Bond Objects - a white paper

Ladislau Bölöni

February 19, 1998

Abstract

This white paper defines the structure of Bond objects and introduces the *shadows*, a middleware independent abstraction for networking. In the Bond system shadows are used to implement inter-object communication, data transfer, remote execution and various distributed services. Locally managed collection of shadows are used to implement virtual networks. The communication of Bond objects is based on a message passing model, using the KQML agent communication language.

1 Introduction

Distributed systems are built from a collection of standard objects like control agents (scheduler, dispatcher, monitor), execution agents, servers and various types of data [2, 10].

The Bond system is a collection of loosely connected communicating objects of various complexity which provides a framework for distributed systems. Bond uses persistent objects, some related to active agents (representing permanent or temporary services) others being passive objects (internal or external data).

A developer can build a distributed system using Bond objects as building blocks with no concern for the low level details of the system.

The Bond system is written in Java, which provides the advantage of portability. For various services the Bond system relies on the underlying middleware which can be CORBA [6], Infospheres [3, 4], MPI [14], Java RMI [15], HTTP [13]. Bond agents use the KQML [11, 12] agent communication language, which allows Bond objects to implement complex behavior without relying heavily on the middleware. KQML also allows Bond agents to communicate with other agents outside the Bond system.

The communication model of the Bond system is based on three high level functions: message passing, object replication and directory service.

Message passing. Bond objects communicate using string messages. Messages are used to implement remote data access and remote procedure calls. Every message is a sentence in the KQML [11] agent communication language.

Message passing is done at the level of local objects using the `say()` function. Communicating between objects is implemented using the *shadow object mechanism.* A shadow object is a local placeholder for a remote object. A message to a remote object is a local function call applied to the shadow of the remote object. Remote message passing happens only between the original object and its shadow.

Object replication. An identical copy of a remote object can be obtained by calling the function `realize()` on the shadow. The replica can be accessed locally, while the master copy can be updated calling the `update()` function on the local copy.

Object migration can be realized using the `migrate()` function on a local copy. This makes the local object the master copy, and target of further updates.

Directory service. The role of the directory service in the Bond system is to create a local shadow of a remote object. The object can be uniquely specified by its name, or can be specified by a query (for example requesting "a scheduler agent").

Shadow objects represent a higher level abstraction for networking. They support (a) messaging, transmission of string messages among objects, (b) directory services and c) remote instantiation of objects. Shadow object relay on a transport mechanism which can be provided by a middleware layer or by transport protocols. The table 1 presents the way in which different transport protocols and middleware systems implement the requirements of the Bond communication model. If a certain service is not provided by the transport mechanism, it should be implemented at the level of Bond shadows.

One of the advantages of using the shadow abstraction is that the only component of the Bond object system which depends on the underlying middleware or transport protocol is the Bond shadow. Porting a Bond networking solution to a new middleware implies only reimplementing the `bondShadow` object, while all the other components can be reused even without being recompiled.

For efficiency reasons two Bond objects may choose to communicate directly using the middleware. For example two agents may decide to use a simple ftp protocol for transfering data. This requires a good coordination between the design of the two agents. The message passing interface however, represents a common denominator of all Bond objects, necessary when a Bond object should interact with unknown objects. For example a monitor agent should be prepared to monitor the execution of any Bond agent, without knowing the details of structure.

Middleware	Message passing	Remote object instantiation	Directory service	Overhead
TCP/IP	no	no	no	minimal
HTTP	no	no	no	minimal
MPI	yes	no	no	small
Java RMI	yes	yes	no	small
Infospheres	yes	no	no	large
CORBA	yes	yes	yes	large

Table 1: The capabilities and the overhead of various communication protocols and middleware systems. A "no" entry implies that the service should be implemented by the shadow object

2 The Bond object hierarchy

The Bond objects can be grouped in a hierarchy of objects. The lower in the hierarchy, the more complex is an object. The top of the Bond object hierarchy is presented in the Figure 1.

bondObject it is the root of the hierarchy. It implements the common fields of all Bond objects (name, unique identifier, address and type). It also implements the messaging function `say`, permitting any Bond object to communicate with any other Bond object using KQML.

bondData is the common ancestor for the objects which represents persistent data in the external storage.