Panel: Aims, Means, and Futures of Object-Oriented Languages

Mike Banahan, Chairman of the European C++ User Group
L. Peter Deutsch, Sun Microsystems Laboratories Inc.
Boris Magnusson, University of Lund
Jens Palsberg, moderator, Aarhus University

Abstract. Panelists will compare and assess the strengths and weaknesses of major object-oriented languages. They will also comment on the possible development and use of those languages and their related tools.

1 Background

Many object-oriented languages are in use today. From the programmer's perspective, they differ both when comparing language features and tool support. These differences have impact on programming style and programmer productivity, and also on how well a language is suited for a particular development project.

This panel focuses on three major object-oriented languages, namely Simula, Smalltalk, and C++. We will compare and assess their strengths and weaknesses, and hazard guesses about future developments of the languages themselves and their related tools.

2 Mike Banahan

The C++ language is currently receiving a lot of attention, especially in the PC and Unix communities who view it as a natural fit to many of the problems of their environments. C++ is viewed as an easy migration path from the most common language already in use there (C), since it is almost entirely upwards-compatible with C and yet offers the promise of stronger type checking, better encapsulation and the chance to implement Object Oriented Concepts. All at the same time!

C++ sets out to do much more than C. It is a considerably larger language and requires a different approach to development than does C. As many organisations have discovered, using a particular language does not guarantee that the software developed will necessarily make the best use of the facilities available.

Strong type checking and Object Orientation sound like a good combination of buzz-words. There is much to be said for it, and the wise use of C++ can be extremely powerful. Working together, those notions force designers to think

hard about the nature of type relationships within program designs. The availability of generic types and exception handling takes the language out of the realm of concepts familiar to most procedural programmers.

As a result, there is not only a new language to learn, but also many programming techniques which are new, exciting and difficult.

Your speaker will be happy to share the insights that he has been able to gain during his eight years' of using C++.

3 Boris Magnusson

Simula was the first object-oriented language and introduced all the now popular concepts associated with o-o (and a few more). The language and its implementations are stable, efficient and reliable and thus provides the framework for many projects. From the language point of view, some of the strength of Simula comes from the combination of o-o constructs it offers. In this company (with C++ and Smalltalk) its tempting to mention compile time typing *and* garbage collection. Simula also offers important unusual mechanisms as unlimited *nesting* of constructs (classes defined inside classes as procedures inside procedures in many procedural languages) and *processes* (co-routines, lightweight processes). Nesting is important to cope with large applications. Processes are important to deal with external communication as in user interfaces and in client-server applications. There are thus mature o-o systems available.

Implementations of o-o languages using traditional techniques and conventional tools in the bottom, such as text editors, linkers and Unix utilities exhibit some common problems:

- Selective loading from libraries does not work so well, since all possible virtual implementations will be included (back side of the coin of dynamic binding)
- More compilation module dependencies means more re-compilations (back side of inheritance and re-use). This can to some extent be fought by fast compilers but as programs grow...
- Utilities such as "grep" does not work so well with o-o languages since it will find all implementations of a procedure (not just the one called from here (back side of several names-spaces created by encapsulation).

These new problems (compared to procedural languages) call for more research in implementation techniques. Tight integration incremental techniques such as those developed in the Smalltalk environment and in the Mjolner project gives some answers, but much more remains to be done before these techniques are in common use.

"SIMULA—Common Base Language" was the title of the definition. Already here we are given a hint of another important direction in o-o development—to see o-o languages as specialized application languages. This is an extremely important way to fight the complexity of large systems—reducing their size by increasing the level of the language used (to say more per source line, in

some sense this is what language development is all about). To make this a viable technique it must be fairly easy to define and implement such application languages, as easy as doing conventional programming.