Object-Oriented Distributed Programming in BETA

Søren Brandt and Ole Lehrmann Madsen

Computer Science Department, Aarhus University
Ny Munkegade, DK-8000 Aarhus C, Denmark

Email: {sbrandt,olmadsen}@daimi.aau.dk

Abstract. This paper describes abstractions that have been designed to support distributed programming in the object oriented programming language BETA. The approach is minimalistic in the sense that a goal is to provide the essential building blocks on top of which other distribution related abstractions may be built. This goal is made easier by demanding for type orthogonal persistence and distribution as the full power of the underlying language may then be used when building higher level abstractions on top of the basic ones.

1 Introduction

This paper describes abstractions that have been designed to support distributed programming in the object oriented programming language BETA [19]. The abstractions are relatively simple, as they are designed to cope only with distribution specifics, whereas e.g. concurrency issues are dealt with by the basic BETA language. In general, when designing programming languages and systems, a goal is to keep them as simple as possible, consisting of a few general orthogonal language constructs. This was one of the main principles behind the design of the BETA object oriented programming language, and it is one of the design principles behind the abstractions in this paper.

In the case of distributed programs, a consequence of the orthogonality principle is that instances of any class must be remotely accessible. If this is not so, a class designed and implemented without distribution explicitly in mind might not be usable unless wrapped in a layer of code providing the ability to be accessed remotely. Especially in OO environments with large class libraries, the usability of these libraries is limited if everything must be rewritten or wrapped to be used in a distributed setting. Furthermore, if the language constructs allowed in the distributed case are limited, the expressive power of the language is limited, leading to more clumsy classes. For example, BETA owes a lot of its expressive power to nested virtual classes, a language construct to our knowledge found in no other programming language, and especially in no distributed programming language.

The principle of orthogonality also applies to object persistence [5] as it should be possible to save any object on stable storage regardless of the class to
which the object belongs. Again, a good example of this is the classes of a large library.

Another gain from the full generality of a powerful programming language is that fewer distribution specific language constructs are needed, thereby keeping the total number of language constructs at a minimum. For example, CORBA [20] includes a type system, an interface definition language and a way of handling exceptions. CORBA is referred to as "language independent". In our opinion there is no such thing as language independence. Instead CORBA defines a new language expressing a subset of a number of other languages. This is of course inevitable in an open system, but should not be confused with "language independence". As a result, a programmer using CORBA IDL to specify remote objects needs to learn and remember yet another programming language to be able to do distributed programming.

The goal thus is a general distributed implementation of BETA allowing instances of any class to be accessed remotely. Preferably, distributed BETA should, modulo network failure and server crashes, be semantically equivalent to non-distributed BETA, the only difference being that objects may be spread around on any number of network hosts and that object references may refer to objects on remote network hosts. By object reference we mean a unique object ID.

A number of issues more or less specific to distributed programming are not directly addressed by this work, as the primary goal is to be able to use the full power of the BETA language when doing distributed programming. Section 5 discusses some of the issues left out, and the reason why they are not regarded as mandatory.

The first applications built using distributed BETA are CSCW applications. The original reason for initiating research into the area of BETA distribution, was a non-distributed hypermedia application to be enhanced to support distributed cooperative work. As the design of the hypermedia is using the full power of the BETA language, it would have required a total redesign and reimplementation of large amounts of code, if distributed access to objects was restricted. As a consequence, we decided to develop a distributed version of BETA. The demands from CSCW applications put on distributed BETA includes support for wide area and heterogeneous networks. Furthermore, it should be possible to integrate the implementation with third party libraries, such as the X window system.

We have identified the following mandatory abstractions needed to be able to describe distributed programs in BETA:

- **Ensemble**: An abstraction modeling the operating system of a physical network host.
- **Shell**: An abstraction modeling executables whose instances are large grain processes within some ensemble.
- **NameServer**: An abstraction modeling mappings from textual object names to object references.

---

1 Computer Supported Cooperative Work