Proceeding of the 18th International Symposium on Reliable Distributed Systems - SRDS' 99, IEE Computer Society. Lausanne, Suiça - Outubro de 1999.

CosNamingFT - A Fault-Tolerant CORBA Naming Service

Lau Cheuk Lung, Joni da Silva Fraga e Jean-Marie Farines
Laboratório de Controle e Microinformática
Univ. Federal de Santa Catarina; Florianópolis - Brazil
lau, fraga, farines@lcmi.ufsc.br

Michael Ogg, Aleta Ricciardi
Information Science Research Center
Bell Laboratories; Murray Hill, NJ - USA
ogg, aleta@research.bell-labs.com

Abstract

This paper describes the design and implementation of a fault-tolerant CORBA naming service - CosNamingFT. Every CORBA object is accessed through its Interoperable Object Reference (IOR), which is registered with the CORBA name service. The name service therefore is a critical gateway to all objects in a distributed system; to avoid having a single point of failure, the name service should be made fault-tolerant. CosNamingFT uses the GroupPac package [6], a CORBA-compliant suite of protocols, to replicate the name server. GroupPac services are built from Common Object Services that function as building blocks to implement fault-tolerant applications. This paper aims to demonstrate the usefulness of some of these object services and to demonstrate the importance of open solutions issues in replicating distributed objects.

Keywords: naming service, fault-tolerant, CORBA, open systems.

1. Introduction

A *name service* plays an important role in any distributed system. Roughly speaking, its primary role is to facilitate access to the system's resources; it is a commonly known "meeting room" for objects, services, and resources. As defined by the CORBA (*Common Object Request Broker Architecture*) specifications, the name service is a critical gateway for all objects in a CORBA-based distributed system. All CORBA objects are registered with the name service, allowing them to be found and used by any other object in the system. Thus, each object that participates in a CORBA-based computation accesses the name service at least one: either to register itself via its IOR (*Interoperable Object Reference*), or to locate other CORBA objects' IORs. One of the OMG's (*Object Management Group*) Common Object Services (COSes) is a name service standard, called CosNaming [9, 10]. The CosNaming specification takes into account the portability, interoperability and reusability aspects that are important concepts in open systems. However, these specifications do not include requirements of fault tolerance that are very important for distributed systems.

One possible way to implement a fault-tolerant name service would to use the COS persistence service [10], a unique interface for persistent storage services for objects' states. We decided not to follow a persistence-based strategy due to concerns over recovery time for applications requiring more stringent availability; to recover from a crash, a new name service must be started and its state loaded from file.

Here, we describe *CosNamingF*T, a name service which adheres to the OMG's CosNaming and uses *primary-backup* replication [2] to achieve fault tolerance. Primary-backup replication can provide continuous service availability (the degree of fault tolerance depends on the model of computation assumed), but cannot guarantee once-and-only-once semantics. Primary-backup replication has a shorter fail-over time than strategies based on persistence, and lower overhead than active replication strategies. We used the GroupPac package [6], a set of common object services for implementing various group-based communication models, for the primary-backup infrastructure. In addition, these service objects can be combined in different ways allowing the implementation of different fault tolerance schemes. This paper demonstrates the usefulness of these services as well as the possibility of obtaining open solutions for replicating distributed objects.

At present, there is no sanctioned COS specification for fault-tolerant CORBA objects through replication and group-based processing, though this is expected within the year [11]. Electra [7] was an *integration-based* approach to build replicated CORBA objects, building the ORB on top of a group-communication subsystem, although any communication substrate could have been used (for example, an adaptor for TCP was as sensible as the original adaptor for Isis [1] and a later for Ensemble (originally called Horus) [14]). While Electra used Isis's active replication model, this was an early implementation choice; Isis's coordinator-cohort (passive replication) model could equally well have been used.

Eternal [8] is an *interception-based* approach, also making use of an existing group-communication subsystem, but in this case intercepting IIOP messages and mapping them to calls of the group-communication system. The advantage of this approach is that no modifications to the ORB are required; neither the ORB nor the objects need ever be aware of being intercepted. The shortcomings are that the approach is only possible if the host operating system permits interception. In both integration and interception, until there are OMG specifications for multicast and group communication semantics, both strategies must be considered "proprietary".

The approach taken by the Object Group Service (OGS) [4] is most similar to ours, in that both adhere to the common object service model of the OMG. The primary difference is in how the group service itself is activated. In OGS the object's interface explicitly inherits from the OGS Groupable interface:

```
interface FOO: mGroupAdmin:Groupable {
  void m0();
};
```

This makes fault tolerance visible to application objects since they must be aware of the existence of the OGS objects to use their services. Application objects convey their invocations and responses, via the Dynamic Invocation Interface and the Dynamic Skeleton Interface, to their associated OGS objects, which then coordinate with each other to perform the operation on the replicas of the object and to return the results appropriately.

The GroupPac package [6] is a set of common object services for implementing various group-based communication models, for different fault tolerance strategies. In contrast to OGS, GroupPac does not require any explicit change to an application's interface *(e.g.,* it is not necessary to inherits any group service interface). Instead, to use GroupPac, the application create a instance of a portable interceptor (from CORBA spec), which implements the mechanisms to invoke GroupPac services. Fault tolerance is not visible to the application.

In Section 2 we present OMG's CosNaming specification. Section 3 describes GroupPac and Section 4 describes the implementation of CosNamingFT with GroupPac. In Section 5 we resent some performance tests for CosNamingFT and comparisons with related work.

2. Name Service in the CORBA Specifications

OMG specifications are intended as a set of standards and concepts for distributed objects in open distributed environments. The heart of the CORBA standards is the Object Request Broker (ORB), which allows a remote object's methods can be invoked transparently in heterogeneous distributed environments. Thus, an ORB is a communication channel for distributed objects. Interoperability between objects is achieved by specifying their interfaces with CORBA's Interface Definition Language (IDL). Translating IDL specifications a host of programming languages (including C, C++, Ada, COBOL, Java) generates the necessary, language-specific interfaces and auxiliary support for object implementations.

To help in the development of distributed applications, OMG specifications included a set of object services that simplify the application designer's task. Such COSes define, for example, how exceptional events should be transmitted, how objects are located and named, how security is provided, or the semantics of transactions. In this way, the COSSes define basic functionality that frequently appears in many distributed applications.

2.1. The Common Object Name Service

The COS name service, commonly called CosNaming, defines how to resolve object names in a CORBA environment. An object is identified by its *name* and its *object reference (i.e.,* its IOR). The object's name corresponds to a set of characters ideally, but not necessarily, expressing a quality or a certain feature associated with it. The IOR is the data necessary to reach the object in a distributed system; it is a character sequence, that once properly converted provides the IP address information of the host, the port, and a local pointer for the object to be accessed and more control information related to the IOR itself. In the CosNaming specification, an association between name and IOR is called a *name bindin*g, which is defined within a *naming contex*t. Each naming context maintains its own set of name bindings; while names must be unique, a naming context permits multiple names to be associated with the same IOR. This multiplicity allows one to swap object instances on the fly, yet retain the same address of a desired service (in the same way, for example, that a

restaurant retains the same physical address irrespective of which of its wait-staff or cooks are on duty). We use *NameContext* throughout this paper to refer to the object that implements the functionality of the name service defined in CosNaming specifications.

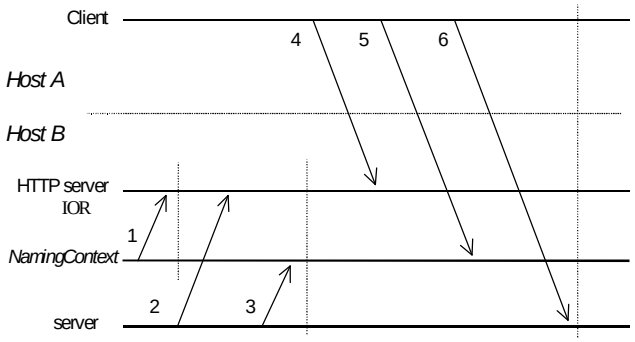


Figure 1. NameContext procedures for the support of client/server interactions.

Figure 1 presents a temporal diagram delineating the phases of interactions between the NameContext, and server and client objects. These phases cover the NameContext start-up, the installation of a server object's IOR at the NameContext (binding procedure) and finally, the phase in which client object obtains the server object's IOR.

A NameContext object in its starting state (step 1 in Figure 1) places its own IOR in a repository (a file in the file system), allowing it to be accessible to all CORBA objects. In our implementation, this repository is a directory that is accessible through an HTTP server (for example: `http://www.lcmi.ufsc.br/~lau/ior`), thereby allowing method invocations through the Internet. Thereafter, other CORBA objects can install their IORs.

```
public class FooServer {
  public static void main (String args[]) {
    try {
      // obtaining name server IOR
      NameContext ncRef = (NameContext)
ORB.resolve_initial_references("NameService");
     :
```

Figure 2. Getting the IOR of the NameContext.

For a server object to install its IOR with the NameContext, it finds the IOR of the NameContext through the `resolve_initial_reference()` method, which accesses the repository (Figure 1, step 2). The Java code implementing this step, is presented in Figure 2.

After getting the IOR of the NameContext, the application server can invoke this service for installing its own IOR and become available via CORBA to all objects in the system (Figure 1, step 3). The code in Figure 3 is how a server named `Foo`, of type `App` installs itself and binds its name/IOR with the NameContext.

```
     :
  // ref - Foo object reference
  Foo ref = new FooServant();
  NameComponent path[] =
    { new NameComponent("Foo", "App") };
  // bind the NameContext reference
  ncRef.bind(path, ref);
}
  catch (Exception e) {
    e.printStackTrace();
}
```

Figure 3. Installing an IOR in the NameContext.

The client's actions in invoking a method of the server are shown in Figure 1, steps 4-6. As with the server, the client must first obtain the NameContext's IOR with the `resolve_initial_references()` method. In a CORBA system, a client must know, *a priori*, the name and type of the server it wishes to use. Thus, in the next step, the client invokes the NameContext to resolve this name. If the name exists in the NameContext, the client is returned the server's IOR, and may thereafter invoke the server's methods. Figure 4 contain the code fragment for these steps.

```
public class Client {
  public static void main(String args[]) {
    try{
      NameContext ncRef = (NameContext)
ORB.resolve_initial_references("NameService");
      // get Foo object reference from name.
      NameComponent path[] =
          {new NameComponent("Foo", "App")};
      Foo ref =
        FooHelper.narrow(ncRef.resolve(path));
      // invoke method m0() of Foo
      ref.m0();
      System.exit(0);
    }
    catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

Figure 4. Client interactions for allowing invocations in the server object.

3. GroupPac Services and CosNamingFT Architecture

3.1. GroupPac Services

The GroupPac package [6] adheres to the OMG orientation in that it is a collection of common object services that provide support building and managing object replication. GroupPac is based on models and concepts specified by the OMG for common object services. In this philosophy, GroupPac provides a set of building blocks – object services – that can be arranged in different ways to arrive at different fault-tolerant schemes. All GroupPac object services are orthogonal and their interfaces are specified using the CORBA IDL.

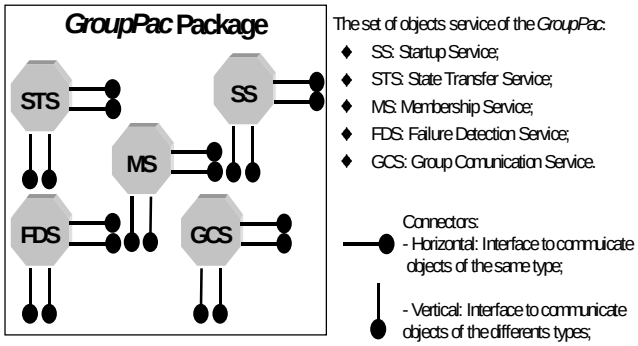


Figure 5. Object services in GroupPac.

Figure 5 depicts the GroupPac Startup Service (SS), State Transfer Service (STS), Membership Service (MS), Failure Detection Service (FDS) and Group Communication Service (GCS). Each one of these object services has got a horizontal and a vertical interface: The former allows communications between object services of the same type; the latter is the way in which the service is made available to other objects in the system.

The SS object is responsible for creating the other package service objects. All other GroupPac service objects are, therefore, dependent on the SS. It is from this object that the necessary configuration for the support of a replicated application is built. The STS object provides functionality for transferring state from one object to another. Some means for transferring state is common to most middleware facilitating replication [1, 15]; it is also common in supporting object migration. The MS object is responsible for managing changes in composition to groups of objects. Ideally, membership changes should be transparent to an application, but there will always be a trade-off between blocking an application and the completeness of the membership change propagation. The FDS object is concerned with detecting the (apparent) failures of objects in a group; obviously failure per se is not detected as much as the unresponsiveness of the object is measured. As expected, the MS object uses the FDS's services: FDS suspicions are reported to the MS object, which then creates a membership list that no longer contains the suspected object. Finally, the GCS object supports various group communication primitives, such as FIFO, causal, or linear order multicasts within a group using point-to-point communications at the ORB level; it depends heavily on the MS object.

3.2. CosNamingFT

Our implementation of CosNamingFT follows the CORBA specifications for name services [10], while using GroupPac to add primary-backup fault tolerance. The primary CosNamingFT replica performs all client requests to the CosNamingFT service. These requests are not blocking in the sense that the primary updates the backups after having responded to the client. Of course, this admits the possibility of state inconsistency between the primary and its backups. In the worst case, a new service may have been registered with the primary CosNaming copy of which the backups are unaware; a request by a client for a registered service's IOR is, in essence, a stateless operation, meaning there will be no significant state difference between the just-failed primary and its backups. The case in which the primary CosNamingFT copy fails before replying to its client is indistinguishable to blocking or non-blocking implementations of primary-backup replication.

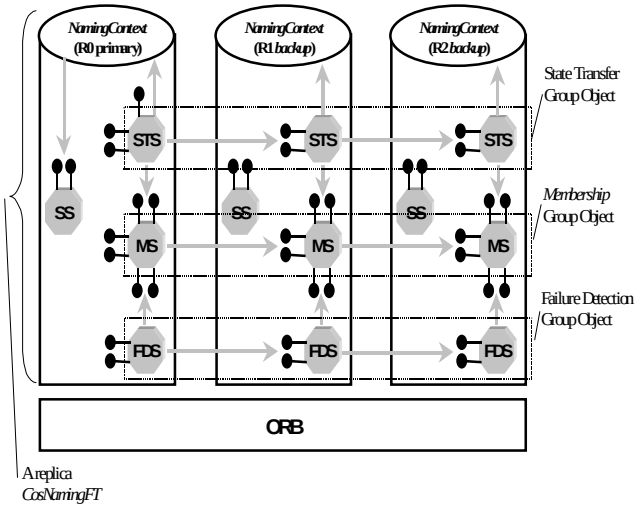


Figure 6. GroupPac service blocks used in CosNamingFT.

After processing a request and performing the corresponding update to replicas, all the correct replicas that compose the CosNamingFT must keep the same context.

GroupPac services are incorporated within CosNamingFT transparently (Figure 6). Each CosNamingFT replica contains a copy of the SS, STS, MS, FDS and NameContext objects, all of which are located in the same Unix process. Each CosNamingFT replica is linked to the others by the grouping of its component GroupPac services. For example, in Figure 6, the MS object of each CosNamingFT replica form a group that supplies a membership service in the CosNamingFT.

3.2.1. Management Service

Between them, the FDS and MS groups supply information about the status (correct/faulty) of each CosNamingFT replica, and control the composition of the CosNamingFT replica members for the CosNamingFT service.

The FDS 'detects' replica crashes in the following manner. The FDS assumes non-blocking, reliable point-to-point communication between pairs in the FDS group; thus, the absence of an expected message cannot be due to its loss by the transmission medium, but may instead be indicative of something wrong with the sender. In Figure 7, each object of the FDS group periodically "pings" its partner (as determined by a virtual ring on the linear ordering the members of the group). If it does not receive a response within some tunable timeout period, it suspects its partner of having crashed and invokes the `leave_group()` method of the MS object in the primary replica.

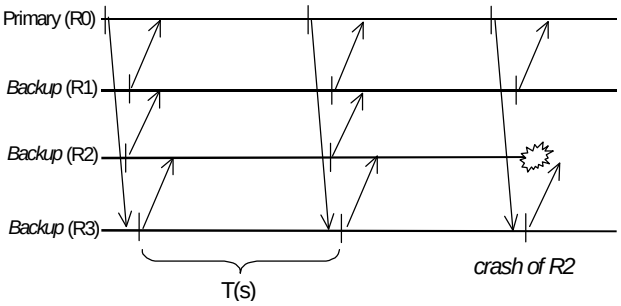


Figure 7. The failure detection service.

This spurs the membership service into action, which uses a centralized, two-phase (three-phase in the worst case) commit protocol reach agreement with the other MS replicas on a new group composition [12]. The MS object of the primary CosNamingFT replica is the coordinator for the group of MS objects. Figure 8 shows a very simple instance of this protocol. The FDS object of replica *R3* suspects replica *R2* and invokes `leave_group(R2)` of the primary replica. This activates the membership protocol, whereby the primary MS object attempts to discover which others remain in the group via the MS-to-MS method `commit()`. The response to a `commit()` method is a simple acknowledgement. After waiting for a predefined-defined timeout, the primary updates the list of active members based on the acknowledgements it received [14, 12] (which must constitute a majority of the previous view).

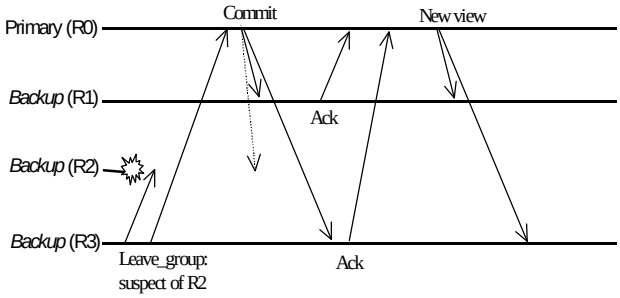


Figure 8. FDS and MS operation when replica *R2* crashes.

When the primary replica is suspected of having 'failed', a new primary must be selected; the candidate preference list follows the order implemented by the virtual ring. In our scheme, the new primary, once ascertained, must install its IOR at the HTTP server. Figure 9 gives a simple example showing how a primary is replaced. Complete details on the reconstruction of the primary follow the procedure described in [12]. Once a new primary has been agreed upon, it installs its IOR at the HTTP server. Since the HTTP repository is unique, only one replica can have its IOR installed there, ensuring at most one replica is ever acting as the primary (recall that all activity involving the primary first checks the HTTP repository).

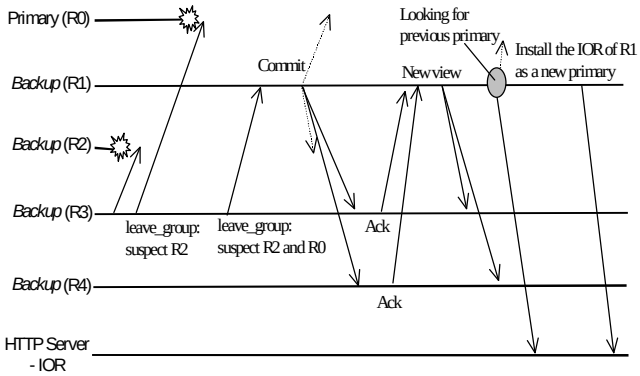
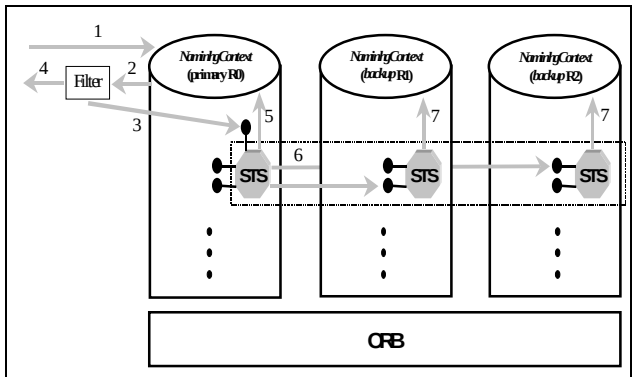


Figure 9. Determining a new primary copy.

3.2.2. State Transfer Service

We use the GroupPac State Transfer Service (STS) to guarantee consistency among CosNamingFT replicas. Figure 10 illustrates the following steps in check-pointing states from primary to backups.



1. Upon a client request to the CosNamingFT service,
2. the reply from the primary replica is intercepted and its IOR is extracted
3. and sent to the STS associated with the primary replica.
4. The intercepted reply takes its natural path, reaching the client.
5. Using the IOR of the primary NameContext, the STS invokes `get_state()` method,
6. and then sends the state to its STS cohorts,
7. which invoke the `set_state()` method of their associated NameContext object.

Figure 10. State Transfer Service.

4. CosNamingFT Implementation

Our implementation adheres to COSS/OMG specifications for the name service (CosNaming); no extension or modification has been made to the standard interface (Figure 11). The use of GroupPac services to implement CosNamingFT did not require any change in the COSS/OMG specifications. All services of this package are written in Java and the interfaces specified according to the IDL/OMG standard. We used Iona's OrbixWeb [5], an ORB also written in Java giving us the benefit of OMG standards and Java portability.

```
module CosNamingFT {
  typedef string Istring;
  struct NameComponent {
    Istring id;
    Istring kind;
  };
  typedef sequence <NameComponent> Name;
  enum BindingType { nobject, ncontext };
    :
  interface NameContext {
    :
    void bind(in Name n, in Object obj)
      raises(NotFound, CannotProceed,
             InvalidName, AlreadyBound);
    void rebind(in Name n, in Object obj)
      raises(NotFound, CannotProceed,
             InvalidName);
    void bind_context(in Name n,
                      in NameContext nc)
      raises(NotFound, CannotProceed,
             InvalidName, AlreadyBound);
    void rebind_context(in Name n,
                        in NameContext nc)
      raises(NotFound, CannotProceed,
             InvalidName);
    Object resolve(in Name n)
      raises(NotFound, CannotProceed,
             InvalidName);
    void unbind(in Name n)
      raises(NotFound, CannotProceed,
             InvalidName);
    NameContext new_context();
    NameContext bind_new_context(in Name N)
      raises(NotFound, CannotProceed,
             InvalidName, AlreadyBound);
    void destroy()
      raises(NotEmpty);
    :
};
```

Figure 11. CosNaming IDL.

Our implementation involves two software components: `NameContextServant.java`, which implements the CosNaming functionality described in the IDL specification, and `NameContextServer.java`, which is responsible for creating and starting the replicas of CosNamingFT including installing the primary's IOR in the HTTP repository and starting GroupPac's SS object. The NameContext interface is composed of

operations to connect (`bind`), to reconnect (`rebind`) or to disconnect (`unbind`) a name to an object, operations for reaching an object by using its name (`resolve`), and operations to create, destroy or list a context of names. In addition, up-call operations such as `get_state()` and `set_state()` are necessary for data transfers between the GroupPac services and the name service.

The IDL specifications for GroupPac services are presented in Figure 12. Each one of these interfaces is represented by a service object at runtime.

```
module GroupPac {
  typedef sequence<any> State;
  typedef sequence<string> MembersCrash_seq;
  // Membership service IDL interface
  interface MembershipService {
    struct ServiceRefs {
      Object fd;
      Object st;
      MembershipService ms;
    };
    struct AllServRefs {
      sequence<string> id_seq;
    // FDS Reference
      sequence<Object> fd_seq;
    // STS Reference
      sequence<Object> st_seq;
    // MS Reference
      sequence<MembershipService> ms_seq;
    };
    boolean join_group(in ServiceRefs
                servRefs, out string rank);
    boolean leave_group(in MembersCrash_seq
                              membersCrashId);
    void view_change(in short newRank,
                in short new_view_number);
    void commit(in short leaderRank,
              in string leaderId,
              in MembershipService leaderRef);
    boolean ack(in string memberAck);
  };
  // Failure Detector IDL
  interface FailureDetector:
            MembershipService {
    void keep_alive();
  };
  // State Transfer IDL
  interface StateTransfer: MembershipService {
    void get_state(in Object obj,
                   in string stg);
    void set_state(in short id,
                   in State state);
:
```

Figure 12. IDL for GroupPac services.

Each GroupPac service is a set of CORBA objects and as such communication between them is through the ORB. The IOR for each of these services *(i.e.,* FDS, STS and SM) is stored in the `AllServRefs` structure of the MS. Access to these IORs is local and guaranteed by inheritance of the Membership Service interface. Thus, each service object can coordinate with its colleagues in the other CosNamingFT replicas by first getting the appropriate IOR from the remote CosNamingFT replica.

When a new replica tries to join to the group, its SS object creates *(i.e.,* instantiates) the GroupPac service objects and puts its reference into `ServiceRefs` structure (Figure 12). Then, the SS object invokes the `join_group()` method of the primary MS object, sending the `ServiceRefs` structure. After the membership protocol has been executed and decided that the new replica can join the CosNamingFT group, the primary MS primary object disseminates the updated `AllServRefs` (including the `ServiceRefs` of the new member) to the other MS objects of the group by invoking the `view_change()` method.

The FDS service uses a simple "keep-alive" method, which each FDS group member invoking this method on its predecessor member of the virtual ring. An FDS member suspects its predecessor upon receiving a CORBA exception signaling a communication failure with the method invocation.

The Context variable is a Java `hashtable` and contains all object name-IOR bindings. The Context variable is the essential state that must be transferred to new NameContext group members, and that must be maintained consistently between the primary and its backups.

The STS methods `get_state()` and `set_state()` translate an object's state *(i.e.,* the Context variable) to or from a standard data format defined for state transfer using the STS. Since a Java hashtable is not defined within CORBA, we convert it to a sequence of bytes and treat it as a CORBA Any. In this way, when the STS object invokes the `get_state()` up-call in the NameContext object (Figure 10, step 5), it separates each binding in the Context variable and converts them to byte sequences.

To implement the interception shown in Figure 10 (step 2) we used OrbixWeb filters. A filter is a device that transparently traps the normal flow in a method invocation; it can be used to add functionality or extra control to a method call. The STS group within CosNamingFT is invoked using a post-marshalling filter to obtain the server object reply and extract the IOR of the primary replica.

5. CosNamingFT Performance Analyses

We now give performance measurements for CosNamingFT. These measurements were taken from the bind (install) and resolve (obtain) method invocations. We evaluate performance in scenarios with different degrees of replication. The execution environment is a local network (10 Mbps Ethernet) of heterogeneous

components: two Sun Ultra 1 running Solaris 2.5, one Axil 240 also running Solaris 2.5, one Pentium 100 and one Pentium 233 MMX, both running Linux and finally, one Pentium 233 MMX running Windows 95.

The tests consisted of one hundred bind operations and the same number of resolve operations. Figure 13 plots the response time as a function of replication degree using the average of these one hundred invocations for both operations. The bind operation curve shows a linear increase in the response time with degree of replication; the cost is about 7 milliseconds (ms). This occurs because the bind operation modifies the state of the NameContext object in the primary replica, making state updates necessary to each backup replica. As one would expect, the resolve operation is insensitive to replication degree.

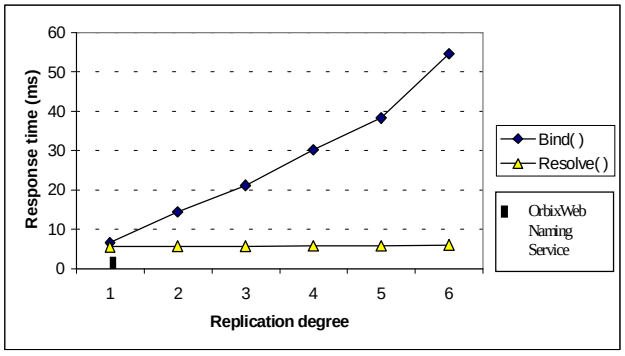


Figure 13. CosNamingFT Performance.

The increase in slope between the fifth and sixth replicas for the bind operation is due to the sixth replica being run on the slower, Pentium 100 machine. A replication degree of three is more than likely sufficient for our purposes; we believe that 21.2 ms response time is indeed acceptable.

In comparison, the native OrbixWeb name service without replication executes bind and resolve operations in 2 ms.

The performance tests reported [7] for Electra's replicatedname service were run on five Sparc 10 and four SparcStation 20 machines on a 10 Mbps Ethernet network, each running SunOS 4.1.3. Measurements indicated a per-replica overhead incurred by the bind operation of about 6 ms using Isis and about 1 ms using Horus/Ensemble. Well aware of the difficulties in comparing results given differences in hardware, operating systems, and so forth, we believe that the benefits seen in Electra over Ensemble are attributed to the highly-optimized group communication support provide by Ensemble *underneath* the ORB. In comparison, we implemented CosNamingFT service as a COSS object, with each GroupPac service object (*e.g.*, membership, failure detection, communication) handled *above* the ORB. For instance, group communication is accomplished by CORBA-object to CORBA-object method invocations, which in OrbixWeb cost approximately 2 ms each (accessing to the HTTP server costs nearly 3 ms). This extra overhead is the reason for the relatively poor performance we observe, and is the cost of conforming to recommendations for common services in the CORBA specifications. Obviously, performance in this model is dependent on the ORB and could improve or deteriorate.

6. Conclusion

The original work on CosNamingFT was done in conjunction with the Nile project [3] at the University of Texas at Austin. Nile is a CORBA-based platform for integrated distributed processing of highly-parallel computations. The original model of fault-tolerant CORBA objects was jettisoned due to the immaturity of platforms, and a model based on COS persistence was adopted until more robust object replication strategies become available[13].

The work presented here aimed to verify the viability of implementing fault-tolerant mechanisms at the application level using standards-based middleware, open system concepts, and common object services. The implementation of this service may be used at no restriction on any CORBA platform allowing, for example, CORBA objects developed in C++ language or others to use this service.

The fault-tolerant extensions to CosNaming discussed here require no changes the CosNaming specification and may be used with no restrictions on any CORBA platform. The GroupPac services were inserted flexibly and transparently. The SS object defines which particular GroupPac object services are required to build the selected fault tolerance framework. Thus, whether one uses GroupPac services or not, there is no need to change the code that implements the name service itself. In addition, following OMG recommendations, all communication between object services is through the ORB and based on IIOP; we have used no communication mechanism that is not in compliance with this standard. OrbixWeb does restrict portability or restrict use of other CORBA/Java platforms. We believe the per-replica cost of our CosNamingFT implementation is not burden-some, though it will assuredly vary from ORB to ORB.

References

[1] K. P. Birman and R. van Renesse. *Reliable Distributed Computing with the Isis Toolki*t. IEEE Computer Society Press, 1994.

[2] N. Budhiraja, K. Marzullo, F. B. Schneider, and S. Toueg. *Distributed Systems (2nd Edition). Edited by S. Mullende*r, chapter 4. The Primary-Backup Approach. Adison-Welsey, 1993.

[3] D. G. Cassel and M. Ogg. The Nile Project. Descriptions, publications, and software available from `www.nile.cornell.edu`, 1999.

[4] P. Felber, R. Guerraoui, and A. Schiper. The Implementation of a CORBA Object Group Service. *Theory and Practice of Object System*s, 4(2):93–106, 1998.

[5] IONA Technologies, Ltd. OrbixWeb Programmer's Guide. Available from `www.iona.com`, 1997.

[6] L. C. Lau, J. S. Fraga, and F. S. de Oliveira. Framework to Support Implementing Fault Tolerant Applications in CORBA. In *Brasilian Symposium on Fault Tolerance - SCTF 9*9. To appear. In Portuguese.

[7] S. Maffeis. A Fault-Tolerant CORBA Name Server. In *15th IEEE Symposium on Reliable Distributed System*s, pages 188–197, 1996.

[8] L. E. Moser, P. M. P. Melliar-Smith, and P. Narasimhan. Consistent Object Replication in the Eternal System. *Theory and Practice of Object System*s, 4(2):81–92, 1998.

[9] Object Management Group. The Common Object Request Broker 2.0/IIOP Specification, Revision 2.0. OMG Document 96-08-04. Available from `www.omg.org`, 1996.

[10] Object Management Group. CORBA Services: Common Object Services Specification. OMG Document 97-xx-yy. Available from `www.omg.org`, 1997.

[11] Object Management Group. Fault-tolerant CORBA Using Entity Redundancy (RFP). OMG Document 98-04-01. Available from `www.omg.org`, 1998.

[12] A. Ricciardi and K. Birman. Using Process Groups to Implement Failure Detection in Asynchronous Environments. In *Tenth PODC*, pages 341–351. ACM, 1991. (also Cornell University TR93-1328).

[13] A. Ricciardi, M. Ogg, and F. Previato. Experience with Distributed Replicated Objects: The Nile Project. *Theory and Practice of Object System*s, 4(2):107–117, 1998.

[14] R. van Renesse, K. P. Birman, R. Friedman, M. Hayden, and D. A. Karr. A Framework for Protocol Composition in Horus. In *14th Principles of Distributed Computing*, pages 80–89. ACM, 1995.

[15] R. van Renesse, K. P. Birman, and S. Maffeis. Horus: A Flexible Group Communication System. *Communications of the AC*M, 39(4):76–83, 1996.