Fault Tolerant Access to Dynamically Located Services for CORBA Applications

Titos Saridakis[†]
Nokia Research Center[†]
PO Box 407
FIN-00045 Nokia Group
FINLAND
Titos.Saridakis@nokia.com

Christos Kloukinas[‡]
INRIA[‡]
Domaine de Voluceau
Rocquencourt BP 105
78153 Le Chesnay Cédex, FRANCE
{Christos.Kloukinas, Valerie.Issarny}@inria.fr

Valérie Issarny[‡]

Abstract

This paper presents an effort that melds two facilities for localizing service providers and for tolerating failures into an assistance, which provides fault tolerant access to dynamically located services for object-based applications.

1 Introduction

The CORBA standard[1] eases the development of distributed applications by providing an object-based framework for structuring them and organizing the additional services needed for their execution. The localization service is captured by CORBA's Trading Common Object Service (COS)[2]. This COS allows application objects to register their interfaces in an interface database, and to query this database in order to obtain a reference to objects with specific interfaces. However, fault tolerance functionalities are not captured in a single CORBA COS; rather, they are scattered among different COSs (*e.g.* Persistent Object, Transactions, Collection Service, etc.). Their coordination leads to complicated application structure, which becomes even worse when the fault tolerance requirements refer to interactions with dynamically located objects, because the Trading Object Service must then be taken into account as well.

FTDA, an assistance for Fault Tolerant Dynamic Access to services provided by application objects, aims at keeping the application structure simple and comprehensible. It is designed as a COS built on top of CORBA's Trading COS and provides a suite of fault tolerance protocols offering ordered, reliable or atomic request delivery and active, semi-active, or passive object replication. The remainder of this paper is organized as follows: Section 2 presents the design of FTDA, Section 3 discusses the implementation of FTDA, and finally Section 4, summarizes with our contributions and a brief comparison with related work.

[1] http://www.omg.org/library/c2indx.html
[2] http://www.omg.org/library/csindx.html

2 Fault Tolerant Dynamic Access

One has three alternatives for dealing with the fault tolerance requirements of a CORBA application: at the application, the middleware, and the ORB level. The first one burdens the application logic and leads to complex application structures. The third one alters the CORBA specifications. The second alternative produces CORBA compliant applications with structural complexity proportional to the bare application logic, but requires the existence of COSs with adequate functionalities. This section elaborates on FTDA, a development assistance designed as a CORBA COS, which provides a variety of fault tolerance protocols for fixed and for dynamically located objects.

Fault Tolerance Protocols. In distributed applications, the failure of an object or a link may cause the whole application to crash. *Process group* [1] is a powerful concept that tolerates this kind of failures by replacing an individual object with a group of object's replicas. An additional mechanism, called *membership*, coordinates the replicas' executions and deals with their failures, so as to make the group of replicas behave as a single, fault tolerant object. FTDA builds upon this concept. Assuming crash processor failures and an asynchronous communication platform with performance and omission failure semantics, it was designed as a layered structure of group-based fault tolerance protocols. The structure of its layers is depicted

in Figure 1 and their functionalities are the following. SET transforms a number of objects into a set represented by a single reference; an action addressed to the set reference must be addressed to each of the set constituents. MULTICAST diffuses requests to each set constituent. ORDERED imposes a delivery order on requests addressed to the same set reference, enforcing the same delivery order for all set constituents. RELIABLE is based on the crash processor failure semantics and provides reliable delivery of requests addressed to a set reference. ATOMIC provides atomic delivery of requests addressed to a set reference, *i.e.* each request addressed to a set reference is either delivered to all set constituents in the same relative order or it is not delivered at all. MEMBERSHIP transforms a set of objects into a process group; when a failure of a set constituent occurs, this protocol updates the set composition, so that the protocols related to request delivery will continue functioning without blocking. REPLICATION duplicates an object and uses MEMBERSHIP to organize the set of object's replicas into a process group, which, thereafter, behaves as a single, fault tolerant object. ACTIVE, SEMI-ACTIVE and PASSIVE are all built on top of REPLICATION and provide the respective kind of replication for the process group. ACTIVE ensures that all group members execute a delivered request and return results to its originator. Using SEMI-ACTIVE, all group members execute a delivered request, but only a single one returns results. Finally, under PASSIVE, a single group member executes a delivered request and returns results; this member is also obliged to periodically checkpoint its internal state.

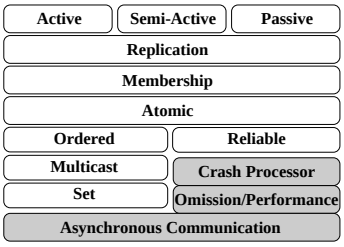

Figure 1: The structure of the FTDA layers

Dynamic Service Location. Some distributed applications may require the FTDA functionalities when accessing services whose server is not known *a priori*. For these, FTDA first employs a dynamic service localization facility, similar to the Trading COS, to locate an object offering the services described in the given interface. Then, it uses the requested replication protocol for binding the object requesting the FTDA service to the dynamically located object. Hence, FTDA offers two distinct service versions for each layer in Figure 1, one for known servers and one for dynamically located servers. Due to space limitations, Table 1 presents the FTDA interface that contains only the declaration of the services corresponding to the ACTIVE protocol. The declarations of the rest of FTDA services are similar to this.

```
typedef Object Srv; typedef Object Conf;
typedef sequence <Any> InterfaceDescr;
typedef sequence <Any> PropertyDescr;
typedef struct SrvIrfc {
    InterfaceDescr ServiceInterface;
    PropertyDescr ServiceProperty; } Service;
exception NoSrvFound { string reason; };
exception ExistingSrv { string reason; };
interface FTDA {
    Object ActiveDynamic(
    in Service service, in short numOfReplicas, in Conf machines)
        raises(NoSrvFound, ExistingSrv);
    Object ActiveStatic(
    in Srv srvRef, in short numOfReplicas, in Conf machines)
        raises(ExistingSrv); };
```

Table 1: The FTDA CORBA interface

The ActiveDynamic service may be requested by an object, which needs active replication to guarantee fault tolerant access to a service, and which does not possess a reference to the server offering the given service. ActiveDynamic takes three arguments: a description of the service to which fault tolerant access is requested, the number of replicas required to provide sufficient fault tolerance guarantees, and an optional suggestion for the deployment of the replicas. The service description contains the interface describing the service and a property that must be satisfied by the server offering the service. The algorithm implemented by ActiveDynamic is briefly explained in the following.

The service description parameter is passed to a dynamic service localization facility, called Localize, which searches a database containing the interfaces declared by existing objects, to find those matching the interface description. Localize returns the list of matching objects, which also satisfy the requested property, or a NoServerFound exception is raised if the list is empty. For each element on the list, ActiveDynamic performs the following process. If the element is not a server shared by all application objects, the requested group of replicas is initialized following the suggested configuration, if possible, and a reference to the object providing access to the group is returned. The same actions are taken if the element is a shared server, which has not been instantiated yet. In case of an instantiated shared server, ActiveDynamic checks whether the instantiated group has the requested number of replicas, in which case that one is returned. Otherwise, the process is repeated with the next list element. If the list is exhausted, an ExistingServer exception is raised, to indicate that servers offering the

requested service do exist, but do not provide the requested fault tolerance guarantees. The ActiveStatic service is simpler since it does not use Localize and it only performs the aforementioned process for a single server.

FTDA vs Existing COSs. Almost all of the functionalities offered by FTDA are provided by, or can be obtained by, combining existing CORBA COSs. The creation of replicas can be handled by the Life Cycle COS, the assembly of replicas into a process group can be supported by the Object Collection COS, the Transaction COS can be used to provide support for reliable communication, a request ordering mechanism can be built on top of the Concurrency COS, the Persistency COS provides the stable storage needed for the passive replication protocol, and the Trading COS is designed explicitly for issues related to dynamic service localization. So, a question that arises naturally is *"why bother with FTDA?"* Because FTDA promotes a clear and comprehensible application structure.

From the software reuse perspective, the fact that the FTDA functionalities are met from existing CORBA COSs is very encouraging. With a modular design and the appropriate implementation of the existing COSs, FTDA needs only to add the code that integrates the primitive fault tolerance functionalities in a CORBA COS that provides fault tolerance support. In theory, this approach can be followed with the existing implementations of the various COSs. However, a majority of these implementations are not designed to cooperate with each other and their integration requires a significant effort from the application developer.

3 FTDA in Practice

To assess the practical benefits of FTDA, we have implemented a prototype based only on a bare ORB, *i.e.* without making use of any CORBA COS. The original reason for not using any existing COS was to extend the capabilities of *FDFS* [2], a CORBA-based system already present in the Aster project[3].

Prototype Implementation. The FTDA prototype uses a dynamic service localization facility, called *Locator*, that we had previously developed for *FDFS*. Although *Locator* provides services similar to the Simple Trader level of the Trading COS, *i.e.* it allows servers to register themselves and clients to query it for a particular service, the two facilities differ in three aspects. First, whereas Simple Trader returns interfaces of the same type or a subtype of the requested one, *Locator* returns interfaces *compatible* to the requested one. An interface I_a is said to be *compatible* to an interface I_b *iff* the contents of I_a are a superset of the contents of I_b, when considering each interface as a set of operations and attributes. The other two differences concern the properties that must be satisfied by the servers in addition to the matching interface. For *Locator*, server properties are predicates and are always dynamic (*i.e.* their values are not cached). Hence, *Locator* implements the Localize facility referred to in the ActiveDynamic description in Section 2.

Besides *Locator*, the prototype consists of a CORBA object offering the FTDA services and a runtime library containing the protocol suite briefly described in the previous section. The FTDA object receives requests for fault tolerant access to a registered service, uses *Locator*, if necessary, to locate it, initializes a fault tolerant server, and returns an object that provides access to the server. This object acts as a client-side proxy. Similarly, each object in the replica group constituting the fault tolerant server is composed of two parts, the native server and the server-side proxy.

The FTDA library functionalities are divided into two categories: the packaging and the fault tolerance functionalities. The latter realize the various FTDA protocols, which provide fault tolerant access to services independently of the service type and the request contents. Figure 2 shows the deployment of the constituents of a fault tolerant server. In the magnifying glass, one may observe that a fault tolerant server consists of a number of replicas, each being an assembly of the native server (*e.g.* an application object), the packaging functionalities, and the FTDA protocol layer. Figure 2 also depicts the procedure for using the FTDA assistance. Let us consider for example, the case when an object requests the ActiveDynamic FTDA service. In step 1, the object contacts the FTDA object, specifying the interface of the service it wants to access and the fault tolerance guarantees it requires. The FTDA object invokes *Locator* in step 2, to obtain a list of references on servers providing the requested interface. That list is used by the FTDA object as described in section 2. If the fault tolerant server must be initiated, the FTDA object does so in step 3. Finally, in step 4 an *"access object"* is returned, which establishes a link with a server that offers the requested services and satisfies the required fault tolerance constraints. Once that link to the fault tolerant server is established, the FTDA object is no longer needed.

[3] http://www.irisa.fr/solidor/work/aster.html

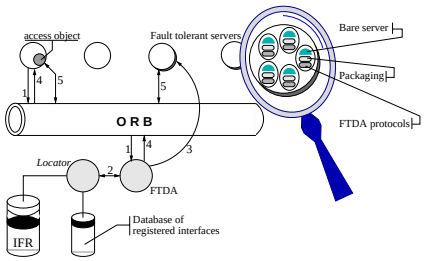

Figure 2: FTDA assistance organization and fault tolerant server composition

Prototype Evaluation. To assess the practical benefits of the FTDA assistance, we used it to obtain fault tolerant file accesses in a CORBA application, which is a simulation of a distributed file system [2]. The application consists of a number of file servers, each having registered its interface with *Locator*, and a number of clients that access files by obtaining a file server reference from *Locator*. In the non fault tolerant version of the application, the client invokes *Locator* once per different file, to obtain a reference to the file server that provides access to the specific file. Except from this *Locator* invocation, the client accesses files in the same way it would have done in a Unix file system.

Modifying the client to request fault tolerant file accesses proved to be a fairly easy task, since FTDA already included the *Locator* functionality. More precisely, the invocation of *Locator* is replaced by a call to the FTDA assistance which, in addition to the arguments passed to *Locator*, takes a few more arguments describing the requested protocol (*e.g.* in the case of ActiveDynamic, the client needs to pass as an argument the number of server replicas). FTDA performs all necessary actions and returns an object providing access to the, now fault tolerant, service. After that, the client may use the service without needing to contact FTDA or any other CORBA COS for fault tolerance related issues. This allows us to conclude that FTDA is easy to use and it does not introduce any complexity in the application structure, since it keeps to a minimum the number of interactions necessary for obtaining fault tolerant access to a specific service. Although we are pleased with the development benefits from FTDA, the performance of the prototype clearly leaves space for improvement.

4 Conclusions

FTDA is not the first effort to provide fault tolerance for CORBA applications. Primitive fault tolerance functionalities are found scattered among the *"official"* CORBA COSs. However, putting them together to obtain a non trivial fault tolerance functionality, like active replication, takes a considerable programming effort. In addition, it alters the rationale of the application structure, since a conceptually stand-alone application behavior is achieved through a number of interactions with various COSs not directly related to fault tolerance.

A number of approaches that offer full fault tolerance support for CORBA applications are mainly based on the integration of fault tolerance protocols with the ORB [3], to extend the standard ORB functionalities (*e.g.* see projects Electra[4] and Bast[5]). The inconvenience with such approaches is that they alter the standard ORB capabilities currently defined by OMG. In contrast, FTDA offers fault tolerance as a COS, which conforms with the CORBA specifications where services beyond those offered by the ORB should be inserted in the system architecture as COSs. Nevertheless, providing fault tolerance functionalities at the ORB level does not impact on the application structure, making the delivery of fault tolerance functionalities as simple as with FTDA. Additionally, an ORB extended with fault tolerance functionalities introduces, by default, much less execution time overhead to the application. The last OMG meeting made clear that this is an advantage significant enough to cause the revision of the ORB specifications in order to provide explicit support for fault tolerance.

The basic conclusion that we have drawn from our experience in designing, implementing and using FTDA is that no technological break through is necessary for facilitating the development of fault tolerant CORBA applications. A careful design and implementation of well understood concepts, suffice to provide transparent fault tolerance services for CORBA objects. Current work on the FTDA is focused on modifying, adjusting and polishing the protocol suite.

References

[1] K. P. Birman. The Process Group Approach to Reliable Distributed Computing. *CACM*, 36(12):37–53, December 1993.

[2] V. Issarny, C. Bidan, and T. Saridakis. Designing an Open-ended Distributed File System in Aster. In *Proc. of the 9th Int. Conf. on Par. and Dis. Computing Sys.*, pages 163–168, September 1996.

[3] S. Landis and S. Maffeis. Building Reliable Dis. Sys. with CORBA. *Theory and Practice of Object Systems (John Wiley)*, 3(1):31–43, April 1997.

[4] http://www.softwired.ch/people/maffeis/electra.html
[5] http://lsewww.epfl.ch/bast/