PIMR: Programming Guide
Version 1.0.0
www.gradsoft.kiev.ua

October 24, 2002

Contents

1 Introduction

`PIMR` ("Repository") is the way to raise availability and to make easier configuration of interfacing CORBA-services. Also you can use repository instead of namesevice To get initial object references to certain services. Repository registers services in its database and when gets a request redirects it to the proper service. Also repository controls availability of registered services and if necessary tries to restart failed service.f

In this realization (version 1.0.0) repository checks availability of services periodically in certain time interval. You can set checking frequency in configuration file by `checkfreq {number}` command. If repository detects service unavailability it tries to revive the service. If service doesn't send the signal about successful restart after certain timeout, it is considered as failed and is not being restarted any more.

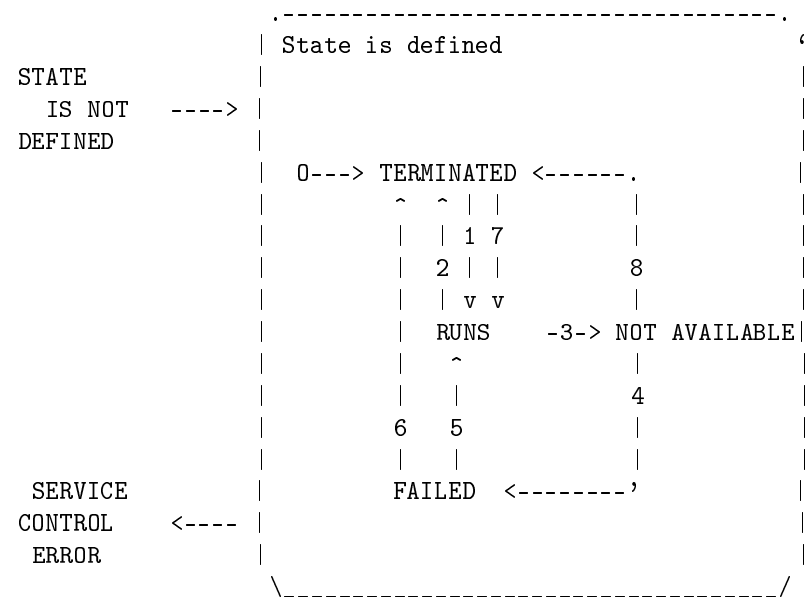
Repository provides application programmer with API described by idl-interfaces

1.1 Service Properties

From static point of view service is characterized by set of interfacing properties and operations. It's possible that some properties don't need to be defined , so they are presented by the set : {flag, property[, property...]}, where flag defines whether a property is defined or not. If flag is set then properties group is defined and you can use these properties. If according flag is not set, then values of the properties group don't have particular sense (though you can read them).

Service Properties:

- IOR reference to service implementation. It's impossible to work with object not having reference to it. IOR(Interoperability Object References) mechanism is the kind of object references. IOR is defined in IIOP, GIOP adaptation, that uses TCP/IP protocol stack. For instance `corbaloc::127.0.0.1:22000/NameService` (Is used to redirect requests and to control services working capacity in addition to callback-interface).
- Object IDL-interface ID. For example `IDL:omg.org/CosNaming/NamingContext:1.0` . (Is used to bind CORBA-object to 'servant-bait')
- IOR reference to CheckCallback interface, implemented by object-service. (no necessary)
- `restart_flag`. Flag that controls whether such properties as restart command and host/port are available or not. If flag is set it means that service should be restarted if its unavailability is detected. The other term fro flag is "always available".
- Host (and port) where server runs. For instance `127.0.0.1:22000` or `localhost:22000`.



Picture 1: Main States Diagram

- Service restart command. For instance `myservice -parameter1 parameter2`.

1.2 Dynamic Part

You can see main service states diagram on 1 picture. It shows main states and possible transitions from one to another:

1 Service startup (going over transient state)

2 Service termination (going over transient state)

3 Running service became unavailable

4 Unavailable service should be available at this moment

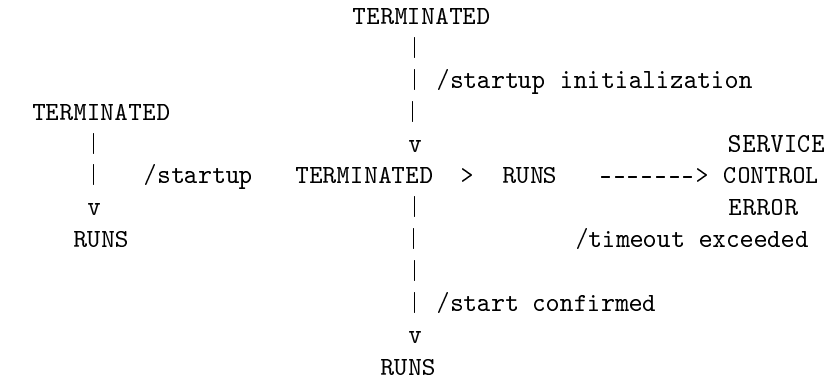
5 Service restart (going over transient state)

6 Failed service termination (going over transient state)

7 Service has been started

8 Service has been terminated

At that transition 8 is forbidden for services with "always available" flag. If "always available" flag isn't set , when repository detects that service is unavailable the service is considered as terminated.



Picture 2: Going Over Transient State

The diagram contains two different transitions from TERMINATED state to RUNS state, and it's not an error, for transitions 1,2,5,6 are executed through additional states (2).

2 Repository API

Repository API is presented in pimr.idl file:

```
#pragma prefix "gradsoft.kiev.ua"

module PIMR
{

 interface CheckCallback;

 interface Repository
 {

   exception UnknownServiceName {};

   /**
    * must be call by client, which know about PIMR after
    * startup.
    **/
   void activated(in string service, in Object instance)
           raises(UnknownServiceName);

   /**
    * we can call checker for service
    **/
```

```
   void setCheck(in string service, in CheckCallback callback)
           raises(UnknownServiceName);

 };

 enum ServiceStatus { Unknown, NotRunning, Correct };

 //
 interface CheckCallback
 {
   ServiceStatus     checkStatus(in string service);
 };


};

#endif
```

Repository interface (as well as RepositoryAdmin, defined in `PIMRAdmin.idl` file) is implemented by repository CORBA-object, which is available by reference:

`corbaloc::repository_host:repository_port/Repository`

And CheckCallback is to be implemented in the service that supports repository. This interface should implement particular CORBA-ping , that is service availability checking. At that, `activated()` or `setCheck()` operations can be called by object-service at startup. These operations inform repository about the object-service activization . The difference between them is that the first operation informs repository about (probably changed) location of object-service and the second one informs about location of the object that implements CheckCallback interface and belongs to the calling service.

Samples that use these interfaces are in the repository package in verb"demoşubdirectory.

Though it's not necessary to call those operation, for the full repository support after service startup first you should call `activated(myIOR)` and then `setCheck(callBack)`. It guaranties that repository will know about object-service location and will be able to check its availability.

When using services which implement partial repository support its possible that repository has correct reference to service but incorrect reference to callback-interface (or the other way round). In that case service will be considered as unavailable.

3 Repository And Object-Service Interaction

Repository and object-service interaction is possible as follows:

- No interaction , except that somebody should register necessary information at the repository. For instance by the configuration file. This case

you can use for the services that don't know about repository, and it is an advantage of the case. And its disadvantage is that the reaction time is longer and it's not very safe.

- After startup service calls `activated(*)` but doesn't implement `CheckCallback` interface.
- Service implements `CheckCallback` interface (and possibly calls `setCallback(*)` after startup, otherwise someone should report service Callback interface reference to repository) but doesn't use `activated(*)`. In this case someone should report object-service reference to repository.
- Service thoroughly supports repository, in sense that implements `CheckCallback` interface (and possibly calls `setCallback(*)` after startup, otherwise someone should report service reference to repository) and calls `activated()` after startup.

In addition you can write service restart information to configuration file. Then this service will have set `restart_flag` (or "always available" flag) and repository will restart it if detects that service is unavailable.

Be careful when combining several interaction cases of the same service, for the situation is possible, when repository has correct object-service reference and incorrect callback-interface reference. Such service will be unavailable.

4 Interaction Of Repository And Object-Client, That Uses Service

Preliminary conditions: repository is running. It's supposed that service is registered at the repository, repository received object reference to service. In such case service will available by the reference:

`corbaloc::repository_host:repository_port/required_service_name`

Client sends request using this corbaloc. Client orb connects to repository and transmits the request. Client orb receives:

- Redirect request instruction. This means that service is available. Instructions contains object address and CORBA implemented mechanism of redirection is not visible to object-client.
- OBJECT_NOT_EXIST exception, which means that repository doesn't have service record or that service is terminated at the moment. It also means that no errors occurred while processing request.
- IMP_LIMIT exception, if repository timeout was exceeded (possible when obtaining information or when operating with service) PS. Probably TRANSIENT is better for this situation, This exception may change in the next version of repository.

- OBJ_ADAPTER exception if service was wrongly configured and therefore is unavailable.
- CORBA system exception with corresponding semantics .

If client orb gets instruction to redirect request, then this and following requests is sent by reference, obtained from repository , implicitly for object-client.

Implementation details: Client orb connects to repository and transmits request. Repository uses dynamic servant - bait to associate it with object available by the references described above. And so repository orb redirects client request to servant - bait, but request is caught by interceptor. Interceptor extracts `ЙНС_УЕТЧЙУБ`, contained in the request, then obtains service information from internal register. Depending on this information interceptor may execute certain operations with service, according to repository functioning algorithm, before `redirecttimeout` will be exceeded. Then interceptor returns the result to client.

For better understanding study samples from `demo\` directory of repository package.

5 Changes

24.09.2002 Initial version is created.