A Bond Domain for Cooperative Environment

October 30, 1998

Contents

1 Bond Domain

A *Bond Domain* consists of a group of users with common interest, a number of services provided by Bond Servers, and cooperative Bond applications relying on the services. The servers providing core services are called *core servers* and they are:

- A *Bond System Monitor*, BSM, provides the failure detection service for the implementation of the fault tolerance. In addition the BSM starts up the other servers and recommends them a directory server to register.

- A *Directory Server*, DS, is a Bond server that cooperates with the local bond directory in each bond program to provide a distributed directory service, which can be utilized by a bond object to find any other remote bond objects.

- A *Persistent Storage Server*, PSS, is a Bond server that provides persistent storage service, which can be utilized by any bond program to save a bond object on the server and reload it some time later.

- An *Authentication Server*, AS, is a Bond server that provides authentication service, which can be utilized by a bond domain to manage a group of user accounts shared in this domain and provide account authentication service to this Bond domain.

Figure 1 shows the example of a Bond domain. A *domain administrator* uses the BSM to start and monitor core servers as well as other servers. Bond objects in this domain can use the various services including the directory service, the persistent storage service, and the authentication service. In this documentation, we introduce the core Bond Servers in the Bond 2.0 package and show how the applications use these servers.

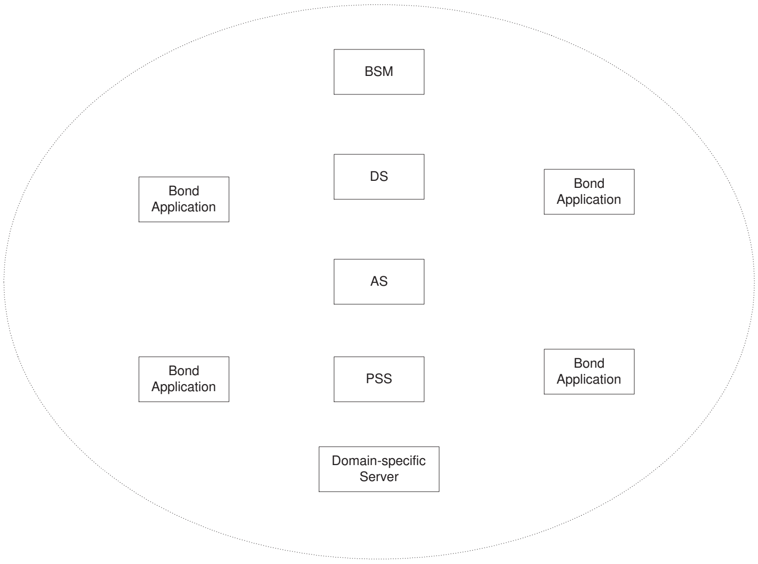

Figure 1: An example of the Bond Domain. It shows the servers and a set of Bond applications. Users in the domain run the applications exploiting the services.

2 The Bond System Monitor

The Bond System Monitor, BSM, is responsible for the fault tolerance of core servers. At this moment the BSM provides the failure detection service for the implementation of the fault tolerance. In addition the BSM starts up the core servers and recommends them a directory server to register.

We start the BSM by

```
%java bond.runnable.initiator.Monitor
```

Figure 2 shows the Graphical User Interface, GUI, of the BSM.

A resident is a container object consisting of several threads of control, a local directory, and a set of objects. The resident can create new objects and threads as requested by various sub-protocols. We start the resident by

```
%java bond.runnable.initiator.Resident
```

Once the resident starts, it finds the address of the BSM from the configuration file and register with that.

We start up servers on the residents. In the *Running Residents* window, top-left, the running residents are listed. In the *Service List* window, top-right, the service list is shown. After selecting one resident and a service we activate the service by pressing *Activate* button. After started, the new server sends back the registering message to the BSM. The *Running Executables* window, bottom, shows all the registered executables including residents and servers. The BSM recommends one directory server, if any, to the registered server.

The BSM detects the *fail-stop failure* of the executables. It expects every registered executable to send it periodic messages. The absence of such messages implies the failure of the executable. The BSM updates the Running Executables window based on such periodic messages. If the number of missing messages exceeds a limit, the BSM excludes the server from the list.

The BSM uses the Bond Monitoring Framework [?] to implement the failure detection. It subscribes to executables asking for the property, *MonitorInfo*, which every executable should

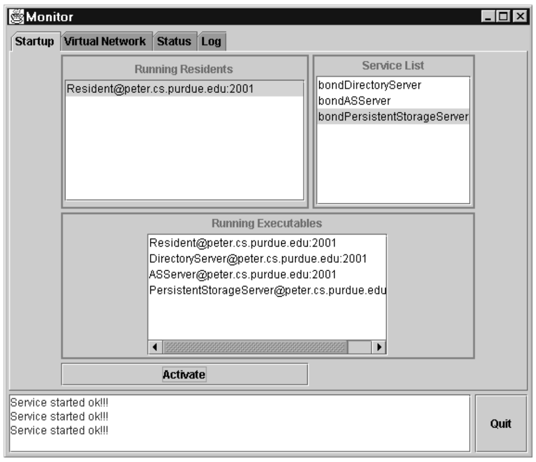

Figure 2: The Bond System Monitor GUI. The top-left window lists the running residents, the top-right window shows the service list. The bottom window lists all the running executables, failures of which the BSM detects.

have. That property has the executable name, location, other optional information, e.g. load, number of users, etc. The subscription requires the time-triggered notifications, which is necessary for the fail-stop failure.

3 Servers

Bond 2.0 implements a set of servers that each of them provides some services that can facilitate the cooperation inside a Bond domain. These servers include the Directory Server (DS), the Persistent Storage Server (PSS) and the Authentication Server (AS). We will discuss the mechanism of each server and the services provided by each server very briefly here.

3.1 DS and Directory Service

DS is a Bond server that cooperates with the local bond directory in each bond program to provide a distributed directory service, which can be utilized by a bond object to find any other remote bond objects. Figure 3 illustrates the DS and directory service.

Inside Bond system, each Bond object must register with its local directory if it wants to receive messages. If the Bond local directory wants to be global searchable, it needs to register itself with a DS. Inside each DS, there is a virtual network that contains all the registered local Bond directories.

In the above figure, if a Bond object want to send a message to object B, it must first get a shadow of object B by calling the 'find' method of its local Bond directory (1).

The local Bond directory will first checks its own repository to see if B is a local Bond object. If it cannot find object B locally, the local directory will send a find message to its registered DS (2).

After the DS gets this find request, it will first check its local cache, to see if someone has found object B before (3).

If DS cannot find object B in its cache, it will broadcast the find request to all the elements in the virtual network that contains all the registered local Bond directories except the one which sent this find request (4).

When each local Bond directory receives this find request, it will check its own repository

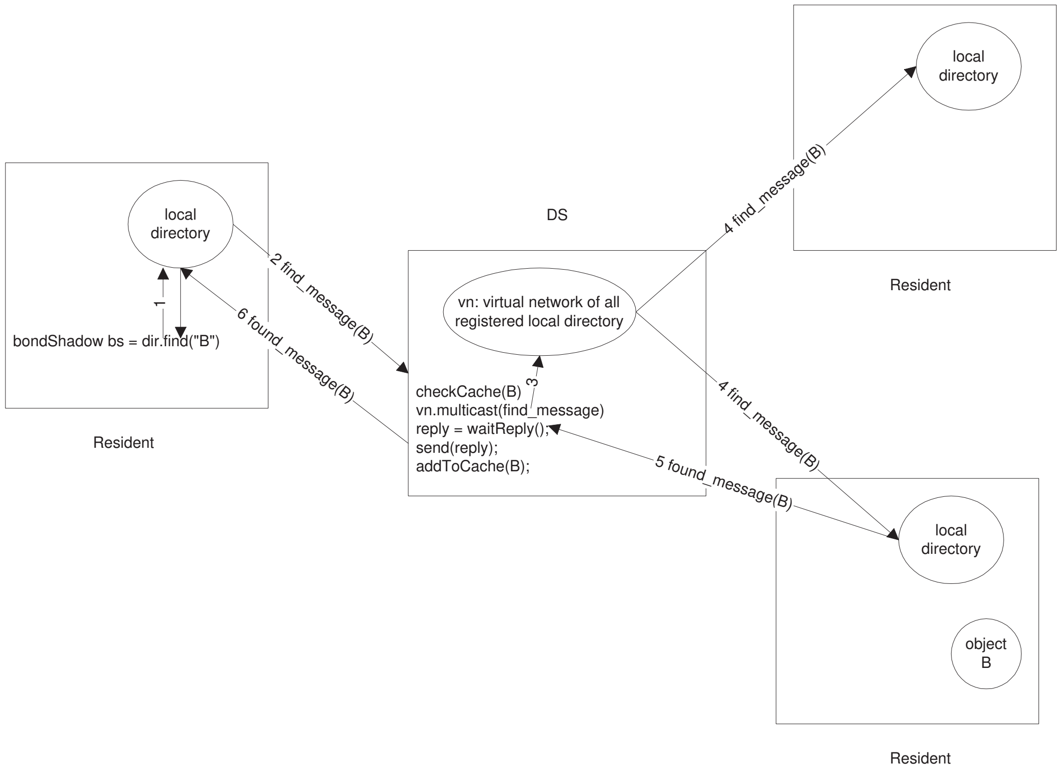

Figure 3: Directory Server and directory service

to see if object B is a local object. If yes, it will send back a found message to the DS (5); otherwise, there is no reply.

When the DS receive the first found message, it will forward it to the local Bond directory where the original find request came from (6). At the same time, DS can also cache the location of object B for serving the future finding request.

At last, the local Bond directory receives this found message, construct a bondShadow for object B, and returns the bondShadow (7).

As we can see from the above discussion, the major function of DS is to cooperate with multiple local bond directories to provide a distributed directory service. DS will also periodically check with all the elements in its virtual network to make sure they are still alive. When the DS is started up, it will display a GUI through which you can see all the current registered local Bond directories, as shown in Figure 4.

3.2 PSS and Persistent Storage Service

PSS is a Bond server that provides persistent storage service, which can be utilized to save a bond object on the server and reload it some time later.

PSS provides two types of services:

- Save or reload a Bond object
- Save or reload a user's Bond workspace

Usually when you want to save or reload a Bond object, you need to send the PSS the bondID of this object. User's Bond workspace is a special bond object, it contains all the bond

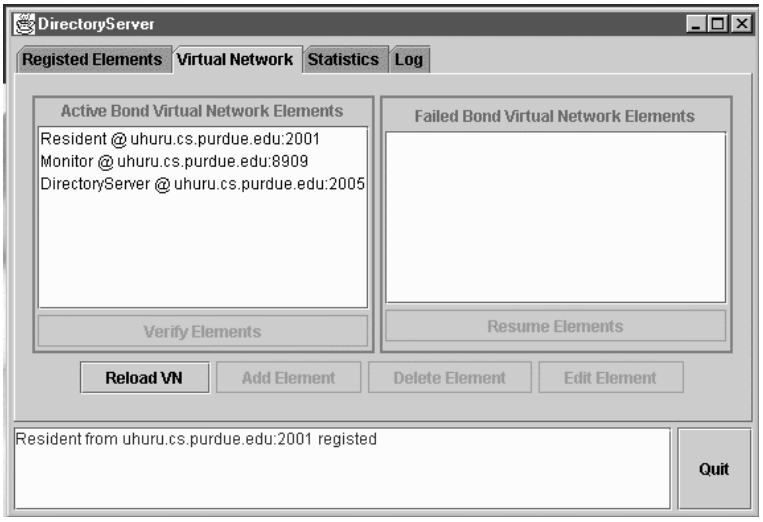

Figure 4: GUI of Directory Server that shows all the registered elements

object the user is working with. When you want to save or reload a user's Bond workspace, you just need to provide the PSS with the user's name.

Bond PSS provides several APIs for accessing the persistent storage service. They are listed below.

- `boolean saveObject(bondShadow pssbs, bondObject bo, bondObject issuer);`
- `bondObject reloadObject(bondShadow pssbs, String objectid, bondObject issuer);`
- `boolean saveWorkspace(bondShadow pssbs, bondWorkSpace bws, String user, bondObject issuer);`
- `bondWorkSpace reloadWorkSpace(bondShadow pssbs, String user, bondObject issuer);`

In these APIs, the parameter pssbs is the bondShadow of a PSS; the parameter issuer refers to the object that issued this API call.

When the PSS is started up, it will display a GUI as shown in figure 5. Through the 'workspaces' tab, the administrator can see all the workspaces stored in this PSS. Through the 'objects' tab, the administrator can see all the bond objects stored in this PSS. Through the 'log' tab, the administrator can also monitor all the save and reload requests received from outside.

3.3 AS and Authentication Service

To prevent themselves from malicious attacks, Bond objects can become secure Bond objects by integrating with the bond security framework. Some types of authenticator inside the Bond security framework, such as password-based authenticator, need the help of an Authentication Server to verify user's password and finish the authentication.

AS is such a Bond server that provides authentication service, which can be utilized by a bond domain to manage a group of user accounts shared in this domain and provide account authentication service to those secure object in this Bond domain. Currently the AS only support password based authentication.

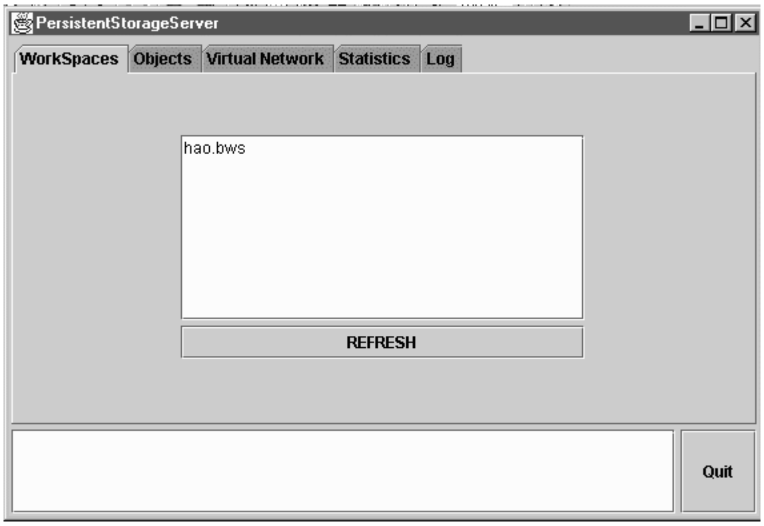

Figure 5: GUI of Persistent Storage Server that shows all the stored workspaces

Bond AS provides several APIs for accessing the authentication service. They are listed below.

```
boolean accountCheck(bondShadow asbs, String user, String password, bondObject bo);
boolean responseCheck(bondShadow asbs, String userid, String challenge, String
response, bondObject bo);
```

In these two API methods, the parameter asbs refers to the bondShadow of an AS, the parameter bo refers to the object who issued this authentication request.

The first method is used if the issuer knows both the username and the password. The second method is used when the issuer does not know the user's password, but it has a response string which is derived from both the challenge string it sent to the user and the user's password. So by using the second API method, a bond object can be safely authenticated without letting the server object knowing its user password.

When the AS is started up, it will display a GUI through which you can manage all the user account and their passwords, as shown in Figure 6.

Through this GUI, the administrator can add user accounts to this AS, delete user accounts from this GUI, or save all the accounts in a back file. Through the 'log' tab, the administrator can also monitor all the authentication requests received from outside so that he can find any possible attacks on a special user account.

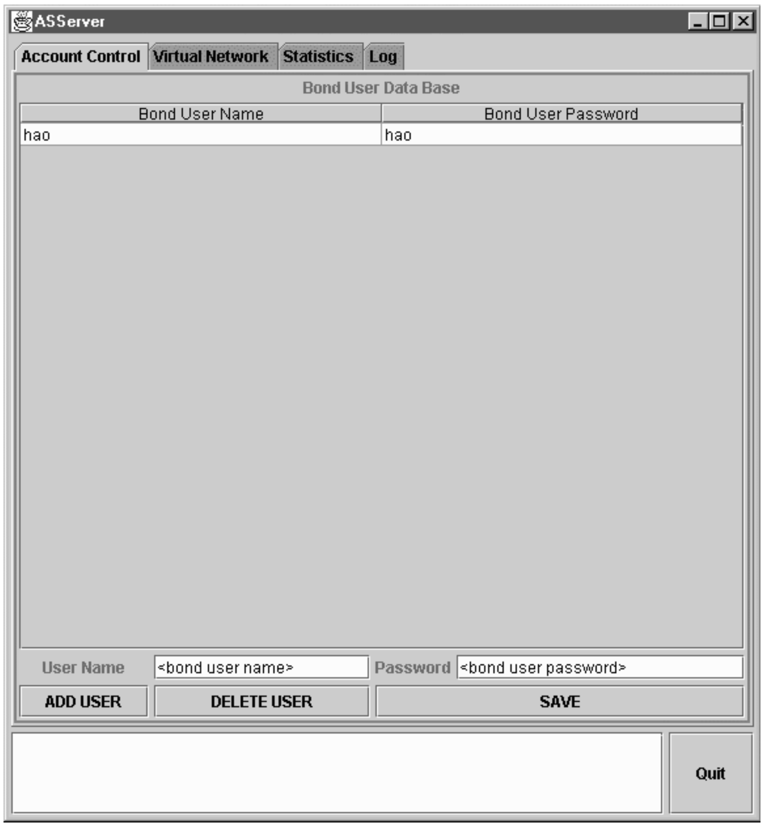

Figure 6: GUI of Authentication Server that shows all the user accounts