Editorial – Object-Oriented Systems

The adjective 'object-oriented' is applied to all kinds of systems, almost regardless of their actual qualities, as though it were simply a mark of respectability without which a system would be seen by the world to be unfashionable and therefore somehow lacking. However, behind the slogans there is something interesting occurring in the way people are thinking about building large, integrated information systems. This special issue on object-oriented systems presents some of the latest research.

The development of object-oriented systems as a discipline has been marked by conferences which have taken place since 1986. In the USA, the annual OOPSLA (Object-Oriented Programming Systems, Languages and Applications) conference organised by ACM SIGPLAN has been extremely well attended. The European Conference on Object-Oriented Programming (ECOOP) first took place in Paris in 1987, then in Oslo in 1988, and came to the UK with ECOOP '89 in Nottingham from 10 to 14 July this year. Each of these conferences has covered a wide range of topics from an object-oriented perspective, including languages, theory, applications, databases, design, operating systems, user interfaces and implementation.

For this special issue, the first six papers have been selected from the ECOOP '89 conference and, where necessary, updated. These, together with the last two papers which were submitted to the Journal in the normal way, represent many of the important research areas for object-oriented technologies.

First Steps Towards Fully Abstract Semantics for Object-Oriented Languages' is one of the few serious attempts in the literature to address the theory of object-oriented languages. This paper discusses observable behaviour, noting that the notion of software reuse is more appropriate for object-oriented languages than the more traditional notion of input–output behaviour. From observational equivalence the condition for a semantics to be fully abstract can be formulated. In the paper, Phillip Yelland defines an extremely simple object-oriented language, and constructs a fully abstract semantics for it.

The topic of concurrency in object-oriented languages is a fertile and active research area. Objects by their nature are an appropriate framework for models of concurrent systems. The paper by Kafura and Lee, 'Inheritance in Actor-Based Concurrent Object-Oriented Languages', investigates the interference between inheritance and concurrency reported in earlier work, and proposes a mechanism called behaviour abstraction which resolves this interference in the context of their experimental language ACT++.

'A Proposal for Making Eiffel Type-safe', by William Cook, investigates some issues in the static type system of the Eiffel object-oriented language. The paper argues that the type system contains errors, which may cause programs which are statically type-correct to produce run-time errors. Alterations to the language are proposed, which it is claimed will correct these errors. Eiffel is a commercially available language with a number of novel features which is gaining in popularity and interest, and this paper is a well-written contribution to its development.

The paper by Raj and Levy, 'A Compositional Model for Software Reuse', proposes a scheme in which objects are composed from simpler entities, in the context of the strongly typed Emerald language. Traditionally, inheritance has been seen as potentially a mechanism for software reuse; however, inheritance has been found not to co-exist easily with either distribution or concurrency. The paper proposes composition as an alternative to inheritance, and analyses some of the ways in which the alternative may be an improvement.

The goal of persistent objects, i.e. objects which reliably outlive the program that created them and form a shared information base for distributed applications, is being actively pursued in many research groups. 'The Treatment of Persistent Objects in Arjuna' describes the management of persistent objects in the Arjuna programming system from the University of Newcastle upon Tyne. Arjuna is designed for fault-tolerant distributed applications, and the paper shows in particular how the persistence property of objects may be supported using the inheritance mechanism of object-oriented languages.

The last paper from the conference in this issue, 'Browsing a la carte in Object-Oriented Databases', by Laenens, Staes and Vermeir, describes a system which uses object-oriented features to define a set of components which can be put together to form a variety of different user-interfaces. The application in this case is an object-oriented database, and the purpose of the facilities described is to simplify the development of end-user applications.

'Writing Concurrent Object-Oriented Programs using Smalltalk-80' by Hopkins and Wolczko is a second paper on concurrency, which proposes the addition of a class 'Future', whose instances represent promises for a result of a computation not yet completed. The paper explores some constructs built on this idea.

The final paper, 'Formal Hierarchical Object Models for Fast Template Matching' by Cooper, studies a method for representing image data efficiently in a hierarchical data structure called a 'repnet' which is suitable for template matching, in particular when searching for objects in pictures.

There is an expanding range of work to be done in the fascinating area of object-oriented technologies. The vision which inspires much of this work sees the technology as a means for integrating, sharing and reusing software. As computers become more and more devices for communication, rather than devices for calculation, technologies such as these for ensuring that the devices on each end of the wire can interpret each other correctly are of critical importance.

S. COOK