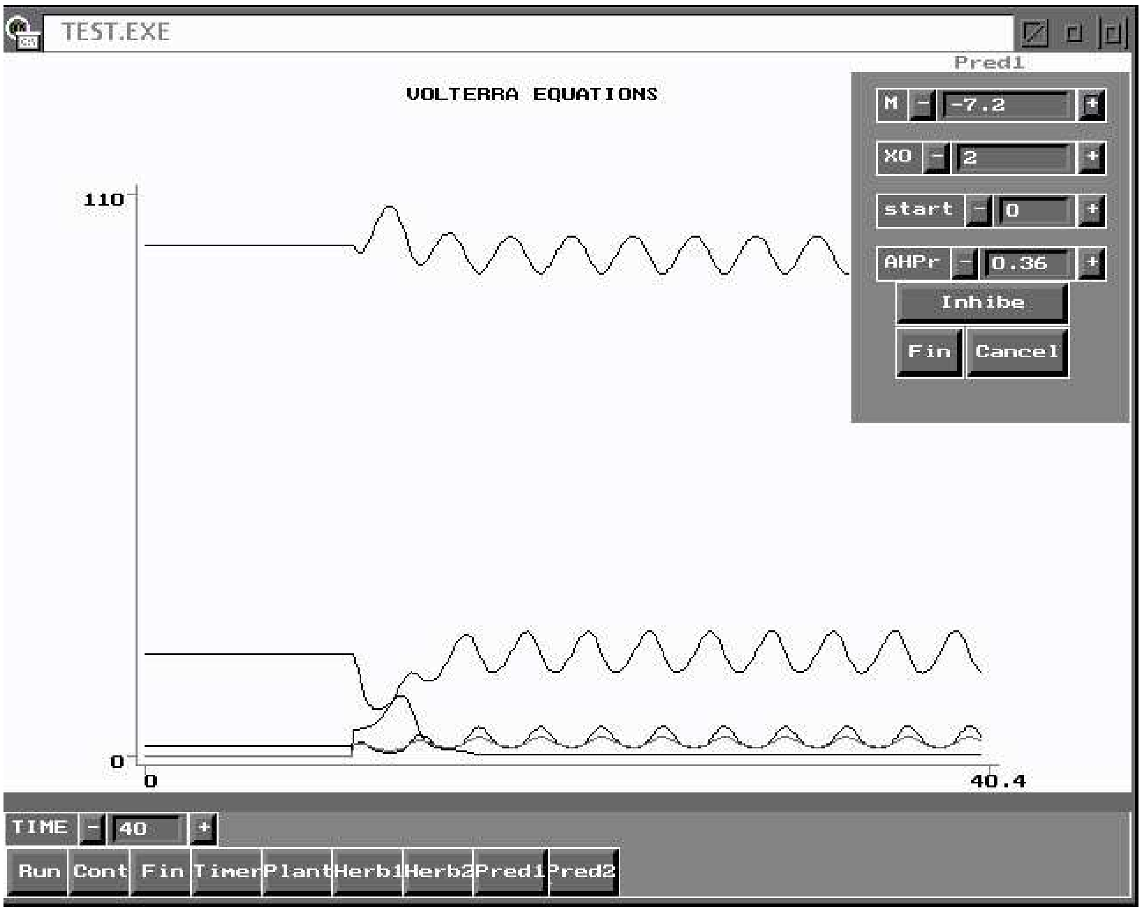
The main advantage of the new compiler, compared with the old one, is the execution speed of the simulation models, which is about one order of magnitude faster, more than balancing the slightly lesser flexibility of use.

The APL2 written OOCSMP compiler has been built as a packaged workspace that may be invoked from a DOS session, either natively, under Windows 95 or OS/2. Another compiler for OOCSMP built by our group [12] generates optionally C++ and JAVA code. In the first case, we can use the same graphics library described above to generate a simulation environment for DOS, or the Amulet graphics interface developed by the Carnegie-Mellon University [13] under Windows 95 and UNIX. In the JAVA case, the standard JAVA user interfaces are used. The compiler also generates an html skeleton embedding the JAVA applets.

These compilers have been used to translate models written for educational purposes. Two of these models are available in the Internet at the following addresses:

- A simulation of the solar system and Newton's universal gravitation law: http://www.ii.uam.es/~epulido/newton/grav.htm

- A simulation of an ecological system: http://www.ii.uam.es/~epulido/ecology/simul.htm



M -7.2
X0 2
start 0
AHPr 0.36
TIME 40

Figure 1: The graphic interface during the execution of the model of an ecological system

The APL2 compiler has also been used in postgraduate courses on continuous simulation, and for medical research, to simulate hepatitis B and AIDS epidemics in Spain [14].

Conclusion

The experience of writing in APL a compiler for a special purpose programming language (a continuous simulation language) was useful for the following reasons:

- Writing the compiler in APL made its construction much faster.

- Translating the models into APL makes available to them all of the APL workspace facilities, providing the compiled models with a great flexibility and allowing the modeller to test "what if" conditions with ease, thus fulfilling one the essential requirements of a good simulation environment.

- However, the resulting models are slow, since they are written in a language which must be interpreted.

Using this experience, we wrote a second compiler in APL2 with the following results:

- It was possible to reuse a large part of the old APL compiler, even though we changed both the source and the target language.

- The effort required to include object-oriented extensions in the source language (OOCSMP) was very small, thanks to the fact that the compiler was programmed in APL2.

- The new compiler generates C++, which means that the executable models are one order of magnitude faster. The automatic generation of a graphics user interface makes it possible to maintain most of the flexibility provided by the old compiler.

References

[1] Y.MONSEF, *Modelling and Simulation of Complex Systems*, SCS Int., Erlangen, 1997.

[2] M.ALFONSECA, "SIAL/71, a Continuous Simulation Compiler", *Advances in Cybernetics and Systems*, Ed. J. Rose, Gordon and Breach, London, Vol. 3, 1974, 1319-1340.

[3] IBM CORP., *Continuous System Modelling Program III (CSMP III) and Graphic Feature (CSMP III Graphic Feature) General Information Manual*, Ontario, GH19-7000, 1972.

[4] KARNOPP, D., "Bond Graph Models for Electrochemical Energy Storage: Electrical, Chemical and Thermal Effects", *Journal of the Franklin Institute*, Vol 324, 1990, pp. 983-992.

[5] LEGASTO JR., A. A.; FORRESTER, J.W.; LYNEIS, J.M. editors, *Systems Dynamics*, North Holland, 1980.

[6] ALFONSECA, M, "APL Continuous System Modelling Program: an Interactive Simulation Language", *Advances in Engineering Software*, Vol.1:2, 1979, 73-76.

[7] IBM CORP., *IBM System/370 APL Continuous System Modelling Program Program Description and Operations Manual*, Moline (Ill.), SH20-2115, 1979.

[8] VOLTERRA, V., *Leçons sur la Théorie Mathématique de la Lutte pour la Vie*, Gauthier-Villars, Paris, 1931.

[9] DAHL, O.J.; NYGAARD, K, "SIMULA - An ALGOL-Based Simulation Language", *Comm. ACM*, 9:9, 1966, pp.671-678.

[10] DIGITALK INC., *Smalltalk/V PM*, Digitalk Inc., Los Angeles, 1990.

[11] STROUSTRUP, B, *The C++ Programming Language*, Addison-Wesley, Reading (Mass.), 1991-1997.

[12] ALFONSECA, M.; PULIDO, E.; LARA, J.; OROSCO, R.: "OOCSMP: An Object-Oriented Simulation Language", *Proc. 9th European Simulation Sympossium ESS97*, SCS Int., Erlangen, 1997, pp. 44-48.

[13] MYERS, B.A.; BORISON, E.; FERRENCY, A.; MCDANIEL, R.; MILLER, R.C., FAULRING, A.; KYLE, B.D.; DOANE, P.; MICKISH, A.; KLIMOVITSKI, A., *The Amulet V3.0 Reference Manual,* Carnegie Mellon University School of Computer Science Technical Report no. CMU-CS-95-166-R2 and Human Computer Interaction Institute Technical Report CMU-HCII-95-102-R2, March 1997.

[14] ALFONSECA, M.; GARUZ, R; TORREA, J.L.: "Some remarks about Hepatitis B and AIDS using mathematical models", *I International Congress on Models and Mathematical Methods Applied to Biology and Medicine*, Alicante, Jul. 1997.