

---

# Intergiciels schizophrènes : une solution fortement interopérable fondée sur les composants adaptables

**Laurent Pautet**

*Ecole Nationale Supérieure des Télécommunications  
46, rue Barrault  
75013 Paris  
Laurent.Pautet@enst.fr*

---

*RÉSUMÉ. La schizophrénie a été introduite dans le domaine des intergiciels afin de résoudre le problème d'interopérabilité entre modèles de répartition, plus connu sous l'acronyme M2M pour Middleware To Middleware. Un intergiciel générique s'instancie en fonction d'un modèle pour donner une personnalité. Un intergiciel schizophrène instancie simultanément plusieurs personnalités cohabitantes et interagissantes. Notre plate-forme schizophrène, PolyORB, découple la partie applicative de la partie protocolaire d'une personnalité grâce à une couche neutre du point de vue du modèle. Cette approche permet de construire des passerelles dynamiques entre intergiciels hétérogènes et offre ainsi une solution souple au problème du M2M. L'architecture de PolyORB s'articule autour de composants adaptables fondés des gabarits de conception classiques ou spécifiques.*

*ABSTRACT. The concept of schizophrenia in the middleware context was introduced in order to solve the distribution model interoperability. A generic middleware is instantiated for a given distribution model to provide a personality. A schizophrenic middleware includes several co-existing and collaborating personalities. Our schizophrenic platform, PolyORB, decouples the application part from the protocol one by defining a neutral layer (independent from the distribution model). This enables the construction of dynamic gateways between middlewares in order to provide a flexible solution to the M2M problem. The PolyORB architecture is composed of adaptative components based on classical or specific design patterns.*

*MOTS-CLÉS : Intergiciels, Interopérabilité, Généricité, Configurabilité, Composants*

*KEYWORDS: Middleware, Interoperability, Genericity, Configurability, Components*

---

## 1. Introduction

Le développement intense de la répartition dans des domaines applicatifs aussi variés que le temps réel ou l'internet amplifie le besoin en de nouveaux modèles de répartition comme CORBA [OMG 98]. La multiplication de ces modèles induit deux problématiques : le développement des intergiciels correspondants et la génération de passerelles entre ceux-ci. En proposant une architecture orientée autour de composants adaptables, la schizophrénie apporte une solution à ces deux problématiques.

En fonction des domaines et des besoins des applications, les modèles de répartition se spécialisent ou se généralisent autour de trois mécanismes fondamentaux :

- l'envoi de messages constituent une réponse efficace aux problèmes de calcul scientifique (MPI, [MPI95]) ou de systèmes d'information (MQSeries, [IBM 97]) ;
- l'appel de sous-programme distant (RPC, [BIR 83]) ou de méthodes sur objets répartis (CORBA, [OMG 98]) fournissent des solutions plus lourdes mais plus structurantes du point de vue du génie logiciel ;
- les objets partagés ou localisés dans un espace partagé éventuellement réparti (DSM) se sont répandus à travers des environnements comme ThreadMarks [KEL 94].

La configurabilité permet d'adapter les plates-formes de communication aux besoins réels de l'application. Une approche classique consiste donc à définir une architecture dont les composants faiblement couplés sont configurables indépendamment. Ainsi, l'utilisateur ajoute ou retire des propriétés de l'intergiciel concerné en sélectionnant statiquement ou dynamiquement les composants appropriés.

La genericité traite de la configurabilité de l'intergiciel en fonction du modèle de répartition. Développer des plates-formes répondant efficacement à la multiplicité des modèles de répartition constitue une barrière technologique pour l'industrie de l'intergiciel. Une approche fréquemment adoptée vise à factoriser les composants pour les réutiliser, les surcharger et les intégrer dans une architecture générique instanciée ou personnalisée en fonction du modèle de répartition.

L'interopérabilité entre modèles de répartition constitue une problématique industrielle émergente, résumée par le concept de M2M pour *Middleware To Middleware* [BAK 01]. Elle est rendue essentielle par la réutilisation de composants préexistants. Ainsi, des composants CORBA doivent pouvoir interopérer avec des composants de *Message Oriented Middlewares* (MOM) [BAN 99]. Le paradoxe de l'intergiciel provient de la contradiction entre son objectif d'interopérabilité au sens classique et l'hétérogénéité inhérente aux multiples modèles de répartition utilisés. Or, l'approche consistant à traduire statiquement une entité d'un modèle de répartition dans un autre ne supporte pas le passage à l'échelle de plusieurs modèles.

L'industrie et le paradoxe des intergiciels constituent deux problématiques ouvertes auxquelles nous apportons une solution en unifiant les concepts de configurabilité, de genericité et d'interopérabilité.

Dans la section 2, nous étudions les solutions adoptées par les intergiciels existants en matière de configurabilité, d'interopérabilité et de généricité. La section 3 expose les raisons pour lesquelles, dans le domaine des intergiciels, la schizophrénie unifie ces trois approches et propose une solution globale aux problèmes d'industrie et d'interopérabilité des intergiciels. La section 4 décrit l'architecture induite par la schizophrénie ainsi que ces composants fondamentaux. Nous indiquons en section 5 les résultats de nos expérimentations.

## **2. Approches suivies par les intergiciels existants**

Pour traiter de l'industrie et du paradoxe des intergiciels, nous avons unifié trois approches complémentaires que nous allons illustrer à travers des intergiciels existants : la configurabilité, l'interopérabilité et la généricité.

### **2.1. Intergiciels fondés sur la configurabilité**

La configurabilité permet de répondre précisément aux besoins réels d'une application ou encore d'adapter un modèle de répartition à des variantes mineures. L'utilisateur sélectionne les composants utilisés dans un intergiciel afin de déterminer par exemple les politiques d'allocation de mémoire. Parmi les travaux examinés dans [PAU 01a], nous nous concentrons sur GLADE, la première implémentation de l'annexe des systèmes répartis d'Ada 95 (DSA) [ISO 95b]. Cet intergiciel se caractérise par une configurabilité proche de celle de TAO [SCH 97].

Le modèle de répartition normalisé d'Ada 95 (DSA) consiste en un sous-ensemble d'entités classiques du langage dont des pragmas restreignent l'usage afin de les doter de propriétés de répartition. Ces pragmas permettent à des sous-programmes et des objets du langage de se comporter comme des sous-programmes distants, des objets répartis et des objets partagés.

GLADE constitue la première implémentation de DSA. Il se compose de GNAT, compilateur Ada 95 de la famille des compilateurs GCC, qui a été enrichi pour les besoins de la répartition ; de GARLIC, système de communication dont la norme impose l'interface avec le compilateur ; et de GNATDIST, outil de déploiement et de configuration d'applications réparties. Porté sur de nombreux systèmes d'exploitation, ce logiciel libre a été validé auprès des organismes accrédités de sorte que GNAT est le seul compilateur proposant l'intégralité de la norme.

Grâce à son langage de description, GNATDIST [KER 96] permet de déployer une application en affectant ses modules à des nœuds logiques. Il permet également de particulariser les composants de la plate-forme de communication GARLIC [PAU 00]. Enfin, l'utilisateur profite des extensions que GLADE apporte aux fonctionnalités de DSA. GLADE s'appuie sur une architecture de composants définis à partir de gabarits de conception [GAM 94]. GNATDIST permet ainsi de contrôler la configuration des ressources de GARLIC notamment les protocoles de communication, les supports

de stockages partagés ou les processus légers traitant les requêtes distantes. Notamment, l'utilisateur peut alléger le système de communication en supprimant la gestion concurrente des requêtes, ce qui s'avère un mécanisme novateur et essentiel pour les systèmes embarqués.

## 2.2. Intergiciels fondés sur l'interopérabilité

L'interopérabilité vise à produire des passerelles entre modèles de répartition, ce qui s'avère un processus délicat en l'absence de fonctionnalités adaptées au sein des intergiciels impliqués dans ce travail de traduction. Nous avons clairement mesuré cette difficulté lors du développement de CIAO [QUI 99], un générateur opérationnel de passerelles statiques entre DSA / GLADE vers CORBA / AdaBroker, notre environnement CORBA libre pour Ada 95 [AZA 99]. Aussi, dans cette section, nous concentrons-nous sur CorbaWeb qui à la différence de CIAO produit des passerelles dynamiques entre le *web* et CORBA.

CorbaWeb [GEI 97] propose des outils pour concevoir, déployer et utiliser des objets CORBA *via* le *web*. L'utilisateur accède aux services CORBA par l'intermédiaire d'un navigateur. Il les consulte comme des documents HTML et exécute des opérations sur ces objets, par exemple à travers des scripts CGI. CorbaWeb se compose de métascripts développés en CorbaScript [MER 97]. Ils produisent des interfaces HTML, qui fournissent une représentation des objets CORBA, ainsi que des scripts, qui exécutent notamment des appels de méthodes sur ces objets.

CorbaWeb relie un client *web* à des objets CORBA sur leur serveur. Il effectue la liaison entre les deux environnements en recevant des requêtes HTTP pour les transformer et les exécuter dynamiquement sous forme de requêtes CORBA. Ces opérations se fondent sur le mécanisme d'invocation dynamique (DII) et sur le référentiel d'interfaces (IR). S'il n'adresse pas directement le problème de l'interopérabilité entre modèles de répartition, CorbaWeb propose une architecture novatrice de passerelle dynamique qui démontre la pertinence de la DII et de l'IR en la matière.

## 2.3. Intergiciels fondés sur la généricité

La généricité étend le concept de configurabilité par la production d'une personnalité ou d'une instance de l'intergiciel en fonction d'un modèle de répartition. Cette approche permet de développer rapidement un environnement pour démontrer la pertinence d'un nouveau modèle de répartition. Parmi les intergiciels génériques, QuarterWare propose une architecture novatrice orientée autour des gabarits de conception [SIN 98]. Dans cette section, nous préférons présenter Jonathan, un intergiciel libre fondé sur les liaisons inspirées du modèle ODP [ODP 95].

Jonathan propose un ORB générique bâti sur une architecture classique mais enrichie de services internes originaux. Le subrogé, ou *Surrogate* [DUM 98] à la base de l'implémentation des liaisons explicites ou implicites, constitue un des composants

novateurs de Jonathan. La notion d'adaptateur d'objets se trouve étendue par rapport à celle de CORBA. Ce mécanisme ne concerne plus uniquement le serveur mais aussi le client, de sorte que le contrôle sur la liaison s'exerce de bout en bout. Le gabarit de conception *Fabrique Abstraite* utilisée pour les liaisons permet de rendre Jonathan configurable en l'occurrence en fonction des protocoles de communication.

Il est possible de mesurer la réutilisation des composants entre personnalités de Jonathan. Le code nécessaire à l'instanciation d'une personnalité s'avère relativement volumineux comparé à celui de la partie générique. En effet, la partie générique se compose essentiellement d'interfaces abstraites et induit la concrétisation de nombreux composants dans une personnalité. `David` constitue une personnalité de Jonathan pour CORBA. S'il offre les fonctionnalités de DII et de DSI, il n'intègre pas le Portable Object Adapter (POA). Jonathan représente 42 % du code utilisé pour la personnalité CORBA. `Jérémie` fournit une personnalité de RMI. Jonathan représente 50 % du code utilisé pour la personnalité RMI. Par ailleurs, d'autres fabriques de liaisons pour ATM [SEI 99] accompagne cet intergiciel.

#### 2.4. Observations

Les gabarits de conception facilitent la mise en œuvre de la configurabilité et de la généricité. L'étude d'intergiciels génériques comme Jonathan (ou QuarterWare) et le développement de nombreux environnements répartis comme GLADE (ou AdaBroker) nous ont démontré qu'une plate-forme générique constitue une solution efficace au problème de l'industrie des intergiciels. De CorbaWeb, nous avons retenu que le mécanisme d'activation de passerelles dynamiques fondé sur l'invocation et l'implémentation dynamiques d'interfaces DII et DSI peut s'appliquer au contexte de l'interopérabilité des modèles de répartition. Pour terminer, il s'avère que l'architecture d'un intergiciel générique ainsi que l'ensemble de ses composants configurables restent à définir plus précisément.

### 3. Définition et objectifs de la schizophrénie

Nous nous sommes fixés comme premier objectif de traiter de l'interopérabilité entre modèles de répartition. Nous avons pu constater à travers CIAO (voir section 2.2) que la mise en présence d'un intergiciel par modèle de répartition et la production d'autant de passerelles que de paires de modèles conduisaient à une explosion combinatoire et à un encombrement inacceptable de la mémoire. Notre démarche a donc consisté à mettre en présence non pas des intergiciels monolithiques mais des personnalités d'un même intergiciel générique. Cette approche présente deux avantages. En premier lieu, le partage de structures communes au sein des personnalités et plus généralement au sein de l'intergiciel générique favorise la production de passerelles dynamiques. Par ailleurs, elle améliore le taux de factorisation et de configurabilité du code et donc facilite l'industrie des intergiciels.

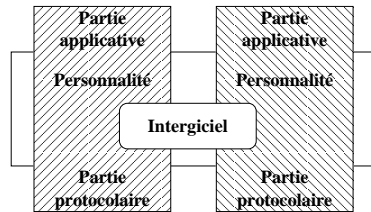


Figure 1. Couplage fort

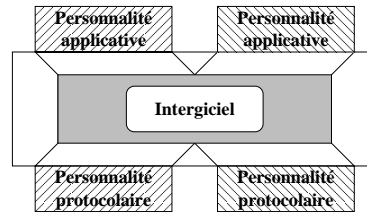


Figure 2. Couplage faible

### 3.1. Observations

Notre approche consiste à fusionner le code de plusieurs personnalités d'un même intergiciel pour favoriser la production de passerelles dynamiques et limiter l'inflation de ressources due à la présence de multiples intergiciels. Cependant, nos études des intergiciels génériques nous amènent à nuancer notre propos.

Partage de code : comme nous l'avons souligné, les intergiciels génériques partagent peu de ressources entre personnalités. Bien que la présence simultanée de personnalités semble possible, le gain en ressources consommées paraît faible *a priori*.

Cohabitation des personnalités : même dans le cas d'une forte factorisation de code, les différentes personnalités cohabitent difficilement, l'intergiciel étant structurellement conçu pour ne supporter qu'une seule instance.

Interaction entre personnalités : pour faciliter les conversions d'un modèle de répartition à l'autre, les différentes personnalités doivent partager et interagir avec leurs ressources. Or, les personnalités concrétisent souvent l'architecture abstraite d'un intergiciel générique avec des composants incompatibles entre instantiations.

Fort couplage des parties applicative et protocolaire : un intergiciel masque l'hétérogénéité entre l'application, le système ou encore le protocole de requêtes (par exemple GIOP). Or, le découplage entre les parties applicative et protocolaire s'avère peu effectif et l'architecture d'un intergiciel correspond fréquemment à la figure 1.

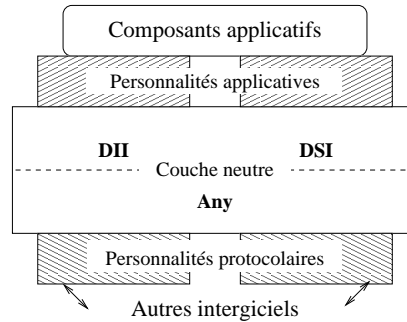
### 3.2. Schizophrénie

Dans [PAU 01a], nous définissons la schizophrénie comme caractérisant chez un intergiciel sa capacité à disposer, simultanément, de plusieurs personnalités afin de les faire interagir efficacement. L'intergiciel satisfait alors les propriétés suivantes :

- cohabitation de personnalités multiples,
- partage du code et des données entre les personnalités,
- découplage des personnalités applicatives et protocolaires.

La mise en application des propriétés de schizophrénie nécessite la définition d'une architecture spécifique bâtie autour d'une couche neutre du point de vue des personnalités, comme l'indique la partie grisée de la figure 2. Cette couche neutre est constituée de composants, indépendants des instanciations, auxquels s'adressent les personnalités applicative et protocolaire pour réaliser leur modèle de répartition respectif. Le principe de la couche neutre s'inspire de celui du langage intermédiaire qui sépare le dorsal et le frontal d'un compilateur.

L'existence de personnalités protocolaires et applicatives découplées par l'intermédiaire de la couche neutre introduit un élément d'architecture essentiel pour la réalisation de passerelles dynamiques. Plus les personnalités applicative et protocolaire utilisent la représentation commune de la couche neutre, plus les conversions entre modèles sont facilitées. Plus la couche neutre s'enrichit de composants concrets, plus le code se trouve factorisé entre personnalités. De plus, un intergiciel schizophrène favorise le déploiement des personnalités applicative et protocolaire. En effet, en l'absence de son équivalent protocolaire, il s'avère possible de développer une personnalité applicative grâce à l'instanciation protocolaire d'un autre modèle de répartition.



**Figure 3.** *Architecture schizophrène*

La couche neutre comporte des composants indépendants des personnalités. A ce titre, le type autodéscriptif Any de CORBA, qui permet de stocker une valeur de type quelconque et une description de ce dernier, constitue une structure de données appropriée pour cette couche comme l'illustre la figure 3. De même, les structures d'invocation et d'implémentation dynamiques de CORBA permettent à la couche neutre d'interagir avec la partie applicative de manière indépendante. Naturellement, ces types de données doivent être adaptés afin de ne pas attirer systématiquement une éventuelle personnalité CORBA.

#### 4. Composants fondamentaux

Notre architecture différencie les composants de l'intergiciel en fonction du modèle de répartition [QUI 01]. Certains d'entre eux, comme le transport, se révèlent fondamentaux pour la mise en œuvre d'un modèle de répartition, alors que d'autres

comme le nommage consistent souvent dans des services externes à l'intergiciel. Les composants fondamentaux rassemblent le référencement (ou encapsulation de l'identité des objets), l'échange de requêtes (couche de transport, liaison vers l'objet destinataire, représentation des données, protocole de requêtes), la gestion des ressources (activation des objets, aiguillage des requêtes). Nous donnons leur définition, leurs propriétés, leurs objectifs et des exemples de réalisation pris dans d'autres intergiciels.

#### 4.1. Description des composants

##### *Référencement*

Chaque objet se voit attribuer une référence composée d'informations transmises auprès de nœuds logiques. Elle désigne sans ambiguïté l'objet au cours de son existence. Un tel référencement s'obtient en utilisant des adresses de points d'accès d'un service de transport tels que les *Transport Service Access Point* décrits dans [ISO 95a]. En associant ces adresses à un identifiant local et unique attribué par l'intergiciel, l'objet dispose d'un identificateur globalement unique.

##### *Transport*

Un nœud doit pouvoir établir un lien de communication avec un objet pour transmettre les requêtes nécessaires à l'exécution de méthodes sur cet objet. L'invocation d'une méthode sur un objet consiste selon la terminologie de la programmation orientée objet à envoyer un message à l'objet. L'envoi de messages physiques dans un système réparti traduit donc fidèlement ce comportement.

##### *Liaison*

Le composant de liaison, défini notamment par l'environnement Jonathan [DUM 98], a pour rôle de créer un point de transport et d'établir une session de protocole à partir des informations contenues dans une référence. Lorsqu'une requête arrive sur un nœud logique, elle est soit traitée localement, soit retransmise à un autre nœud. Dans le cas local, il la transmet à l'entité concrète implémentant l'objet. Sinon, il fait appel à un subrogé, ou *Surrogate*, pour faire suivre la requête.

Notre composant de liaison généralise cette approche en traduisant une requête entre nœuds logiques distants dotés de couches protocolaires différentes. Le composant de liaison crée un subrogé (ou une passerelle dynamique) dont il adapte le comportement en fonction de l'objet ciblé après échanges d'information entre la couche neutre et les référentiels d'interface des modèles de répartition concernés.

##### *Représentation*

Les données d'une requête sont traduites dans une représentation adéquate pour rester cohérentes à travers un système hétérogène communiquant par réseau. Pour assurer l'interopérabilité entre matériels et systèmes, les nœuds logiques partagent une

représentation commune et normalisée. Par exemple, les intergiciels d'un appelant et d'un appelé conviennent d'une représentation des entiers commune.

#### *Protocole*

L'intergiciel met en œuvre un protocole pour la transmission de requêtes entre instances des exécutifs de répartition. Un protocole pour objets répartis s'articule autour des primitives fondamentales `Request` et `Reply` qui mettent en œuvre les appels de méthodes à distance. D'autres primitives complètent cet ensemble, notamment celles annulant des requêtes en cours. CORBA définit dans ce but GIOP.

#### *Activation*

L'intergiciel active et désactive les objets d'implémentation qui traitent les opérations sur les objets de l'application. Lorsqu'il reçoit une requête, il s'assure que la référence correspond à un véritable objet d'implémentation. S'il ne trouve pas l'objet destinataire parmi les objets présents, il crée un objet d'implémentation dynamiquement ou en recharge un à partir d'un état sauvegardé sur un support persistant.

#### *Aiguillage*

A la réception d'une requête, un nœud logique lui affecte une ressource de traitement, par exemple un processus léger. L'intergiciel le sélectionne parmi un groupe de processus existants ou en crée un nouveau. Si possible, le traitement des requêtes doit être partagé mais il peut également s'effectuer en fonction de plusieurs politiques. Celles-ci, exposées dans [SCH 95], déterminent les critères de création et de sélection du processus léger chargé d'une requête. Comme nous l'indiquons en 2.1, GLADE gère de manière originale le groupe de processus légers chargés des requêtes distantes.

## **4.2. Structure des composants**

Tous ces composants s'appuient sur la couche neutre. Ils interagissent avec les objets des couches applicatives (*Référencement*, *Activation*, *Aiguillage*) ou protocolaires (*Référencement*, *Transport*, *Liaison*, *Représentation*, *Protocole*) au travers de gabarits de conception. Grâce à ceux-ci, nous définissons des relations entre entités sans imposer plus de connaissances sur celles-ci que nécessaires. Cette approche ainsi que l'utilisation de structures auto-descriptives de type `Any` indiquée en 3.2 assure une forte implication de la couche neutre dans la réalisation d'une personnalité.

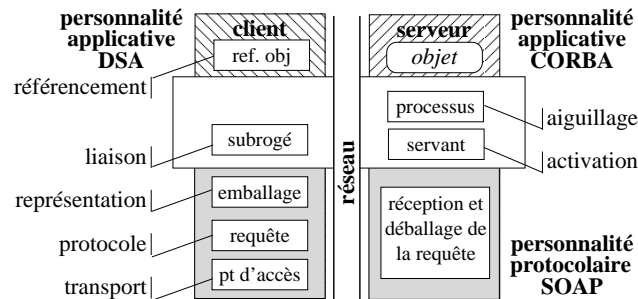
Parmi les gabarits de conception utilisés, nous avons repris des gabarits comportementaux tels que *Wrapper Facade* ou *Acceptor - Connector* ainsi que des gabarits de concurrence *Half-async - Half-sync* ou *Reactor* [SCH 96]. Ces gabarits avaient déjà trouvé leur place dans des projets précédents comme GLADE et AdaBroker.

Nous avons introduit de nouveaux gabarits comme *Components* et *Annotation*. *Components* enrichit dynamiquement les méthodes offertes par un objet et rend explicite l'équivalence entre un appel de méthode et l'envoi d'un message [ING 78].

*Annotation* fournit une fonctionnalité équivalente afin d'enrichir dynamiquement un objet de nouveaux attributs ou données.

### 4.3. Déroulement d'un appel distant

Dans l'exemple de la figure 4, un client DSA disposant d'une référence d'objet interagit avec le serveur CORBA correspondant en utilisant le protocole SOAP [W3C00]. Lorsque le client invoque une méthode sur la référence d'objet, la partie applicative se charge d'interpréter, grâce au composant de *Référencement*, les informations de localisation fournies par la référence. Un objet logique, le *subrogé*, créé par le composant de *Liaison*, se charge d'établir une liaison avec le serveur et exploite ainsi les données de la référence. Il interroge le composant de *Transport* pour obtenir un point d'accès. Les paramètres de méthode sont ensuite emballés par le composant *Représentation* et l'invocation de la méthode s'opère par la construction d'une requête grâce au composant *Protocole*. Après transmission, la personnalité SOAP du destinataire identifie à partir de la requête l'objet physique concerné, la méthode invoquée et les paramètres utilisés. Pour effectuer l'appel, le serveur alloue grâce au composant d'*Activation* un objet logique, le *servant*, correspondant à l'objet physique. Le composant d'*Aiguillage* fournit quant à lui un processus léger assurant l'exécution de la méthode. Ces éléments disponibles, la méthode est exécutée et l'éventuel résultat suit le chemin inverse de la requête.



**Figure 4.** Déroulement d'une requête

## 5. Réalisations

Nous avons réalisé PolyORB, un intergiciel schizophrène pour Ada 95. Nous nous sommes appuyés sur nos développements précédents, GLADE et AdaBroker, destinés à être remplacés par des personnalités de PolyORB. Nous leur avons emprunté des gabarits de conception et des éléments de configuration pour les étendre et les intégrer dans PolyORB. Nous avons réalisé en premier lieu une personnalité SOAP afin de développer des services web. Notre connaissance de leurs modèles a facilité la

réalisation de la personnalité CORBA et plus récemment de celle de DSA. Enfin, nous avons produit une personnalité de MOM, adaptation pour Ada 95 de JMS [SUN 99].

**SOAP** : la personnalité SOAP [W3C00] nous a conduit à développer le composant de *Protocole* pour HTTP et celui de *Représentation* pour XML. Ce travail ne porte que sur la personnalité protocolaire et ne propose pas d'interface de construction de requêtes. L'utilisateur élabore donc manuellement ses requêtes en fonction du composant. Nous prévoyons de fournir des interfaces proches de la DII ou de la DSI.

**CORBA** : AdaBroker nous a facilité le développement de la personnalité CORBA ainsi que la mise en œuvre de la couche neutre en reprenant les aspects réflexifs des types `Request` et `ServerRequest`. Cependant, nous avons adapté ces composants afin qu'ils n'attirent pas dans la couche neutre la personnalité CORBA. Le composant de *Représentation* pour CORBA, inspiré d'AdaBroker, implémentent toutes les versions de GIOP. Le composant d'*Activation* consiste en un POA complet. Nous avons constaté que la partie générique représente 66 % de la personnalité CORBA, soit 30 000 lignes de code comparées aux 15 000 dédiées à CORBA.

**DSA** : la personnalité DSA a été récemment achevée. Ce travail a été facilité en raison du savoir-faire acquis lors du développement de GLADE et de l'étude d'interopérabilité faite autour de CIAO, générateur de passerelles de DSA vers CORBA. Ce travail constitue un travail important de réécriture du compilateur GNAT. Il aborde la mise en œuvre de mécanismes plus riches et plus complexes que ceux de CORBA. Il a donc induit des améliorations essentielles à PolyORB. Le composant d'*Aiguillage* reprend des travaux effectués dans GLADE et AdaBroker afin d'implémenter les allocations de processus légers lors de requêtes distantes. Au travers de ce travail, nous nous sommes attachés à démontrer l'interopérabilité entre DSA et CORBA, ce qui nous a amené à enrichir le composant de *Liaison*, support des passerelles dynamiques.

**MOM** : nous avons instancié PolyORB pour obtenir un MOM, nommé MOMA (*Message Oriented Middleware for Ada*). Nous avons ainsi adapté pour le langage Ada 95 la spécification de JMS. Cette personnalité supporte les composants de *Protocole* HTTP et GIOP. De même, le composant de *Représentation* peut correspondre à celui de CDR ou à celui de XML. La mise en œuvre d'un MOM aux fonctionnalités très distinctes de celles des ORB nous a conduit à enrichir le traitement de l'asynchronisme au sein des composants de PolyORB.

## 6. Conclusion et perspectives

Nous avons introduit le concept d'*intergiciel schizophrène* comme étant un intergiciel générique disposant de plusieurs personnalités cohabitantes et interagissantes. Nous avons proposé une architecture pour de tels intergiciels. Elle découple la partie applicative de la partie protocolaire grâce à une couche de composants neutre du point de vue du modèle de répartition. Par ailleurs, notre architecture s'articule autour de sept composants indispensables au traitement des requêtes distantes.

Grâce à sa couche neutre, un intergiciel schizophrène fournit automatiquement un mécanisme de passerelle et juggle l'explosion combinatoire qu'introduit l'interopérabilité entre modèles de répartition. Par ailleurs, il accorde une place importante à la factorisation de composants fondamentaux que nous avons identifiés. La personnalisation ne nécessite donc que la concrétisation d'un nombre limité de composants et le développement d'un volume réduit de code spécifique. Les réalisations que nous avons menées ont démontré qu'un intergiciel schizophrène induit un fort taux de factorisation du code et fournit également une forte configurabilité.

PolyORB [PAU 01b] constitue le premier intergiciel configurable, personnalisable et interopérable entre modèles de répartition. Techniquement, sa couche neutre généralise les mécanismes d'invocation et d'implémentation dynamique d'interfaces présents dans CORBA. PolyORB met en œuvre le principe de schizophrénie afin de résoudre un problème de génie logiciel synthétisé par le concept de M2M (*Middleware To Middleware*) : l'hétérogénéité des modèles de répartition. Aussi, cet intergiciel nous a-t-il permis de valider notre approche sur des exemples parfois complexes en faisant interopérer de manière effective des services répartis conçus dans des modèles de répartition différents comme CORBA, DSA ou MOM/JMS.

## 7. Bibliographie

- [AZA 99] AZAVANT F., COTTIN J.-M., NIEBEL V., PAUTET L., PONCE S., QUINOT T., TARDIEU S., « CORBA and CORBA Services for DSA », *Proceedings of SIGAda'99*, ACM Press, octobre 1999, <http://www.infres.enst.fr/~pautet/papers/pautet99adabroker.ps>.
- [BAK 01] BAKER S., « Middleware To Middleware », *Proceedings of the 3rd International Symposium on Distributed Objects and Applications (DOA'01)*, septembre 2001.
- [BAN 99] BANAVAR G., CHANDRA T., STROM R., STURMAN D., « A Case for Message Oriented Middleware », *Int'l Symposium on Distributed Computing*, 1999, p. 1-18.
- [BIR 83] BIRRELL A. D., NELSON B. J., « Implementing Remote Procedure Calls », *Proceedings of the ACM Symposium on Operating System Principles*, Bretton Woods, NH, 1983, Association for Computing Machinery, page 3.
- [DUM 98] DUMANT B., HORN F., TRAN F. D., STEFANI J.-B., « Jonathan : an Open Distributed Processing environment in Java », *IFIP Int'l Conference on Distributed Systems Platforms and Open Distributed Processing*, Londres, 1998, Springer Verlag, p. 175-190.
- [GAM 94] GAMMA E., HELM R., JOHNSON R., VLISSIDES J., *Design Patterns : Elements of Reusable Object-Oriented Software*, Addison Wesley, Massachusetts, 1994.
- [GEI 97] GEIB J., GRANSART C., MERLE P., *CORBA : des concepts à la pratique*, 1997.
- [IBM 97] IBM, « MQ Series », 1997, <http://www-3.ibm.com/software/ts/mqseries>.
- [ING 78] INGALLS D., « The Smalltalk-76 Programming System Design and Implementation », *Conference Record of the Fifth Annual ACM Symposium on Principles of Programming Languages*, Tucson, Arizona, janvier 1978, p. 9-16, <http://users.ipa.net/~dwithth/smalltalk/St76/Smalltalk76ProgrammingSyst%em.html>.
- [ISO 95a] ISO, *Information technology – Open Systems Interconnection – Basic Reference Model : The Basic Model*, ISO, février 1995, ISO/IEC 7498-1 :1994.

- [ISO 95b] ISO, *Information Technology – Programming Languages – Ada*, ISO, février 1995, ISO/IEC/ANSI 8652 :1995.
- [KEL 94] KELEHER P., « ThreadMarks : Distributed Shared Memory on standard workstations », *1994 Winter Usenix Conference*, 1994, p. 115-131.
- [KER 96] KERMARREC Y., NANA L., PAUTET L., « GNATDIST : A Configuration Language for Distributed Ada 95 Applications », *Proceedings of TRI-Ada'96*, ACM Press, 1996, p. 63-72, <http://www.infres.enst.fr/~pautet/papers/pautet96gnatdist.ps>.
- [MER 97] MERLE P., « CorbaScript - CorbaWeb : propositions pour l'accès à des objets et services distribués », Thèse de doctorat, Université des Sciences et Technologies de Lille (USTL), janvier 1997.
- [MPI95] MPI, Forum, « MPI : A Message-Passing Interface Standard », juin 1995, <http://www.mpi-forum.org/docs/mpi-11.ps.Z>.
- [ODP 95] ODP, « ODP Reference Model : Overview, ITU-T -- ISO/IEC Recommendation X.901 -- International Standard 10746-1 », 1995.
- [OMG 98] OMG, *The Common Object Request Broker : Architecture and Specification, revision 2.2*, OMG, février 1998.
- [PAU 00] PAUTET L., TARDIEU S., « GLADE : a Framework for Building Large Object-Oriented Real-Time Distributed Systems », *Proceedings of ISORC'00*, Newport Beach, California, USA, juin 2000, IEEE Computer Society Press.
- [PAU 01a] PAUTET L., « Intergiciels schizophrènes : une solution à l'interopérabilité entre modèles de répartition », Habilitation à Diriger des Recherches, Université Pierre et Marie Curie – Paris VI, décembre 2001, <http://www.infres.enst.fr/~pautet/papers/pautet01hdr.ps>.
- [PAU 01b] PAUTET L., QUINOT T., KORDON F., TARDIEU S., AZAVANT F., NIEBEL V., PONCE S., GINGOLD T., « PolyORB », 2001, <http://libre.act-europe.fr>.
- [QUI 99] QUINOT T., KORDON F., PAUTET L., « CIAO », 1999, <http://adabroker.eu.org/ciao>.
- [QUI 01] QUINOT T., PAUTET L., KORDON F., « Architecture for a reuseable object-oriented polymorphic middleware », *Proceedings of PDPTA'2001*, Las Vegas, Nevada, Etats-Unis, juin 2001, <http://www.infres.enst.fr/~pautet/papers/pautet01architecture.ps>.
- [SCH 95] SCHMIDT D., VINOSKI S., « Comparing Alternative Server Distributed Programming Techniques », *SIGS C++ Report*, vol. 7, n° 8, 1995.
- [SCH 96] SCHMIDT D. C., STAL M., ROHNERT H., BUSCHMANN F., *Patterns for Concurrent and Networked Objects*, vol. 2 de *Pattern-Oriented Software Architecture*, John Wiley & Sons, 1996.
- [SCH 97] SCHMIDT D., CLEELAND C., « Applying patterns to develop extensible and maintainable ORB middleware », *CACM*, vol. 40, n° 12, 1997, ACM Press.
- [SEI 99] SEINTURIER L., LAURENT A., DUMANT B., GRESSIER-SOUDAN E., HORN F., « A Framework for a Real-Time Communication Based Object Oriented Industrial Messaging Services », rapport, 1999, CNET, France Télécom-CNET.
- [SIN 98] SINGHAI A., SANE A., CAMPBELL R., « Quarterware for Middleware », *Proceedings of ICDCS'98*, IEEE, mai 1998, <http://choices.cs.uiuc.edu/singhai/Papers/icdcs98.pdf>.
- [SUN 99] SUN, « Java Message Service », 1999.
- [W3C00] W3C, « Simple Object Access Protocol (SOAP) 1.1 », mai 2000, W3C note, <http://www.w3.org/TR/SOAP/>.