MetaFT – A Reflective Approach to Implement Replication Techniques in CORBA

Lau Cheuk Lung, Joni Fraga
Laboratório de Controle e Microinformática - LCMI-DAS-UFSC
Campus Universitário - Caixa Postal 476 – Trindade
CEP 88040-900 - Florianópolis – SC - Brazil
e-mail: {lau, fraga}@lcmi.ufsc.br

Carlos Alberto Maziero
Programa de Pós-Graduação em Informática Aplicada - PPGIA
Pont. Univ. Católica do Paraná - PPGIA
Curitiba – PR - Brazil
e-mail: maziero@ppgia.pucpr.br

Abstract

A model was introduced in [Fraga97] for integrating replication techniques in heterogeneous systems. The model adopts a reflective structure based on the meta-object approach [10]. Also, this model is founded in ORBs that support group communication in heterogeneous environments. The OMG still does not have specifications for fault tolerance. The MetaFT model has a great flexibility allowing, for instance, to modify the coordination protocols according to the fault tolerance level desired, without any implications for the application code. The advantage is that it allows to use different meta-object protocol to assist different quality of service (QoS) parameters to fault tolerance. This paper explicit our experiences in developing replication techniques following the model using two different CORBA platforms and presents some performance analyzes.

Keywords: distributed systems, fault-tolerance, CORBA, computational reflection.

1. Introduction

This paper presents our experiences, in the ambit of the ASAP project, about the integration of replication techniques in heterogeneous distributed systems. The project adopts an open architecture, following the distributed objects standard - CORBA/OMG [14]. At present, there is no sanctioned COS specifications for fault-tolerant CORBA objects through replication, though this is expected within the year. Our propositions attempt not to change the OMG orientations.

In [6] we presented an integration model for replication techniques in CORBA. This model uses the concepts of the reflexive computation [10], aiming to minimize the implications of the replication techniques over the application programming. The computational reflection is used in this context to separate the application algorithms of the replica coordination protocols. This separation introduces a great flexibility in the system for allowing the change of coordination protocols without interfere in the application code, not even to imply in changes at the execution support level, what would be hard if we consider heterogeneous distributed systems. The ORBs, with specific facilities for group communication, support the implementation of the replication techniques according to the proposed integration model. The integration model was used with the Electra [11] and Orbix+Isis [8] systems.

This paper concentrates on those implementation experiences, describing aspects related with the means offered by those CORBA platforms to facilitate the implementation of the replication techniques. The potentialities of those ORBs and the proposed model are evaluated in this paper starting from the efforts that we did, programming different replication techniques structured according to that integration model. With the objective of illustrating the use of the model in the considered environment and the implementation aspects involved in the two ORBs mentioned, we present in this text our experience for implementing the leader/followers replication technique [16].

In section 2 we present the description of the CORBA platforms used. Section 3 describes the integration model – MetaFT. In section 4 the integration model on ORBs. In section 5, general considerations about our implementation experiences using Electra and Orbix+Isis are presented. Finally, in section 6 the final conclusions of this work.

2. The CORBA platforms used

2.1. CORBA architecture

The 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.

In the development of distributed applications, the CORBA/OMG specifications included a set of object services that simplify the application designer's task. In this way, the COSS (*Common Object Services Specifications*) was introduced describing a set of services (interfaces) that provide basic functions for using and implementing application objects.

2.2. ELECTRA

Electra [11] is an ORB compatible with the CORBA specifications, presenting support for objects group. For the development of distributed applications, this model combines the benefits of the CORBA standard with the power of lower level tools, such as: Isis [2], Horus [17], Transis [1], Consul [12], Chorus [4], and others. Electra is implemented in C++ and the IDL for C++ mapping is dealing with specified for OMG [13]. The Electra communications can be given in the reliable multicast or point-to-point style. The client makes use of a same model of invocation of method, independent if the server is an object singleton or a group of objects. In Electra, the invocations can be synchronous, asynchronous or semi-synchronous (deferred-synchronously), using static or dynamics interface in the CORBA model. Two way of group communication is available in Electra:

- Transparent: the group is seen as a simple and highly available object where the requisitions are submitted as in a conventional ORB that is, the client only receives a single result of the group;
- No transparent: permits the access, in an invocation, to the results of each individual member of the group of objects.

Electra supports groups in the active replication model, making use of the tools of lower level, like Isis or Horus, in the supply of services of reliable multicast and group management (figure 1). The state transfer and membership services, supported by those tools are available as Electra's operations in BOA (Basic Object Adapter). In the BOA interface of the Electra were added mechanisms to activate groups and to select the multicast protocol to be used. The CORBA environment class is used for the selection of invocation style and mainly, to pass exceptions of the server to the client.

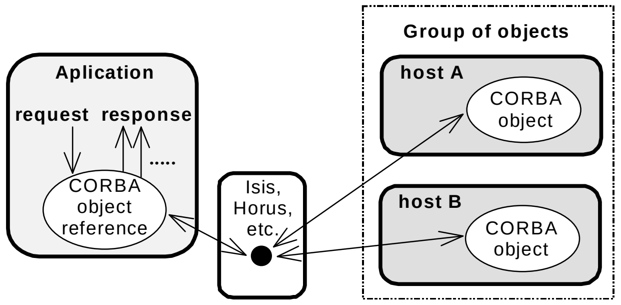

Figure 1. Group Communication on Electra

2.3. ORBIX+ISIS

The Orbix+Isis system [8] is a commercial product developed by ISIS Distributed Systems Inc. and IONA Technologies, Ltd. As the previous ORB, it allows a system to be built as a group of objects interacting according to CORBA/OMG standard. Orbix+Isis system simplifies the development and the integration of distributed fault-tolerant applications. Each object has an interface specified using the IDL/CORBA language. The ISIS resources in the implementation of the group abstractions, in this case, simplify the ORB. In the Orbix+Isis model, servers may be formed by group of objects where the client/server interactions are reliable. Membership mechanisms, state transfer and reliable multicast using different types of order, supplied by ISIS, are available at ORB interfaces to the applications in the Orbix+Isis.

Groups of objects can be defined over two execution styles: group processing and event stream. In group processing, each object member shares the same interface and semantics of equivalent implementation, what assures that each replica member in an identical way process the same service request. Executions in the group processing style allow three communication ways:

- *Multicast:* a request is diffused for all members of a group and only one result is returned to the client, maintaining the server transparency. The multicast style corresponds to the active replication model [18]. On the other side, the client can have access to the response from all members of the group; for that, it is necessary to build a smart proxy (a proxy object supplies group support in the client side). The proxy would inherit the behavior of a default proxy class and to add the capacity to send results to the client.
- *Client's Choice:* in this style the client request is executed in just one of the group members. This communication style has the aim to improve the performance in client/server interactions. It is created for read-only operations. The `choosier` function in the client side determines the member for receiving the request.
- *Coordinator/Cohort:* in this model, a coordinator executes the requested operation and then, sends the results to the client and, to the other replicas (cohorts) for updating their states (checkpoint). A `choosier` function determines the coordinator for each request processing submitted by the client. If the coordinator fails the `choosier` function is automatically invoked for the choice a new coordinator.

In the group processing style, Orbix+Isis still offers membership services, state transfer and ordering of messages. State transfer and membership are key services for different models of group, because they allow new objects to join a group and to become a normal replica of the group.

In the event-stream processing style, the clients use one-way asynchronous communications. They send messages (events) to object members of a group, called “event receivers”. This execution style follows the publisher/subscriber model: clients send asynchronous messages to the “event stream” which are responsible for sending to the subscribers. The “event stream” also maintain copies of these events. Event receivers can join or leave the subscribers group any time. The subscriber participation in the group can be programmed for the reception of a certain amount of events. Events-stream model corresponds to a loosely coupled connection between client and servers.

The Orbix+Isis architecture is presented in figure 2. It consists of C++ library and a run-time support, which implement the functionality of group processing and event stream. This structure involve, together, a conventional ORB (Orbix) and the Orbix+Isis group support. The Orbix communication layer treats point-to-point communications. The Orbix+Isis support handles group communications using Isis facilities and then, offering bases for distributed fault-tolerant applications.

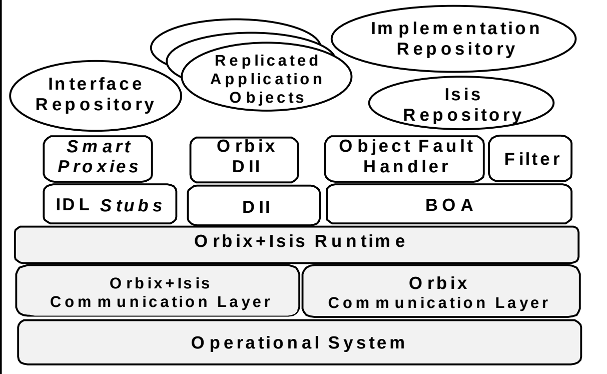

Figure 2 The Orbix+Isis architecture

The IDL interfaces in Orbix+Isis generate in compilation time the necessary structures for group processing. Using these structures (stubs, proxy, etc.), the client connects and communicates with an objects group like a singleton object. The Isis Repository (IsR) is a file hierarchy of the Orbix+Isis, similar to the implementation repository in the Orbix. Each Orbix+Isis server has a file in IsR, which defines the activation procedures and the configuration of the server group. The performance aspects and the execution style of the group are specified in the Isis Repository (IsR). The available information in these files allow, for example, changes in the application involving the group execution style, without changing the application code.

3. Integration model

3.1. Computational reflection

The essence of the computational reflection paradigm is a system that executes processing on itself, modifying their behaviors. The reflexive paradigm is introduced into the object-oriented programming following the meta-objects protocols [10] where the functional and no-functional aspects are separate using base-object and meta-objects, respectively. A base-object describes in their methods the application functionality, while the associated meta-object executes the control policies that determine the behavior of its corresponding base-object. The calls to the base-object methods are trapped in the sense of invoking meta-methods that allow to modify the behavior of the base-object or to add functionalities to the corresponding calls in base level.

3.2. MetaFT - The integration model

The reflexive paradigm allows assigning to the base-objects the functionality of a replicated application, while meta-objects execute the coordination protocol to control the replicas execution. The coordination reflected allows the use of different replication techniques with the same base-object maintaining their characteristics. For changing the replication technique, changes are concentrated in the meta-level.

The reflexive structure proposed for incorporating replication techniques in open systems is represented in figure 3. Each replica is mapped to one base-object (replica_base), which is associated a meta-object (meta-controller) that assumes coordination functions of the replication technique used. Crash failures were assumed in the replication models in our experiments. As we admitted a strong coupling between controller and associated replica, under a crash failure, both, controller and associated replica will stop their processing activities.

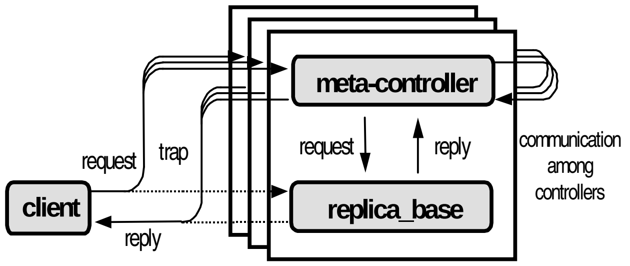

Figure 3 Reflective structure for the replication model.

The integration model of replication techniques in the CORBA context is shown in figure 4. In that figure, the client is structured in a base-client that represents the behavior of the application, and a meta-client, that does not possess active function in our implementation, but that could be used in managing the replicated client, or to implement mechanisms of handling exceptions in the client. The structure of each server replica is similar to that of the client: a replica-base object, carrying out the replicated services; a meta-controller, responsible for the coordination protocol of the replication technique; and meta-objects special, identified generically as meta-communication, that it control so much on the client side as in the server side the access to the support supplied by a CORBA platform. These entities concentrate the set of clients stubs, of server stubs (the stubs for communication of the replicas) and the BOA with the support for group management, generated starting from the compilation process of the IDL specification of the interface of the object server. The use of the term “abstract object” given to the meta-communication on the model follows some authors [7] and has the sense of a simple separation for greater clarity. In reality, these interfaces are generated as a set of methods that will be composed with inheritances multiple in the client and controller meta-objects.

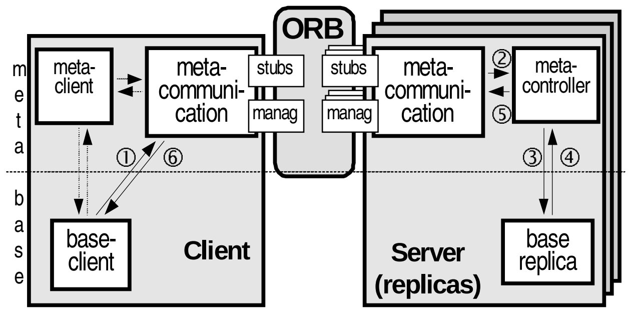

Figure 4. Structure of the model on a CORBA support.

The numbered arrows in figure 4 indicate the normal way of a client request: The request made by the client base (1) is then broadcast using a stub appropriated in the client meta-communication. In each replica, the meta-communication, by means of a local stub, receives the request and transfers it to the meta-controller (2), which then activates the local replica (3). On receiving the reply (4), the meta-server executes the coordination protocol, by means of the meta-communication so as to interact with other replicas. The processing and interactions on the level of the meta-controllers are conditioned at this time by the replication model utilized. Later, the reply is then sent back to the client (5 and 6).

This model can be used with several replication techniques; the differences essentially will concentrate on the replicated meta-controllers (servers). In some techniques the meta-communication entities may gain functionality besides that of concentrating methods of access to the services of the CORBA support. For instance, in the use of active replicas with voter or adjuster mechanisms, the implementation of the voting or adjustment can be programmed in a more simplified way on the client side. The Transparency could be achieved in this case, implementing these mechanisms in the client meta-communication entity, which, with the addition of this functionality, takes on the characteristics of a real object.

4. Using integration model in CORBA

4.1. Leader/Followers replication according to the integration model

In the sense of illustrating one of the several examples of replication techniques developed by us with the mentioned model, we described in this section our experience with the leader/followers technique. In this replication model all the replicas are active and execute the same code, but the leader replica is the responsible to handle the interactions with clients, and for the decisions that affect the determinism of replica [18]. When receiving a request the leader disseminates the same among the followers. The processing of the method is the same in all replicas, however only the leader sends the results to the client.

The figure 5 presents the meta-controller's code regarding the leader/followers technique. To each base-method has a meta-method associated in the controller (base_method_1 and meta_method_1, figure 5). The controller's actions are described brief in the mentioned figure. A temporal diagram involving the interactions among replicas in the processing of a requisition of client is shown in figure 6.

```
class meta_controller_1_im {
// declaration of variables

   method meta_method_1(parameters){
     // declared in IDL
     base_method_1 (parameters);
     meta_control (parameters);
  };
  // declaration of other meta-methods
     :
};
```

```
class meta_controller_2_im {
  method meta_control(parameters){
    concluded := false;
    semaphore := false;
    my_id := rank_system ();
    leader := 0;
    while not concluded do
      leader := leader + 1;
      if (my_id = leader) then
      // I am the leader
        group.reply_leader(reply);
        return;
      // response comes back to
      // the client
      else
        wait(semaphore or timeout);
        if not concluded then
          group.closing ();
        end;
      end;
    end;
  };
  method closing(){
  // this method is declared in IDL
    if (leader ∈ membership) then
      concluded := true;
    end;
  };
  method reply_leader(reply) {
  // this method is declared in IDL
    if (my_reply ≠ reply) then
      my_reply : = reply;
      semaphore := true;
    end;
  };
};
```

Figure 5. Code of Meta-controller of the leader/followers replication technique.

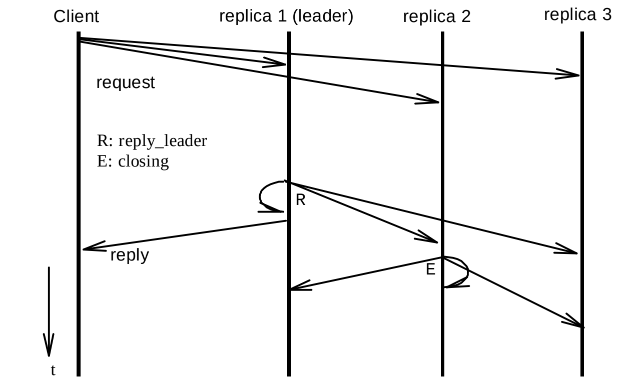

Figure 6. Temporal diagram of the *leader/followers* replication technique.

The leader's indication is determined by the replica with my_id equal to 1, the oldest of the group (rank_system = 1, figure 5). The used support (Horus and

ISIS) adopt rank mechanisms, what supplies to the implementations of the replication techniques that depend on privileged replica, the advantage of choosing a leader without the need of any change of message among replicas. The use of the closing method has as purpose the detection of failure of the privileged replica (leader) what is simplified by the used platforms. The closing method is implemented using the membership lists supplied by the CORBA platform. Those membership lists are up-to-date for the BOA, to each arrives or leaves of replica in the group.

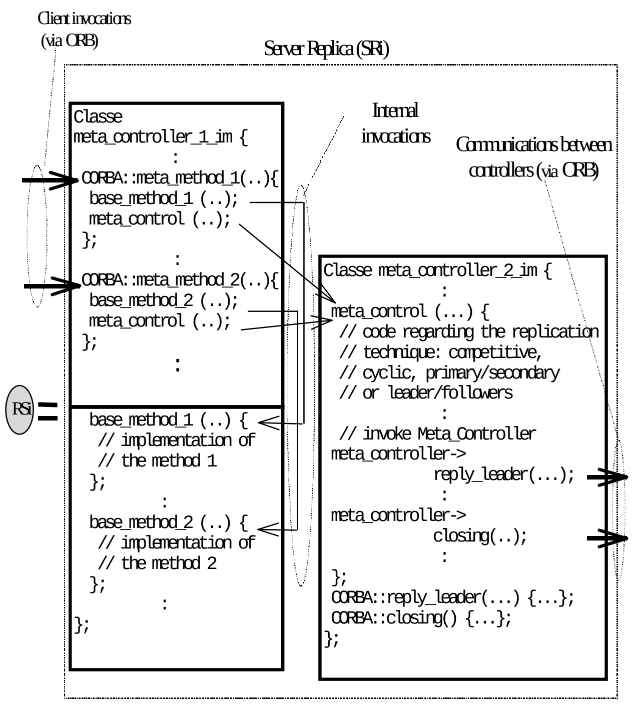

Figura 7. Format of the separation of server code.

In figure 6, after the processing of the requested method, the leader (replica 1) is shown sending the results to the client and their followers. The followers once satisfied the wait condition (figure 5 and 7), executes the closing method. The leader replica cannot signal the end processing to the other members of the group because when the results return to the client, its execution is finished. The activation of the closing method fits the at least a follower replica that will provoke the execution of the same in all the other replicas. This method has the function of certifying if the response was sent indeed by the leader (concluded condition of the while loop). The verification of the closing is simple: if after the diffusion of the closing method the leader is still alive (to belong to the membership of the group) then the response was sent indeed. Otherwise, a new leader is chosen and the process is repeated. In the algorithm, the activation of closing method is transmitted to all the replicas of the group, in ordered totally way.

In our implementation, the base methods were simulated with operations of an application of bank service and were developed separated on behalf of the replica coordination (meta-methods). The figure 7 emphasizes this separation that is the base of the reflexive paradigm. In our experiments different meta-objects, implementing different replication techniques, were changed, changing the behavior in the coordination of replica of the service, without affecting the functionality of the base level.

4.2. Implementing the model using the ELECTRA system

The initial step for the implementation of a system on any CORBA platform is the description in IDL of the meta-controller's interface. The interface consists of the declaration of each method offered by the replicated server to the client. Besides this, it is necessary to declare a second interface, composed by the methods that handle the management of the replication technique, in the interactions among the different replicas of the service. The code in IDL of both CORBA interfaces of replicated server, in dealing with the specifications described in the previous section, is shown in figure 8. The meta_controller_1 interface specifies the access of client to the replicated service, while the meta_controller_2 interface declares the necessary methods to the intra-replica interactions.

```
// IDL

interface meta_controller_1
{
   // Descrition of the data types employed

   // Descrition of the server methods
   boolean meta_method_1 (parameters);
   ...
   boolean meta_method_n (parameters);
};

interface meta_controller_2
{
   // Descrition of the meta_controller methods
   boolean replt_leader (in reply);
   boolean closing ();
};
```

Figure 8. IDL Interface of the replicated server.

The Electra ORB version 1.0 does not support preemptive threads, what limits the concurrency degree to treatment the client's requests, besides the interactions among the meta-controllers of the replicated service. This restriction brings difficulties in the implementation of replication techniques. To treat this limitation, we adopted a solution that consists of separating the meta-controller's functionality in two UNIX processes. The first one is concern the client/server interactions and activation of method of the base-object and, the second one is involved with the interactions among replicas (among the meta-controllers of the replicas). It fits to stand out that both interfaces are actually two faces of a same server (or, in our case, of a same group of objects). Therefore, the two interfaces of the figure 8 were compiled separately, and in runtime the system woks like two groups of objects (group meta_controller_1 and meta_controller_2).

In the process of compilation of the interfaces of an application, Electra generates the whole support automatically for the communication (stubs) among the involved entities, also including the functionalities for groups management of the BOA class. The programmer is responsible for the description of the base-object of the application (meta_controller_1) and management of the replication technique (meta_controller_2), filling the bodies of the methods defined in the interfaces (figure 8). With that implementation outline, which is illustrated in figure 9, them base-object client and server stay out of any activities that are not related to the application itself. All the aspects related to the management of the replication and the interactions in CORBA context are concentrated at the meta-objects level (meta_controller_2).

4.3. Implementing the model using the Orbix+Isis

The Orbix+Isis, if compared to the Electra, is a commercial product, that presents more means for processing and management of groups [section 2.3]. But, the main advantage in relation to the Electra is that it provide multithreads support. In implementation of the integration model using Orbix+Isis, each pair associated base-object/meta-object shares a same UNIX process, doing with that the interactions between both are local, without need of ORB. The concurrency among base-object and meta-object inside a process are satisfied through the use of a threads library provided by the Isis tool. The initial step of the implementation is also the compilation IDL of the two interfaces of the figure 8. In this case, the compilation creates an only group of processes in runtime. The figure 9 explicates the process of generation of the client and server codes starting from IDL.

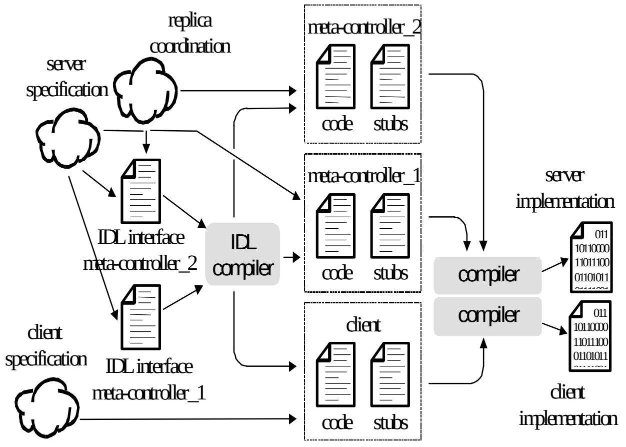

Figure 9. Application building process

4.4. Performance analyze

In this item we present performance measurements carried out in our laboratory, to check our experiences using Electra and Orbix+Isis to implement the MetaFT. These measurements were taken from request method invocation. MetaFT performance was raised considering different replications degrees. The environment considered in these performance tests, was composed of a local network (10 Mbps Ethernet) heterogeneous using two machines Sun Ultra 1 with Solaris 2.5, one Axil 240 also with Solaris 2.5, one Pentium 100 and one Pentium 233 MMX, both running Linux and finally, one Pentium 233 MMX with Windows 95.

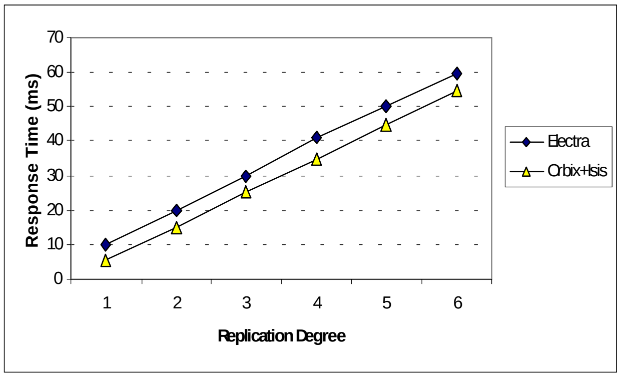

Figure 10. *MetaFT* Performance using Electra and Orbix+Isis.

The tests carried out consisted of a hundred request operation calls. In figure 10, the response time (axis Y) related to the replication degree (axis X) had been determined considering the average of these one hundred invocations. The request operation curve represents an increase in the response time whenever the MetaFT degree of replication grows. Using Electra or Orbix+Isis to implement the MetaFT the growth factor of the response time rate per replica added was around 10 milliseconds. But, Orbix+Isis presents a better performance - around less 5 milliseconds.

5. General considerations

The multithreads mechanism that Electra operates is called TPP (Thread Per Process), this model imposes that the server executes one request in each time, the beginning to the end, multiple servers objects that share a same process cannot receive several invocations in the same time. In some cases this model can reach a deadlock, for instance: supposes that a server A, when being invoked by the client, make a request to the server B and that during the processing of this request, B activate A, this would provoke a deadlock in the system, because A can only process one request per time (figure 11).

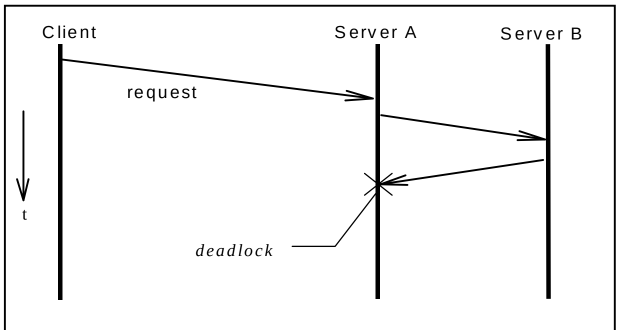

Figure 11. deadlock caused by the B replica.

The separation in two processes, meta-controllers 1 and 2 in Electra, emphasizes the difficulties of implementation of the integration model when uses the TPP approach. The communication among the two processes is made through ORB. This causes an low performance once the activation through ORB should be considered the total time in the call of a method in this context (to locate the remote object, to prepare the object implementation to receive the request, to transport the data and to return to the client the results in a transparent way).

The multithread mechanism that we needed to implement the proposed integration model is called TPR (Thread Per Request). With this mechanism, a new thread is created for each invocation that arrives to the server and it avoids deadlock problems that can happen in TPP approach. Orbix+Isis uses a multithreads support in TPR, therefore the meta-controllers compose just one process, guaranteeing an gain of performance (figure 10), once the communications among meta_controller_1 and meta_controller_2 (figure 7) are made internally without to use the ORB. Both CORBA platforms used in this work presents means that would allow the implementation of mechanisms of detection of less restrictive faults (such as temporal, value, etc.).

In relation to the crashes that eventually happens in the evolution of the system, the integration model concentrates at meta level operations to recovery the replication degree, independent of the type of replication technique used. If the number of active replicas in the group to drop below a preset limit, the oldest replica takes the initiative of throwing new replicas, reestablishing the ideal population. The code regarding these recovery procedures is equivalent for two ORBs considered and is based on a membership test (view.number < quorum_min), inserted in the body of the view_change method. Our approach of state transfer (checkpoint) of the replicas differs of that proposal in [5], which the state updating are given through meta-methods making updates in public attributes of their associated replicas, with the exclusive use of coordination protocols. In our approach we used more primitive of support and less protocols at coordination in Meta level, simplifying the meta-controllers codes. The state recovery, in our system, is based on the join primitive, that it should be offered by the BOA interface of the CORBA support, and invoked through the view_change method.

Due to the fact of the used language (C++) do not provide specific support to the computational reflection, for instance, the automatic trap of invocation of a method in base level to the respective meta level, in our implementations the reflection is implemented artificially, through the direct activation to the meta-method, starting from the client code (figure 7). The use of a language supporting reflection, like Open C++ [3], could to eliminate this problem, but in this case the CORBA/IDL compiler would have to support the mapping for that language. Inside of OMG there is a group working in the standardization of the use of meta-objects protocols in CORBA, more information about that can be obtained in [15].

The computational reflection allows the independence of codes of the replicas in relation to the coordination protocols, driving to a great flexibility in the system: to change of technique or to alter it to reach to degrees of fault tolerance wanted can result in simply to change the coordination protocols in meta-level, without to imply in any alteration to the application algorithms, or still in changes at level of execution support, what would be difficult in heterogeneous systems. The use of the reflection to implement fault tolerance techniques is not new [5], [6], and even the separation between the coordination and the replicas has already been recommended in [16].

The presented model integrates reflection concepts and group processing in heterogeneous environments. The accomplished implementation makes intensive use of the functionalities of CORBA platforms, what implied in a simplification of the coordination needs in the replication techniques implemented. Moreover, the use of a CORBA platform allowed the implementation of our application on a heterogeneous execution environment (group of machines with operational system Solaris 2.5 and Linux 2.0, in a local network), facilitating the interoperability aspects.

The structure of integration proposed was shown quite flexible, could to support several replication techniques easily. Until the moment we implemented the technical primary/secondary [2], leader/followers [9], competitive [16] and cyclic redundancy [16], using the same integration model. The necessary changes to the substitution of the replication technique in the integration model are limited to the interface the IDL meta-controllers and to their codes, which implement the protocols of corresponding coordination. Then, we can to build different replication techniques to assist the different quality of service (QoS) parameters of a fault-tolerant application. The QoS parameters could be, for instance: recovery time, response time, types of fault to tolerate, etc.

6. Conclusion

The implementations presented in this paper verified the viability to implement mechanisms of fault tolerance on level of the application over CORBA platform using a reflective approach. The implementations were made accomplishing the separations recommended for the computational reflection. These same implementations makes extensive use of the primitive of the Electra and Orbix+Isis support, what does not also imply in the loss of portability of the codes of the application. In other word, changes in the support are reflected in the meta-level; the codes that reflect the functionalities of the application are not affected. The obtained results, although the limitations of Electra, were considered satisfactory. The objective of this paper was discussing the MetaFT model in the implementation aspects, considering our recent experience with Orbix+Isis.

References

[1] Amir, Y., Dolev, D., Kramer, S. and Malki, D., “Transis: Comunication Subsystem goes High Availability.” In 22nd International Symposium on Fault-Tolerant Computing, IEEE, July 1992.

[2] K. P. Birman, “The Process Group Approach to Reliable Distributed Computing”, Technical Report TR 91-1216, Cornell University Computer Science Department, Ithaca, N.Y., July 1991.

[3] S. Chiba, “Open C++ Programmer´s Guide”, Technical Report 93-3, Department of Information Science, University of Tokio, 1993.

[4] Chorus Systemes, “Chorus Simulator v4 r1 - Programmer's Guide”, http:\\www.chorus.com, 1992.

[5] J. Fabre, V. Nicomette, T. Pérennou, R. J. Stroud and Z. Wu, “Implementing Fault Tolerant Applications using Reflective Object-Oriented Programming”, Proceedings of the 25th IEEE International Symposium on Fault-Tolerant Computing, Pasadena (CA), June 1995.

[6] J. Fraga, C. Maziero, Lau L. and O. Loques “Implementing Replicated Services in Open Systems Using Reflective Approach”, Proceedings of the 3th IEEE International Symposium on Autonomous Decentralized Systems - ISADS 97, Berlin - Germany, April 1997.

[7] O. Hagsand, H. Herzog, K. P. Birman and R. Cooper, “Object-Oriented Reliable Distributed Programming”, IEEE, 2nd International Workshop on Object-Orientation in Operational Systems, I-WOOOS/1992.

[8] Isis Distributed Systems Inc., IONA Technologies, Ltd. “Orbix+Isis Programmer's Guide”, 1995. Document D070-00.

[9] M. C. Little, “Object Replication in Distributed System”, PhD. Thesis, University of Newcastle upon Tyne Computing Laboratory, September 1991.

[10] P. Maes, “Concepts and Experiments in Computational Reflection”, OOPSLA 87 Proceedings, pp. 147-156, October 1987.

[11] S. Maffeis, “Adding Group Communication and Fault-Tolerance to CORBA“, In Proceedings of the 1995 USENIX Conference on Object-Oriented Technologies, Monterey - CA, June 1995.

[12] Mishra, S., Peterson, L. L., and Schlichting, R. D. “Consul: Communication Substrate goes Fault-Tolerant Distributed Programs.” Distributed Systems Engineering Journal 1,2 Ten. 1993.

[13] Object Management Group, “IDL C++ Language Mapping Specification”, OMG Document 94-9-14, 1994.

[14] Object Management Group, “The Common Object Request Broker 2.0/IIOP Specification“, Revision 2.0, OMG Document 96-08-04, 1996.

[15] Object Management Group, “Meta-Object Facility”, RFP 5, OMG Document 96-05-02, 1996.

[16] D. Powell, “Delta-4 Architecture Guide”, Esprit II P2252, Delta-4 Phase 3, August 1991.

[17] Robbert V. Renesse and Kenneth P. Birman, “Protocol Composition in Horus” Dept. of Computer Science of the Cornell University, Sea 1995.

[18] F. B. Schneider, “Implementing Fault-Tolerant Service Using the State Machine Approach: Tutorial”, ACM Computing Survey, 22(4):299-319, December 1990.