Programming Language Support for Large-Scale Distributed Systems

Position Paper

Patrick Eugster
Department of Computer Science
Purdue University

1 Introduction

Programming distributed systems is to date an overly onerous task. The engineering of such systems contradicts most programming models, design, and development methodologies, as these focus on application logic. Experts are then required to reconcile this application logic with synchronization and fault-tolerance mechanisms.

It has become common practice to deal with distribution-related issues by building on specific middleware, programmed through libraries. This approach is mainly motivated by the belief that it is the only way to achieve interoperability, portability, and independence.

We believe programming language support for distribution is badly needed. With increasing scales of systems this argument becomes even more valid, as scale impacts the design of programs. Providing such support does not necessarily mean hardwiring a set of specific abstractions into a language. By providing generic mechanisms, and leveraging syntactic macros, one can provide language constructs and mechanisms supporting a wide variety of distributed interaction models and architectural styles, as well as interfacing with other languages.

2 Programming Distributed Systems with Libraries

When looking at current programming languages, it seems striking how little support is proved for distributed programming. In practice, it seems to be commonly accepted that distribution-related issues are best dealt with through libraries, in particular, because of interoperability.

Indeed, there is a set of "classic" tools and building blocks for distributed systems, such as application and web servers, and library frameworks such as CORBA or Web Services based on standard protocols and description languages, which established their roles in the field. It is true that by adding support to a language x, that support will not be necessarily embraced by a component written in language y. It is unlikely that in the future all language research and development efforts will converge towards one, ultimate, programming language.

Interoperability is indeed a property worth striving for. However, there are alternative means of achieving it. Programming language support does not necessarily mean hardwiring a protocol; many language runtime systems nowadays provide interfaces for customization and to smoothly integrate with the outside world.

Besides, it is common knowledge also that *data* prevails, and the legacy code problem is not as stringent in practice as sometimes claimed. Modern languages provide support for interfacing with, and wrapping, legacy code. Legacy should not be an excuse for not looking ahead and solving problems that should have been tackled long ago.

3 Programming Language Support for Distributed Systems

We argue for programming language support for distributed programming.

- *We need programming language* syntax *support*, to guide programmers in devising safe code, as the debugging of distributed applications can quickly become a nightmare.

  Take the example of a replicated server: if the server is multi-threaded, non-determinism might ensue which may hamper the consistency of replication protocols. As shown by experiences such as CORBA providing replication, transactions, or persistence in a library framework, combining different such mechanisms can involve disastrous complexity. Web Services are on the way to repeat errors made in CORBA. Consider the concept of compensation introduced to deal with aborts of long-running transactions has so far only been applied sparsely precisely because compensation clauses are strongly application-dependent. In CORBA the difficulty of combining services, and the overhead incurred by such integrations, has motivated the introduction of ORB services, more light-weight services implementable within the ORB itself. In Java RMI for instance, marshaling and type inclusion mechanisms are intertwined – no real type identifiers are generated and attached to objects sent over the wire, which makes it hard to consider such a framework to be strongly typed, let alone type safe.

- *We need programming* runtime *support*. This support can provide efficiency by inferring optimizations from the executed code, and can deal with certain distribution-related issues transparently to the application thus relieving the programmer's burden.

  As illustrated by byte code transformations performed in the Java virtual machine upon class loading, only the runtime can know what code is actually executing. This is the best place to deal with redundant invocations incurred by n-to-m invocations in replication, perform optimizations in the context of transactions, transparently steer agreement with remote hosts on the fundamentals of remote interaction, such as type conformance. With intermediate architectures such as application servers, many mechanisms end up being duplicated. Concurrency control for instance appears in the application logic, and is intermingled with transactions for remote interaction. The same argument as for language syntax applies for runtime support: providing support does not mean hardwiring particular protocols or mechanisms. A runtime environment can be designed to provide interfaces through which specific modules can be plugged in, and policies can be configured. Making use of a common runtime such as .NET or even the Java virtual machine allows different languages to exploit these mechanisms.

4 Conclusions

We believe programming language support for distributed systems is badly needed – the larger the scale of the targeted systems the more exhaustive support is required.

Programming language support does not necessarily mean hardwiring primitives into a language. Such support can be achieved in a flexible and interoperable manner by providing (1) *generic* language support e.g. through syntactic macros, and through a (2) *modular and open* runtime environment e.g. with inherent support for remote objects and interfaces for transactions, replication, or garbage collection adapted to such objects. The key to devising proper support consists in identifying the fundamental *distributed* aspects of objects which permeate the program and thus are reflected in the language syntax, and which aspects can be dealt with in the runtime environment.