Article

Vers une tolérance aux fautes standard, interopérable et transparente

Mohamed Taha BENNANI, Unité de recherche MOSIC, Ecole Nationale d'Ingénieurs de Tunis, BP 37, Le Belvédère 1002, Tunis, Taha.Bennani@enit.rnu.tn.

Date de publication : 30 décembre 2006

Résumé

Contrairement à l'aspect fonctionnel qui représente le service rendu par une application, les mécanismes de tolérance aux fautes, comme la réplication, représentent un aspect non-fonctionnel. Découpler ces deux aspects permettrait d'avoir une meilleure productivité. La réflexivité, jusqu'alors non standard, est un moyen puissant de réaliser un découplage entre les aspects applicatifs et les aspects non-fonctionnels orthogonaux d'un système. Dans ce cadre, nous nous focaliserons sur les applications distribuées utilisant l'intergiciel CORBA pour identifier un potentiel réflexif standard, à savoir les intercepteurs de requêtes. Nous présenterons notre plateforme à composants COTS, Component-Off-The-Shelf, de tolérance aux fautes DAISY, *Dependable Adaptative Interceptors and Serialization-based sYstem*, qui fournit des mécanismes de réplication transparents et découplés des applications distribuées. Après l'identification des deux approches de découplage, nous terminons par la discussion des limites conceptuelles des intercepteurs CORBA dans chacune d'entre elles.

Abstract

Unlike the functional aspect which represents the service provided by the application, the fault tolerance, like the replication, represents a non-functional aspect. Separating these two aspects improve the applications design and maintenance. Reflection is a powerful means to separate these cross-cutting concerns. We will focus on the standard reflexive potential provided by the CORBA middleware to know if the portable request interceptors could handle fault tolerance issues. We will present our fault tolerant and components based platform, namely DAISY (Dependable Adaptative Interceptors and Serialization-based sYstem) which provides replications mechanisms in a transparent and lightweight way. Two approaches of cross-cutting separation will be identified and, finally, the conceptual limits of CORBA portable request interceptors are discussed.

Table des matières

Texte intégral

1. INTRODUCTION

Plusieurs projets ont eu pour objet de munir les intergiciels implémentant la norme CORBA standard, de mécanismes de tolérance aux fautes, à savoir l'utilisation d'un système de communication de groupe, d'algorithmes de consensus ou de politiques de réplication. L'objectif était de définir une plate-forme qui serait utilisée d'une façon standard pour le développement d'applications distribuées tolérant les fautes. Ces premiers travaux étaient confrontés à deux dilemmes :

- La « spécificité » par rapport à la « portabilité » ; en effet, des projets tels que Eternal (Moser, Melliar-Smith et al., 1999) (Narasimhan, Moser et al., 2001) ont essayé d'exploiter la spécificité d'un système d'exploitation particulier alors que d'autres projets tels que ORBIX+ISIS (IONA, 1994) ont tenté de coupler un intergiciel à un système de communication de groupe, ce qui nuit à la portabilité de la solution proposée.

- Le choix de la « transparence » par rapport à « l'interopérabilité » ; en effet, certains travaux, à l'instar d'Electra (Maffeis, 1995), avaient pour objectif de fournir les mécanismes de tolérance aux fautes d'une manière transparente pour les applications ; cette transparence nécessite une certaine intrusion (c'est-à-dire, une modification de l'implémentation) au niveau de l'intergiciel. Ceci limite l'interopérabilité requise par la norme CORBA entre les ORB standard et ceux qui sont tolérants aux fautes. En revanche, les approches inter-opérables, comme OGS (Felber, Guerraoui et al., 1998), impliquent les développeurs des applications dans la mise en œuvre de la tolérance aux fautes, ce qui fragilise la mise en œuvre et rend l'approche non transparente.

Suite à ces travaux et à l'intérêt croissant d'une approche standardisée, l'OMG a proposé une nouvelle norme FT-CORBA (OMG 2002a) définissant un ensemble de services assurant la gestion de la réplication des services et un nouvel identificateur des références d'un groupe d'objets. Ces modifications rendent les applications légataires inappropriées.

Un système réflexif (Smith, 1982) (Maes, 1987) (Taiani, 2004) est défini par deux niveaux : un « niveau de base » qui représente le service que doit rendre l'application, et un « méta-niveau » qui représente des fonctionnalités qui lui sont orthogonales telles que la tolérance aux fautes. Les interactions entre ces deux niveaux sont régies par la réification, l'introspection, l'intercession comportementale et l'intercession structurelle. La réification assure la notification au méta-niveau des évènements comportementaux et structurels du niveau de base. L'observation de la structure du niveau de base par le méta-niveau est fournie par l'introspection. L'intercession structurelle et l'intercession comportementale assurent, respectivement, le changement de la structure et du comportement du niveau de base par le méta-niveau.

Parallèlement aux travaux classiques, évoqués plus haut, plusieurs travaux ont essayé de tirer profit de la réflexivité pour fournir des systèmes tolérant les fautes en séparant les aspects fonctionnel et non-fonctionnel ; ceci assure la transparence de la mise en œuvre de l'utilisation des mécanismes de tolérance aux fautes. Dans DynamicTAO (Roman, Kon et al., 1999) et OpenORB (Clarke, Blair et al., 2001) (Saikoski, Coulson et al., 2003) la réflexivité est utilisée pour assurer la tolérance aux fautes de l'intergiciel de communication, alors que dans FRIENDS (Fabre et Pérennou, 1998), la réflexivité est utilisée pour assurer la tolérance aux fautes des applications réparties. Dans l'ensemble de ces travaux, la réflexivité est introduite d'une manière

propriétaire et non standard.

La première partie de cet article est consacrée à la description des différents mécanismes réflexifs, spécifiques et standards au niveau des intergiciels. Nous mettrons en particulier l'accent sur les mécanismes standards qui seront utilisés pour le développement de la plate-forme DAISY (Bennani, 2005). L'illustration des différents composants de cette plate-forme, destinée à munir des applications distribuées CORBA de mécanismes de réplication (i.e. passive, semi-active et active), fera l'objet de la seconde partie. Dans la troisième partie, nous présenterons deux approches différentes pour la mise en œuvre de la plate-forme DAISY. Ces deux approches se distinguent par l'emplacement des composants de tolérance aux fautes par rapport au support d'exécution, à savoir l'intergiciel. Nous terminons cette partie par la comparaison de ces deux approches sous l'angle de leur adaptabilité par rapport aux applications et de la complexité de leur mise en œuvre. Dans la dernière partie, nous présenterons les limites conceptuelles, en termes architecturaux, des intercepteurs CORBA.

2. MÉCANISMES RÉFLEXIFS DANS CORBA

L'intérêt croissant que suscite la réflexivité a influencé, dans un premier temps, quelques industriels qui ont doté leur intergiciel de mécanismes réflexifs spécifiques (i.e. propriétaires), et dans un second temps, l'OMG qui propose de munir la norme CORBA de mécanismes réflexifs standards.

2.1. Mécanismes réflexifs spécifiques

Il existe trois mécanismes réflexifs propriétaires : les mandataires intelligents (Wang, Parameswaran et al., 2001), les filtres (IONA, 2000a) et les transformateurs de requêtes (IONA, 2000b). Les premiers réécrivent les mandataires standards générés par le compilateur IDL pour y introduire des mécanismes non-fonctionnels. Les filtres se déclinent en deux types (des filtres associés aux requêtes et d'autres associés aux objets) permettent d'ajouter des prétraitements ou des post-traitements lors de l'envoi des requêtes et de la réception des réponses. Les transformateurs de requêtes définis par l'intergiciel ORBIX ont pour objectif de modifier le tampon des données relatives à une requête `CORBA::Request` juste avant que l'invocation d'une opération ne soit activée au niveau du serveur, et que la réponse ne soit retournée au client.

2.2. Mécanismes réflexifs standards

L'OMG a défini deux types d'intercepteurs portables, l'intercepteur de références des objets CORBA (*IOR interceptor*) et l'intercepteur des requêtes (*request interceptor*). C'est ce second type d'intercepteurs que nous mettrons en avant dans cet article, puisque nous l'utiliserons afin d'ajouter les traitements nécessaires aux mécanismes de réplication d'une manière transparente.

Comme le montre la figure 1, ce type d'intercepteur peut être utilisé pour traiter les requêtes à la fois du côté du client, et du côté du serveur. Nous nommons ceux qui se trouvent du côté client PIC *Client Portable Request Interceptors* et ceux qui se trouvent du côté serveur PIS *Server Portable Request Interceptors*.

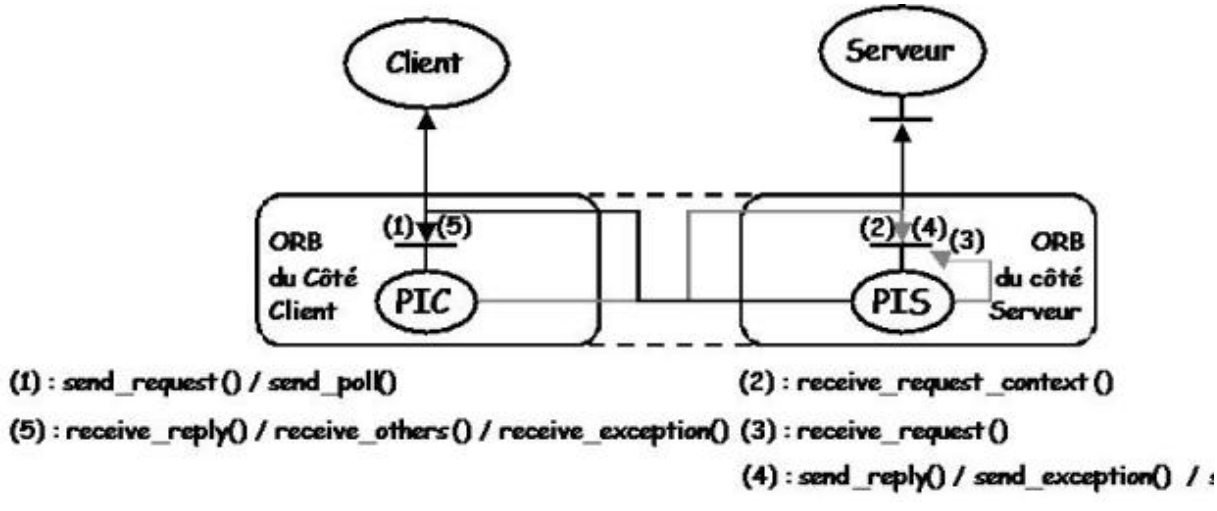

Figure 1. Interfaces des intercepteurs de requêtes PIC et PIS

Ce type d'intercepteur possède une interface fixe ; les différentes méthodes qui y sont définies représentent l'insertion des traitements à réaliser avant envoi et réception des requêtes et des réponses. Par ailleurs, l'activation de ces méthodes est assurée par l'ORB.

- Lorsque le client envoie une requête et que celle-ci est empaquetée dans un message GIOP, l'ORB appelle les méthodes `send_request` ou `send_poll` selon que la méthode est synchrone ou asynchrone (1).

- Une fois envoyée via le réseau, la requête arrive au niveau de l'ORB du côté serveur. Avant que la requête ne soit dépaquetée, l'ORB active la méthode `receive_request_context` (2). Lors de l'activation de cette méthode, les paramètres de la requête ne sont pas observables.

- Avant que la requête n'arrive au niveau du squelette du serveur cible, l'ORB active la méthode `receive_request` du PIS (3). À ce niveau, les paramètres de la requête sont observables (mais pas modifiables).

- Au retour de la réponse, et lorsqu'elle est transformée en un message GIOP, l'ORB active les méthodes `send_reply` et `send_others` selon que le service réalisé est synchrone ou asynchrone (4). L'ORB active la méthode `send_exception` lorsque le serveur génère une exception durant l'élaboration d'une réponse. Cette méthode est également activée dans le cas où un problème de communication surgirait lors du retour de la réponse et avant son interception par le PIS, ou même quand une des quatre méthodes antérieures de l'interface du PIS déclenche une exception.

- Une fois que la réponse arrive au site client et que le message IIOP est transformé en un message sous le format GIOP, l'ORB active les méthodes `receive_reply` ou `receive_others` si la réponse est bien arrivée, ou bien la méthode `receive_exception` si une exception a été soulevée pendant l'élaboration d'une réponse ou lors de la transmission du message (5).

Une fois activés, ces intercepteurs peuvent observer le contenu des messages qui transitent entre clients et serveurs (i.e. réification).

En outre, la norme CORBA a défini des méta-données standards qui peuvent être interprétées par les intercepteurs. Par exemple, du côté client, l'intercepteur PIC peut accéder aux données contenues dans la structure `ClientRequestInfo` et, du côté serveur, l'intercepteur PIS peut accéder aux données contenues dan la structure `ServerRequestInfo` (OMG 2002b). Ces méta-données contiennent des informations relatives au nom du serveur cible, à la politique de transmission de la requête, à la liste des paramètres, etc.

Par ailleurs, les intercepteurs peuvent rediriger une requête vers un serveur différent du serveur initial et ils peuvent inhiber l'invocation d'une requête (i.e. intercession comportementale). Par contre, ils ne peuvent pas observer directement l'objet auquel ils sont associés (i.e. ils ne fournissent pas le mécanisme d'introspection) et ne peuvent pas agir directement sur l'état d'un objet (i.e. ils ne fournissent pas le mécanisme d'intercession structurelle).

3. ARCHITECTURE DE DAISY

DAISY, Dependable Adaptative Interceptors and Serialization-based sYstem, est une plate-forme fournissant des mécanismes de tolérance aux fautes distribués sur le bus à objets CORBA. Cette plate-forme que nous avons conçue nous permettra d'analyser les limites des mécanismes réflexifs standards.

3.1. Spécification

La plate-forme DAISY(cf. figure 2) fournit les mécanismes de réplications classiques : réplication passive, réplication semi-active et réplication active. Ces mécanismes lui permettent d'étendre le comportement des applications client/serveur. Ainsi, pour initialiser l'application serveur, l'administrateur choisit un mode de réplication parmi les trois. Le client, pour sa part, initialise l'application cliente puis demande à travers cette application les services offerts par l'application serveur.

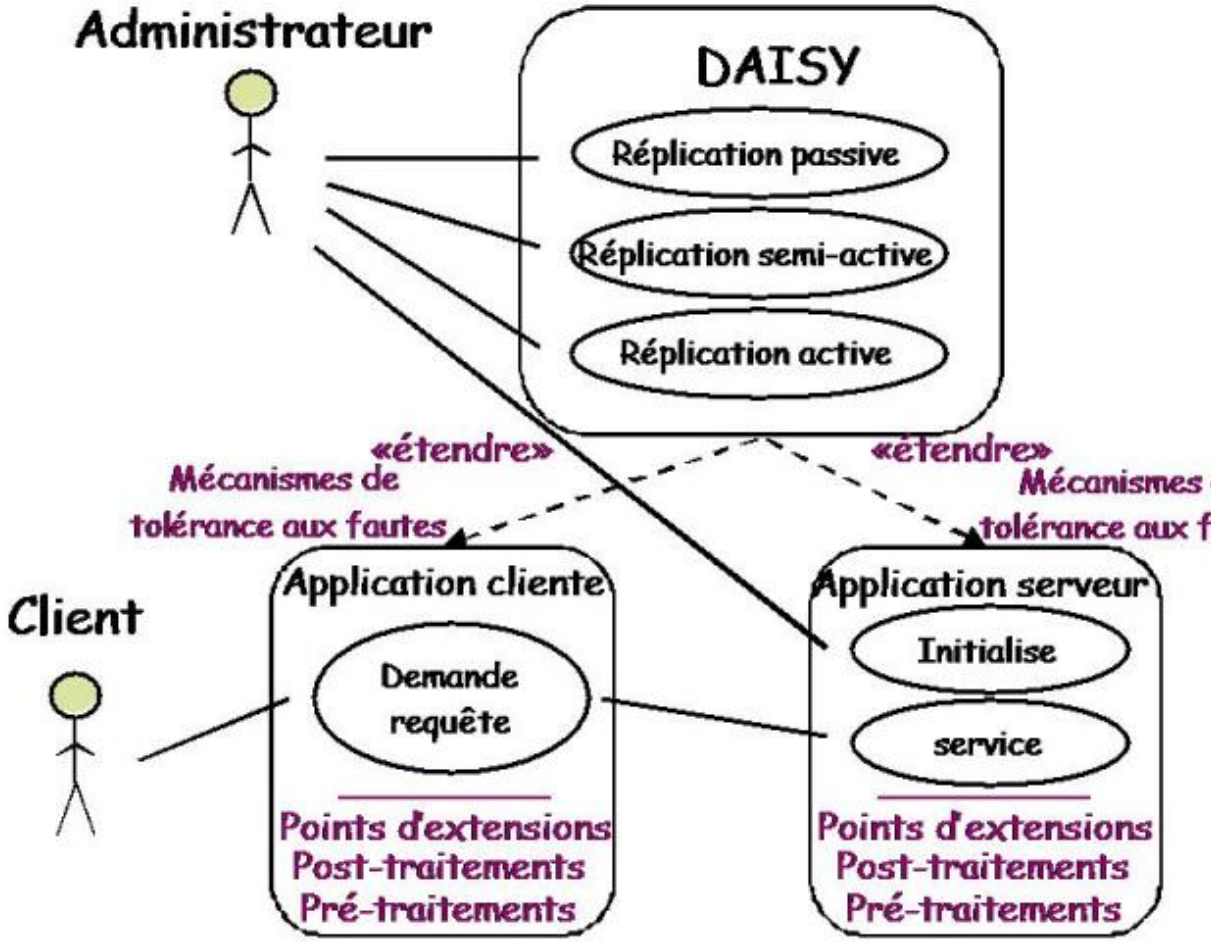

Figure 2. Diagramme de cas d'utilisation de DAISY

Les réplications passive et semi-active répondent aux mêmes modèles de fautes. Ces deux mécanismes ne tolèrent que les défaillances par arrêt d'un serveur. La réplication active présente un modèle de faute différent qui lui permet de tolérer aussi bien les défaillances par arrêt que les défaillances par valeur. La tolérance des fautes en valeur nécessite un vote sur les sorties des répliques ; cela implique que ces serveurs doivent être déterministes. Afin d'assurer ce déterminisme, la plate-forme garantit la diffusion atomique des requêtes et la gestion de l'appartenance d'un groupe de répliques.

3.2. Définitions (Principes)

La figure 3 montre l'architecture à composants de DAISY.

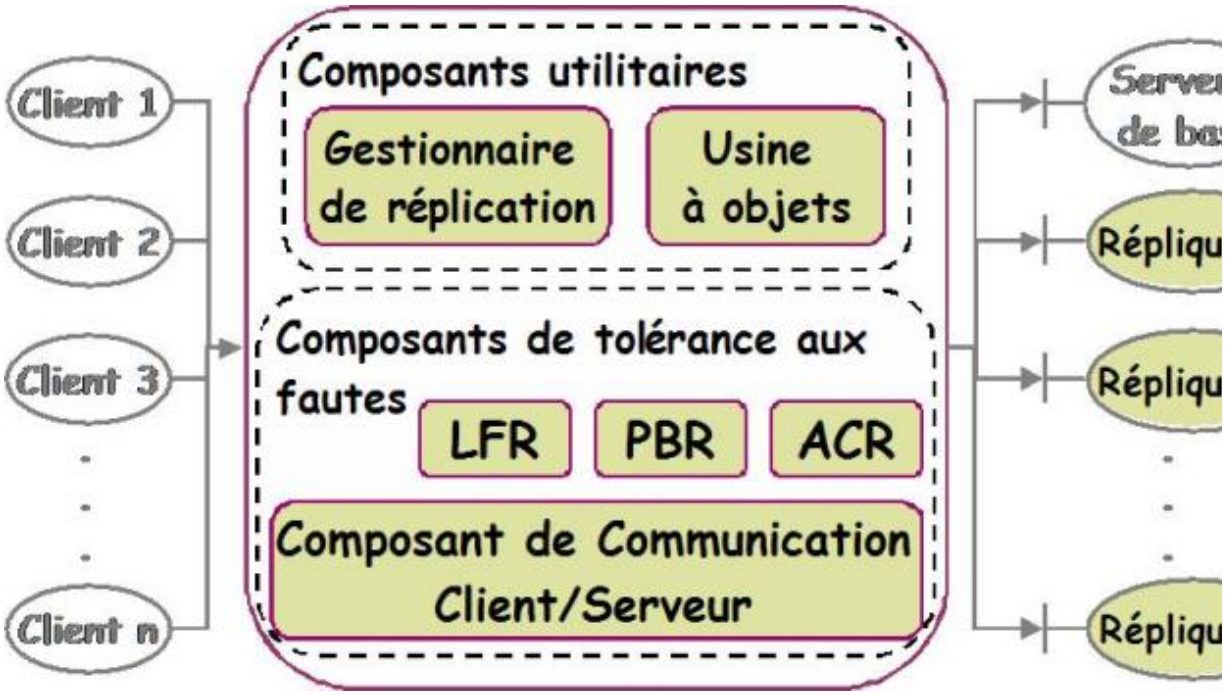

Figure 3. Vue Générale de l'architecture

L'ensemble de ces composants est formé de deux familles différentes. Les composants de tolérance aux fautes, assurant la mise en œuvre des mécanismes de réplication, et les composants utilitaires qui assurent le contrôle de la plate-forme.

L'utilisation d'une telle architecture (i.e. par composants) permet de séparer les différents types de mécanismes. L'activation d'un mode de réplication n'entraîne pas l'activation des composants assurant les deux autres modes. Ainsi, si une erreur s'est introduite dans le code d'un type de réplication, elle n'aura pas d'effet sur le code des deux autres. Ceci limite la propagation des erreurs entre les différents composants. Par ailleurs, cette décomposition se prête mieux à l'utilisation de composants génériques qui sont indépendants des applications auxquelles ils sont associés (i.e. changement de l'application qui est associée au serveur de base et les applications clientes qui l'utilisent).

3.3. Les composants de tolérance aux fautes

3.3.1. Composant de communication client/serveur

Le CC-C/S, composant de communication client/serveur, est propre à chaque client et a pour mission d'assurer la transparence de l'envoi des requêtes par rapport au client, l'identification de chaque requête sortante, la récupération des vues du groupe de répliques et la protection du client envers les défaillances des serveurs. Par ailleurs, il rend les mécanismes de tolérance aux fautes, fournies par les composants ACR, PBR et LFR, transparents au client puisqu'il assure le rôle de mandataire pour ces mécanismes.

Dans le cas d'un fonctionnement en absence de fautes, toute requête est identifiée d'une manière unique, avant d'être délivrée, pour faciliter son repérage par les autres composants de tolérance aux fautes. Si un serveur membre du groupe de réplique défaillirait par arrêt, le CC-C/S masquerait cette défaillance en essayant d'établir un nouveau lien de communication avec un autre serveur qui fonctionne.

3.3.2. Composants de réplication passive et semi-active

Les composants PBR et LFR possèdent chacun un ensemble de répliques qui se décline en deux types de serveurs : le serveur primaire et les serveurs répliques dans le cas de la réplication passive, et le serveur meneur et les serveurs suiveurs dans le cas de la réplication semi-active. La seule différence qui existe entre les deux composants est que le premier envoie un état, tandis que le second envoie des messages de synchronisation.

Dans le cas de la réplication passive, la requête envoyée par le composant de communication arrive au niveau du composant PBR associé au serveur primaire. Le composant PBR délivre la requête au serveur d'application, et au retour de la réponse, il récupère l'état du serveur primaire pour l'envoyer aux serveurs répliques pour qu'ils mettent à jour leurs états. Si le composant PBR est associé à un serveur réplique, il est alors chargé d'une part de la mise à jour de l'état du serveur répliqué, et d'autre part, de la transition vers un nouveau primaire lors de l'occurrence d'une défaillance par arrêt du serveur primaire.

Dans le contexte de la réplication semi-active, la requête diffusée par le composant de communication arrive au niveau des différents composants LFR. Ces composants sont associés d'une part au serveur meneur, et d'autre part, aux serveurs suiveurs. Dans le premier cas, le composant LFR envoie des messages de synchronisation pour imposer un ordre identique de traitement des requêtes au niveau des différents suiveurs. Dans le second cas, le composant LFR met en attente toutes les requêtes jusqu'à ce qu'il reçoive l'ordre de synchronisation. Par ailleurs, le LFR agit de la même façon que le PBR du côté réplique dans le sens où il change de comportement lorsque le serveur auquel il est associé se trouve élu comme nouveau meneur en cas de défaillance.

3.3.3. Composants de réplication active

Les composants ACR implémentent un protocole de réplication active. Ils assurent la diffusion des messages en respectant un ordre total. De plus, comme la réplication active tolère les défaillances par valeurs, ces composants assurent le vote pour pallier à de tels problèmes.

Lorsque l'application évolue en absence de fautes, nous utilisons un protocole d'ordre asymétrique (Ezhilchelvan, 1995) pour assurer un ordre total pour la délivrance des requêtes. Ce type de protocole utilise un membre du groupe, que nous appelons principal, pour séquencer et ordonner les messages. Ainsi, lorsque la requête arrive au niveau du composant ACR du serveur principal, il la diffuse aux différents ACR des serveurs répliques (subsidiaires). Ensuite, lors du retour des différentes réponses, le composant ACR du serveur principal assure le vote et retourne la réponse majoritaire au client.

Lorsqu'un serveur, principal ou subsidiaire, défaille en valeur, le composant ACR du serveur principal réinitialise son état à partir de l'état d'un autre serveur jugé correct.

Lorsqu'un serveur subsidiaire défaille, l'ACR du serveur principal se charge de sa détection puis notifie le gestionnaire de réplication de cette défaillance par arrêt. En revanche, si le serveur principal défaille, c'est le composant ACR d'un autre serveur subsidiaire qui le détecte, s'assure de cette défaillance et notifie par la suite le gestionnaire de réplication.

3.4. Les composants utilitaires

Nous distinguons deux composants utilitaires, l'usine à objets et le gestionnaire de réplication.

L'usine à objets permet d'automatiser la création de répliques. Cette automatisation a un double objectif. Le premier est de pallier la dégradation du niveau de tolérance aux fautes lors de l'occurrence d'une défaillance d'une réplique de l'application. Le second objectif est de masquer à l'administrateur du système l'insertion de la plate-forme lors du lancement d'une application. Ceci protège l'ensemble de l'application d'une mauvaise initialisation qui peut compromettre la mise en œuvre des mécanismes de tolérance aux fautes.

Le gestionnaire de réplication fournit une interface à l'administrateur pour pouvoir choisir le mode de réplication initial, les paramètres du modèle temporel du système, le nombre de répliques à déployer et les machines qui vont héberger les différentes répliques.

4. DIFFÉRENTES VARIANTES DE DAISY

Nous présentons dans cette partie deux approches différentes pour mettre en œuvre l'architecture DAISY. Dans la première approche, les composants de tolérance aux fautes sont présentés sous forme de services, c'est-à-dire à l'extérieur de l'ORB. En revanche, dans la seconde approche, ces composants sont intégrés dans l'intergiciel.

4.1. Approche à composants externes

4.1.1. Principe

La figure 4 montre comment nous utilisons l'intercepteur de requêtes CORBA pour mettre en œuvre une architecture à composants externes à l'ORB.

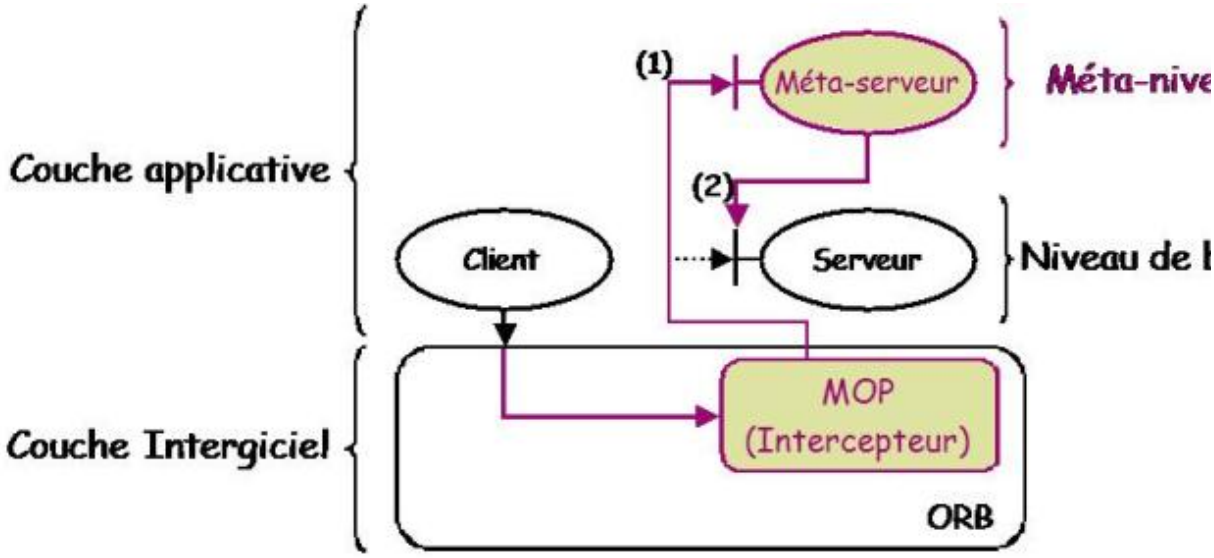

Figure 4. Protocole à méta-objets sur étagère (Intercepteur CORBA)

Dans cette première variante de DAISY, nous considérons l'intercepteur comme un support pour mettre en œuvre un protocole à méta-objets *MOP* à l'exécution (Killijian, 2002). Avec ce type de protocole, le lien entre serveur (i.e. Niveau de base) et le méta-serveur qui lui est associé (i.e. Méta-niveau) ainsi que les lois qui régissent la réification (1) et l'intercession (2) sont définies lors de l'exécution de l'application.

En effet, lors de l'initialisation de l'ORB, l'intercepteur de requêtes récupère l'IOR du méta-serveur pour initialiser le lien entre le niveau de base et le méta-niveau. Ce lien est, par la suite, activé chaque fois qu'une requête est interceptée.

Lors de l'interception d'une requête à destination du serveur, l'intercepteur peut choisir de la rediriger au méta-serveur en soulevant l'exception `ForwardRequest` (i.e. réification) ou de la transmettre au serveur destinataire. Par ailleurs, le méta-serveur peut aussi bien invoquer le serveur cible qu'élaborer une réponse et la retourner au client (i.e. intercession comportementale).

La différence entre les protocoles à méta-objets à l'exécution, tels que CLOS (Kiczales, 1991) ou OpenC++v1 (Chiba, 1993), par rapport à un MOP à base d'intercepteurs réside dans leur niveau de définition au sein d'un système. Les premiers sont définis au niveau du langage, c'est-à-dire dans la couche applicative, en revanche, le second est défini au niveau de l'intergiciel.

4.1.2. Prototype de la réplication passive

Nous présentons dans la figure 5 l'architecture de la variante à base de méta-objets externes de la plate-forme DAISY mettant en œuvre la réplication passive.

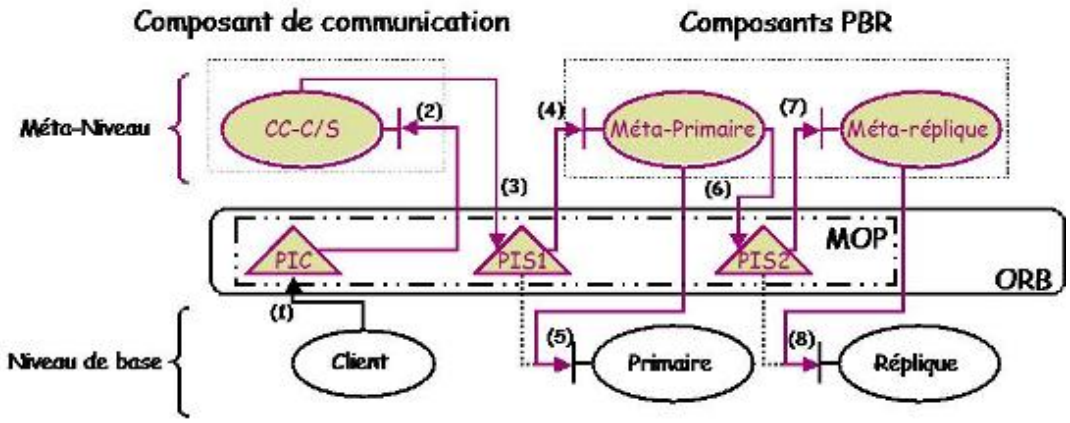

Figure 5. Réplication passive à méta-objets externes

Notre architecture est composée de deux niveaux : le méta niveau et le niveau de base. Le méta-niveau renferme les composants de tolérance aux fautes, à savoir les composants PBR et le *CC-C/S*. Ces composants, qui sont des méta-objets, sont aussi des objets CORBA identifiés d'une manière unique à travers un IOR. Le niveau de base est composé d'un client, d'un serveur primaire et d'un serveur réplique. En observant et en contrôlant le comportement et l'état des objets du niveau de base, les composants de tolérance aux fautes leur fournissent les mécanismes de la réplication passive.

Le lien entre le niveau de base et le méta-niveau est assuré par un protocole à méta-objet composé de trois intercepteurs PIC, PIS1 et PIS2.

Lors de l'initialisation des objets du niveau de base, l'intercepteur PIC récupère l'identificateur du composant de communication client/serveur pour le lier avec l'objet *client*. Ensuite, l'intercepteur PIS1 récupère l'identificateur du serveur *méta-primaire* pour le lier avec le serveur primaire. Enfin, l'intercepteur PIS2 assure l'association entre le méta-objet *méta-réplique* et l'objet *réplique*. L'ensemble de ces actions représente la phase d'initialisation du protocole à méta-objet (i.e. le lien entre le niveau de base et le méta-niveau).

Lors de l'exécution de l'application, toute requête issue du client est interceptée par le PIC (1) qui la réifie au CC-C/S (2). Ce dernier envoie ces requêtes au serveur primaire qui seront interceptées par le PIS1 (3). Celui-ci les réifie au serveur méta-primaire (4). Ce méta-objet traite les requêtes qui lui sont parvenues par l'intermédiaire du PIS1 puis les transmet au serveur *Primaire* (5). Par ailleurs, le méta-primaire récupère l'état du serveur primaire (6) puis élabore une requête de mise à jour de l'état du serveur réplique. L'intercepteur PIS2 réifie cette requête au serveur méta-réplique (9) qui la traite, puis assure la mise à jour du serveur réplique (10). Cette phase représente le mode de fonctionnement du protocole à méta-objet qui se résume :

1. en la réification de toute requête sortante d'un client vers le méta-objet qui lui est associé.
2. en la réification de toute requête à destination d'un serveur vers son méta-objet associé.
3. en la transmission de toute requête à destination d'un objet et provenant de son méta-objet. Cette transmission concerne les requêtes simples (i.e. intercession comportementale) la récupération de l'état (i.e. introspection) et la mise à jour de l'état (i.e. intercession structurelle).

C1 : Problème et incidence sur la conception de DAISY	L'intercepteur ne permet pas au méta-primaire de récupérer ou de contrôler la structure de l'objet de base ni la définition de son interface. C'est pour cette raison qu'une application doit hériter les méthodes d'observations et de contrôles.

L'existence des intercepteurs CORBA est liée aux objets auxquels ils sont associés lors de l'initialisation (i.e. serveur et client) ; l'arrêt d'un serveur provoque la destruction immédiate du lien qui le lie à son méta-serveur. Il en est de même pour le lien entre le client et le serveur CC-C/S. Nous soulignons qu'un méta-objet dont l'objet est détruit peut être lié à un nouvel objet; Par ailleurs, chaque intercepteur est capable de détecter la défaillance d'un méta-objet.

4.1.3. Prototype de la réplication active

Nous présentons dans la figure 6 l'architecture de la variante à base de méta-objets externes de la plate-forme DAISY, dans laquelle nous traitons la réplication active.

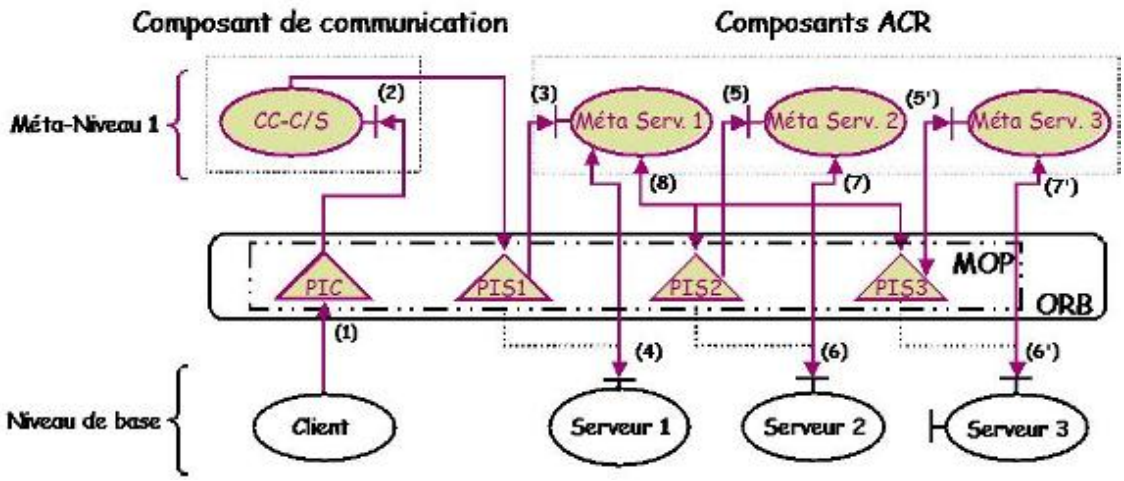

Figure 6. Réplication active à méta-objets externes

Comme pour la réplication passive, le méta-niveau renferme les composants de tolérance aux fautes, à savoir les composants ACR et le composant de communication client/serveur. Le niveau de base est composé d'un client et de trois serveurs d'applications qui forment le groupe de répliques.

Le lien entre le niveau de base et le méta-niveau est assuré par un protocole à méta-objet composé des intercepteurs PIC, PIS1, PIS2 et PIS3.

A l'initialisation, l'intercepteur PIC récupère l'identificateur du composant de communication client/serveur pour le lier avec l'objet client. Ensuite, chaque intercepteur assure l'association entre l'objet et le méta-objet qui l'observe et le contrôle, par exemple PIS1 lie l'objet Serveur 1 au méta-objet *Méta Serv.1*. Cette phase représente la phase d'initialisation du protocole à méta-objet.

Lors de l'exécution de l'application, toute requête issue du client est interceptée par le PIC (1) qui la réifie au *CC-C/S* (2). Celui-ci envoie ces requêtes au Serveur1 qui seront réifiées par le PIS1 au Méta Serv.1 (3). Ce méta-objet traite les requêtes qui lui sont parvenues puis les transmet à l'objet du niveau de base qui lui est associé Serveur 1 (4). De plus, le Méta Serv.1 diffuse aux autres serveurs du groupe de répliques toutes les requêtes qui lui parviennent. Ces requêtes sont réifiées par les intercepteurs PIS2 et PIS3 aux méta-objets Méta Serv.2 et Méta Serv.3 (5, 5'). Chaque méta-serveur invoque par la suite l'objet du niveau de base qui lui est associé (6, 6'). Après avoir traité les requêtes, les serveurs retournent leurs réponses au méta-objet Méta Serv.1 (8) en passant par leurs méta-objets respectifs (7, 7'). Enfin, le Méta Serv.1 réalise le vote et retourne la réponse majoritaire au client.

4.2. Approche à composants intégrés

4.2.1. Principe

La figure 7 illustre l'utilisation de l'intercepteur de requêtes CORBA pour mettre en œuvre une architecture à composants intégrés à l'ORB.

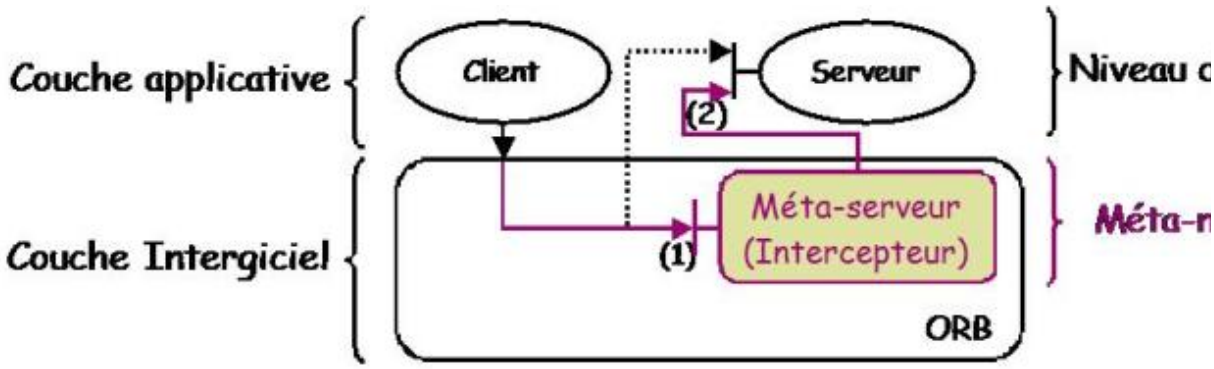

Figure 7. Méta-objet sur étagère (Intercepteur CORBA)

Dans cette seconde variante de DAISY, nous considérons l'intercepteur comme un support pour mettre en œuvre une stratégie de réplication de services (active ou passive).

En effet, toute requête émise par un client est réifiée par l'ORB à l'intercepteur de requêtes (1). Cette réification est assurée par l'activation de l'une des méthodes appartenant à son interface. Ces interfaces sont indépendantes de l'application. En d'autres termes, cette interface ne dépend pas des objets du niveau de base, ni de leurs interfaces ; elle n'est pas modifiable. L'intercepteur est un méta-objet particulier, dans le sens où il ne peut pas modifier le contenu des requêtes qui lui sont réifiées ; son action se limite à l'observation de l'information qu'elles véhiculent. Les seules actions autorisées par la norme CORBA aux intercepteurs sont relatives à l'intercession comportementale (2). Cette dernière se matérialise en la transmission de toute requête réifiée au serveur cible, ou bien en sa redirection vers un autre serveur équivalent, ce qui veut dire, vers un serveur qui possède la même interface que le serveur original.

C2 : Problème et incidence sur la conception de DAISY	*L'intercepteur possède une méta-interface non modifiable. C'est pour cette raison que les méthodes destinées à l'extension de cette interface doivent être héritées par l'application.*

Nous savons qu'il existe un intercepteur de requêtes du côté client et un autre du côté serveur. Nous stipulons ainsi que tout intercepteur qui se trouve du côté client représente le méta-objet de la souche du client (i.e. méta-stub). En revanche, l'intercepteur du côté serveur est le méta-objet du serveur (i.e. méta-serveur).

C3 : Problème et incidence sur la conception de DAISY	*L'intercepteur ne peut pas avoir de méta-objet (i.e. méta-intercepteur). C'est pour cette raison qu'on ne peut pas observer et changer son comportement.*

Comme le lien entre objet et méta-objet doit être régi par un protocole *MOP*, la norme CORBA en a défini un à l'exécution.

En effet, l'établissement de lien entre objet et méta-objet (i.e. la sélection) se fait lors du chargement de l'application. L'intergiciel utilise un initialisateur permettant d'associer un intercepteur à une application (i.e. client ou serveur). Cette association représente le lien qui est créé entre le méta-objet (i.e. intercepteur) et l'objet (i.e. l'application). Cette phase représente la phase d'initialisation du protocole à méta-objet (i.e. le lien entre le niveau de base et le méta-niveau).

C4 : Problème et incidence sur la conception de DAISY	*Un ensemble d'intercepteurs ne peut être associé à une application qu'en série, ce qui nécessite la cohérence de leurs mécanismes non-fonctionnels. C'est pour cette raison que la coexistence de deux applications avec leur ensemble d'intercepteurs peut ne pas être possible.*

Lors de l'exécution de l'application, le protocole à méta-objet assure l'observation et le contrôle d'un seul concept du modèle à objet CORBA, à savoir les requêtes.

Pour que les intercepteurs observent les requêtes, l'intergiciel applique un protocole simple qui se déroule de la manière suivante :

- réifie toute requête issue d'un client à l'intercepteur de requête client PIC.

- réifie toute requête à destination d'un serveur à l'intercepteur de requête serveur PIS.
- délivre toutes les requêtes réifiées par les PIC et PIS au serveur cible; sauf dans le cas où l'intercepteur déclencherait l'exception `Forward_Request`.

Les intercepteurs sont intimement liés aux applications auxquelles elles sont associées. La défaillance de l'application par arrêt entraîne l'arrêt de son intercepteur.

C5 : Problème et incidence sur la conception de DAISY	*Le cycle de vie de l'intercepteur est lié à celui de l'application. C'est pour cette raison que la défaillance d'une application affecte une partie de l'ORB et des mécanismes non-fonctionnels qui y sont définis.*

4.2.2. Prototype de la réplication passive

Nous présentons, dans la figure 8, l'architecture de la variante à base de méta-objets intégrés de la plate-forme DAISY. Dans cette architecture, nous traitons la mise en œuvre de la réplication passive.

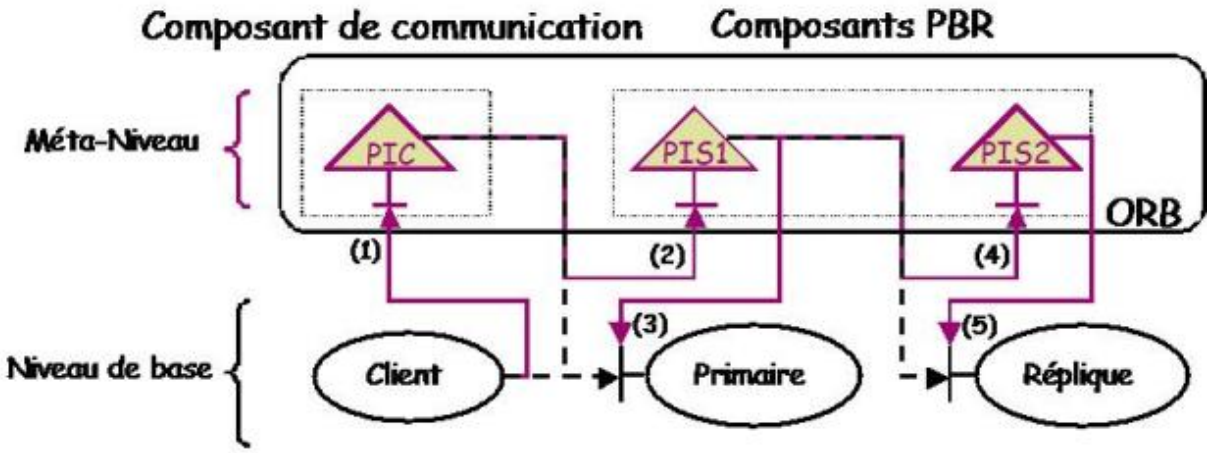

Figure 8. Réplication passive à méta-objets intégrés

Comme le montre cette figure, le méta-niveau renferme les composants de tolérance aux fautes, à savoir le CC-C/S et les composants PBR qui sont des intercepteurs dans cette seconde variante. Le CC-C/S est représenté par le PIC et les composants PBR par les PIS1 et PIS2. Le niveau de base est composé d'un client et de deux serveurs d'applications, le serveur Primaire et le serveur Réplique.

Lors de l'exécution de l'application, toute requête issue du client est réifiée par le PIC (1). Le composant de communication envoie ces requêtes au serveur primaire qui seront réifiées par le PIS1 (2). Ce méta-objet analyse les requêtes qui lui sont parvenues puis les transmet au serveur Primaire (3). Par ailleurs, le PIS1 récupère l'état du serveur primaire puis élabore une requête de mise à jour de l'état du serveur réplique. Cette requête est réifiée à l'intercepteur PIS2 (4) qui la traite, puis assure la mise à jour du serveur réplique (5).

Pour que les intercepteurs puissent envoyer des requêtes, ils récupèrent les IOR des différents objets du niveau de base. Ces identificateurs sont utilisés par la suite pour la mise en œuvre des mécanismes de réplication passive et non pour la mise en œuvre d'un protocole à méta-objet, comme cela était le cas pour la première variante de la plate-forme DAISY.

4.2.3. Prototype de la réplication active

Nous présentons, dans la figure 9, l'architecture de la variante à base de méta-objets intégrés de la plate-forme DAISY, dans laquelle nous traitons la mise en œuvre de la réplication active.

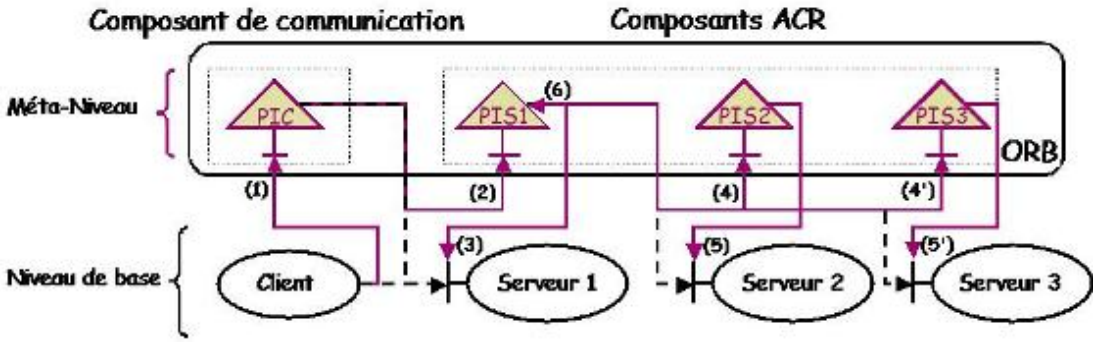

Figure 9. Réplication active à méta-objets intégrés

Le méta-niveau renferme les composants de tolérance aux fautes, à savoir les composants ACR et le composant de communication client/serveur. Le composant de communication est représenté par le *PIC* et les composants *ACR* se trouvent au niveau des intercepteurs *PIS1*, *PIS2* et *PIS3*. Le niveau de base est composé d'un client et de trois serveurs d'applications : le *Serveur 1*, le *Serveur 2* et le *Serveur 3*. Ces trois serveurs forment le groupe de répliques. En observant et en contrôlant le comportement et l'état des objets du niveau de base, les différents intercepteurs leur fournissent les mécanismes de la réplication active.

Lors de l'initialisation des objets du niveau de base, nous leur associons des intercepteurs qui jouent le rôle de méta-objets. L'intercepteur PIC qui représente le composant de communication client/serveur est associé à l'objet client. Les intercepteurs PIS1, PIS2 et PIS3 qui jouent le rôle des composants ACR sont associés aux serveurs Serveur 1, Serveur 2 et Serveur 3 respectivement. Cette phase représente la phase d'initialisation du protocole à méta-objet (i.e. le lien entre le niveau de base et le méta-niveau).

Lors de l'exécution de l'application, toute requête issue du client est réifiée au PIC (1) qui joue le rôle du composant CC-C/S. Ces requêtes qui sont transmises au Serveur 1 seront réifiées à l'intercepteur PIS1 (2). Ce méta-objet analyse les requêtes qui lui sont parvenues puis les transmet à l'objet du niveau de base qui lui est associé Serveur 1 (3). Pour assurer les mécanismes de réplication active, l'intercepteur PIS1 diffuse toutes les requêtes qui lui parviennent aux autres serveurs du groupe de répliques (i.e. Serveur 2 et Serveur 3). Ces requêtes sont réifiées aux intercepteurs PIS2 et PIS3 (4, 4'). Chaque méta-objet invoque par la suite l'objet du niveau de base qui lui est associé (5, 5'). Après avoir traité les requêtes, les serveurs retournent leurs réponses au méta-objet PIS1 (6) en passant par leurs méta-objets respectifs PIS2 et PIS3. Enfin, le PIS1 applique le vote et retourne la réponse majoritaire au client.

4.3. Comparaison et choix entre les deux variantes

Les deux approches présentées permettent de fournir les différents mécanismes réflexifs, nécessaires à la mise en œuvre de la réplication, avec pour chacune des avantages et des limites (voir Tableau 1).

Tableau 1. Les mécanismes réflexifs offerts par les deux versions de DAISY

Mécanismes réflexifs DAISY	Réification	Intercession comportementale	Introspection	Intercession structurelle
À méta-objets externes	++	++	++	++
À méta-objets intégrés	+	+	++	++

Avec la première variante, le choix du méta-objet peut être effectué aussi bien lors du chargement de l'application que lors de son exécution. En outre, nous pouvons définir des règles de réification implémentant un filtrage par exemple. Ni le choix du méta-objet, ni même la définition de règles de réification ne sont possibles avec la seconde variante. La réification est assurée par l'intergiciel d'une manière statique et non modifiable (i.e. toutes les requêtes sont réifiées au niveau des intercepteurs).

Comme les intercepteurs ne peuvent que rediriger une requête vers un autre serveur en soulevant l'exception `ForwardRequest`, l'intercession comportementale offerte par la seconde variante est très limitée. En revanche, un méta-serveur, de la seconde variante, a la possibilité de changer les paramètres d'une invocation avant de l'appliquer au serveur du niveau de base. Aussi, il peut même invoquer une méthode différente de celle qui lui est réifiée au niveau du serveur d'application.

Par ailleurs, pour les mécanismes d'intercession structurelle et d'introspection, les deux versions sont

équivalentes, puisque ces mécanismes sont dépendants des méta-interfaces offertes par les serveurs d'applications.

Les deux variantes permettent de mettre en œuvre les quatre mécanismes de réplication classique. Seule la variante à composants intégrés, altère les mécanismes de la réplication active. En effet, dans le cas où le serveur associé au serveur qui diffuse la requête défaille en valeur, le *PIS* ne peut pas retourner une réponse qu'il aurait récupérée d'un autre serveur.

L'utilisation d'un service de communication de groupe facilite la mise en œuvre de mécanismes de réplication. Munir la plate-forme DAISY d'un service de communication de groupe peut être envisagé dans la première variante puisque les méta-objets ne sont pas limités en termes de contrôle et d'observation et qu'ils possèdent des interfaces modifiables. En revanche, avec une interface non modifiable et un mode d'activation lié au flot des requêtes, les intercepteurs sont mal adaptés à la mise en œuvre d'un système de communication de groupe.

La seconde variante de l'architecture de DAISY est mieux adaptée pour la mise en œuvre des mécanismes de tolérance aux fautes pour deux raisons :

Dans la première variante, les intercepteurs CORBA sont utilisés pour mettre en œuvre un protocole à méta-objets. Les mécanismes de réplication sont intégrés au niveau des méta-objets qui se trouvent au niveau applicatif. Ceci limite l'évaluation des intercepteurs par rapport à leur utilisation dans la mise en œuvre de la tolérance aux fautes. En revanche, dans la seconde variante, les intercepteurs CORBA jouent le rôle de méta-objets et intègrent les mécanismes de réplication. Ceci nous donnera plus d'éléments pour juger de la possibilité d'intégrer la tolérance aux fautes au niveau de la couche intermédiaire d'une manière standard.

Le potentiel de réification offert par le protocole à méta-objet à base d'intercepteurs CORBA, se limite à la re-direction d'une invocation à un méta-serveur ayant une interface équivalente à celle qui est proposée par le serveur de base. C'est-à-dire que l'interface du méta-serveur doit contenir les méthodes définies au niveau de l'interface du serveur de base. L'aspect non fonctionnel est, par conséquent, caché derrière cette interface. Ceci est très contraignant par rapport au développement des composants de tolérance aux fautes car ils doivent tenir compte des interfaces des applications sous-jacentes. En conséquence, cette architecture n'est pas réutilisable facilement.

5. LES LIMITES CONCEPTUELLES DES INTERCEPTEURS CORBA

5.1. Protocole à méta-objects à base d'intercepteurs CORBA

Ce type de MOP doit prendre en compte les différents éléments du concept à objet CORBA. Ainsi, il doit couvrir les notions d'objet, d'interface et de requête qui véhiculent les interactions entre les objets CORBA. En utilisant les deux variantes d'intercepteurs de requêtes CORBA (i.e. *PIS* et *PIC*), un méta-objet peut observer et contrôler toutes les requêtes sortantes d'une application cliente ou à destination d'un serveur. Cependant, si ces intercepteurs peuvent notifier les méta-objets de la destruction des objets qui leur sont associés, ils sont incapables d'assurer cette notification lors de l'initialisation des objets.

En revanche, ces intercepteurs ne permettent pas l'observation de l'objet et de son interface. Pour résoudre ce problème, il faut que l'application hérite une interface particulière renfermant des méthodes d'observation et de contrôle de son état et de sa structure ; c'est l'approche que nous avons utilisée pour le développement de la plate-forme DAISY (Bennani, 2004).

5.2. Meta-Objets de tolérance aux fautes à base d'intercepteurs CORBA

Nous montrons dans le tableau 2 les limites des intercepteurs CORBA dans le cas où on les utiliserait comme des méta-objets pour mettre en œuvre des applications tolérants les fautes.

Tableau 2. Limites des intercepteurs en tant que méta-objets

	Aspect Réflexif			Aspect Tolérance aux fautes	
Intercepteurs CORBA	Non circulaire	Absence de méta-interface adaptable	Association unique d'un ensemble d'intercepteurs	Élargissement de la zone de confinement des erreurs	Non évolutif

Dans les systèmes réflexifs, l'aspect circulaire (Smith, 1982) permet d'associer un méta-objet à tout objet ou méta-objet. Cette notion permet de munir une application de plusieurs méta-niveaux, c'est-à-dire de plusieurs mécanismes non fonctionnels. Cet aspect n'est pas fourni par la norme CORBA actuelle puisqu'un intercepteur ne possède pas son propre IOR.

La communication entre les méta-objets, du même niveau, de niveaux différents ou avec des objets externes, se fait par l'intermédiaire des méta-interfaces. Cette communication est rendue difficile car les intercepteurs n'ont qu'une interface unique et non modifiable.

La prise en compte d'un ou de plusieurs méta-objets dans un méta-niveau donné relève de la flexibilité du protocole à méta-objet qui les gère. Cette flexibilité, qui permet le changement de méta-objets, élargit le potentiel des solutions architecturales. Les intercepteurs actuels n'ont qu'une seule et unique manière d'association, en série.

Dans les systèmes critiques, plus la zone de confinement d'erreur est bien isolée plus les effets de corrélation des erreurs sont limités. Les intercepteurs actuels sont systématiquement détruits lorsque leur objet CORBA associé est détruit. Ceci élargit la zone de confinement des erreurs, puisque la défaillance de l'application n'entraîne plus son arrêt, mais aussi l'arrêt d'autres composants de l'intergiciel sous-jacent.

Comme ces systèmes sont souvent des systèmes à longue durée de vie, les besoins en termes de tolérance aux fautes peuvent évoluer. Par conséquent, l'ajout ou la modification de ces fonctionnalités orthogonales par le biais des intercepteurs CORBA doivent être dynamiques et indépendants des applications, et ce pour des raisons de disponibilité. Jusqu'à la dernière spécification de l'intergiciel CORBA, les intercepteurs de requêtes ne peuvent pas être insérés ou retirés lorsque l'application est en cours d'exécution.

6. Conclusions et perspectives

Nous avons présenté, dans cet article, les limites que posent les intercepteurs CORBA standards pour le développement d'architectures tolérant les fautes et réflexives. Nous avons discuté ces limites selon deux points de vue différents pour construire un protocole à méta-objet à l'exécution et comme des méta-objets de tolérance aux fautes.

Pour le premier aspect, nous avons montré que les intercepteurs CORBA assurent : la réification de la destruction d'un objet ainsi que les interactions entre objets. En revanche, ils n'assurent pas la réification de la création d'un objet, et l'intercession structurelle qui permet la communication entre méta-objet et objet.

En revanche, ces intercepteurs ne peuvent pas fournir l'observation et le contrôle de l'état des objets (i.e. introspection et intercession structurelles).

Pour le second aspect, où nous considérons les intercepteurs comme des méta-objets de tolérance aux fautes, nous avons distingué deux types de limites : celles qui sont relatives à la réflexivité et celles qui sont relatives à la tolérance aux fautes.

Sur le plan de la réflexivité, nous avons présenté la pauvreté de la méta-interface offerte par ces intercepteurs, leur affectation statique aux applications et leur incapacité d'être réflexifs.

Sur le plan de la tolérance aux fautes, nous avons démontré que leur utilisation introduirait un élargissement de la zone de confinement des erreurs à l'espace contenant l'application et ses intercepteurs associés.

A partir de ces travaux, nous nous proposons de formuler des recommandations pour la définition d'une nouvelle génération d'intercepteurs compatibles avec les intercepteurs actuels. Ces nouveaux intercepteurs doivent être plus adaptés au développement des mécanismes non fonctionnels, en particulier, ceux qui sont relatifs à la tolérance aux fautes.

Bibliographie

Bennani, M. T., Blain, L. et al. (2004). Implementing Simple Replication Protocols using CORBA Portable Interceptors and Java Serialization. Proceedings de l'International Conference on Dependable Systems and Networks (DSN 2004), Florence, juin 2004, 549-554.

Bennani, M. T., (2005). Tolérance aux fautes dans les systèmes répartis à base d'intérgiciels réflexifs standrads. Mémoire de doctorat. France : Institut National des Sciences Appliquées de Toulouse.

Chiba, S. et Masuda, T. (1993). Designing an Extensible Distributed Language with a Meta-Level Architecture.

Proceedings de l'European Conference on Object Oriented Programming (ECOOP'93), Kaiserslautern, juillet 1993, 482-501

Clarke, M., Blair, G. S., et al. (2001). An Efficient Component Model for the Construction of Adaptive Middleware. Proceedings de l'International Conference on Distributed Systems Platforms, Heidelberg, 2001, 160-178.

Ezhilchelvan, P. D., Macedo, R. A. et al. (1995). Newtop: A Fault-Tolerant Group Communication Protocol. Proceedings de la 5th International Conference on Distributed Computing Systems (ICDCS'95), Vancouver, mai 1995.

Fabre, J. -C. et Pérennou, T. (1998). A Metaobject Architecture for Fault-Tolerant Distributed Systems: the FRIENDS Approach", IEEE Transactions on Computers, Special Issue on Dependability of Computing Systems, 1, 47, janvier 1998, 78-95.

Felber, P., Guerraoui, R. et al. (1998). The implementation of a CORBA object group service. Theory and Practice of Object Systems, 2, 4, avril 1998, 93-105.

IONA (1994). An Introduction to ORBIX+ISIS, IONA Technologies Ltd and Isis Distributed Systems, Inc.

IONA (2000a). Filtering Operation Calls. Orbix Programmer's Guide C++ Edition (I. T. PLC, Ed.), 319-338.

IONA (2000b), Transforming Requests. Orbix Programmer's Guide C++ Edition (I. T. PLC, Ed.), 401-408.

Karablieh, F. et R. A. Bazzi, R. A. (2002). Heterogeneous Checkpointing for Multithreaded Applications. Proceedings du 21st Symposium on Reliable Distributed Systems (SRDS'02), Osaka, 2002, 140-149.

Gregor Kiczales, G., Des Rivières, J, et al. (1991). The Art of the Metaobject Protocol. Londres: MIT Press.

Killijian, M. -O, Ruiz-Garcia, J.-C. et al. (2002). Portable serialization of CORBA objects: a Reflective Approach. Proceedings de l'ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA), Seattle, 2002.

Maes, P (1987). Computational Reflection. Mémoire de doctorat, Bruxelles: Vrije Universiteit.

Maffeis, S. (1995). Run-Time Support for Object-Oriented Distributed Programming. Mémoire de doctorat, Suisse: Université de Zurich.

Moser, L. E., Melliar-Smith P. M., et al. (1999). A Fault Tolerant Framework for CORBA. Proceedings du 29th International Symposium on Fault-Tolerant Computing, Madison, 1999.

Narasimhan, P., Moser, L. E. et al. (2001). State Synchronization and Recovery for Strongly Consistent Replicated CORBA Objects, Proceedings de l'International Conference on Dependable Systems and Networks (DSN'2001), Washington, DC, 2001, 261-270.

Object Management Group. (2002a). Fault Tolerant CORBA. Common Object Request Broker Architecture: Core Specification OMG formal/02-12-06, 939-1043.

Object Management Group. (2002b). Portable Interceptors. Common Object Request Broker Architecture: Core Specification OMG formal/02-12-06, 789-853.

M. Roman, M., Kon, F., et al. (1999). Design and Implementation of Runtime Reflection in Communication Middleware: The DynamicTAO Case. Proceedings du 19th IEEE International Conference on Distributed Computing Systems Workshops on Electronic Commerce and Web-based Applications/Middleware. Austin, 1999, 122-127.

Saikoski, K. et Coulson, G. (2003). Supporting Dependable Distributed Applications Through a Component-Oriented Middleware-Based Group Service. Proceedings du Workshop on Software Architectures for Dependable Systems (WADS), Portland, 2003, .99-122.

Smith, B. C. (1982). Procedural Reflection in Programming Languages. Macuscrit de doctorat, Etats-Unis: Massachussetts Institute of Technology.

Taiani, F. (2004). La réflexivité dans les architectures multi-niveaux. Manuscrit de doctorat. France: Université Paul Sabatier de Toulouse.

Wang, N., Parameswaran, K. et al. (2001). The Design an Performance of Meta-Programming Mechanisms for Object Request Brocker Middleware, Proceedings du USENIX, Boston, juin 2001.

Pour citer cet article

Mohamed Taha BENNANI. «Vers une tolérance aux fautes standard, interopérable et transparente». e-TI - la revue électronique des technologies d'information, Numéro 3, 30 décembre 2006, http://www.revue-eti.netdocument.php?id=1078.