Modelling and Analysis of Distributed Program Execution in BETA Using Coloured Petri Nets

Jens Bæk Jørgensen and Kjeld Høyer Mortensen

University of Aarhus, Computer Science Department
Ny Munkegade, DK-8000 Aarhus C, Denmark
{jbj, km}@dadim.aau.dk

Abstract. Recently, abstractions supporting distributed program execution in the object-oriented language BETA have been designed. A BETA object on one computer may invoke a remote object, i.e., an object hosted by another computer. In this project, the formalism of Coloured Petri Nets (CP-nets or CPN) is used to describe and analyse the protocol for remote object invocation. In the first place, we build a model in order to describe, understand, and improve the protocol. Remote object invocation in BETA is modelled on the level of threads (lightweight processes) with emphasis on the competition for access to critical regions and shared resources. Secondly, the model is analysed. It is formally proved that it has a set of desirable properties, e.g., absence of dead markings.

Topics: System design and verification using nets; higher-level net models; computer tools for nets; experience with using nets, case studies; application of nets to protocols.

1 Introduction

In this project, the formalism of Coloured Petri Nets (CP-nets or CPN) [Jen92] is used to describe and analyse the protocol for remote object invocation in the object-oriented language BETA [MMPN93].

The project is divided into a construction stage and an analysis stage. In the construction stage a model is built in order to describe and understand the considered protocol. Several meetings are held between the modellers and the designer of the protocol. In the process, the designer increases his own understanding. As a consequence, a number of changes are made.

In the analysis stage, the protocol is verified. It is formally proved that it has a set of desirable properties. E.g., we prove that the protocol has no deadlocks, that certain BETA objects always have the chance to do remote object invocations (liveness), and that a monitor construction correctly ensures exclusive access to a critical region. We apply recently developed computer tools for formal analysis of CP-nets, an occurrence graph tool and a place invariant tool.

The rest of this paper is structured as follows: In Sect. 2, the system supporting distributed program execution in BETA and the protocol for remote
object invocation are introduced. Sect. 3 describes the constructed CPN model and Sect. 4 its analysis. In Sect. 5 related work is discussed. Finally, in Sect. 6, we draw some conclusions.

2 Description of the DistBETA System

The system considered in this project will be called the DistBETA system [Bra94, BM93]. The DistBETA system is a framework for distributed program execution in BETA. It includes the protocol for remote object invocation. In this section we first introduce a set of concepts from the DistBETA system that are used to describe the protocol. Then we explain the protocol itself.

The following three concepts are relevant for the remote object invocation protocol:

- **Ensemble:** Is a representation of the operating system on a computer connected to a network.
- **Shell:** Is similar to a process. Shells exist inside ensembles. A shell can communicate with another shell in a remote ensemble or in its own ensemble. Moreover, a shell can communicate directly with its ensemble.
- **Thread:** Each shell contains at least one user thread executing the main program and exactly one listener thread taking care of incoming requests from the network.

The framework that the application programmer uses to support distributed program execution contains a class called the **RPC handler**. The RPC handler includes the necessary primitives for serialization (marshalling) and communication. The framework is more general than RPC\(^1\). An object can act both as a client and as a server, allowing arbitrarily long invocation-chains wandering through many different computers. The parameters passed in an invocation are not just values as for RPC. They may be objects or references to objects, to which messages can be sent resulting in object invocations.

In the following we describe the protocol for remote object invocation (see Fig. 1). Suppose that an active object \(o_1\) wants to invoke another object \(o_2\). The two objects are physically separated on two computers, Host1 and Host2 respectively. Each object has a unique object identifier (OID)\(^2\). The sequence of events is as follows:

1. \(o_1\) looks up the OID of the object to be invoked (\(o_2\)) in a table containing OIDs of remote objects. The table is local to the shell of \(o_1\). \(o_1\) allocates

---

\(^1\) RPC is an acronym for remote procedure call. For an introduction, see [Tan92]. In this paper the term "RPC" means remote object invocation.

\(^2\) The purpose of an OID is to have a database key. Some objects may be persistent, i.e., they may survive between program executions, and are typically stored in a database on a permanent storage (as a hard-disk). The OID is then used to retrieve the object again or can even be used to get type information about the object.