Programming Languages for Distributed Applications

Seif HARIDI
Swedish Institute of Computer Science,
S-164 28 Kista, Sweden.

Peter VAN ROY
Département INGI, Université catholique de Louvain,
B-1348 Louvain-la-Neuve, Belgium.

Per BRAND
Swedish Institute of Computer Science,
S-164 28 Kista, Sweden.

Christian SCHULTE
German Research Center for Artificial Intelligence (DFKI),
D-66123 Saarbrücken, Germany.

Received 5 March 1998
Revised manuscript received 22 March 1998

Abstract Much progress has been made in distributed computing in the areas of distribution structure, open computing, fault tolerance, and security. Yet, writing distributed applications remains difficult because the programmer has to manage models of these areas explicitly. A major challenge is to integrate the four models into a coherent development platform. Such a platform should make it possible to cleanly separate an application's functionality from the other four concerns. Concurrent constraint programming, an evolution of concurrent logic programming, has both the expressiveness and the formal foundation needed to attempt this integration. As a first step, we have designed and built a platform that separates an application's functionality from its distribution structure. We have prototyped several collaborative tools with this platform, including a shared graphic editor whose design is presented in detail. The platform efficiently implements Distributed Oz, which extends the Oz language with constructs to express the distribution structure and with basic primitives for open computing, failure detection and handling, and resource control. Oz appears to the programmer as a concurrent object-oriented language with dataflow synchronization. Oz is based on a higher-order, state-aware, concurrent constraint computation model.
Keywords: Network Transparency, Fault Tolerance, Mobile Objects, Distributed Algorithms, Oz.

§1 Introduction

Our society is becoming densely interconnected through computer networks. Transferring information around the world has become trivial. The Internet, built on top of the TCP/IP protocol family, has doubled in number of hosts every year since 1981, giving more than 20 million in 1997. Applications taking advantage of this new global organization are mushrooming. Collaborative work, from its humble beginnings as electronic mail and network newsgroups, is moving into workflow, multimedia, and true distributed environments. Heterogeneous and physically-separated information sources are being linked together. Tasks are being delegated across the network by means of agents. Electronic commerce is possible through secure protocols.

Yet, despite this explosive development, distributed computing itself remains a major challenge. Why is this? A distributed system is a set of autonomous processes, linked together by a network. To emphasize that these processes are not necessarily on the same machine, we call them sites. Such a system is fundamentally different from a single process. The system is inherently concurrent and nondeterministic. There is no global information nor global time. Communication delays between processes are unpredictable. There is a large probability of localized faults. The system is shared, so users must be protected from other users and their computational agents.

1.1 Identifying the Issues

A distributed application should have good perceived behavior, despite the vicissitudes of the underlying system. The application should have good performance, be dependable, and be easily interfaceable with other applications. How can we achieve this?

In the current state of the art, developing a distributed application with these properties requires specialist knowledge beyond that needed to develop an application on a single machine. For example, a new client-server application can be written with Java RMI. An existing application can be connected with another through a CORBA implementation (e.g., Orbix). Yet in both cases the tools are unsatisfactory. Simply reorganizing the distribution structure requires rewriting the application. Because the Java specification does not require time-sliced threads, doing such a reorganization in Java may require profound changes to the application. Furthermore, with each new problem that is addressed, e.g., adding a degree of fault tolerance, the complexity of the application increases. To master each new problem, the developer must learn a complex new tool in addition to the environment he or she already knows. A developer experienced only in centralized systems is not prepared.