Petri Net Based Behavioural Specification of CORBA Systems

Rémi Bastide, Philippe Palanque, Ousmane Sy, Duc-Hoa Le, and David Navarre

LIHS-FROGIS, Université Toulouse I, Place Anatole France
F-31042 Toulouse CEDEX, France
{bastide, palanque, sy, leduchoa, navarre}@univ-tlse1.fr

Abstract. CORBA is a standard proposed by the Object Management Group (OMG) in order to promote interoperability between distributed object systems. CORBA provides a programming-language neutral Interface Definition Language (IDL) that describes the syntactic aspects of services supported by remote objects. However, CORBA IDL does not provide any means to specify the behaviour of objects in an abstract and formal way. Behaviour specification is provided either in plain English, or directly in the programming language chosen for the implementation. We propose the use of Cooperative Objects, a dialect of object-structured high-level Petri nets, as formalism for behavioural specification of CORBA systems. We detail at the syntactic and semantic level how this formalism supports the features of the CORBA object model. We present a realistic case study to demonstrate our approach.

Keywords: Distributed systems, CORBA, behavioural specification, high-level Petri nets.

1 Introduction

CORBA [16], [21] (Common Object Request Broker Architecture) is a standard proposed by the Object Management Group (OMG) in order to promote interoperability between distributed object systems. The appearance of an industrial standard is an indication that the field of object-oriented distributed computing has moved, in the past few years, from experimental research projects to mainstream commercial products.

CORBA proposes an Interface Definition Language (IDL), independent from any programming language (although closely patterned after C++) and object-oriented, supporting specialisation of interfaces through inheritance. A CORBA-IDL interface specifies at a syntactic level the services that a client object can request from a server object that implements this interface. The interface details the services supported and their signature: a list of parameters with their IDL type and parameter-passing mode, the IDL type of the return value, the exceptions that may possibly be raised during the processing of the service.
interface Example {
  exception reject;
  integer op1(in string a);
  void op2(inout float b, out integer c)
    raises(reject);
}

Fig. 1. An example of CORBA IDL

Fig. 1 illustrates the definition of an interface in CORBA IDL. This text defines one interface (Example) supporting two services (op1 and op2). The figure also illustrates the syntax for the various parameter-passing modes (in, inout or inout) and of the exceptions. The keyword exception defines an exception type, while the keyword raises specifies what types of exceptions may be raised by a service.

A recognised limitation of CORBA is that it defines remote object classes in terms of their interface only. CORBA IDL covers only the syntactic aspects of the possible use of a remote object and does not cover any semantic or behavioural description, while this information is obviously of prime importance for the clients. By behavioural aspect, we mean:

• The constraints on the order of invocation of the services described in the interface.
• The concurrency constraints of the remote object: is it able to support concurrent access to its services, or does it enforce a serialisation on the concurrent invocations?
• The conditions under which an exception might be raised during the processing of a service.

What CORBA lacks is an abstract way to specify the semantics of an IDL interface without constraining its implementation, much in the same way that an Abstract Data Type [8] (ADT) specification can be used for specifying the semantics of a sequential data type.

The present paper aims at providing a suitable solution to the problem of behavioural specification of distributed objects, in the context of CORBA. The paper is organised as follows: We first detail the requirements for a behavioural specification formalism suited to CORBA. We then present how the Cooperative Objects formalism needs to be adapted in order to support fully the CORBA model. Section 3 presents a significant case study of specification using our approach. Section 4 explains how the formalism can be used to enable rapid prototyping of distributed systems.