

Engineering Distributed Object Systems: A Case of Déjà Vu?

Innes Jelly¹, Peter Croll², Ian Gorton³, Guido Wirtz⁴

¹Sheffield Hallam University, UK. Email: i.jelly@shu.ac.uk

²University of Sheffield, UK. Email: p.croll@dcs.shef.ac.uk

³CSIRO, Australia. Email: iango@syd.dit.csiro.au

⁴University of Münster, Germany. Email: guidow@math.uni-muenster.de

Recent advances in network technologies and the performance of workstations has led to the widespread use of distributed systems in industry and commerce. Reduced costs of the components have accelerated the introduction of large scale network systems. This has been matched by increased user demands for advanced information access and handling. In manufacturing applications, integration between control and information systems is becoming possible; financial and retail institutions are turning to distributed systems to support sophisticated information storage and analysis tasks, and the advent of multi-media technologies has increased uptake of distributed systems for commercial, leisure and educational purposes. Access to internet services is now accepted as a necessary facility for the many aspects of modern living.

Distributed systems provide the foundation for a wide range of applications, and it is clear that the demand for better, faster and smarter systems will continue from a growing body of individual and corporate users. New network technologies will emerge to meet these market demands [1].

The range of platforms and propriety software involved in many distributed systems is considerable. In order to facilitate the development of these heterogeneous systems, there has been a rapid uptake of distributed object technology. This provides for easy interoperability between different applications and is thus a key feature in the construction of advanced distributed systems. Its importance can be judged by the speed with which system developers have embraced this form of middleware support.

However the rapid expansion in the use of distributed object systems, driven by market forces, has

meant that much of the software construction have been based on ad hoc design solutions. The design, implementation and re-use of software for complex distributed systems remains a difficult task [2]. As Daniels says "what the technology *doesn't* tell us is how to design our system" [3]. It is necessary to address the role of good software engineering practice in the development of these systems, and issues relating to the full software life cycle require further exploration.

It is perhaps instructive to look at other areas where new hardware technologies have led the way in system development. In the area of parallel computers the sophistication of the hardware has been achieved by vast investment. However less attention has been given to the production of effective techniques and industrial strength tools for associated software development. These are just not adequate to allow users to capture the enhanced computing power of parallel machines, and build robust, portable applications at a reasonable cost. In this field many of the hardware manufacturers have abandoned large scale multi-processor machines (or gone out of business) because the uptake of their machines has been limited by the problems associated with software development [4].

The history of computing would seem to indicate that good software engineering practice lags behind the advances in hardware but is the key issue in containing costs and providing user satisfaction. With conventional systems the advent of software quality standards, accepted software process models and use of rigorous specification, design and testing techniques has arrived late on the scene but is now becoming established.

The development of good software engineering methods for distributed systems should therefore be a

matter of crucial concern. Ad hoc methods do not, in general, produce reliable, maintainable and scaleable systems.

With distributed object technology it would seem obvious to look towards well recognised object oriented analysis and design techniques to provide the framework for the software development process. Unfortunately this is not as straightforward as it first appears. These methods have arisen from conventional “mainstream” computing, and are conspicuously lacking in support for many of the issues that are central to distributed systems, eg partitioning and allocation [5].

The challenge is therefore the development or adaptation of good software engineering practice to incorporate the specific requirements for distributed systems, and thus give more rigorous support for their analysis, specification, design and verification. This requires not only the definition of new methods but the establishment of dialogue between academic researchers and the practitioners working with distributed object systems.

It has been our aim in organising this minitrack to address these issues, and we believe that the papers presented here will make a significant contribution to the development of methods for distributed object systems. By including papers which describe the development of real applications as well as those proposing new methods, we hope to encourage a synthesis of best current practice with new research results.

References

[1] Ted Lewis, “The Next 10,000 Years: Part 2”, IEEE Computer, May 1996

[2] Douglas C Schmidt “Using Design Patterns to Develop Reusable Object-Oriented Communication Software”, Comm ACM, Vol. 38, No. 10, October 1995

[3] John Daniels, “ORBs at Your Service?”, Object Expert, Vol. 1(4), May/June 1996

[4] Cheri Pancake, Invited Talk, EPSRC Workshop on Portable Tools for Parallel Systems, University of Southampton, UK, July 1996

[5] G C Low, G Rasmussen and B. Henderson-Sellers, “Incorporation of Distributed Computing Concerns into Object Oriented Methodologies”, Journal of Object Oriented Programming, Vol. 9, No. 3, June 1996