Architectures Fault Tolerant CORBA
Eternal et IRL

François Trifin

Facultés Universitaires Notre-Dame de la Paix, Institut d'informatique,
B-5000 Namur, Belgique

Résumé L'intégration de la tolérance aux fautes dans les systèmes distribués n'est pas encore couramment présente mais le deviendra avec certitude dans les prochaines années. Nous présentons deux stratégies d'implémentation de la spécification Fault Tolerant CORBA de l'Object Management Group (OMG). FT-CORBA vise à prodiguer au système CORBA (Common Object Request Broker Architecture) la résistance aux fautes de manière à maintenir un service hautement disponible.

1 Introduction

Beaucoup d'applications distribuées différentes ont besoin d'être tolérantes aux fautes. Du grand système critique, tel un système de contrôle du trafic aérien, au moins grand, telle une application de production industrielle ; de plus en plus d'applications ont besoin de maintenir en fonctionnement leurs services en cas de défaillance d'un sous-système.

La spécification FT-CORBA [1], publiée pour la première fois par l'OMG au début de l'année 2000 et ensuite incluse dans la version 3.0 de CORBA, définit une série de services et d'améliorations du système CORBA, en vue d'apporter la tolérance aux fautes aux systèmes distribués. Cette spécification est le résultat d'un effort de standardisation d'un ensemble de systèmes créés dans le but d'offrir la tolérance aux fautes aux applications utilisant CORBA.

FT-CORBA étant une spécification, il existe plusieurs stratégies d'implémentation possibles de l'architecture. Ainsi, un système répond à la norme FT-CORBA si celui-ci fournit les interfaces et implémente un sous-ensemble des fonctions spécifiées dans la spécification FT-CORBA.

Nous développons dans ce document un aperçu général de la spécification FT-CORBA. Nous présentons ensuite deux architectures conformes à cette spécification. La première architecture, l'Interoperable Replication Logic (IRL), a été réalisée au "Dipartimento di Informatica e Sistemistica" de l'Université de Rome "La Sapienza" et possède un prototype encore en développement [2]. La seconde architecture, Eternal, issue du "Departement of Electrical and Computer Engineering" de l'Université de Californie, est une implémentation propriétaire d'Eternal Inc. [3]. Enfin, nous terminons par une discussion sur les architectures FT-CORBA.

Nous supposons dans cet article que le lecteur possède un minimum de connaissances du système CORBA. Celui-ci pourra se référer aux documents suivants [1,4] si tel n'est pas le cas.

2 Principes de tolérance aux fautes

Deux stratégies font partie de l'état de l'art et sont actuellement utilisées en entreprise pour supporter les fautes et permettre la haute disponibilité des services. La première, la plus développée, facile à mettre en oeuvre et d'un coût très faible, est la tolérance lors du traitement des transactions.

Typiquement, un serveur unique est utilisé mais soit celui-ci est dupliqué en plusieurs miroirs ou soit celui-ci a accès à un disque dupliqué. Si une transaction est complètement réussie, alors l'ensemble des disques ou l'ensemble des miroirs doit être mis à jour de manière atomique.

Cependant, cette stratégie n'est pas transparente pour le client. En effet, dans le cas d'une faute, l'ensemble des transactions actives est alors abandonnée et redémarrée à la charge du client. Elle n'offre donc pas la tolérance aux fautes au client mais seulement au serveur. Bien que simple dans le cas de serveurs fonctionnant indépendamment, il devient très difficile, avec cette technique, de maintenir en tous points du système une information cohérente dans le cas d'interactions plus complexes entre plusieurs serveurs.

La seconde stratégie appliquée, plus difficile à mettre en place lors du développement d'une application, utilise une véritable logique de réplication des composants. L'idée générale, pour un système tolérant aux fautes, est alors de répliquer en plusieurs exemplaires chaque composant du service. Chaque exemplaire, appelé réplique, est alors réparti sur différents hôtes reliés par un réseau de communication. Pour surmonter la perte d'un ou plusieurs composants, un système de détection des fautes et de récupération des composants est utilisé.

Il est à remarquer que nous ne parlons pas ici de fautes d'implémentation ou de conception mais de fautes dues à la non-réponse du service. Une faute peut être le crash d'un composant dû à, par exemple, une erreur physique. Il est cependant impératif que le fonctionnement du composant ait été toujours correct avant le crash. Une faute peut être aussi l'omission d'une réponse par un composant, la perte d'une réponse par le matériel de communication ou encore une réponse obsolète après avoir dépassé un temps imparti. Mais dans le cas où un composant donne une réponse incorrecte, seule la réplication active avec vote permet de ne pas en prendre compte.

Il existe plusieurs logiques de gestion de la réplication qui ont toutes leurs avantages et inconvénients. Les principales techniques de réplication sont :

Réplication active Dans la logique de réplication active, toutes les répliques d'un objet exécutent indépendamment la méthode invoquée, de manière à ne pas recommencer la méthode si une réplique ne fonctionne pas correctement, évitant ainsi une perte de temps.

Réplication active avec vote La réplication active avec vote correspond à la réplication active avec l'addition d'un système de vote à la réception et à

l'emission d'une requête. Celle-ci n'est reçue/envoyée que si la majorité des demandes/réponses sont identiques.

Réplication passive La réplication passive est une logique de réplication où seule une des répliques, appelée principale, exécute la méthode invoquée.

Réplication passive à froid Cette forme de réplication passive archive l'état de la réplique principale et utilise la dernière archive créée pour charger une autre réplique en cas de récupération.

Réplication passive à chaud La réplication passive à chaud est une forme de réplication passive où les répliques non-principales ont leur état régulièrement mis à jour à partir de la réplique principale.

Lors de l'application d'une logique de réplication, il faut distinguer les objets avec état et sans état. Un objet est sans état si son comportement n'est pas affecté par son histoire d'invocation. (Par exemple, un composant dont la fonction n'est que la lecture d'un élément dans une base de données) Les logiques de réplication passive à froid et à chaud n'ont donc pas d'intérêt si l'objet est sans état. La cohérence des répliques sera dite forte si, à la fin de chaque invocation, les répliques possèdent toutes le même état et que leurs séquences d'invocation ont été identiques.

De plus, il faut aussi tenir compte du caractère déterministe ou non-déterministe du composant répliqué. L'objet possédant, par exemple, plusieurs sous-processus, ne sera pas déterministe. Dans ce cas, la logique de réplication active ne pourra pas être appliquée.

Enfin, dans le cadre du développement d'une application distribuée, la logique de réplication, ainsi que la logique de communication impliquée par celle-ci, est généralement directement intégrée à l'application. Il existe des toolkits de réplication[1] (*Group communication toolkit*) apportant une aide au développeur. Mais celui-ci est toujours fortement impliqué dans l'implémentation de la logique de réplication, ce qui alourdit sa tâche et entraîne la création d'erreurs.

3 La spécification Fault Tolerant CORBA

De la même manière que le système CORBA permet de gérer une application distribuée, FT-CORBA permet d'éviter au développeur d'implémenter la tolérance aux fautes lors du développement de son application.

La spécification FT-CORBA se base sur trois mécanismes, à savoir la redondance, la détection des fautes et la récupération des composants défaillants. Un objet tolérant aux fautes possède plusieurs répliques qui sont gérées par l'infrastructure d'une façon transparente pour l'application. De plus, le système a comme exigence de n'accepter aucun endroit où une faute unique pourrait mettre en défaillance un des mécanismes.

[1] Par exemple Totem (Eternal)[5], Newtop[6], JGroup (IRL)[7].

3.1 Identification et gestion des répliques

Pour créer une couche d'abstraction autour de l'ensemble des répliques d'un objet, ceux-ci possèdent une même référence IOR (Interoperable Object Reference) particulière, appelée IOGR (Interoperable Object Group References). Ce système permet au client de ne pas être au courant ni de la réplication des objets, ni des fautes que ceux-ci peuvent commettre. De cette manière, la logique de réplication est totalement dissimulée pour l'utilisateur de l'objet. Cela permet aussi d'effectuer un minimum de changements de l'ORB client désirant bénéficier de la tolérance aux fautes. Techniquement, l'IOGR est un IOR composé de plusieurs profils, chacun d'eux pointant sur une réplique dans le cas d'une réplication passive et sans état, ou sur un ou plusieurs portails organisant les accès aux répliques d'un objet dans le cas d'une réplication active.

A chaque objet répliqué est attaché un ensemble de propriétés tels que son style de réplication, le nombre initial de répliques, le nombre minimum de répliques possibles, etc... Ces propriétés peuvent être définies à la création ou dynamiquement après la création.

3.2 Mécanismes de tolérance aux fautes

Un ORB client doit fournir au moins deux services : la réinvocation transparente et la redirection transparente. Lorsque le client désire émettre une requête au serveur, l'ORB client utilise un des profils de l'IOGR. Si l'opération échoue, le mécanisme de redirection permet de sélectionner un autre profil de l'IOGR et de réessayer l'invocation.

De plus, le mécanisme de redirection est utilisé pour diffuser les modifications sur l'appartenance des répliques à un objet répliqué. Lorsque l'ORB client émet une requête, celui-ci inclut la version de l'IOGR qu'il utilise, de telle manière que l'ORB serveur puisse détecter si cette version est obsolète. Si la version ne correspond plus, l'IOGR du client est alors mis à jour.

La réinvocation transparente permet de ne pas invoquer un objet plus d'une fois. En archivant les invocations et leurs réponses[2], un client peut donc réobtenir les informations d'une requête sans risquer d'exécuter deux fois une même méthode. Pour éviter que la journalisation des invocations soit trop importante, un temps limité de conservation est ajouté aux éléments du journal.

Du côté serveur, la spécification FT-CORBA étend le standard CORBA avec de nouveaux mécanismes divisés en trois catégories : la gestion de la réplication, la gestion des fautes et la gestion de la récupération.

3.3 Gestion de la réplication

La gestion de la réplication est effectuée par le gestionnaire de réplication. Celui-ci s'occupe de la création, de la suppression et de la gestion des répliques d'un objet. Pour créer un objet, le gestionnaire s'appuie sur des fabriques locales créées par le programmeur de l'objet et déployées sur chaque hôte.

[2] L'archivage des requêtes et des réponses est réalisé côté serveur

Lors de la création d'un objet répliqué, le gestionnaire crée les différentes répliques, collecte leurs IOR et retourne l'IOGR correspondant. Le gestionnaire peut être responsable, par exemple, de maintenir un minimum de répliques en vie. Il permet aussi un certain nombre d'options, tels le type de réplication ou encore la possibilité de laisser soit l'application, soit l'infrastructure contrôler les répliques et/ou leur cohérence.

3.4 Gestion des fautes

La gestion des fautes concerne la détection du crash d'une réplique, la création, la notification et l'analyse de rapports. Ces activités sont respectivement réalisées par le détecteur de fautes, le notifieur de fautes et l'analyseur de fautes. Le détecteur surveille chaque réplique demandée grâce à leur interface *PullMonitorable*. Le notifieur, informé par les détecteurs, s'occupe de transmettre un rapport de fautes au gestionnaire de réplication ainsi qu'à tous les composants ayant souscrit au service. Il est à noter que l'analyseur de fautes est un élément optionnel dans la spécification.

3.5 Gestion de la récupération

Dans le cas d'une cohérence des répliques gérée par l'infrastructure, la gestion de la récupération est basée sur deux mécanismes : la journalisation et la récupération. Ceux-ci sont possibles grâce à l'utilisation de deux interfaces, *Checkpointable* et *Updateable*, qui permettent de lire et d'écrire l'état des répliques. La journalisation enregistre périodiquement l'état, les requêtes et les réponses des composants. Ces informations sont utilisées par la récupération lors de la création d'un nouveau composant.

3.6 Répartition des composants FT-CORBA

La spécification FT-CORBA introduit la notion de domaine tolérant aux fautes (FT-domaine). Un FT-domaine est composé de plusieurs hôtes CORBA interconnectés. Chaque réplique d'un objet est hébergée sur des hôtes distincts d'un même FT-domaine, un hôte pouvant supporter plusieurs FT-domaines en même temps.

Une seule instance logique du gestionnaire de réplication et du notifieur de fautes doit être créée sur un FT-domaine. En effet, pour que le système entier soit tolérant, chaque composant gérant la logique de réplication doit également être répliqué.

Enfin, la notion d'FT-infrastructure reprend tout les mécanismes et interfaces offerts par une architecture FT-CORBA.

4 Limitations de Fault-Tolerant CORBA

FT-CROBA souffre de plusieurs limitations [1]. Cependant, celles-ci peuvent être surmontées par une implémentation spécifique d'un vendeur.

Vendeur unique Tout les hôtes d'un même FT-domaine doivent utiliser l'ORB et l'FT-infrastructure d'un même vendeur, ceci pour assurer l'interopérabilité et la complète tolérance aux fautes au sein d'un même domaine.

Compatibilité entre ORBs Un ORB non FT-CORBA ne peut supporter la tolérance aux fautes offerte par un ORB serveur FT-CORBA.

Comportement déterministe Dans le cas d'une logique de réplication contrôlée par l'infrastructure, c'est à dire lorsque la réplication n'est pas gérée par l'application, les objets de l'application et de l'ORB doivent avoir un comportement déterministe, aussi bien dans le cas de la réplication active que passive.

Séparation du réseau Le système est dépendant de la résistance de l'infrastructure de communication et ne peut fonctionner correctement si un sous-ensemble d'hôtes n'est plus accessible en raison de fautes de communication.

Faute malicieuse Dans le cas d'une réponse erronée n'entraînant pas de crash, seule la réplication active avec vote permet de surmonter l'anomalie. Cependant, ce type de mécanisme est très lourd en terme de volume de communications.

Erreur de conception Aucune protection n'est prévue dans le cas de fautes dues à des erreurs de programmation ou de design, ou toutes autres causes affectant l'ensemble des répliques d'un objet, de l'ensemble de l'ORB ou encore de l'ensemble des systèmes d'exploitation des hôtes.

5 Présentation d'IRL

L'Interoperable Replication Logic (IRL)[7,8,9,10] est une architecture conforme à la spécification FT-CORBA. L'idée principale d'IRL est de permettre l'utilisation des répliques d'un objet sur des sites très éloignés grâce à l'utilisation d'une architecture trois tiers (3T).

Réaliser une cohérence forte des répliques sur un grand système de communication asynchrone (Internet par exemple) n'est pas facile. Un système de communication est dit asynchrone s'il n'existe pas de contrainte temporelle forte sur l'échange des informations. L'idée d'une architecture 3T est de rassembler les éléments cruciaux, c'est à dire, dans notre cas, les éléments gérant la logique de réplication, au sein d'une structure de communication synchrone (Un LAN par exemple). Donc, seuls les composants ayant besoin d'une forte cohésion pour fonctionner efficacement sont répartis sur un réseau de communication synchrone.

L'architecture IRL répartit les éléments en trois parties. De manière conceptuelle, le premier tiers encapsule les différents clients, le tiers du milieu englobe la logique de réplication et le dernier tiers reprend les répliques côté serveur.

L'*IRL Object Group Handler* (OGH) est un nouvel élément par rapport à la spécification FT-CORBA. Un OGH existe pour chaque objet répliqué avec état. Celui-ci enregistre les références IOR de chaque réplique ainsi que les informations sur leur style de réplication. Il joue le rôle de portail du tiers du milieu dans

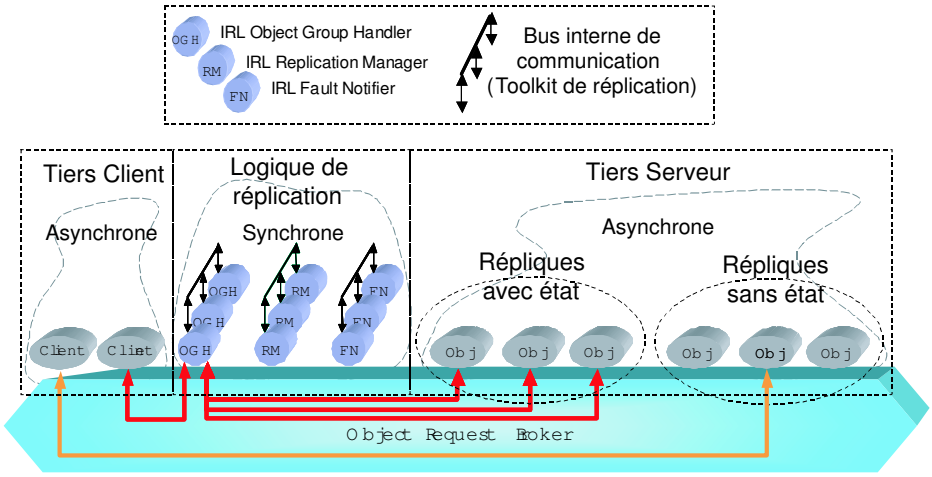

Fig. 1. L'architecture trois tiers d'IRL - Source fig. [7]

l'architecture 3T et il est donc l'intermédiaire entre les clients et les répliques, de manière à garder la cohérence des états des répliques. Pour réaliser cela dans le cas d'une réplication active des répliques, l'OGH reçoit toutes les requêtes adressées aux répliques d'un objet, impose un ordre de passage, les transmet ensuite une à une à toutes les répliques, collecte les réponses et finalement retourne l'une d'elles.

L'*IRL Replication Manager* correspond au gestionnaire de réplication FT-CORBA. Dans le cas d'un contrôle de la cohérence par l'infrastructure, le gestionnaire de la réplication d'IRL est chargé de créer un OGH répliqué et de retourner au client un IOGR correspondant à ces répliques d'OGH. Dans le cas d'objet sans état, l'IOGR renvoyé correspond directement aux répliques de l'objet (cfr. fig. 1).

L'*IRL Fault Notifier* correspond au notifieur de fautes FT-CORBA. Celui-ci utilise les *IRL Local Failures Detectors* comme détecteurs de fautes FT-CORBA. Le notifieur de fautes d'IRL vérifie la bonne santé des détecteurs par l'échange d'un message. Si l'un de ces derniers ne répond pas après un certain temps prédéterminé, le notifieur transmet un rapport de fautes au gestionnaire de la réplication ainsi qu'à tous ses autres souscripteurs.

L'*IRL Factory*, deuxième élément nouveau de l'architecture IRL, a comme tâche de créer, sur son hôte, les composants OGH, les détecteurs de fautes, le gestionnaire de la réplication et le notifieur de fautes si nécessaire. Le lancement de l'architecture IRL est alors effectué par la création d'une *IRL Factory* sur chaque hôte du FT-domaine. Celle-ci assure sa survie en étant répliquée en deux exemplaires sans état. Ces deux répliques s'échangent un message "Je suis en vie" et peuvent recréer leur partenaire en cas de faute.

Conformément à la spécification FT-CORBA, les composants d'IRL doivent être eux-mêmes répliqués. Le mécanisme de réplication des éléments du tiers

du milieu est réalisé grâce à une logique de réplication utilisant un bus interne de communication basé sur le protocole TCP/IP. En pratique, cette logique de réplication est sous-traitée à un toolkit de réplication (JGroup [7]).

Un prototype d'IRL a été réalisé par l'équipe à l'origine de l'architecture. Dans ce prototype, les composants peuvent être séparés en deux groupes : ceux propres à un hôte, c'est à dire l'*IRL Local Failures Detectors* et l'*IRL Factory*, et ceux propres au FT-domaine, c'est à dire l'OGH de chaque objet, l'*IRL Fault Notifier* et l'*IRL Replication Manager*. Ces derniers sont regroupés dans le tier synchrone du milieu de l'architecture 3T.

Enfin, le prototype utilise, côté client, un intercepteur CORBA, l'*IRL Object Request Gateway* (ORGW), permettant d'ajouter les mécanismes de redirection et de réinvocation aux clients ORB non-FT-CORBA. Du côté serveur, chaque réplique d'un objet est cachée derrière un *IRL Incoming Request Gateway Component* (IRGW) qui utilise les mêmes interfaces que l'objet et reçoit toutes les requêtes qui lui sont destinées. L'IRGW réalise cela en utilisant le *Dynamic Skeleton Interface* et l'*Interface Repository* de CORBA.

6 Présentation d'Eternal

Le système Eternal [11,6,12] a contribué activement à l'élaboration de la spécification FT-CORBA par l'OMG et fut la première architecture commerciale répondant à la norme. Nous présentons donc dans cette section la nouvelle architecture Eternal [6,12] répondant à la norme FT-CORBA.

Eternal est un système conçu pour offrir plus que la tolérance aux fautes. En effet, il offre aussi la possibilité d'utiliser un service de répartition de la charge et un service de mise à jour transparente des composants d'une application CORBA.

Ce dernier service est très intéressant car il utilise les mécanismes de réplication pour permettre une mise à jour automatique de l'application pendant qu'elle continue à fonctionner normalement [11].

Les éléments d'Eternal sont de deux types. Les éléments du premier type, à savoir le gestionnaire de la réplication, le détecteur de fautes et le notifieur de fautes, sont des objets CORBA qui peuvent être à l'occasion des objets répliqués. Les éléments du second type, non-CORBA, sont placés sur chaque hôte du FT-domaine entre l'ORB et la couche TCP/IP (cfr. fig. 2).

Le gestionnaire de la réplication d'Eternal répartit la gestion de ses tâches en trois composants : le gestionnaire de propriétés, la fabrique générique et le gestionnaire d'objets répliqués. Il est à noter que les services de ces trois composants n'ont pas de particularités propres intéressantes par rapport à la spécification FT-CORBA. Il en est de même pour le détecteur et le notifieur de fautes.

La particularité d'Eternal se trouve dans l'implémentation des éléments du second type. Ceux-ci utilisent un intercepteur sur chaque hôte pour intercepter les messages IIOP de l'application destinés à la couche TCP/IP. Chaque intercepteur est réalisé avec les librairies propres du système d'exploitation de l'hôte.

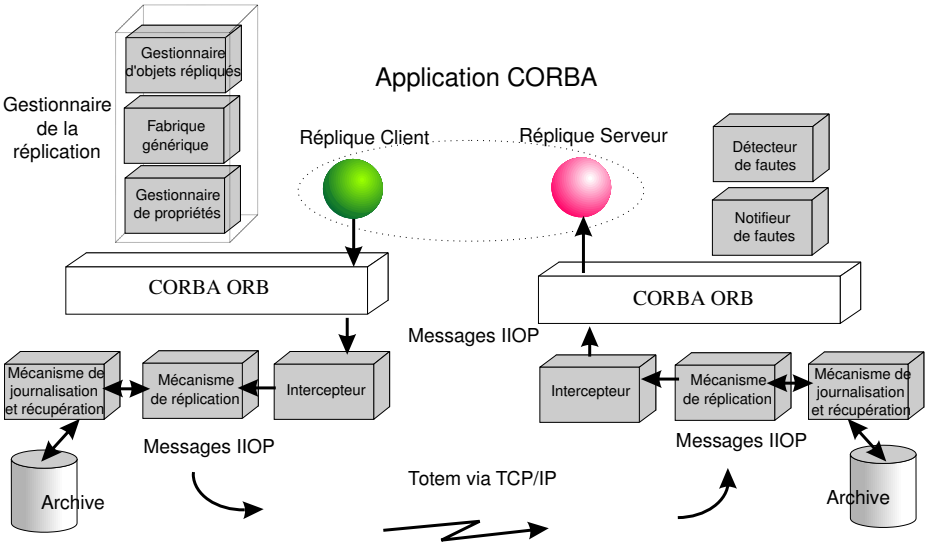

Fig. 2. Placement des différents mécanismes de l'architecture Eternal. Ceux-ci sont situés au-dessus et en-dessous de l'ORB de CORBA - Source fig. [6]

Les messages interceptés sont redirigés vers le mécanisme de réplication d'Eternal (*Eternal Replication Mechanisms*) permettant de réaliser la logique de réplication à travers la diffusion multicast de messages IIOP réinjectés dans la couche TCP/IP . Le mécanisme de réplication d'Eternal est, en réalité, soustraite au système Totem [5] qui gère la logique de réplication de manière cohérente. Totem est un toolkit de réplication utilisé lors du développement d'une application distribuée. Il permet d'intégrer directement à l'application un système de tolérance aux fautes.

Afin d'implémenter simplement la logique de réplication et de ne pas répéter deux fois une même invocation, un identifiant unique est ajouté à chaque invocation d'un objet répliqué. Ce mécanisme n'est possible que si les objets ont un comportement déterministe.

Prenons l'exemple d'un objet client répliqué invoquant une méthode sur un objet serveur lui aussi répliqué. Étant donné que les requêtes des différentes répliques du client sont identiques et interviennent dans le même ordre (ie. déterministe), le système FT-CORBA peut numéroter d'une manière identique toutes les requêtes. Dans le cas d'une réplication active, la première requête arrivant à un hôte hébergeant une réplique de l'objet serveur est transmise à l'objet. Étant donné que toutes les requêtes arrivant sur chaque hôte sont journalisées, les autres requêtes provenant des autres répliques de l'objet client seront ignorées. Pour les réponses, le principe est appliqué en sens inverse. Nous voyons donc bien que ce mécanisme n'est possible que si une numérotation identique des requêtes existe et donc que si les objets ont un comportement déterministe.

Le mécanisme de récupération d'Eternal utilise l'*Eternal Recovery Mechanisms* pour journaliser et assurer la sauvegarde de l'état des objets répliqués. Celui-ci distingue trois types d'état pour un élément : l'etat applicatif, l'état ORB de l'objet propre à l'ORB du vendeur et l'état infrastructurel maintenu par Eternal. L'état applicatif contient les valeurs des données conçues par le développeur.

Enfin, un composant spécifique d'Eternal, le planificateur [6,12], permet de préserver la cohérence d'un objet répliqué possédant plusieurs sous-processus (*multithread*) en le rendant déterministe. Les planificateurs d'Eternal réalisent cela en contrôlant la gestion des sous-processus de manière à reproduire le même comportement sur chaque réplique. Les planificateurs, répartis sur chaque hôte, exécutent les mêmes actions de manière déterministe. Un planificateur contrôle la création, l'activation, la désactivation et la destruction des sous-processus.

7 Discussion sur les architectures

L'architecture IRL permet de surmonter deux limitations de la spécification FT-CORBA. Premièrement, elle permet de fonctionner sur des ORB non-FT-CORBA non modifiés grâce notamment à l'utilisation d'intercepteurs et de composants purement CORBA. Deuxièmement, l'architecture est interopérable, c'est-à-dire que l'FT-infrastructure peut être répartie sur des systèmes CORBA de différents vendeurs tant que ceux-ci respectent le standard ORB utilisant le protocole IIOP.

L'architecture a également l'avantage de ne pas surcharger le client grâce à l'interception et à la légèreté des mécanismes utilisés. De plus, les composants utilisés côté réplique sont aussi très légers.

IRL est une architecture trois tiers permettant d'obtenir une cohérence forte des répliques physiquement installées sur des sites très éloignés. Cependant, le regroupement de la logique de réplication peut entraîner, dans un système tolérant aux fautes, l'apparition d'un point unique susceptible de bloquer le système entier en cas de faute. En effet, la majorité des réseaux du type LAN en entreprise n'ont malheureusement qu'une seule passerelle vers l'extérieur. Un minimum de redondance dans l'infrastructure de communication est donc obligatoire au niveau du tiers du milieu.

Le système Eternal est indépendant de l'ORB et permet de lever les mêmes limitations qu'IRL. Par contre, en raison de son système d'interception des messages du protocole IIOP, Eternal n'est pas indépendant du système d'exploitation.

Eternal a la particularité de permettre la réplication d'objets non-déterministes grâce à son intervention dans la planification des sous-processus. De plus, un des plus grands avantages d'Eternal sur les autres architectures est qu'il permet de répliquer les objets, qu'ils soient clients ou bien serveur. En effet, l'architecture IRL ne supporte pas qu'un objet répliqué invoque un autre objet répliqué. Enfin, l'architecture Eternal est pour le moment la seule à implémenter toutes les fonctionnalités de la spécification FT-CORBA.

Beaucoup d'autres architectures ont été développées pour offrir la tolérance aux fautes aux objets CORBA mais peu respectent déjà la norme FT-CORBA. Seuls Eternal, IRL et DOORS [13] sont connus pour leur compatibilité à la spécification. Les stratégies d'implémentation de ces architectures sont réparties en trois types : Intégration, Interception et Service [7].

La stratégie d'intégration (ex.AQuA [14]) répartit les mécanismes de tolérance aux fautes directement dans l'ORB et en est donc dépendant. Eternal fait partie de la stratégie d'interception en raison de son intercepteur de messages IIOP. IRL et DOORS implémentent une stratégie de service en déployant pour le développeur les interfaces et objet CORBA permettant la tolérance aux fautes.

8 Conclusion

Dans ce document, nous avons introduit les notions du principe de tolérance aux fautes pour rendre des services hautement disponibles. Les mécanismes de la spécification Fault Tolerant CORBA ont été introduits et deux stratégies d'implémentation de celle-ci, IRL et Eternal, ont été détaillées. Enfin, les avantages et les inconvénients de ces deux architectures ont été exposés.

Tout porte à croire que dans quelques années, étant donné la simplicité d'installation et d'utilisation des systèmes de tolérance aux fautes, le service de tolérance aux fautes FT-CORBA sera devenu aussi commun que l'utilisation de services de sécurité pour CORBA. De plus, il est aussi fort probable qu'une majorité des architectures utilisées pour ajouter la tolérance aux fautes aux objets CORBA respecteront la norme Fault Tolerant CORBA en raison de sa standardisation.

Références

1. OMG : Object management group : The common object request broker architecture and specifications. version 3.0.2. OMG Document formal edition - formal/02-12-02 (December 2002)
2. IRL : (Site internet du projet) http ://www.dis.uniroma1.it/˜irl.
3. Eternal : (Site internet d'eternal inc.) http ://www.eternal-systems.com.
4. OMG : (Tutoriel corba) http ://www.omg.org/gettingstarted/orb_basics.htm et http ://www.omg.org/gettingstarted/corbafaq.htm.
5. Moser, L., Melliar-Smith, P., Agarwal, D., Budhia, R., Lingley-Papadopoulos, C. : Totem : A fault-tolerant multicast group communication system. Communication of the ACM **39(4)** (1996) 54–63
6. Moser, L., Melliar-Smith, P., Narsimhan, P. : Eternal - a component-based framework for transparent fault-tolerant corba. Software - Practice and Experience **32** (2002) 771–788
7. Baldoni, R., Marchetti, C. : Three-tier replication for ft-corba infrastructures. Software - Practice and Experience **33** (2003) 767–797

8. Marchetti, C., Baldoni, R., A.Virgillito, F.Zito : Failure management for ft-corba applications. In : Prooceedings of the 6th IEEE International Workshop on Object Oriented Real-time Dependable Systems (WORDS'01). (2001)
9. Baldoni, R., Marchetti, C. : Three-tier replication for ft-corba infrastructures. Technical report, (Dipartimento di Informatica e Sistemistica - Université de Rome 1 La Sapienza)
10. Baldoni, R., C.Marchetti, R.Panellaa, L.Verde : (Handling ft-corba compliant interoperable object group references)
11. Moser, L., Melliar-Smith, P., Narsimhan, P., Tewksbury, L., Kalogeraki, V. : The eternal system : An architecture for enterprise applications. In : Proceedings 3rd Internationnal Enterprise Distributed Object Computing Conference (EDOC'99), IEEE Computer Society Press (1999) 214–222
12. Moser, L., Melliar-Smith, P., Narsimhan, P. : (Strong consistent replication and recovery of fault-tolerant corba applications)
13. Natarajan, B., Schmidt, D., Aniruddha Gokhale, S.Y. : Doors : Towards high-performance fault tolerant corba. In : Proceedings 3International Symposium on Distributed Objects and Applications, IEEE Computer Society : Los Alamitos (2000) 39–48
14. Ren, Y. : AQUA : A Framework for Providing Adaptive Fault Tolerance to Distributed Applications. PhD thesis, University of Illinois (2001)