République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur
et de la Recherche Scientifique
Université de Batna
Faculté des sciences de l'ingénieur
Département d'informatique

Université de Batna
Rue Chahid Boukhlouf Mouhamed El-hadi
05000 Batna www.univ-batna.dz

Thèse de Magistère

Thème

Définition d'un style architectural pour la description de systèmes logiciels à base de composants de type COTS, selon une approche « services WEB »

présentée par

Kamel MANSOURI

pour obtenir le diplôme de Magistère spécialité Informatique

Soutenue publiquement le 13 Novembre 2005 devant le jury composé de :

Dr. M. Benmouhammed	M.C. Université de Constantine, Présidant,
Dr. M.S.Kheriddine	C.C. Université de Batna, Rapporteur,
Dr. A.Zidani	M.C. Université de Batna, Examinateur,
Dr. M.K.Kholladi	M.C. Université de Constantine, Examinateur.

SEPTEMBRE 2004

Remerciement

Je remercie,

MOHAMMED SALAH KHIREDDIN
Pour m'avoir encadré, conseillé et soutenu pendant toute la durée de ce travail.

DJEMAI ARAR
Pour ses nombreux conseils, sa gentillesse et sa serviabilité.

TOUS LES PROFESSEURS DE L'INSTITUT INFORMATIQUE.

ABSTRACT

The development of big software applications is oriented toward the integration or interoperation of existing software components (like COTS and legacy systems) . This tendency is accompanied by a certain number of drawbacks for which classical approaches in software composition cannot be applied and fail. COTS-based systems are built in ad-hoc manner and it is not possible to reason on them no more it is possible to demonstrate if such systems satisfy important properties like Quality Of Service and Quality Attributes.

The recent works issued in web field allow the definition and the use of a complex web service architecture. Languages such as WSFL, XLANG and BPEL4WS support these architectures called Services Oriented Architectures.

The definition of software systems using these languages benefits some existing technical solutions such as SOAP, UDDI, etc., that permit the distribution, the discovery and the interoperability of web services.

However, these languages do not have any formal foundation. One cannot reason on such architectures expressed using such languages: properties cannot be expressed and the system dynamic evolution is not supported.

On the other hand, software architecture domain aims at providing formal languages for the description of software systems allowing to check properties (formal analyses) and to reason about software architecture models.

In this work, we proposes a formalisation of COTS-based system (their structure, their behaviours) using architectural styles. The ADL used is π-ADL (based on the π-calculus, supporting style description). The memory will also present our approach consisting in refining an abstract architecture to an executable and services-oriented one.

Keywords : COTS-based system, software architecture, styles, services oriented architecture.

Table des matières

CHAPITRE I. INTRODUCTION 1

I.1 Introduction a la problématique 3
I.2 Pourquoi les systèmes à base de COTS ? 3
I.3 Les COTS et les services WEB 3
I.4 L'approche centrée architecture 4
I.4.1 Présentation 4
I.4.2 Les enjeux des travaux sur l'architecture logicielle 5
I.4.2.1 Processus de conception devenu inefficace 5
I.4.2.2 Evolution vers un processus centré architecture 5
I.4.3 La notion de style architectural 6
I.4.3.1 Langage de Description Architecturale (LDA) 7
I.5 Problématique 7
I.6 Organisation du document 7

CHAPITRE II. ETAT DE L'ART 9

II.1 Les systèmes à base de COTS 10
II.1.1 Définition du COTS 10
II.1.2 Comparaison entre COTS et composant 11
II.1.3 Classification de COTS 11
II.1.4 Modélisation des systèmes à base de COTS 11
II.1.4.1 UML (Unified Modeling Language) 12
II.1.4.2 Les Langages de Description Architecturale (LDA) 12
II.1.5 Technologies et normes pour l'intégration de composant 12
II.1.5.1 SUN (EJB) 13
II.1.5.2 Microsoft(COM/DCOM) 14
II.1.5.3 OMG(CORBA) 14
II.1.6 Problèmes d'intégration des COTS 15
II.1.7 Bilan 16
II.2 Les services WEB 18
II.2.1 Pourquoi les services WEB dans notre approche? 18
II.2.2 Définition d'un service WEB 18
II.2.3 Langage de description de service WEB 19
II.2.3.1 La Grammaire de WSDL 19
II.2.4 SOAP 20
II.2.4.1 Définition 20
II.2.4.2 Grammaire SOAP 20
II.2.5 UDDI 21
II.2.5.1 Définition 21
II.3 Les workflow 22
II.4 Quelque workflow 24
II.4.1 WSFL d'IBM 24
II.4.1.1 Définition 24
II.4.1.2 Un exemple WSFL 24
II.4.1.3 Présentation générale de la syntaxe de WSFL 26
II.4.2 XLANG de Microsoft 27
II.4.2.1 Définition 27
II.4.2.2 Les actions XLANG 28
II.4.2.3 Les actions de contrôle XLANG 28
II.4.2.4 un exemple XLANG 30
II.4.3 BPEL4WS de IBM, Microsoft et BEA 30
II.4.3.1 Définition 30
II.4.3.2 Structure de BPEL4WS 31
II.4.3.3 Traitement d'erreurs 31

II.4.3.4 Exemple 31
II.4.4 Autres langages 34
II.4.4.1 WSCI de SUN 34
II.4.4.2 WSCL de HP 34
II.4.4.3 BPMLde BPMI 35
II.4.5 Bilan 35
II.5 Le serveur Biztalk 37
II.5.1 Introduction 37
II.5.2 Le moteur BizTalk Server 39
II.5.3 Relier les applications 40
II.5.3.1 Envoyer et recevoir des Messages: les adaptateurs 40
II.5.3.2 Traitement des messages : les pipelines 41
II.5.3.3 Choix des Messages: les Souscriptions (subscriptions) 42
II.5.4 Définition d'un Processus Métier 42
II.5.4.1 Orchestration 43
II.5.4.2 Le moteur des règles de gestion 45
II.5.5 Bilan 45
II.6 Les langages de description architecturale 47
II.6.1 Définition 47
II.6.1.1 Style 48
II.6.2 Présentation de quelque ADL 49
II.6.2.1 UNICON-2 49
II.6.2.2 AESOP 52
II.6.2.3 ARMANI 55
II.6.2.4 π-ADL 58
II.6.3 Bilan des ADL 60
II.6.4 Le langage formel π-CALCUL 62
II.6.4.1 Introduction : 62
II.6.4.2 Concept de Mobilité 62
II.6.4.3 Syntaxe 63
II.6.4.4 Sémantique 64
II.6.4.5 Le π-calcul asynchrone 64
II.6.4.6 Le π-calcul polyadique 65
II.6.4.7 Le π-calcule d'ordre supérieur 65
II.6.4.8 Bilan 65

CHAPITRE III. PROPOSITIONS 67

III.1 Rappel de la problématique 69
III.2 Proposition d'une architecture de référence 70
III.3 Définition des services principaux fournis par UDDI-COTS 71
III.4 Description des éléments de l'architecture 72
III.4.1 Service de publication (UDDI) 72
III.4.2 Service Contrôle de processus (UC) 75
III.4.3 Service orientation (OR) 77
III.4.4 Représentation externe du COTS 78
III.4.5 L'attachement des composants 82
III.5 Génération de code (π-ADL à XLANG) 83
III.6 Bilan 85
III.7 Formalisation d'un exemple avec nos styles architecturaux 87
III.7.1 Description de l'exemple 87
III.7.2 Création de l'architecteur du système 87
III.7.2.1 Projection du système selon la topologie 87
III.7.2.2 Description du système en π-ADL 88
III.7.2.3 Raffinement du système 89
III.7.3 Bilan 93

CHAPITRE IV. BILAN GENERALE 95

IV.1 Bilan de travail 97
IV.2 Conception de système 97

IV.3 Le raffinement 98
IV.4 Conclusion.. 98

REFERENCES BIBLIOGRAPHIQUES 101

ANNEXE A : DESCRIPTION DU STYLE 107

ANNEXE B : L'UC DE L'EXEMPLE EN XLANG 115

ANNEXE D : PUBLICATION REALISEE 125

Index des figures

Figure 1 Processus de developpement centre architecture 6
Figure 2 Style chateau 6
Figure 3 Architecture de la plate-forme EJB 13
Figure 4 Modele abstrait du composant CORBA............15
Figure 5 Modele de communication a travers un service WEB 18
Figure 6 L'information circule sous forme XML 19
Figure 7 Representation d'un UDDI............22
Figure 8 Modele de deploiement de service WEB 22
Figure 9 Les couches de protocole d'un service WEB 23
Figure 10 Le diagramme du processus metier totalSupplyFlow 26
Figure 11 XLANG importante une partie de WSDL............28
Figure 12 Modele de Gestion de stock en EAI............38
Figure 13 Modele de Gestion de stock en B2B 38
Figure 14 composant de moteur Biztalk............39
Figure 15 pipeline de reception 42
Figure 16 Representation d'une communication UniCon............49
Figure 17 Architecture Aesop 53
Figure 18 Representation d'un System Armani............55
Figure 19 Elements π-ADL............58
Figure 20 transmission de valeur dans π-calcul 62
Figure 21 mobilite dans π-calcul 63
Figure 22 les deux modeles proposes 70
Figure 23 Encapsulation d'un COTS 72
Figure 24 Representation graphique de UDDI............74
Figure 25 Representation graphique de UC............75
Figure 26 Representation graphique de OR............77
Figure 27 Unification de l'interface de COTS 78
Figure 28 le tableau de parametre utiliser par une interface WSDL 81
Figure 29 raffinement............84
Figure 30 diagramme de sequence 88
Figure 31 Representation du systeme selon notre architecture............88
Figure 32 Fiche de demande d'article............89
Figure 33 Flux de control suivi par le serveur BizTalk 92
Figure 34 Architecture centric development process 129
Figure 35 COTS wrappers............129
Figure 36 The reference architecture............131
Figure 37 Architecture centric development process 132

CHAPITRE I. INTRODUCTION

I.1 Introduction a la problématique

La tendance actuelle pour réaliser des applications de grandes tailles est orientée vers les systèmes modulaires (modules, objets ou composants), et pour les faire communiquer, on utilise des interfaces et des outils adéquats. En plus, la conception, le développement et la maintenance de ces applications posent des problèmes difficiles à résoudre. Les concepteurs sont confrontés à des contraintes de réutilisation de code existant, d'installation d'applications dans des contextes matériels et/ou logiciels qui peuvent varier avec le temps, des contraintes d'administration, d'évolution des applications, etc. La conception d'applications avec les langages de programmation classiques permet difficilement de respecter ces contraintes. Ces langages sont intéressants pour programmer des logiciels mais sont inadéquats pour décrire la structure et le comportement d'une application. Ainsi, des travaux portant sur les architectures logicielles sont nés les langages de description architecturale (LDA[1]).

I.2 Pourquoi les systèmes à base de COTS[2] ?

Il s'agit de considérer les logiciels de demain comme des assemblages d'outils logiciels déjà existants au sein des entreprises ou disponibles sur le marché plutôt que de nouvelles applications issues de longs et coûteux processus de développement [Verjus-2001] .

L'objectif est d'assembler un ensemble d'outils logiciels pour permettre d'exploiter les fonctionnalités propres à chacun d'eux. Cet assemblage doit être aussi simple que possible à mettre en œuvre et offrir des possibilités d'évolution en fonction des nouveaux besoins exprimés par les utilisateurs.

Au cours de ces dernières années, plusieurs travaux ont porté sur les approches orientées composants, c'est-à-dire création des applications par l'assemblage des composants.

Nous nous intéressons à l'assemblage de composants existants, disponibles (COTS).

Les composants qui entrent dans la construction des systèmes à base de COTS, sont généralement

- de grandes tailles,
- développes avec des langages différents,
- fonctionnent dans des milieux hétérogènes et bien souvent distribués.

I.3 Les COTS et les services WEB

Trois standards sont aujourd'hui utilisés pour développer des systèmes distribuer. CORBA/IIOP (Common Object Request Broker Architecture / Internet Inter-ORB Protocol), DCOM (Distributed Component Object Model) et RMI (Remote Method Invocation).

[1] LDA (langages de descriptions architecturaux), dans le reste de document, nous utilisons l'acronyme anglo-saxon ADL.
[2] COTS (Component of the shelf) : Composants logiciels à l'étagère (composants logiciels disponibles sur le marché). Nous détaillons ce concept dans les parties qui suivent.

Ces trois standards permettent d'accéder à une fonction d'une autre application sur une machine distante et ce, de la même manière que l'appel d'une fonction locale, bien souvent indépendamment des plates-formes et des langages utilisés.

Ces modèles fonctionnent indépendamment des plates-formes[1] et des langages de programmations, offrent des mécanismes de récupération d'espace mémoire, de sécurité, de gestion du cycle de vie des objets [Nicolle-2002], mais sont : complexes, peu compatibles avec les pare-feu, difficilement inter-opérables entre eux, donc ils restent souvent confinés à l'intérieur d'un seul domaine.

D'autre part, on trouve les services WEB: simple a l'implémenté, sa structure est basée sur XML, inter-opérable avec les autres standards, utiliser pour système de grand taille, orienté services. Or dans la plus part des systèmes à base COTS, ces derniers sont vus comme des fournisseurs de service.

Ainsi il nous paraît intéressant d'envisager une architecture orientée services WEB pour les systèmes à base de COTS.

I.4 L'approche centrée architecture

La recherche dans le domaine des architectures logicielles a permis d'introduire une nouvelle dimension à cette notion d'architecture. Ainsi [Luckman et al-1995] établissent qu' « une architecture spécifie les éléments d'un système et l'interaction entre ces éléments afin de satisfaire les besoins du système ».

I.4.1 Présentation

Par analogie au secteur du bâtiment, on peut établir un certain nombre de points communs intéressants, comme exemple l'utilisation d'un plan avant de construire le bâtiment. Ce plan fixe les méthodes et les matériaux à employer selon le but à atteindre et permet d'assigner les différentes tâches aux personnes compétentes. Dans cet objectif de partage du travail, on montrera sur des plans détaillés une seule partie de la construction : le plan des canalisations aux plombiers, le plan électrique aux électriciens, et ainsi de suite. Ces différentes vues de la construction permettent d'étudier, à chaque fois, un point particulier indépendamment des autres intervenants, tout en étant assuré que les canalisations, par exemple, seront placées au bon endroit à la fin. De la bonne conception du plan découle enfin des propriétés globales (résistance, poids, possibilité de rajouter des pièces, etc.) démontrées une fois pour toutes [Blanc-1999].

Comme pour l'architecture d'un bâtiment, une architecture logicielle représente à la fois la structure d'une application et le comportement de celle-ci. Ainsi, suivant le niveau d'abstraction auquel nous nous situons, les blocs de construction de notre architecture peuvent être modélisés par des composants qui suivent une configuration architecturale puis par raffinements[2] successifs, nous évoluons vers l'implémentation jusqu'à obtenir une application exécutable.

La réalisation classique de logiciels de grande envergure devient inefficace dès lors qu'elle est conçue d'un bloc, du cahier des charges à l'implémentation sans étapes de validation qui permettent de contrôler le respect des spécifications et des contraintes. L'architecture logicielle met en œuvre des mécanismes de

[1] L'indépendance n'est pas totale, nous trouvons le même mécanisme développer pour chaque plate-forme.

[2] Le raffinement consiste à passer d'un niveau de spécification abstrait à un niveau proche de l'implémentation par l'ajout successif de détails de conceptions.

formalisation qui offrent, tout au long du processus, la garantie du bon déroulement du processus.

I.4.2 Les enjeux des travaux sur l'architecture logicielle

Les travaux sur les architectures logicielles ont commencé il y a plus de dix ans aux Etats Unis avec pour objectif de décrire formellement des systèmes logiciels en termes de structure, de comportement et de tenter d'apporter aux besoins exprimés, des solutions formalisées et évolutives. La formalisation de ces architectures est nécessaire pour pouvoir exprimer et vérifier des propriétés telles que la complétude, la cohérence, la vivacité, la sûreté, etc. [Telisson-2002].

I.4.2.1 Processus de conception devenu inefficace

Le processus de conception classique d'une application qui consiste à implémenter le code exécutable directement après l'élaboration des spécifications à partir d'un cahier des charges devient obsolète et coûteux dès lors qu'on cherche à développer une application de grande envergure et très fiable.

En effet, dans ce type de conception, la tendance est de déployer l'application pour la valider avec le client. Si l'application n'est pas validée le cahier des charges est redéfini et le processus recommence. Les coûts engendrés atteignent rapidement des sommes importantes.

La solution pour tenter de pallier ces problèmes est d'adopter lors de la conception, un processus centré sur les architectures logicielles.

I.4.2.2 Evolution vers un processus centré architecture

Un processus de conception centré sur les architectures logicielles (Figure 1), introduit plusieurs notions à différents niveaux dans le processus de conception classique [Boehm-1986]:

- *élaboration* d'une architecture à un haut niveau d'abstraction,
- *validation* de l'architecture; si le comportement[1] n'est pas celui escompté, le cahier des charges est redéfini et le processus recommencé,
- *raffinement* de l'architecture : de nouvelles architectures de plus en plus détaillées sont construites en respectant les architectures d'un niveau d'abstraction plus haut,
- *validation* de chaque nouvelle architecture avant génération du code.

Cette approche permet d'effectuer des validations au fur et à mesure de la conception. Cela permet de s'approcher de l'implémentation étape par étape en respectant les comportements déjà validés. La construction des architectures moins abstraites se fait par des méthodes formelles assurant la conservation des propriétés lors du passage d'un niveau à un autre. Ainsi lorsqu'une erreur a été corrigée en amont elle ne peut pas apparaître plus tard [Bolusset et al-2000].

[1] La validation du comportement est réalisée à l'aide d'outils de simulation.

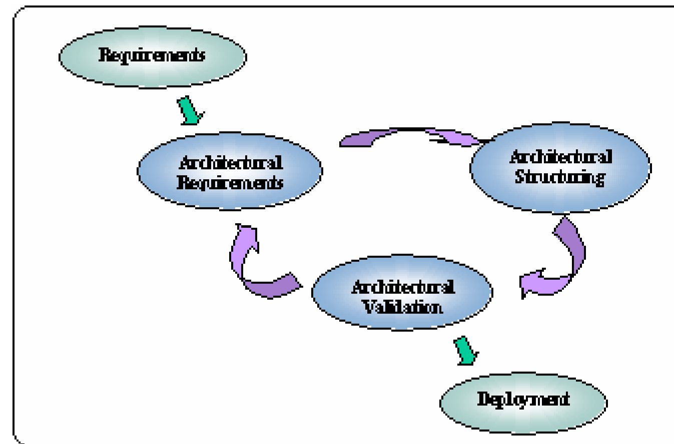


Figure 1 Processus de développement centré architecture

I.4.3 La notion de style architectural

Un style permet de définir une famille d'architectures qui sont reliées par des propriétés structurelles et sémantiques [Garlan-1995]. L'objectif de ces styles est d'une part de fixer des règles et des contraintes topologiques, d'autre part de permettre à terme la réutilisation de code dans de nouveaux problèmes.

Pour rester dans le domaine du bâtiment, prenons l'exemple donné par [Leymonerie-2001]. Nous définissons un style château (Figure 2). Ce style a des éléments structuraux, par exemple, des murs et des tours ainsi que des règles de configurations : les tours sont implantées sur les coins des murs.

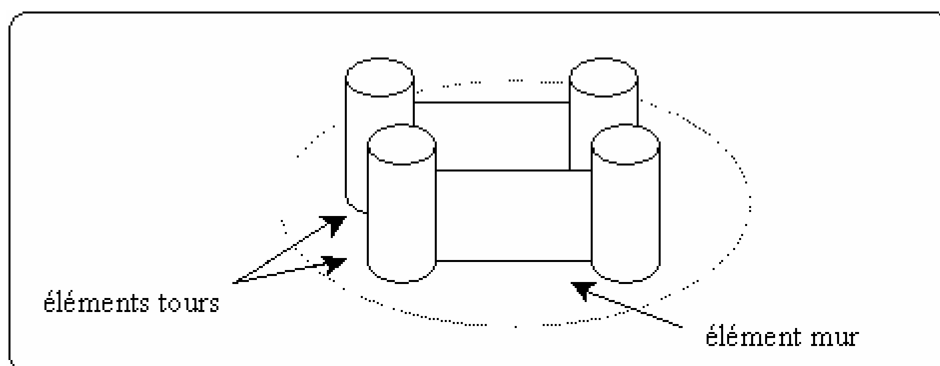


Figure 2 Style château

Nous verrons plus loin en abordant les langages de description architecturale qu'il existe une certaine hiérarchie entre les styles, ainsi nous parlons sur les sous style pour spécifier un style. Ce sous style doit respecter les contraintes et les règles de configurations imposées par le style de niveau supérieur. Dans l'exemple du château, un château fort est un sous style de château.

Afin de décrire les éléments des styles, il est nécessaire de mettre en œuvre un langage de description architectural.

I.4.3.1 Langage de Description Architecturale (LDA)

Les langages de description architecturale permettent de spécifier des architectures d'applications en offrant un moyen pratique et abstrait de description d'un système complexe [Medvicovic et al-1997]. Ils permettent d'exprimer la configuration d'une application en terme de composition d'unités logicielles élémentaires et d'éventuelles contraintes de déploiement.

Quelques langages de descriptions sont présentés ultérieurement dans ce document (section II.6).

I.5 Problématique

Les travaux portant sur les systèmes à base de COTS ont permis de concevoir de manière informelle des applications à base d'outils logiciels autonomes et hétérogènes sans établissement d'une architecture de référence.

Le cadre de notre travail a pour objectif de définir une topologie d'assemblage des systèmes à base de COTS, aussi définir des styles architecturaux qui permettront, aux architectes, de formaliser ces systèmes comme un ensemble de services WEB dont le fonctionnement et les interactions sont à définir.

I.6 Organisation du document

Ce document présente une solution pour la modélisation architecturale des systèmes à base de COTS.

Une première partie porte sur l'état de l'art dans là quelle en présente:

- Les COTS ou nous invoquons les travaux de recherches concernant l'assemblage de ces composants.
- Les services WEB ou nous choisissons un service qui pourra jouer un rôle plus efficacement, dans notre système.
- Les ADL (langages de descriptions architecturaux) afin de dégager le langage le plus approprié pour nos travaux.

La seconde partie porte sur notre solution proposée, où nous présentons l'approche que nous avons choisie, pour modéliser un style architectural pour les systèmes à base de COTS.

Enfin, nous conclurons sur une synthèse de notre travail puis nous mettrons en avant les perspectives en vu de la continuation de nos travaux.

Des annexes à ce document décrivent :

- Les différents éléments qui constituent le style architectural « annexe A ».
- Une description de l'unité de contrôle en Xlang
- En fin une publication réalisée dans le cadre de ce travail

CHAPITRE II. ETAT DE L'ART

II.1 Les systèmes à base de COTS

Cette partie présente les travaux effectués sur les systèmes à base de COTS: Essentiellement les travaux effectués par l'institut de génie logiciel (Software Engineering Institut) de Carnegie Mellon au USA. Leurs travaux portent sur les caractéristiques d'intégration de COTS en terme de coût, de difficulté, d'évolution, etc. Dans notre cas, nous avons restreint l'étude aux mécanismes d'intégration de COTS seulement.

II.1.1 Définition du COTS

Le terme de composant serait issu de l'évolution de la programmation logicielle. Les fonctions, méthodes ou procédures regroupées en « librairies » auraient abouties à la création de composants. On trouve plusieurs définitions au terme de composant logiciel.

- [Verjus-2001] parle « d'entités logicielles possédant certaines propriétés ».
- [Orfali et al-1999] définissent un composant comme « un fragment de logiciel assez petit pour qu'on puisse le créer et le maintenir, et assez grand pour qu'on puisse l'installer et en assurer le support ».
- [Medvicovic et al-1997] écrivent « un composant est une unité de calcul ou de stockage ».
- Selon la littérature, un COTS (Component of the shelf) signifiée un composant logiciel commercialisé ou composants logiciels à l'étagère. C'est-à-dire un produit d'habitude utilisé pour des usages généraux. Il peut signifier aussi que le COTS ne doit pas être développer ou modifier par l'utilisateur, mais soutenu et évolué par le fournisseur qui effectue la maintenance et gère son évolution. Ainsi que les droits de propriété intellectuelle [Maurizio-2000].

D'une manière générale, COTS est un composent (partie) logiciel disponible immédiatement pour l'intégration dans lequel aucun code source n'est accompagne (le code source et la documentation interne est habituellement indisponible Boîte noire), les composants logiciels qui incluent le code source modifiable sont considérés comme des composants de réutilisation (modes internes autorisés, Boîte blanche).

Le COTS peut être développé comme une bibliothèque ou comme un programme exécutable autonome. Clairement, le code source peut être modifié seulement s'il est disponible (pour paramétrer certaines tâches par exemple). Cependant, parfois même si le code source est disponible il n'est pas modifiable, mais il est employé seulement pour la documentation et la compréhension. Ainsi, Verjus a rajouté [Verjus-2001] [Verjus-2001] ces trois propriétés:

- qu'aucun contrôle sur les fonctionnalités ou la performance du produit logiciel ne peut être assurée,
- que les composants de type COTS ne sont souvent pas conçus pour interopérer avec d'autres,
- qu'aucun contrôle ne peut être assuré sur l'évolution du composant,
- que les comportements des distributeurs/éditeurs de composants de type COTS divergent largement (ils sont donc imprévisibles).

Un certain nombre de technologies, telles que COM, CORBA, Entreprise JavaBeans, ont été définies pour normaliser l'utilisation des composants. Et pour les rendre moins dépendants des langages de programmation, des environnements de développement, des logiciels d'exploitation et des protocoles de gestion de réseau.

II.1.2 Comparaison entre COTS et composant

Les composants ont évolué dans la communauté orientée objet. Ils signifient bien souvent des morceaux de programme relativement grands (comportant beaucoup de classes et fonctions). Ce sont donc des unités permettent de composer/ décomposer un système. Dans ce sens une définition plus formelle, est : un composant est une unité de composition avec des interfaces dépendent explicitement d'un contexte, où le contexte signifie le langage de programmation, le logiciel d'exploitation, l'interface utilisateur, les services de gestion de réseau, la base de données et les services de transaction. Dans le meilleur des cas, un composant peut être branché et être utilisé dans n'importe quel contexte; dans ce cas un récipient (appelé bien souvent conteneur) est chargé de cacher l'exécution de ces services tout en offrant aux composants des interfaces normalisées (comme CORBA, JavaBeans, EJB, COM/DCOM, etc.). En résumé quelques composants sont des COTS mais pas l'inverse [Maurizio-2000].

II.1.3 Classification de COTS

Carney [Carney-1997] a classifié les COTS selon:

- L'origine du COTS : c'est-à-dire un simple COTS ou une composition de plusieurs COTS.
- Les permissions de modification : accès (ou non) au code pour la révision (retouché étendu), pour modifier certaines fonctionnalités.
- La structure du COTS (Emboîtage) : le COTS peut être emboîté comme bibliothèque liée dynamiquement (EX : DLL), programme exécutable autonome ou être une combinaison des deux approches ci-dessus.
- Le type de fonction fourni par le COTS : selon ce critère, on peut classer les COTS dans deux catégories

    1) Horizontal : Les fonctions fournies par le COTS ne sont pas spécifiques à un domaine, mais peuvent être réutilisées à travers des différents domaines d'application (Ex : protocoles de gestion de réseau, ODBC, etc.).

    2) Vertical : les fonctions fournies par le COTS sont spécifiques a un domaine (Ex : comptabilité, planification de ressource, commande satellite,…).

II.1.4 Modélisation des systèmes à base de COTS

La conception des systèmes à base de COTS est modélisée selon les architectures composant/connecteur, parmi ces architectures on peut mentionner :

II.1.4.1 UML (Unified Modeling Language)

OMG a crée un méta modèle composant/connecteur qui repose sur quatre modèles spécifiques[Riveill-2000]:

- **un modèle abstrait** qui étend la notion d'objet CORBA en introduisant le concept de composant. Ce concept prolonge celui des objets dans les voies de la modularité, de l'encapsulation et de la réutilisation. Plusieurs interfaces peuvent être associées à un composant. Proches de celles utilisées pour décrire les objets, elles sont plus complètes puisqu'elles contiennent aussi les événements pouvant être émis par les composants et ceux pouvant être reçus;

- **un modèle de programmation** qui fixe un cadre de développement des composants et offre un langage pour la description des propriétés non fonctionnelles (multi-threading, transaction, persistance, sécurité) ;

- **un modèle de déploiement** qui permet de décrire la manière d'assembler et de déployer un ensemble de composants pour construire une application ;

- **un modèle de container** qui définit les interfaces d'une machine virtuelle permettant l'exécution des composants. Les containers prennent en charge la gestion des transactions, de la sécurité et de la persistance. Les containers assurent aussi la diffusion des événements émis.

- **Un langage de script (IDL Script)** permettant de réaliser dynamiquement des assemblages de composants complète ces modèles.

Malgré cette richesse d'UML (concept de méta modèle), aussi son moment d'inertie (norme pour modéliser un problème, indispensable pour rédiger un cahier de charge, etc.), sa faiblesse majeure est sa base, qui est semi-formelle. Ce qui ne donne pas une vérification de système.

II.1.4.2 Les Langages de Description Architecturale (LDA)

C'est un langage qui sert à décrire la structure logicielle d'une application en termes de composants interagissant. Il permet de s'attacher uniquement à la description de l'architecture logicielle d'une application sans prendre en considération les détailles algorithmiques (le point fort du point de vue modélisation des systèmes à base des COTS). Cette structure, textuelle ou graphique, formelle ou semi formelle, constitue un élément important à la documentation de développement, permet des vérifications de propriétés ainsi que la génération des squelettes de l'application dans un langage de programmation cible [Maurizio-2000]. Nous détaillerons les concepts de ces langages plus tard (section II.6).

II.1.5 Technologies et normes pour l'intégration de composant

Enormément d'efforts ont été consacrés à la normalisation des interactions entre les composants. Parmi ces travaux, nous trouvons les modèles proposés par SUN, OMG, et Microsoft.

II.1.5.1 SUN (EJB)

SUN a créé la spécification EJB, et a propose une architecture permettant de construire une application en Java dont la partie serveur est construite à partir de composants appelés "Entreprise Beans (EB)" [Sun-1999].

II.1.5.1.1 Vue générale

En général, le modèle EJB n'impose pas d'implémentation, il définit des patrons de conception que chaque programmeur doit respecter. Ceux-ci identifient différents blocs fonctionnels, définissent des interfaces et la manière de les utiliser. Les composants du modèle EJB s'exécutent au sein d'un support d'exécution appelé EJB Container (conteneur), lui-même contenu dans un EJB Server (serveur EJB).

Deux interfaces principales sont spécifiées : l'interface entre un client et un composant serveur, et l'interface entre un composant serveur et le conteneur dans lequel il est déployé.

La première permet à un client (qui peut être un composant serveur) de créer ou de chercher un composant, d'interagir avec lui, et de le détruire.

La deuxième interface permet au conteneur de gérer les composants qu'il contient. Il peut ainsi rendre passif un composant (c'est-à-dire le stocker temporairement sur disque pour faire de la place en mémoire) ou au contraire demander à un composant de mettre à jour sa représentation persistante.

II.1.5.1.2 Vue détaillée

Un composant est lui-même composé de trois objets distincts appelés EJB Home (usine), EJB Object (relais) et EJB Instance (instance), comme le montre la

Figure 3. L'EJB Instance est une instance d'une classe Java fournie par le développeur du composant. Cette classe implémente l'interface entre le client et le conteneur. L'usine et le relais sont deux objets, générés par la plate-forme EJB, qui implémentent uniquement l'interface d'accès au composant par un client. L'usine regroupe les méthodes de création, recherche et destruction. Elle est commune à toutes les instances de ce type de composant. Le relais regroupe les méthodes applicatives du composant (business methods). Le client n'interagit jamais directement avec l'instance. Toutes ses requêtes passent par l'usine ou le relais, qui délèguent ensuite leur traitement soit vers l'instance, soit vers le conteneur. Ce mécanisme d'interposition permet d'exporter au client uniquement la partie qui le concerne et de cacher à celui-ci les méthodes liées à la gestion du composant qui sont exploitées par le serveur.

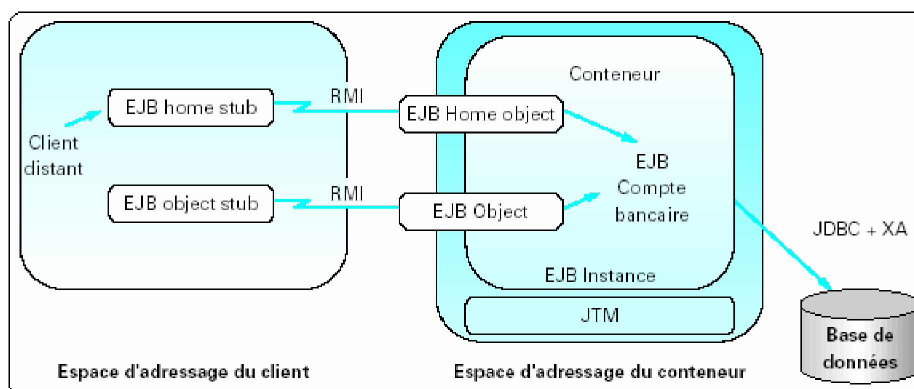


Figure 3 Architecture de la plate-forme EJB

Les applications complexes en Java deviennent, avec cette approche, plus "simples" à écrire puisque plusieurs aspects (transactions, persistance, sécurité, répartition) sont déjà gérés. D'autre part, la plate-forme et le bus logiciel sont indépendants des applications [Verjus-2001] .

II.1.5.2 Microsoft(COM/DCOM)

Microsoft a créé le modèle COM (Component Object Model) pour permettre la construction de composants binaires réutilisables, accessibles à travers une ou plusieurs interfaces. Celles-ci permettent l'accès au composant qui peut être construit à partir de composants existants par un processus de délégation[1] ou d'agrégation[2]. La définition des composants faite au niveau du binaire et sans modification du logiciel permet de ne pas se préoccuper du langage initial de programmation du composant [Riveill-2000]. Puis elle a étendu ce modèle avec DCOM/COM+ pour permettre l'accès à des objets COM quelle que soit leur localisation physique. La manière de procéder est proche de celle utilisée dans les modèle d'appel de procédure à distance (RPC). Ce modèle impose des contraintes technologiques assez lourdes (environnement homogène, langages supports, et structure binaire surtout) pour le développeur, mais permet cependant de faire inter-opérer des composants et produits logiciels fournis par des éditeurs différents [Riveill-2000].

II.1.5.3 OMG(CORBA)

La spécification CORBA était destinée à l'interopérabilité de composants en milieux hétérogènes. Plusieurs produits commerciaux implémentant cette spécification ont vu le jour, avec un certain succès d'ailleurs. Au fil du temps, des groupes de travail se sont constitués dont les objectifs étaient d'enrichir la spécification et l'ajout de nouvelles fonctionnalités dont les objets métiers, les interfaces de domaines, etc.

Les principales limitations de la spécification CORBA sont les suivantes :

- Aucune proposition standard pour le déploiement des implémentations des objets ;
- Limitation des possibilités d'extension des fonctionnalités des objets ;
- La disponibilité des services objet CORBA n'est pas définie à l'avance ;
- Pas de gestion standard du cycle de vie des objets.

A la suite de plusieurs remarques sur l'absence de spécifications sur les composants d'une part, et l'émergence de propositions concurrentes pour la spécification de composants (EJB, COM/DCOM) d'autre part, l'OMG (Object Management Group) a décidé de définir une spécification des composants qui viendrait enrichir la spécification CORBA 3 : c'est le modèle de composant CORBA ("CORBA Component Model" - CCM). Ce modèle prend en compte les aspects diffusion et déploiement d'applications. Nous notons que la proposition sur les composants CORBA s'est fortement inspirée du modèle EJB de Sun (notons d'ailleurs que les deux propositions sont entièrement compatibles) [Riveill-2000].

[1] Délégation permet à un composant d'implémenter une partie de son interface par un composant interne. L'interface de l'objet interne n'est pas visible.
[2] Agrégation permet à un composant d'exporter dans son interface, l'interface d'un objet interne. L'interface de l'objet interne est hébergée sur différents serveurs

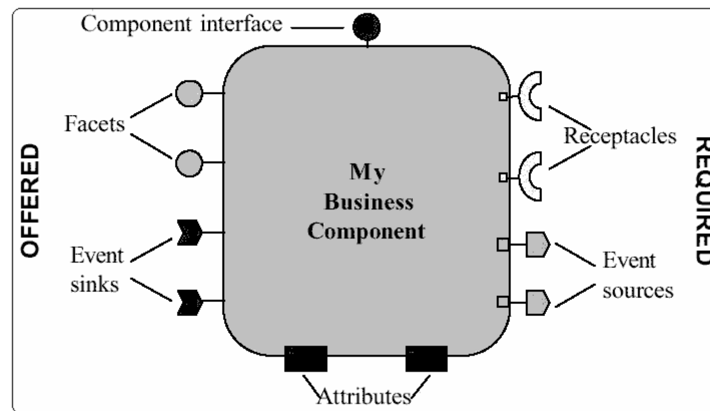


Figure 4 Modèle abstrait du composant CORBA

II.1.6 Problèmes d'intégration des COTS

Le problème principal d'intégration des COTS dans un système, est la divergence de différentes architectures de COTS qui composent le système [Davis-1997], ce qui rend l'interaction entre les COTS difficiles. Deux catégories principales d'erreurs peuvent être distinguées dans les interactions entre les COTS :

- **Syntaxique** : la représentation des règles syntaxiques de l'interaction, par exemple, le nom de la fonction appelée; les types, et l'ordre des paramètres.

- **Sémantique-pragmatique** : définit les spécifications sémantiques et pragmatiques de l'interaction, c'est-à-dire, quelle fonctionnalité est exécutée par le composant et comment. Par exemple, appelant la fonction «SQRT» calcule la racine carrée de son seul argument et renvoie le résultat à l'appelant. Cependant, quelle fonction «SQRT» doit-on appeler? Celle des entiers ou celle des réels? Le COTS est-il capable de choisir la fonction adéquate? Dans ce domaine on ne peut pas séparer les deux vues (sémantique, pragmatique) [Maurizio-2000].

Certains fournisseurs affirment que leurs COTS ne causent pas de problème, puisqu'ils suivent une norme standard bien spécifique, tel que CORBA. Cette réponse, même si elle comporte un peu de vérité. Les effets secondaires générés par les différentes interactions ne sont pas simplement les résultats de méthodes de communication, mais aussi aux :

- Capacités des COTS qui composent le système, tel que par exemple : Un COTS ne gère qu'un nombre limité de threads (programme fils) à un temps donné Ou, un COTS ne supporte pas une charge qui dépasse certain limite pour une ressource donnée (Ex : cinq requêtes SQL par seconde).

- Communication entre sous-ensembles qui composent les COTS: Un COTS composer qui communique avec un autre COTS simple ce qui engendre un problème de synchronisation aux niveaux des données partagé (la bijection est violé : le COTS simple suppose qu'il communique avec un sel COTS, mais en réalité il communique avec plusieurs COTS). Ou deux COTS

simples ne savent pas qu'ils sont entrain de partager une ressource à un seul point d'entré.

Ces deux derniers problèmes sont nommés « architectural mismatches » [Allen-1996][Abd-Allah-1996].

Certaines de ces failles peuvent être détectées par les caractéristiques architecturales du domaine. Telles que les différences de types dans les interfaces de domaine additionnels (unités, systèmes du même rang, fréquences), qui dépassent les spécifications générales de l'interface (EX : CORBA) [Cristina-1999].

Un travail réalisé par Cris Tina Gacé et Barry Boehm [Cristina-1999] pour résoudre certains problèmes de ce type. Ils sont basés sur l'utilisation de la méthode formelle Z pour construire des dispositifs conceptuels qui aident à décrire le système et analyser leur comportement.

Après une analyse de divers modèles architecturaux et leurs descriptions communes [Shaw-1996], ils ont conçu un ensemble de dispositifs conceptuels pour représenter les systèmes distribués, a base d'évènements, à boucle fermé (closed-loop feedback control), à Couches (Layering), procédurale, orienté objet, tuyau et filtrer (pipe and filtre), tableau noir (blackboard), temps réel, etc.

Bien que ces dispositifs aient une grande utilité pour l'évaluation des risque dans un système à base de COTS très tôt, leur utilisation dépend de la description interne donné par le fournisseur, qui est dans leur avis des données secrets qui peuvent réduire les avantages commerciaux du COTS par apport à ces concurrents.

II.1.7 Bilan

Nous venons de voir les COTS et leurs caractéristiques. Ainsi un ensemble d'architectures pour modéliser les systèmes à base de COTS.

Au vu des concepts propres au système à base de COTS présentés dans cette partie, l'architecture doit satisfaire les critères suivants :

- Type d'interaction entre les COTS,
- L'architecture d'interaction entre les COTS,
- Contrôle de système,
- Les types d'interface.
- Permettre de spécifier des contraintes et des propriétés,

Nous résumons ces architectures dans le tableau suivant :

Modèle de COTS	COM	EJB	CORBA	UML
Interaction des COTS	RPC (DCOM)	RMI	Bus CORBA	/
Architecture	Point à point	Client/serveur	/	Méta modèle
Contrôle de système	Anarchie	Centraliser	/	/
Interface	Binaire	Machine virtuelle	Machine virtuelle	Pas d'implémentation
Spécification des contraintes	Non	Non	Non	Informelle (OCL)

Selon ce tableau, nous voyons que la technologie UML n'a pas un modèle d'implémentation, elle serve seulement pour modéliser l'architecture. Ainsi son caractère semi-formel n'autorise pas la vérification des propriétés du système.

Pour les trois autres architectures, la comparaison peut être faite à deux niveaux :

JavaBeans est un environnement Java, permettant d'écrire des composants logiciels s'exécutant sur n'importe quelle station. L'accent est mis sur les qualités de portabilité plutôt que d'efficacité. Dans la solution EJB, un constructeur propose une interface; d'autres constructeurs réalisent des implémentations de cette interface et proposent éventuellement des extensions au modèle d'origine. La portabilité est assurée par l'unicité du langage de programmation et par la spécification précise des différentes interfaces (interface interne à la plate-forme EJB mais aussi interface propre à la machine virtuelle Java dans le cas présent).

COM/DCOM est un environnement pour plate-forme Windows, où l'intégration se fait au niveau du code binaire lui-même. L'effort a porté sur la possibilité de programmer dans de très nombreux langages des composants COM, de les intégrer au niveau du code objet pour construire de nouvelles applications efficaces. Dans la solution Microsoft, un constructeur propose seul un ensemble de solutions et décrit comment les utiliser. La portabilité est assurée par l'unicité du matériel, du système d'exploitation et l'intégration peut se faire au niveau du binaire.

CORBA favorise la possibilité d'écrire des composants dans différents langages de programmation et d'interconnecter des composants qui s'exécutent sur n'importe quel type de plates-formes. Dans la solution CORBA, on a affaire à un processus ouvert ou chacun peut proposer des interfaces, les faire adopter et mettre en œuvre des interfaces précédemment adoptées. La portabilité des solutions n'est pas toujours garantie, mais l'interopérabilité de l'ensemble par un bus CORBA est possible. À chaque instant, de nouveaux services peuvent être ajoutés, remplacés, utilisés. Le processus d'évolution est ouvert.

Malgré la richesse proposer par chaque architecture, l'interopérabilités entre les COTS issus de ces architectures est très difficile. Et puisque le fruit de notre étude est de faire communiquer des COTS issus de marché, c'est-à-dire développer selon des architectures différant, ce critère potentiel défavorise ces trois architectures. D'autre part nous trouvons les services WEB indépendant des plat-formes et facilement inter-opérable avec les autres architectures. Dans la partie qui suive nous allons étudier cette technologie de très proche.

II.2 Les services WEB

Dans cette partie, nous présentons les services WEB et leurs relations avec notre étude. Ensuite nous choisissons un pour nos travaux.

II.2.1 Pourquoi les services WEB dans notre approche?

Comme nous avons dit, les COTS sont hétérogènes et distribués. Pour les faire communiquer, on utilise des protocoles d'interactions qui peuvent cacher cette hétérogénéité d'une par, et d'autre par cacher l'aspect distribué du système. Puisque les services WEB: sont simple à l'implémenter, leurs structures basées sur XML, inter opérable avec les autres standards, compatibles avec les pare-feu, etc.

Pour ces raisons, nous avons choisi les services WEB pour jouer le rôle de glue dans notre étude.

II.2.2 Définition d'un service WEB

Les services WEB sont des composants logiciels encapsulant des fonctionnalités et accessibles via des protocoles standards du WEB. Le service réalise, à la demande sur le WEB, une tâche bien définie. On propose ici les DTD (Document Type Definition) les plus utilisé pour les applications orientées workflow : XLANG par Microsoft, WSFL par IBM et d'autres formalismes tel que WSCI de SUN, WSCL de HP et BPML de BPMI

Les deux premier (WSFL, XLANG) sont défini au moyen d'un langage standard WSDL (WEB Service Definition Language), précisant les méthodes pouvant être invoquées, leur signature, et les points d'accès du service (URL, port, etc.).

Ces méthodes sont accessibles via le protocole standard SOAP (Simple Object Access Protocol). La requête et la réponse sont des messages XML transportés par HTTP. A fin de localisé un Service WEB enregistré, on utilise le protocole standard UDDI (Universal Discovery, Definition & Integration) [Nicolle-2002].

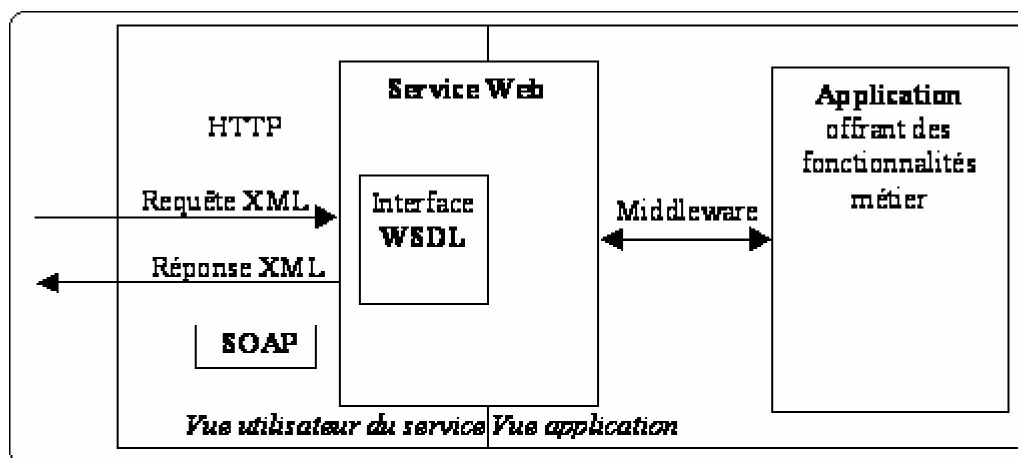


Figure 5 Modèle de communication a travers un service WEB

Un service WEB est accessible depuis n'importe quelle plate-forme (WAP, WEB...), ou langage de programmation. On peut utiliser un service WEB pour exporter des fonctionnalités d'une application et les rendre accessibles via des protocoles standard.

Le service WEB sert alors d'interface d'accès à l'application, et dialogue avec elle.

II.2.3 Langage de description de service WEB

Le WEB Service Description Language (WSDL) est une grammaire dérivée d' XML. Elle permettant de fournir les spécifications nécessaires à l'utilisation d'un service WEB, en décrivant les méthodes, les paramètres et ce qu'il retourne. WSDL est donc un langage de description des services[Donsez-2002].

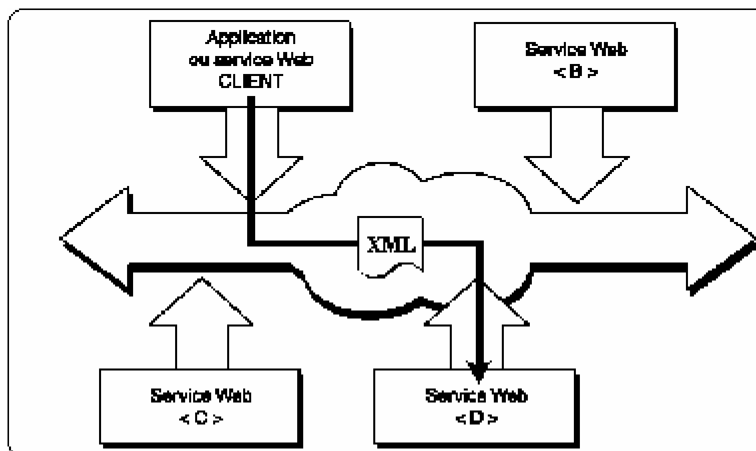


Figure 6 L'information circule sous forme XML

Le fichier WSDL présente l'interface publique du service et permet la génération automatique des stubs et skeletons des proxy à l'aide des outils automatiques.

II.2.3.1 La Grammaire de WSDL

Nous décrivons le schéma XML du langage WSDL

<types> : Contient les définitions de types utilisant un système de typage (comme XSD).

<message> : Décrit les noms et types d'un ensemble de champs à transmettre(Paramètres d'une invocation, valeur du retour, ...)

<porttype> : Décrit un ensemble d'opérations. Chaque opération a zéro ou un message en entrée, zéro ou plusieurs message de sortie ou de fautes

types d'opérations :

One-way : Le point d'entrée reçoit un message (<input>).

Request-response : Le point d'entrée reçoit un message (<input>) et retourne un message corrélé(<output>) ou un ou plusieurs messages de faute (<fault>).

Solicit-response : Le point d'entrée envoie un message (<ouput>) et reçoit un message corrélé (<input>) ou un ou plusieurs messages de faute (<fault>).

Notification : Le point d'entrée envoie un message de notification (<ouput>)

Les champs des messages constituent les paramètres (in, out, inout) des opérations.

<binding> : Spécifie une liaison d'un **<porttype>** à un protocole concret (SOAP1.1, HTTP1.1, MIME, ...). Un porttype peut avoir plusieurs liaisons.

<port> : Spécifie un point d'entrée (endpoint) comme la combinaison d'un <binding> et d'une adresse réseau.

<service> : Une collection de points d'entrée (endpoint) relatifs.

II.2.4 SOAP

II.2.4.1 Définition

SOAP est un protocole léger basé sur XML, utilisé pour échanger des informations. Il est composé d'une enveloppe décrivant le message, en particulier le contenu, le destinataire, et la façon dont le message doit être traité par ce dernier. Il dispose d'un ensemble de règles d'encodage pour les types de données spécifiques à l'application, d'une convention pour représenter les appels de procédure distante et leur réponse.

Ces informations sont stockées dans un paquet de type MIME qui peut être transmis sur HTTP ou un autre protocole WEB [Adolphe-2002].

II.2.4.2 Grammaire SOAP

```
//enveloppe du message

<soap:Envelope

//espace de nommage pour l'enveloppe SOAP

 xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/"

//on définit le type d'encodage du message

SOAP:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"

//espace de nommage pour les types de variables

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:xsd="http://www.w3.org/2001/XMLSchema"

//espace de nommage pour l'appel de méthodes
xmlns:tns="http://soapinterop.org/">

//corps <soap:Body>
```

```
//espace de nommage pour la fonction getStateName <m:getStateName

xmlns:m="http://soapware.org/">

//on spécifie un argument à passer à getStatName <statenum

xsi:type="xsd:int">41</statenum>

</m:getStateName>

// gestion d'erreurs <soap:fault> </soap:fault>

</soap:Body> </soap:Envelope>
```

II.2.5 UDDI

II.2.5.1 Définition

Universal Description, Discovery and Integration (UDDI) a été créé par IBM, Microsoft et Ariba. C'est une architecture répartie qui permet aux fournisseurs de services WEB (Business providers), d'enregistrer leurs services, et aux applications de rechercher les services correspondant à leurs besoins, de façon normalisée.

UDDI est donc un annuaire distribué de services WEB. Cette architecture se comporte elle-même comme un service WEB dont les méthodes sont appelées via le protocole SOAP.

UDDI est composée de deux parties :

- L'UDDI Business Registry: annuaire d'entreprises et de services WEB contenant des informations, au format XML, organisées en trois catégories [Lomme et Martin-2001] :

    1. Les Pages blanches: noms, adresses, contacts, identifiants, des entreprises enregistrées. Ces informations sont décrites dans des entités de type Business Entity. Cette description inclut des informations de catégorisation permettant de faire des recherches spécifiques dépendant du métier de l'entreprise

    2. Les Pages jaunes donnent les détails sur le métier des entreprises, les services qu'elles proposent. Ces informations sont décrites dans des entités de type Business Service.

    3. Les Pages vertes regroupant les informations techniques sur les services proposés. Les pages vertes incluent des références vers les spécifications des services WEB, et les détails nécessaires à l'utilisation de ces services (interfaces implémentées, information sur les contacts pour un processus particulier, description du processus en plusieurs langages, catégorisation des processus, pointeurs vers les spécifications décrivant chaque API).

- Les interfaces d'accès à ces annuaires, et les modèles de données [Donsez-2002].

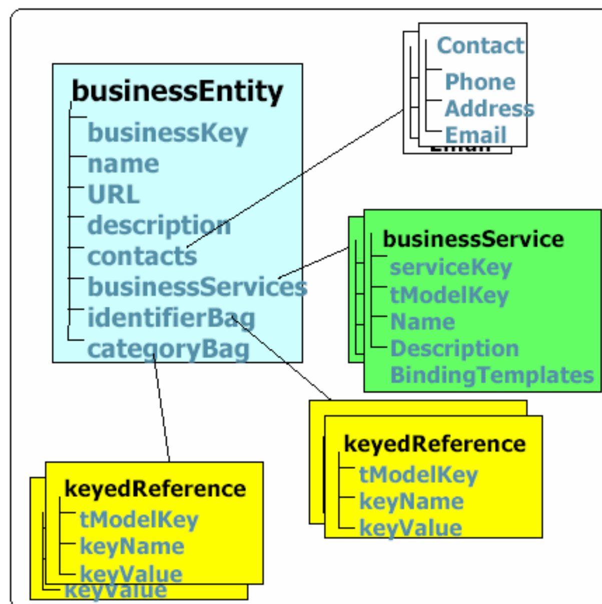


Figure 7 Représentation d'un UDDI

II.3 Les workflow

L'un des buts principaux des services WEB est de permettre aux développeurs d'intégrer leurs applications et leurs services avec tout le monde, indépendamment des systèmes d'exploitation sur lesquels fonctionnent leurs applications, indépendamment du modèle objet dont elles dépendent et indépendamment du langage avec lequel elles sont écrites.

Les développeurs ne peuvent faire cela que s'ils ont à leur disposition un modèle de programmation des services WEB. La figure suivante présente ce modèle.

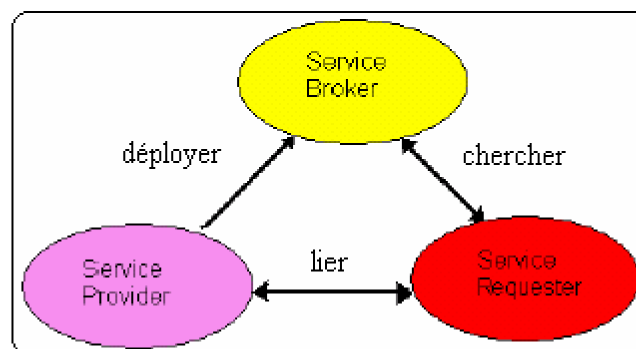


Figure 8 Modèle de déploiement de service WEB

Il est articulé autour de trois composants:

- Le Service Broker est un intermédiaire entre les deux autres. Il permet au fournisseur du service de publier la description de son service. Le client y trouve les informations nécessaires à l'utilisation du service publié.

- Le Service Provider met à la disposition des clients potentiels ses services et les publie.

- Le Service Requester cherche les informations auprès de l'intermédiaire et s'il trouve un service WEB conforme à ses attentes, se met en relation avec le fournisseur de service.

L'utilisation récursive de ce modèle de programmation lorsque le service invoqué fait lui-même appel à d'autres services WEB est une réelle innovation. D'autre part, cet aspect est totalement masqué aux yeux du client final. On le voit, les services WEB peuvent interagir de façon complexe.

L'intégration des services entre-eux pose cependant quelques problèmes:

- Processus partenaires multiples: Lors de l'intégration il faut tenir compte des temps d'exécution de chaque service WEB. Si un service prend trop de temps il peut perturber l'ensemble de la chaîne et il peut être judicieux de le remplacer par un service équivalent. Cela nécessite de mettre en place un procédé de découverte dynamique des services et une procédure de contrôle de processus.

- Contrôle des transactions: Lors de l'utilisation simultanée de plusieurs services, il peut arriver que l'un d'entre eux n'aboutisse pas. Dans cette éventualité, certaines opérations doivent être annulées ou différées tant que la procédure défaillante n'aura pas abouti avec succès. C'est par exemple le cas du paiement électronique qui doit simultanément débiter le compte du client et créditer celui du fournisseur.

- Gestion des exceptions: Quel que soit le processus en cause, l'application qui exploite plusieurs services doit connaître le service responsable et agir en conséquence.

Pour résoudre ces problèmes, des laboratoires, tel que IBM, Microsoft, etc., ont développé le concept de "workflow" et proposé des outils logiciels supportant leurs nouveaux langages de gestion des services WEB. Ces langages sont basés sur XML et utilisent les technologies présentées auparavant (Figure 9).

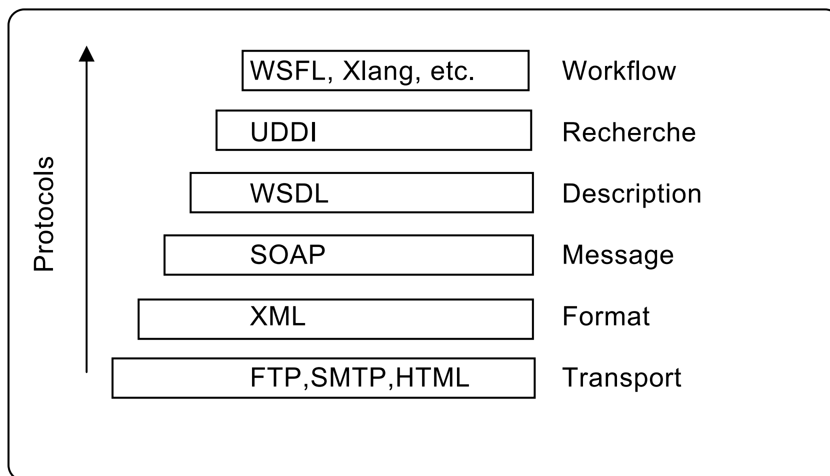


Figure 9 Les couches de protocole d'un service WEB

II.4 Quelque workflow

II.4.1 WSFL d'IBM

II.4.1.1 Définition

WSFL est une proposition de standard, par IBM [Leyman-2001], offrant deux styles de composition de services WEB :

- La description explicite de la succession des étapes et de l'enchaînement des appels aux opérations des services WEB, appelée processus métier.
- Un modèle d'interactions de services WEB pris deux à deux, le contrat. C'est le premier cas qui se rapproche le plus de la procédure au sens des langages de programmation. WSFL utilise l'expression modèle de flux (flow model) pour désigner ce style de spécification qui s'apparente à la programmation dans un langage de script. Le second cas, appelé modèle global (global model) dans WSFL, décrit une collection de liens entre opérations de services WEB (en WSDL), prises deux à deux, sans indiquer de structure de contrôle explicite.

WSFL permet de créer des modèles récursifs : une composition de services WEB est elle-même considérée comme un service WEB, utilisable à son tour dans d'autres compositions. Enfin, WSFL propose des modèles d'interactions dit hiérarchique et de point à point, reflétant des modes d'organisation différents [Chauvet-2000].

II.4.1.2 Un exemple WSFL

L'exemple suivant, illustration du modèle de flux, représente un processus métier simplifié, une prise de commande, mettant en jeu un fournisseur et un transporteur dont on suppose, pour chacun d'entre eux, que leurs services sont accessibles via des services WEB WSDL, nommés ici Supplier et Shipper.

```
<flowModel name="totalSupplyFlow" serviceProviderType=
"totalSupply">
<serviceProvider name="vendeur" type="supplier">
<locator type=”static” service=”qualitySupply.com”/>
</serviceProvider>
<serviceProvider name="transporteur" type="shipper">
<locator type=”static” service=”worldShipper.com”/>
</serviceProvider>
<activity name=”processPO”>
<performedBy serviceProvider=” vendeur”/>
<implement>
<export>
<target portType=”totalSupplyPT”
operation=”sendProcOrder”/>
</export>
</implement>
</activity>
```

```
<activity name="acceptShipmentRequest">
<performedBy serviceProvider="transporteur"/>
<implement>
<export>
<target portType="totalSupplyPT"
operation="sendSR"/>
</export>
</implement>
</activity>
<activity name="processPayment">
<performedBy serviceProvider="vendeur"/>
<implement>
<export>
<target portType="totalSupplyPT"
operation="sendPayment"/>
</export>
</implement>
</activity>
<controlLink source="processPO" target=
"acceptShipmentRequest"/>
<dataLink source="processPO" target=
"acceptShipmentRequest">
<map sourceMessage="anINVandSR" targetMessage="anSR"/>
</dataLink>
</flowModel>
```

Ce document WSFL (simplifié) montre un processus métier (flowModel) orchestrant deux rôles (serviceProvider) dans trois activités (activity). Un lien de contrôle (controlLink) indique la succession des deux premières activités et le lien de données (dataLink) spécifie les données qui sont échangées dans cet enchaînement. La description simplifiée de l'activité indique quel rôle doit l'implémenter (performedBy) et nomme, dans l'élément export, l'opération et le port du service WSDL correspondants. La figure ci-dessous présente un diagramme équivalent aux spécifications de cet exemple.

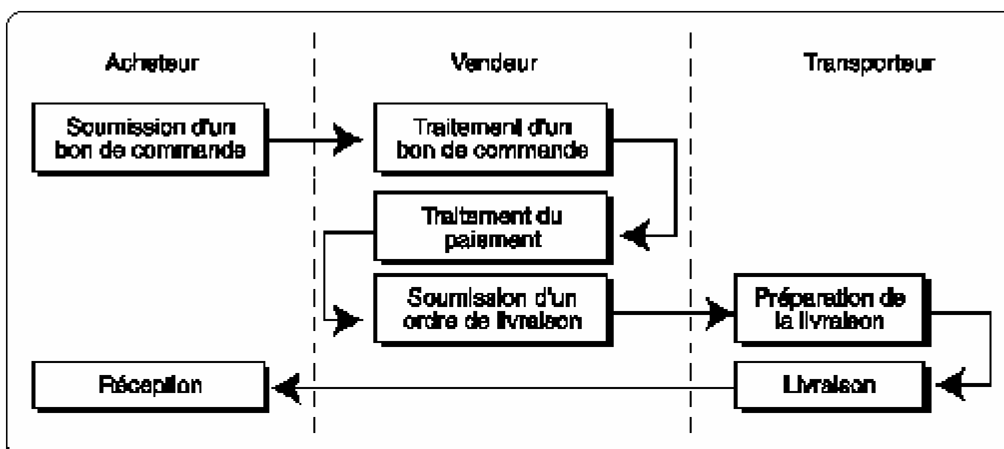


Figure 10 Le diagramme du processus métier totalSupplyFlow

```
<globalModel name="maCommande"
serviceProviderType="supplyChain">
<serviceProvider name="leFournisseur" type="supplier"/>
<serviceProvider name="leTransporteur" type="shipper"/>
<serviceProvider name="leTraitementCommande" type="totalSupply">
        <export>
                <source portType="supplyLifeycle"
operation="spawn"/>
                <target portType="manageChain"
operation="order"/>
        </export>
</serviceProvider>

<plugLink>
        <source serviceProvider=" leTraitementCommande"
portType="totalSupplyPT" operation="sendProcOrder"/>
        <target serviceProvider=" leFournisseur" portType="suppSvr"
operation="procPO"/>
</plugLink>

<plugLink>
        <source serviceProvider=" leTraitementCommande"
portType="totalSupplyPT" operation="sendPayment"/>
        <target serviceProvider=" leFournisseur" portType="suppSvr"
operation="recPay"/>
</plugLink>

<plugLink>
        <source serviceProvider=" leTraitementCommande"
portType="totalSupplyPT" operation="sendSR"/>
        <target serviceProvider="leTransporteur"
portType="shipSvr" operation="recSR"/>
</plugLink>
</globalModel>
```

Dans cet exemple, on se réfère au document WSFL précédent, spécifiant le processus totalSupply. On y trouve les déclarations, en préambule, des différents rôles et du modèle (totalSupply) concerné ; suit la description des trois connexions mettant en correspondance les ports et opération de l'interface publique du processus (la balise Source) avec ceux du service chargé de leur implémentation (la balise Target).

II.4.1.3 Présentation générale de la syntaxe de WSFL

Le document WSFL, dans sa généralité, décrit donc un ou plusieurs modèles (processus ou contrat) visant à capturer une succession d'échanges de messages et de données qui constituent un dialogue entre services WEB. Chacun des modèles, ou compositions, précise, d'une part, la succession des étapes à enchaîner pour l'accomplissement du dialogue et, d'autre part,

l'interface publique du dialogue à laquelle se réfèrent ses différents acteurs. Six éléments XML principaux pour spécifier une composition

Les éléments (flowSource et flowlink) : indiquent les données en entrée et en sortie de la composition.

L'élément (serviceProvider) : indique les services WEB participant à la composition.

L'élément(activity) décrit une opération ou une étape du dialogue.

Les éléments (controlLink et dataLink) spécifient respectivement comment se succèdent les étapes et les données qui sont passées d'une étape à la suivante.

La description des services WEB dans WSFL spécifie uniquement la nature des opérations qu'ils doivent implémenter (type de port et opération au sens WSDL), alors qu'un document WSDL représente l'implémentation du service [Chauvet-2000].

II.4.2 XLANG de Microsoft

II.4.2.1 Définition

XLANG est le format proposé par Microsoft pour représenter en XML. L'orchestration d'activités qui constituent un processus métier. Motivé par l'émergence des standards comme (SOAP et WSDL,…).XLANG est le format intermédiaire de stockage de l'environnement de développement BizTalk Serveur. Ces fichiers XLANG sont exécutés par le serveur BizTalk en phase de production. XLANG, comme WSFL d'IBM mais avec une division différente des tâches, couvre les points suivants :

- Flot de contrôle séquentiel et parallèle.
- Transactions longues.
- Corrélations des messages entre eux.
- Gestion des défaillances et des erreurs.
- Découverte dynamique des services.
- Contrats multipartites.

XLANG s'appuie sur WSDL en réutilisant un certain nombre des concepts définis dans cette dernière norme. XLANG reprend, en particulier, totalement la description WSDL d'un service en termes de groupe de ports et de liaisons à des protocoles de transport, chaque port étant constitué à son tour d'opérations caractérisées par un échange de messages.

Il est possible d'associer des champs d'information supplémentaires à un message constitutif d'une opération. Le service XLANG est un service WSDL doté un jeu de balises spécifique

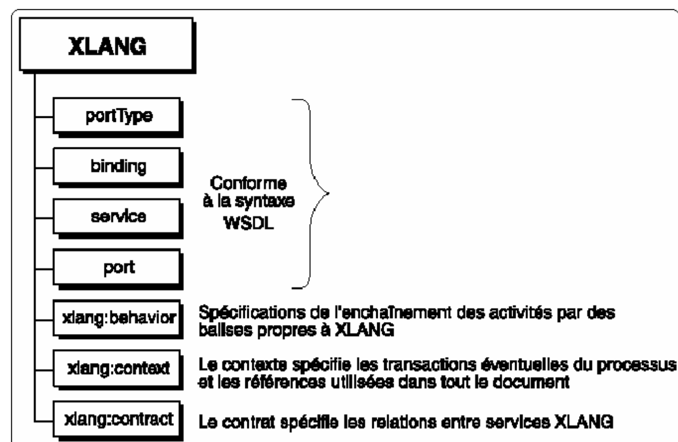


Figure 11 XLANG importante une partie de WSDL

II.4.2.2 Les actions XLANG

XLANG définit quatre types d 'actions élémentaires dans la représentation du workflow entre participants.

Operation: Une opération au sens de WSDL, c'est-à-dire un échange de messages, faisant référence à un port d'un service donné :

<action operation="AskTradePrice"portType=operation="pRequest">

DelayFor: Suspend l'exécution pendant le temps indiqué :

<delayFor period="RequestTimeOut">

DelayUntil: Suspend l 'exécution jusqu 'à un temps indiqué :

<delayUntil clock="ShippingDeadline">

Raise: Signalisation des défaillances : <raise signal="Invalid logon">

II.4.2.3 Les actions de contrôle XLANG

L'enchaînement de l'exécution est contrôlé par des commandes similaires a un langage de programmation.

Empty : Processus nul

<empty/>

Sequence : Exécution séquentielle d'une ou plusieurs actions

<xlang :sequence>
<xlang :action

```
operation="AskTradePrice"
portType= operation="pRequest">
<xlang :action
operation="SendTradePrice"
portType=operation="pRequest">
</xlang :sequence>
```

Switch : Exécution conditionnelle

```
<switch>
<branch>
<case>…</case>
<sequence>…</sequence>
</branch>
<branch>
<case>…</case>
<sequence>…</sequence>
</branch>
</switch>
```

While : Boucle

```
<while>
<case>…</case>
<sequence>…</sequence>
</while
```

All : Exécution en parallèle d'actions

```
<all>
<sequence>…</sequence>
..<sequence>…</sequence>
<sequence>…</sequence>
</all>
```

Pick : Le processus attend que se produise l'un des événements prévus et exécute alors les actions qui lui sont associées. Le premier événement déclenche la séquence et le bloc pick n'est exécuté qu'une seule fois

```
<pick>
<eventHandler>
<action operation="accusé reception"/>
<sequence>…</sequence>
```

```
</eventHandler >
```

```
<eventHandler >
<action operation="échec"/>
<sequence>…</sequence>
</eventHandler >
</pick>
```

II.4.2.4 un exemple XLANG

Dans exemple ci-dessous, on relie deux services, provider et user, définis respectivement dans les fichiers XLANG provider.xlg et user.xlg. Ces descriptions sont incluses par la balise (d'origine WSDL) <import>. Il est ensuite précisé comment les ports des deux services sont connectés.

```
<?xml version="1.0"?>
<definitions name="StockQuoteContract"
targetNamespace="http://example.com/stockquote/contract"
xmlns:tns="http://example.com/stockquote/contract"
xmlns:xlang="http://schemas.microsoft.com/biztalk/xlang/"
xmlns:provider="http://example.com/stockquote/provider"
xmlns:user="http://example.com/stockquote/user"
xmlns="http://schemas.xmlsoap.org/wsdl/">

<!-- import the provider and user namespaces -->
<import namespace="http://example.com/stockquote/provider"
  location="http://example.com/stockquote/provider.xlg"/>

<import namespace="http://example.com/stockquote/user"
  location="http://example.com/stockquote/user.xlg"/>
<xlang:contract>
<xlang:services refs="provider
StockQuoteProviderService
user:StockQuoteUserService"/>
<xlang:portMap>
<xlang:connect
port="provider:StockQuoteProviderService/pGetRequest"
port="user:StockQuoteProviderService/pSendRequest"/>
<xlang:connect
port="provider:StockQuoteProviderService/
pSendResponse"
port="user:StockQuoteProviderService/pGetResponse"/>
</xlang:portMap>
</xlang:contract>
</definitions>
```

II.4.3 BPEL4WS de IBM, Microsoft et BEA

II.4.3.1 Définition

BPEL4WS correspond à la fusion des langages WSFL d'IBM et XLANG de Microsoft. Cette spécification a été publiée en août 2002, accompagnée de deux autres spécifications : WS-Coordination et WS-Transaction. WS-Coordination décrit un framework extensible pour fournir des protocoles de coordination d'applications distribuées. WS-Transaction décrit des types de coordination transactionnelle utilisables avec WS-Coordination.

BPEL4WS est sensiblement identique à WSCI en terme d'architecture et de fonctionnalités. Il intègre les spécifications XLANG et WSFL, et spécifie à la fois le mode de conception, de mise en production et de maintenance des processus métier en prenant en compte SOAP, ebXML et RosettaNet et les annuaires UDDI.

II.4.3.2 Structure de BPEL4WS

Puisque il inspire sa partie statique de WSDL,BPEL4WS intègre directement les notions de [<types>,<message>,<porttype>,<binding>,<port>,<service>] dans sa syntaxe. De plus intègre d'autre notion tel :

- La section < variables > définit les variables de données déclarées comme les messages en WSDL. Permettent aux processus de conserver des données, ou l'historique des processus.

- La section <partnerLinks> définit les différentes parties qui agissent l'un sur l'autre au cours de processus métier. elle est caractérisée par type de lien son rôle

- La section de < faultHandlers > contient des traitements qui doivent être exécutées en réponse aux données erronées pour l'invocation des services ou d'autorisation.

La structure de la section de traitement principale est définie par l'élément <sequence>,qui déclare l'ordre d'exécution des activités. De même façon que XLANG,BPEL4WS contiens des actions de contrôle tel que :

- <receive> pour la réception d'un message

- <few> pour la concurrence et la syntonisation

- <pick> attend l'occurrence d'un ensemble d'événements, puis exécute l'activité liée à cet l'événement.

- <while> répété l'exécution d'une activité tant que la condition est juge vrai.

- <scope> fournie le comportement d'une activité. Chaque section <scope> a une activité primaire qui définit son comportement normal. L'activité primaire peut être une activité complexe, avec beaucoup d'activités fils.

II.4.3.3 Traitement d'erreurs

La manipulation des erreurs dans BPEL4WS peut être considérée comme un bascule de traitement normal dans une portée. Son but unique est de débarrasser du travail partiel et non réussi.

II.4.3.4 Exemple

Dans l'exemple suivant, on va présenter l'utilisation de BPEL4WS, pour décrire un simple service de transport maritime <shippingService >. Ce service envoi les articles selon un ordre de deux façons. Les articles sont embarqués ensemble ou par tranche.

Description de service : ce service a un comportement d'interaction bipartite entre un client et le service. Ceci est modelé dans la définition suivante de < partnerLinkType >

```
<plnk:partnerLinkType name="shippingLT"
xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/">
<plnk:role name="shippingService">
        <plnk:portType name="shippingServicePT"/>
</plnk:role>
<plnk:role name="shippingServiceCustomer">
        <plnk:portType name="shippingServiceCustomerPT"/>
</plnk:role></plnk:partnerLinkType>
```

La définition de la partie statique de service (messages « type de message » et portType « les opérations ») :

```
<wsdl :definitions
targetNameSpace="http://ship.org/wsdl/shipping"xmlns:ship= ...>
<message name="shippingRequestMsg">
        <part name="shipOrder" type="ship:shipOrder"/>
</message>
<message name="shippingNoticeMsg">
        <part name="shipNotice" type="ship:shipNotice"/>
</message>
<portType name="shippingServicePT">
        <operation name="shippingRequest">
        <input message="shippingRequestMsg"/>
        </operation>
</portType>
<portType name="shippingServiceCustomerPT">
        <operation name="shippingNotice">
        <input message="shippingNoticeMsg"/>
        </operation>
</portType>
</wsdl:definitions>
```

Les propriétés concernant le comportement de service sont:

- L'identification de bateau qu'est employée pour relier la notice (s) de bateau avec l'ordre de bateau (shipOrderID).
- L'ordre de transport soit complet ou par tranche (shipComplete).
- Le nombre total d'article (itemsTotal).

- Le nombre d'articles visés à une notification de bateau de sorte que, quand les expéditions partielles sont acceptables, nous puissions employer ceci, avec itemsTotal, pour dépister la réalisation globale de l'expédition (itemsCount) ici sont les définitions pour les propriétés et leurs noms d'emprunt

```
<wsdl:definitions
targetNamespace="http://example.com/shipProps/"
xmlns:sns="http://ship.org/wsdl/shipping"
xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-
process/">
<wsdl:types>
<xsd:schema>
        <xsd:restriction base="xsd:int">
                <xsd:minInclusive value="1"/>
                <xsd:maxInclusive value="50"/>
        </xsd:restriction>
        </xsd:simpleType>
</xsd:schema>
</wsdl:types>

<bpws:property name="shipOrderID" type="xsd:int"/>
<bpws:property name="shipComplete" type="xsd:boolean"/>
<bpws:property name="itemsTotal" type=" xsd:int "/>
<bpws:property name="itemsCount" type=" xsd:int "/>
<bpws:property name="numItemsShipped" type=" xsd:int "/>
<bpws:propertyAlias propertyName="tns:shipOrderID"
        messageType="sns:shippingRequestMsg"
        part="shipOrder"
        query="/ShipOrderRequestHeader/shipOrderID"/>
<bpws:propertyAlias propertyName="tns:shipOrderID"
        messageType="sns:shippingNoticeMsg"
        part="shipNotice"
        query="/ShipNoticeHeader/shipOrderID"/>
<bpws:propertyAlias propertyName="tns:shipComplete"
        messageType="sns:shippingRequestMsg"
        part="shipOrder"
        query="/ShipOrderRequestHeader/shipComplete"/>
<bpws:propertyAlias propertyName="tns:itemsTotal"
        messageType="sns:shippingRequestMsg"
        part="shipOrder"
        query="/ShipOrderRequestHeader/itemsTotal"/>
<bpws:propertyAlias propertyName="tns:itemsCount"
        messageType="sns:shippingNoticeMsg"
        part="shipNotice"
        query="/ShipNoticeHeader/itemsCount"/>
</wsdl:definitions>
```

En fin, la description de processus est :

```
receive shipOrder
switch
        case shipComplete
                send shipNotice
        otherwise

                itemsShipped := 0
                while itemsShipped < itemsTotal
                itemsCount := opaque
                send shipNotice
                itemsShipped = itemsShipped + itemsCount
```

II.4.4 Autres langages

II.4.4.1 WSCI de SUN

WSCI (WEB Services Choregraphy Interface) de SUN est un langage XML qui décrit le flux de messages échangés par un WEB Service qui interagit de manière ordonnée avec d'autres services. Tout comme WSFL, WSCI propose deux niveaux de description[BEA et al-2002] :

la description du comportement du WEB Service sous la forme d'une interface ou d'une API, d'un point de vue du flux de messages, des transactions ou encore des corrélations entre ces messages. Ce comportement est exprimé sous la forme de dépendances temporelles et logiques entre les messages échangés en proposant des mécanismes de :

- règles d'enchaînement
- corrélation
- gestion des erreurs
- gestion des transactions

La description des liens entre les opérations des WEB Services qui collaborent au travers de l'interface précédente, d'une manière similaire au modèle globale WSFL.

WSCI définit la manière dont un WEB Service doit réagir au regard des messages SOAP qu'il reçoit. Des réactions qui s'expriment en termes des tâches à effectuer, de règles de séquencement et de corrélation de ces tâches ou encore de traitement d'exceptions [Levy-2002].

II.4.4.2 WSCL de HP

WSCL (WEB Services Conversational Language) est une proposition de langage XML représentant simplement les interactions entre WEB Services, proposée par Hewlett-Packard dans la poursuite du développement d'une plate-forme de WEB Services fondée sur sa précédente plate-forme e-Speak. Le document WSCL est constitué de trois blocs :

- les schémas XML des documents échangés durant l'interaction

- la description des interactions elles-mêmes
- la description de transactions réglant le passage d'une interaction `a l'autre

La spécification WSCL vise à la simplicité de l'expression externe des interactions entre WEB Services. En contraste avec WSFL et XLANG, WSCL ne spécifie aucunement comment est créé le contenu des messages échangés.

A la manière de WSDL, WSCL se traduit par une interface qui précise le comportement d'un WEB Service. Cette interface peut ensuite être publiée sur un annuaire UDDI en vue d'être retrouvée puis prise en compte par d'autres WEB Services [**Erreur ! Source du renvoi introuvable.**].

II.4.4.3 BPMLde BPMI

BPML (Business Process Modeling Language) est un langage XML qui permet de décrire des processus métier génériques. Il gère notamment les transactions, les compensations et les exceptions. BPML a été conçu pour être exploité dans un contexte multi-participants, où un référentiel de processus s'impose. Son niveau d'abstraction permet par exemple à un service de sous-traiter à un autre le traitement d'un processus selon les modalités propres à ce sous-traitant. La syntaxe de BPML 1.0 est très semblable à celle de WSCI [**Erreur ! Source du renvoi introuvable.**].

II.4.5 Bilan

Nous concluons, les services WEB ne sont que des mécanisme de communication entre les applications distribuées, inspirer des RPC(Remote Procedure Call) puis évoluer dans le temps vers XML-RPC, XP, etc. jusqu'a WSFL,XLANG,BPEL4WS.ces dernier qui peuvent même créer des processus tel que les langage de programmation et orchestrer des processus indépendant.

Dans le tableau ci-dessous, une petite comparaison entre les services WEB et les langages de programmation.

Langages de programmation	**services WEB: WSFL,XLANG,WSCL**
impératifs/procéduraux	flots de données et de contrôle entre services WEB
Appels de procédures ou RPC	Envoi de messages HTTP ou MIME.coder sous SOAP
Structures de contrôle (if,while , etc.) explicites	Structures de contrôle et de synchronisation (fork,join ,loop ,etc.)explicites
Signature des procédures et des Fonctions	Documents WSDL
Bibliothèque de procédures et de Fonctions	Annuaires UDDI (par exemple)

Et d'après étude de différent langage de description des services WEB, on peut résumer ces formalismes dans le tableau suivant :

	WSDL	**WSFL**	**XLANG**	**BPEL4WS**
Contrats	Point a point	Point à point	Multipartites	Multipartites
flot de contrôle	Séquentiel	Séquentiel	Séquentiel /parallèle	Séquentiel /parallèle
corrélations des messages entre eux	NON	NON	OUI	OUI
Description de protocole	Simple	Simple	complexe	/
découverte dynamique des services	NON	/	OUI	/

Gestion des erreurs	NON	NON	OUI	OUI (puissant)

Au vu de ce tableau et des critères précédemment établis, notre choix se porte sur le langage Xlang, pour ce qu'il apporte en matière :

- de description du comportement d'un système à base de COTS,
- l'existence d'environnement qui implémente la technologie Xlang (serveur BizTalk (une description détaillée sera aborder dans le chapitre suivant).

II.5 Le serveur Biztalk

II.5.1 Introduction

Les applications ne sont pas des ensembles d'informations isolés. Même si certaines ont un mode de fonctionnement centré sur elle-même, la réalité est que l'interaction des applications entre elles est devenue la norme. Mais relier des applications signifie bien plus que de simplement échanger des octets. Au fur et à mesure que les organisations se dirigent vers une architecture orientée services (Service-Oriented Architecture ou SOA), le but réel est la création de processus qui unifient les applications dans un ensemble cohérent. C'est d'ailleurs dans cette perspective que s'inscrit BizTalk Server.

En effet, Biztalk permet de relier les applications hétérogènes, mais aussi de créer et de modifier graphiquement des processus métier qui utilisent les services qu'offrent ces applications. Elle dispose, Aussi : d'un mécanisme de spécifications des règles de gestion, de meilleurs outils de gestion et de contrôle des applications ainsi qu'une prise en charge du Single Sign-On (Authentification unique d'entreprise) pour l'accès à ces applications.

BizTalk Server peut être utilisé dans une grande variété de situations. Traditionnellement, BizTalk Server a été utilisé afin de répondre à des problématiques d'intégration et parmi la palette des scénarios possibles, deux sont primordiaux. Le premier consiste à faire communiquer des applications au sein d'une même organisation et se réfère communément à la notion d'intégration applicative d'entreprise (Enterprise Application Integration ou EAI), tandis que l'autre, souvent appelé intégration business-to-business (B2B), met en relation des applications appartenant à différentes organisations.

Le schéma ci-dessous (Figure 12) montre un exemple simple de l'utilisation de BizTalk Server, dans le cadre d'une problématique EAI. Dans ce scénario, une application de gestion de stock, s'exécutant peut être sur un mainframe IBM, remarque la faible disponibilité d'un article en stock et initialise une requête destinée à en commander un plus grand nombre d'exemplaires. Cette requête est envoyée à l'application BizTalk Server (étape 1), laquelle émet une requête à destination de l'application ERP de l'organisation, de manière à générer un ordre d'achat (étape 2). L'application ERP, qui peut très bien s'exécuter sur un système Unix, renvoie l'ordre d'achat requis (étape 3), puis, l'application BizTalk Server informe une application achat, reposant peut être sur Windows et construite via le .NET Framework, que l'article devrait être commandé (étape 4).

Dans cet exemple, chaque application communique au moyen d'un protocole différent. Par conséquent, l'infrastructure de messagerie du moteur BizTalk Server doit être capable d'échanger avec chaque application dans son style natif de communication. Notez également qu'aucune application ne connaît l'ensemble du processus métier. Ce processus métier, ainsi que l'intelligence requise pour coordonner toutes ses composantes, est mis en oeuvre dans l'application BizTalk Server.

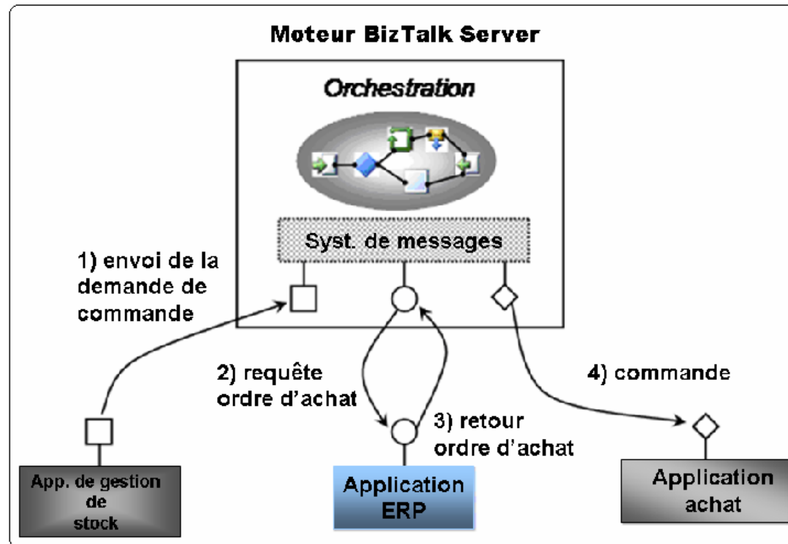


Figure 12 Modèle de Gestion de stock en EAI

Mettre des applications en relation au sein d'une organisation est important mais faire communiquer des applications qui s'étendent sur de multiples organisations peut quelquefois représenter une valeur ajoutée commercialement supérieure. La Figure 13 montre un exemple simple de ce type de scénario d'intégration B2B. Dans ce cas, l'organisation qui achète, située en haut de la figure, exécute une application BizTalk Server qui interagit avec deux fournisseurs. Le fournisseur A utilise également BizTalk Server, ce qui offre un accès indirect à son application de distribution. Le fournisseur B utilise une plate-forme d'intégration d'un autre éditeur et se connecte à l'application BizTalk Server de l'acheteur au moyen, disons, de services Web. Le fournisseur B exécute le même processus métier que les autres : une définition BEPL de ce processus a pu être envoyée par la société cliente, qui a exporté cette définition depuis BizTalk Server.

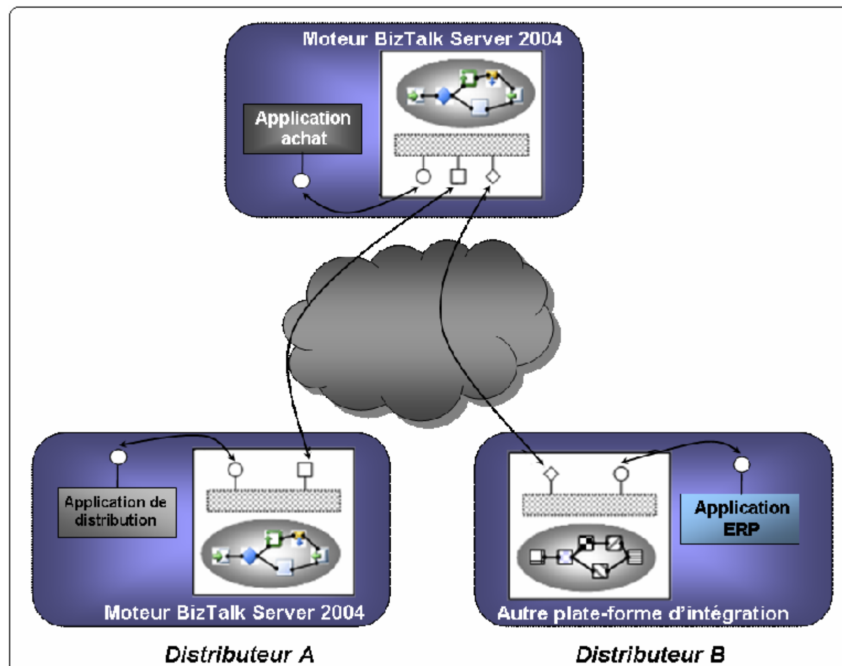


Figure 13 Modèle de Gestion de stock en B2B

L'intégration d'applications — qu'elles résident au sein d'une même société ou qu'elles soient disséminées parmi de multiples organisations — à l'intérieur d'un processus métier unique, constitue l'un des objectifs fondamentaux de BizTalk Server.

II.5.2 Le moteur BizTalk Server

Afin de permettre à ses utilisateurs de créer un processus métier qui s'étend sur de multiples applications, le moteur de BizTalk Server doit proposer deux éléments principaux : un moyen de spécifier ce processus métier, ainsi que certains mécanismes pour communiquer entre les applications qu'il utilise. La Figure 14 montre les principaux composants du moteur de BizTalk Server qui adressent ces deux problèmes.

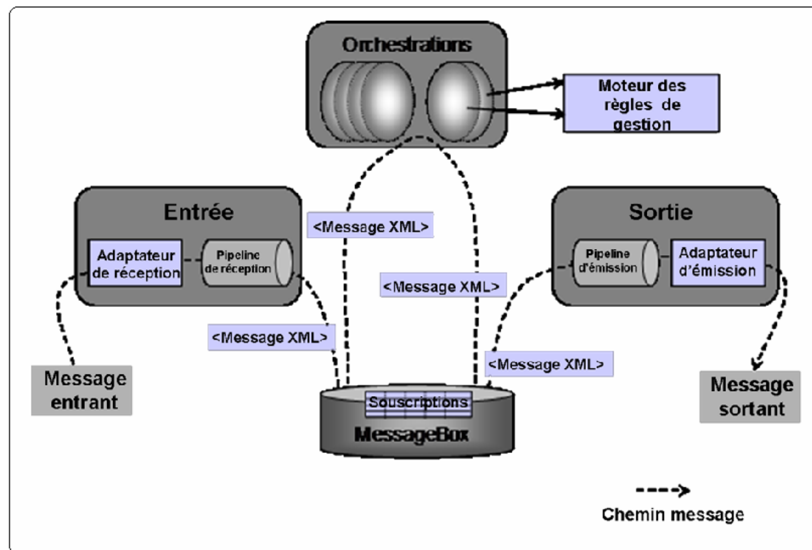


Figure 14 composant de moteur Biztalk

Comme le montre le schémas, un message est reçu via un adaptateur de réception. Différents adaptateurs correspondent à différents mécanismes de communication et ainsi, un message peut être reçu en accédant à un service WEB, en lisant un fichier ou en utilisant un autre moyen. Le message est ensuite traité via un pipeline de réception. Ce pipeline peut contenir divers composants qui réalisent des actions telles que la conversion du format natif du message vers un format de document XML, la validation de sa signature digitale, etc. Le message est ensuite acheminé vers une base de données appelée MessageBox, laquelle repose sur SQL Server.

Un processus métier est mis en oeuvre sous la forme d'une ou plusieurs orchestrations, chacune d'entre elles correspondant à du code exécutable. Cependant, ces orchestrations ne sont pas créées en écrivant du code, via un langage tel que C#. Au lieu de cela, un analyste financier ou un développeur organise graphiquement un groupe défini de formes, au moyen d'un outil propre, adapté à chacun, afin d'exprimer les conditions, boucles et autres comportements du processus métier. Les processus métier peuvent également utiliser le moteur des règles de gestion qui offre un moyen plus simple et plus aisé pour définir les règles d'un processus métier.

Chaque orchestration crée des souscriptions (subscriptions) afin d'indiquer le type de messages qu'elle s'attend à recevoir. Lorsqu'un message répondant à ces critères arrive dans la MessageBox, il est expédié vers son orchestration cible, laquelle effectue ensuite les actions requises par le processus métier. Le résultat de ce traitement a typiquement pour effet la production d'un autre message, par le processus métier, lequel est sauvegardé dans la MessageBox. Ce message est à son tour traité par un pipeline d'émission, qui peut le convertir du format interne XML utilisé par BizTalk Server vers le format requis par sa destination, ajouter une signature digitale, etc. Le message est ensuite envoyé via un adaptateur d'émission, qui utilise un mécanisme approprié pour communiquer avec l'application cible.

II.5.3 Relier les applications

Une bonne intégration passe par un échange efficace des messages entre applications. Etant donné la diversité des modes de communication qui existent, le moteur de BizTalk Server doit supporter une grande variété de protocoles et de formats de message. Comme décrit par la suite, une portion significative du moteur de BizTalk Server est dédiée à établir cette communication. Un point important à garder à l'esprit, le moteur ne fonctionne qu'avec des documents XML en interne. Quel que soit le format dans lequel un message arrive, il est systématiquement converti en document XML après sa réception. De même, si le récepteur d'un document ne peut accepter son format XML, le moteur le convertit dans le format attendu par l'application cible

II.5.3.1 Envoyer et recevoir des Messages: les adaptateurs

Le moteur de BizTalk Server devant s'adresser à un large éventail d'autres logiciels, il repose sur une gamme d'adaptateurs pour réaliser cette opération. Un adaptateur est l'implémentation d'un mécanisme de communication tel qu'un protocole particulier. Un développeur peut déterminer quels adaptateurs utiliser dans une situation donnée. Il peut choisir l'un des adaptateurs fournis en standard par BizTalk Server, par exemple, ou en utiliser un lié à une application populaire telle que SAP, ou même créer un adaptateur personnalisé. Dans tous les cas, cet adaptateur est construit sur une base technologique standard appelée le Framework adaptateur. Nouveauté de BizTalk Server, ce framework fournit un moyen commun de créer et d'exécuter des adaptateurs. Il permet également d'utiliser les mêmes outils pour gérer à la fois les adaptateurs standards et les adaptateurs personnalisés. BizTalk Server propose les adaptateurs suivants :

- Adaptateur services Web : permet l'envoi et la réception de messages via SOAP, au-dessus d'HTTP. SOAP étant le protocole principal des services Web, cet adaptateur constitue la fondation sur laquelle s'appuie BizTalk Server pour interagir dans un environnement services Web. Comme d'habitude avec les services Web, les URL sont utilisées pour identifier les systèmes émetteurs et récepteurs.

- Adaptateur MSMQT : permet l'envoi et la réception de messages via le Message Queuing de BizTalk (MSMQT). MSMQT est une implémentation du protocole MSMQ qui peut recevoir et envoyer des messages MSMQ dans/depuis la MessageBox. Il ne remplace pas MSMQ, mais constitue plutôt un moyen efficace d'utiliser le transport MSMQ avec BizTalk.

- Adaptateur Fichier : permet la lecture et l'écriture de fichiers au sein du système de fichiers Windows. Les applications impliquées dans un processus métier peuvent invoquer souvent le même système de fichiers,

soit localement, soit à travers le réseau, l'échange de messages au moyen de fichiers peut constituer une option pratique.

- Adaptateur HTTP : permet l'envoi et la réception d'informations via HTTP. Le moteur de BizTalk Server expose une ou plusieurs URL, afin de permettre aux autres applications de lui envoyer des données. Il peut lui même utiliser cet adaptateur pour envoyer des données aux autres URL.

- Adaptateur SMTP : permet l'envoi de messages via SMTP. Les adresses mail standards sont utilisées pour identifier les parties émettrices.

- Adaptateur SQL : permet la lecture et l'écriture d'informations depuis/dans une base de données SQL Server.

Quel que soit l'adaptateur utilisé pour recevoir des données, les messages qu'il reçoit doivent être traités avant d'être accessibles par une orchestration. De même, les messages de sortie d'une orchestration ont souvent besoin d'être traités avant d'être envoyés par un adaptateur. La façon dont ce traitement est effectué est expliquée dans la prochaine section.

II.5.3.2 Traitement des messages : les pipelines

Les applications sous jacentes à un processus métier communiquent en échangeant des types de documents variés : commandes, factures, etc. Une application BizTalk Server qui exécute un processus métier doit être capable d'exécuter correctement les messages qui contiennent ces documents. Ce traitement peut impliquer de multiples étapes et est réalisé par un pipeline message. Les messages entrants sont traités par un pipeline de réception, tandis que les messages sortants le sont par un pipeline d'émission.

Ainsi, par exemple, bien que de plus en plus d'applications soient capables d'interpréter les documents XML, beaucoup—probablement la majorité aujourd'hui, même—ne le peuvent. Le moteur de BizTalk Server ne travaillant en interne qu'avec des documents XML, il doit offrir un moyen de convertir les autres formats en XML et vice versa. D'autres services peuvent être également requis, tel que l'authentification de l'émetteur du message. Afin de gérer ces tâches d'une façon modulaire et composable, un pipeline est construit à partir d'un certain nombre de phases, chacune d'entre elle contenant un ou plusieurs composants .NET ou COM. Chaque composant gère une partie spécifique du traitement du message. Le moteur de BizTalk Server offre plusieurs composants standards qui adressent la plupart des cas courants. S'ils ne suffisent pas, les développeurs peuvent également créer des composants personnalisés, à la fois pour le pipeline d'émission et pour le pipeline de réception.

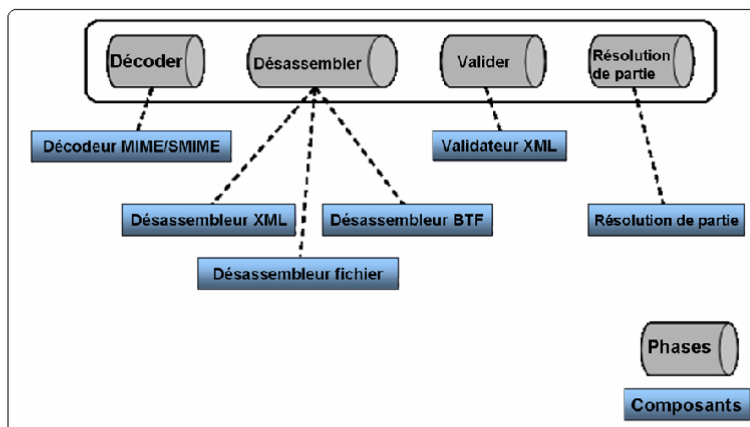


Figure 15 pipeline de réception

La Figure 15 illustre les phases d'un pipeline de réception, en parallèle aux composants standards offerts à chacune d'entre elles. Ces phases et leurs composants associés sont :

- Décoder : BizTalk Server fourni un composant standard pour cette phase : le Décodeur MIME/SMIME. Ce composant peut gérer les messages, ainsi que tous les attachements qu'ils contiennent, qu'ils soient au format MIME ou MIME sécurisé (S/MIME). Le composant convertit les deux types de messages en XML. Il peut également décrypter les messages S/MIME et vérifier leurs signatures digitales.

- Désassembler : Trois composants standards sont fournis. Le Désassembleur de fichiers convertit les fichiers en documents XML. Ces fichiers peuvent être de type positionnel, chacun des enregistrements possédant les mêmes longueur et structure, ou de type délimité, ce qui correspond à un format où un caractère désigné est utilisé pour séparer les enregistrements, dans le fichier. Le second composant standard, le Désassembleur XML, analyse les messages entrants qui sont déjà au format XML. Le troisième composant standard, qui n'est pas souvent utilisé de nos jours, est le Désassembleur BTF. Il accepte les messages envoyés via le mécanisme sûr de messagerie défini par le framework BizTalk (BTF).

- Valider : BizTalk Server fourni un composant Validateur XML pour cette phase. Comme son nom le suggère, ce composant valide un document XML produit par la phase de désassemblage, par rapport à un schéma ou un groupe de schémas spécifiés. Il retourne une erreur si le document n'est pas conforme à l'un de ces schémas.

- Résolution de partie : Le seul composant standard de cette phase, le composant de Résolution de partie, essaie de déterminer l'identité de l'émetteur du message. Si le message a été signé digitalement, la signature est utilisée pour rechercher une identité Windows dans la base de données de configuration de BizTalk Server. (Comme décrit plus loin, la base de données est également utilisée par plusieurs outils d'administration.) Si le message contient l'identificateur de sécurité authentifié (SID) d'un utilisateur Windows, cette identité est utilisée. Si aucun des deux mécanismes ne réussit, une identité anonyme par défaut est assignée à l'émetteur du message.

II.5.3.3 Choix des Messages: les Souscriptions (subscriptions)

Une fois qu'un message a fait son chemin, via un adaptateur et un pipeline de réception, la question suivante qui se pose est de savoir où est-ce qu'il doit être acheminé. Une orchestration constitue le plus souvent la cible d'un message, mais il est également possible qu'un message aille directement vers un pipeline d'émission, ceci permettant d'utiliser le moteur de BizTalk Server comme un pur système de messagerie. Dans l'un ou l'autre des cas, la correspondance entre ces messages et leur destination est effectuée via les souscriptions.

II.5.4 Définition d'un Processus Métier

Envoyer des messages entre des applications hétérogènes est une condition sine qua non à la résolution des problèmes que BizTalk Server adresse. Mais dans la plupart des cas, c'est seulement un moyen d'arriver à une fin : le but réel

est de définir et d'exécuter des processus métier en fonction de ces applications. Le moteur de BizTalk offre deux technologies pour réaliser cette opération :

- les orchestrations
- le moteur des règles de gestion.

Cette section décrit les deux.

II.5.4.1 Orchestration

Un processus métier peut être mis en oeuvre directement via un langage tel que C# ou Visual Basic.NET. Mais la création, la maintenance et la gestion de processus métier complexes, au moyen de langages de programmation conventionnels, peuvent représenter un défi. De la même manière, un processus créé de cette façon risque d'apparaître comme un ensemble d'applications disparates, plutôt que comme un processus métier cohérent. Ainsi, BizTalk Server permet la création graphique d'un processus métier. Cette façon de faire peut être plus rapide que de construire le processus directement via un langage de programmation et en facilite la compréhension, l'explication et la modification. Les processus métier générés de cette façon peuvent être également contrôlés plus aisément.

Réussir à créer un processus métier automatisé requiert généralement une collaboration entre des développeurs orientés technologie et des utilisateurs. Ainsi, BizTalk Server propose des outils appropriés à ces deux populations. Les outils des développeurs s'exécutent au sein de Visual Studio.NET, un environnement dans lequel les professionnels se sentent chez eux. Cependant, la plupart des utilisateurs lambda ne trouvent pas Visual Studio spécialement adapté à leur profil, c'est pourquoi il leur est possible d'utiliser Visio. Dès lors, les informations créées dans les outils basés sur Visual Studio peuvent être importées dans les outils basés sur Visio et vice versa, ce qui aide ces deux types de population à travailler ensemble, lors de la création d'un processus métier. Une fois créé, ce processus, connu sous le nom d'orchestration , est automatiquement transformé en assemblies standards qui s'exécutent au sein du.NET Framework.

Pour un développeur, la création d'une orchestration repose sur trois outils principaux : *l'Editeur BizTalk* pour générer des schémas XML, le *Mappeur BizTalk* pour définir les correspondances entre ces schémas et le *Concepteur d'Orchestration* pour spécifier le flux des processus métier. Tous les outils de développement de BizTalk Server sont hébergés au sein de Visual Studio.NET, ce qui permet de disposer d'un environnement cohérent. Cette section décrit le fonctionnement de chacun de ces outils, ainsi que la façon dont ils coopèrent.

II.5.4.1.1 Création de Schémas: l'éditeur BizTalk

L'orchestration travaille avec des documents XML, chacun d'entre eux étant conforme à un schéma XML quelconque. Il doit donc exister un moyen pour définir ces schémas. Pour réaliser cette opération, le moteur de BizTalk Server fournit *l'Editeur BizTalk*. Cet outil permet la création de schémas, qui sont essentiellement les définitions des types et des structures d'un document informationnel, exprimées au moyen du langage de définition de schéma XML (XSD).

II.5.4.1.2 Correspondance entre schémas: le Mappeur BizTalk

Une orchestration qui met en oeuvre un processus métier reçoit typiquement des documents et en envoie d'autres. Il est courant qu'une partie de l'information des documents réceptionnés soit transformée d'une certaine façon, avant d'être

transférée aux documents envoyés. Un processus de préparation de commandes, par exemple, peut recevoir une commande concernant un certain nombre d'articles, puis renvoyer un accusé de réception indiquant que la commande a bien été envoyée. Il est ainsi possible que des informations appartenant à la commande, telles que le nom et l'adresse du client, soient copiées des champs de la commande reçue vers ceux de son accusé de réception. Le *Mappeur BizTalk* peut être utilisé pour définir une transformation, ou un fichier de mappage, entre un document et un autre.

La transformation décrite dans un fichier de mappage peut être simple, telle que la copie d'un nom et d'une adresse d'un document vers un autre. Les copies directes de données comme celles ci sont exprimées via une *liaison*, représentée dans le *Mappeur BizTalk* par une ligne reliant les éléments appropriés du schéma source à leurs homologues du schéma de destination. Des transformations plus complexes sont également réalisables, au moyen de *fonctoids*. Un fonctoid est un morceau de code exécutable qui peut définir des transformations arbitraires complexes entre schémas XML et le *Mappeur BizTalk* le représente comme une boîte placée sur la ligne qui relie les éléments à transformer. Comme quelques unes de ces transformations sont assez courantes, BizTalk Server inclut un certain nombre de fonctoids de base. Ces fonctoids de base sont regroupés dans des catégories qui incluent les suivantes :

- Les fonctoids mathématiques qui exécutent des opérations telles que additionner, multiplier et diviser sur les valeurs des champs du document source et stockent les résultats dans l'un des champs du document cible.

- Les fonctoids de conversion qui convertissent une valeur numérique vers son équivalent ASCII et vice-versa.

- Les fonctoids logiques qui peuvent être utilisés pour déterminer si un élément ou un attribut doit être créé dans le document cible, en fonction d'une comparaison logique entre des valeurs spécifiées dans le document source. Ces valeurs peuvent être évaluées à l'aide d'opérateurs d'égalité, supérieur à/inférieur à, etc.

- Les fonctoids cumulés calculent des moyennes, des sommes, ainsi que d'autres valeurs, à partir de divers champs du document source, puis stockent le résultat dans un seul des champs du document cible.

- Les fonctoids de base de données peuvent accéder aux informations stockées dans une base de données.

Il est également possible de créer des fonctoids personnalisés directement en XSLT ou au moyen de langages .NET comme C# et VB.NET. Les fonctoids peuvent également être combinés en séquences qui enchaînent le résultat de sortie de l'un comme entrée d'un autre.

II.5.4.1.3 Définition des processus métier: le concepteur d'Orchestration

Un processus métier est un ensemble d'actions dont les effets conjugués répondent à un besoin commercial essentiel. Grâce au moteur de BizTalk Server, un développeur peut employer le concepteur d'Orchestration afin de spécifier graphiquement ces actions. Plutôt que d'en exprimer les étapes avec le langage Xlang, cet outil permet à un développeur de créer une orchestration en assemblant une série de formes de façon logique.

II.5.4.2 Le moteur des règles de gestion

Le concepteur d'orchestration, ainsi que l'éditeur BizTalk et le *Mappeur BizTalk*, est un moyen efficace de définir un processus métier et les règles qu'il implique. Il est quelquefois utile, cependant, de pouvoir disposer d'un moyen plus aisé pour définir et modifier les règles de gestion. Pour cette raison, BizTalk Server fourni le moteur des règles de gestion, afin de permettre aux utilisateurs plus orientés gestion de directement créer et modifier les jeux de règles de gestion. Ces règles sont créées via un outil appelé le *Editeur des règles de gestion*, puis, exécutées directement par le moteur. Il s'agit d'une nouvelle technologie introduite dans BizTalk Server 2004 et qui constitue l'une des fonctionnalités les plus intéressantes du produit.

Afin de mieux percevoir l'utilité d'une telle approche, pensez à ce que requiert la modification d'une règle de gestion, implémentée au sein d'une orchestration. Un développeur doit d'abord ouvrir l'orchestration dans Visual Studio.NET, modifier les formes appropriées (et peut être les objets .NET ou COM qu'elle invoque), puis, construire et déployer l'assembly modifié. Ceci requiert également l'arrêt et le redémarrage de l'application BizTalk Server qui utilise cette orchestration. Si, au lieu de cela, cette règle de gestion est mise en oeuvre au moyen du moteur des règles de gestion, elle peut être modifiée sans recompiler ni relancer quoi que ce soit. Tout ce qu'il est nécessaire de faire est d'utiliser le Editeur des règles de gestion pour modifier la règle désirée, puis de déployer le nouveau jeu de règles.

Une orchestration synthétise le flux d'un processus métier en l'exprimant graphiquement plutôt qu'avec du code. De même, le moteur des règles de gestion permet d'exprimer les règles d'un processus métier selon un niveau d'abstraction plus élevé. Ensemble, ces deux technologies offrent une approche efficace pour créer la logique commerciale

II.5.5 Bilan

Les fondations de BizTalk Server sont représentées par l'intégration de diverses applications. Au fur et à mesure que la technologie de Gestion des processus métier (BPM) progresse, BizTalk Server constitue de plus en plus l'épicentre de relations commerciales de toutes sortes. Cette évolution présente des bénéfices évidents. Imaginez-vous créer des mécanismes standards natifs de contrôle de processus métier, par exemple, via une logique reposant sur le .NET Framework (ou d'ailleurs, sur un serveur d'application Java). Au fur et à mesure que les organisations intensifient leur adoption des architectures orientées services basées sur les services Web, la capacité de BizTalk Server à exposer diverses applications comme un ensemble de services Web, dont les documents commerciaux reposent sur des schémas XML courants, devient de plus en plus attractive.

L'objectif de BPM qui est la gestion centralisée et uniforme des processus métier, conduit à l'efficacité et à l'adaptabilité des organisations. Mais quoi qu'il arrive, il est clair que BizTalk Server est en train de monter en puissance. BizTalk Server représente un pas significatif en avant par rapport à ses concurrent, de par le nombre de services prépondérants qu'il ajoute. De même, il constitue une solide fondation pour le futur.

II.6 Les langages de description architecturale

Comme nous l'avons expliqué (section I.4.3.1), l'intégration des systèmes à base de COTS requiert un langage indépendant des langages de programmation afin que le développeur d'un système ne soit pas obligé de rendre lui-même les entités interopérables au sein d'un environnement hétérogène.

II.6.1 Définition

Les ADL permettent de spécifier des architectures logicielles en offrant un moyen pratique et abstrait pour une description compréhensible d'un système complexe. Ils permettent l'expression de la configuration d'une application en termes de composition d'unités logicielles élémentaires et d'éventuelles contraintes de déploiement [Telisson-2002].

Chacun de ces langages fournit une manière de décrire l'architecture d'un logiciel selon son utilisation finale. Certains langages ont le mérite d'aider à la vérification formelle de propriétés, de permettre la simulation d'une architecture, ou alors d'aider de manière directe à la mise en œuvre et l'exécution du programme. En effet les buts des ADL peuvent être classés essentiellement en deux catégories, ceux qui sont relatifs aux spécifications formelles et ceux qui sont relatifs à l'implémentation d'applications.

Un ADL se compose communément de trois éléments [Medvicovic et al-1997] : les composants, les connecteurs et les configurations.

- **Composant :** Dans un ADL, les composants sont des entités logicielles, qui en prenant part à une architecture plus large, permettent la construction d'une application. Par le mécanisme de la description d'architectures à l'aide de langages spécifiques, les ADLs, on vise à réutiliser ces entités logicielles. Pour cela, le composant est l'élément indispensable de tous les ADL. C'est avant tout une entité d'intégration et de structuration de logiciels, dans la mesure où il peut permettre une hiérarchisation de l'application, spécifiée indépendamment de l'implémentation. Les composants logiciels sont, au même titre que les objets, décrits par une **interface** qui explicite la liste des procédures et données, les types et éventuellement les événements pouvant être visibles par les autres composants du système (Interface est un ensemble de points d'interactions entre le composant et le monde extérieur qui permettent l'appel des services). Cette interface exhibe, en plus, les informations et les appels qui lui sont nécessaires chez les autres composants du système. Par ailleurs, les composants dérivent d'un type qui représente le mode de mise en œuvre. Cet aspect de typage impose des vérifications, dans la phase d'analyse, de la faisabilité et de la validité de l'architecture.

- **Connecteur :** Entité de gestion des interactions entre les composants, le connecteur permet de définir le comportement des communications. Les connecteurs sont donc des abstractions essentielles pour la maintenance et la réutilisation des composants. Ils permettent de séparer la phase de programmation en deux niveaux : la programmation classique des différents modules logiciels et le codage des connexions entre ces modules. Ils contiennent les informations concernant les règles d'interconnexion de composants, telles que le protocole, la spécification des échanges de données ainsi que les transformations éventuelles du format d'échange. Chaque connecteur dérive, comme les composants, d'un type qui représente

le mécanisme et les propriétés de communication : type de communication, règles de synchronisation, etc.

- **Configuration :** C'est la description de l'ensemble des composants logiciels nécessaires pour le fonctionnement d'une application, ainsi que celle de leurs communications. Elle permet de gérer la complexité de l'application en raffinant peu à peu son architecture en un ensemble hiérarchique.

II.6.1.1 Style

Rappelons que l'enjeu de nos travaux est de modéliser un ensemble de styles architecturaux pour les systèmes à base de COTS. Les langages décrits dans la section suivante supportent tous la notion de style

De plus, l'utilisation d'un langage de description d'architecture est un moyen pratique pour décrire la dynamique d'une application. Il serait intéressant dans notre problématique générale que ceux-ci s'adaptent particulièrement aux systèmes répartis où la configuration de l'application pourrait être reconfigurée en fonction des contraintes des sites.

Afin de dégager le langage qui répond au mieux à nos attentes, nous allons étudier les possibilités et les limites offertes par les différents ADL mis à notre disposition : UniCon-2, Aesop, Armani, Darwin et π-ADL.

II.6.2 Présentation de quelque ADL

II.6.2.1 UNICON-2

Version de UniCon, pour Universal Connector, langage élaboré au sein du laboratoire de Carnegie Mellon à l'Université de Pittsburgh, UNICON-2 [DeLine-1996] a la particularité de supporter les styles architecturaux et de construire à partir d'eux des systèmes complexes. Une description d'architecture dans UNICON-2 s'appuie sur deux concepts principaux, les composants et les connecteurs.

Nous établirons dans cette partie, une synthèse sur les avantages et inconvénients des différentes approches de ce langage.

II.6.2.1.1 Modèle

Le modèle décrit le système sous forme de deux entités principales, les composants et les connecteurs. Ces deux entités sont décrites par un nom et une spécification (interface) contenant un type (rôle).

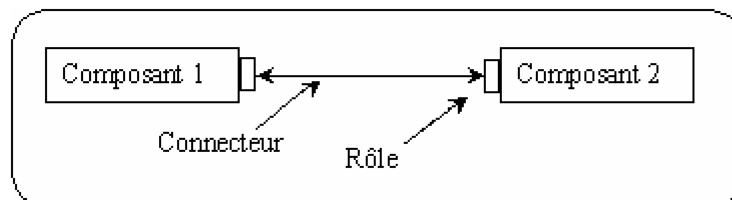


Figure 16 Représentation d'une communication UniCon

II.6.2.1.2 Composant

Son interface spécifie les fonctions que le composant exporte et importe dans son implémentation. Un composant définit aussi son type, ce qui permet de nombreuses vérifications au niveau de l'architecture.

Comme dans la plupart des langages de description d'architecture, nous verrons que deux types d'implémentation de composants existent. Les composants primitifs et les composants composites qui sont eux même un assemblage de composants et de connecteurs.

II.6.2.1.3 Type de composants

UNICON-2 présente un système de typage fort pour ses composants. En plus de l'expression de la fonctionnalité globale du composant, les types expriment des restrictions sur les propriétés, pour la communication, le partage de données et autres interactions.

II.6.2.1.4 Syntaxe des composants

UNICON-2 présente un système de typage fort pour ses composants. En plus de l'expression de la fonctionnalité globale du composant, les types expriment des restrictions sur les propriétés, pour la communication, le partage de données et autres interactions. La syntaxe des composants se distingue par deux parties, l'interface qui contient soit le type du composant, des propriétés sous forme de liste, et l'implémentation ; soit le lien entre l'interface et le code logiciel. L'implémentation primitive ou composite fait référence à un fichier source qui contient du code de programmation traditionnel.

II.6.2.1.5 Connecteur

Comme son nom l'indique, un connecteur permet d'établir des connexions entre les composants. Il spécifie les protocoles, les interactions pour les échanges de données. Comme pour le composant, le connecteur possède une implémentation

II.6.2.1.6 Protocole

Il définit les interactions permises entre les différents composants de l'application. En fonction du type de connecteur, le protocole définit des rôles. Il existe un certain nombre de types de rôle qui, suivant le type de connecteur, définissent un protocole particulier. Par exemple le protocole RPC pour Remote Process Call est spécifié par deux types de rôle, definer pour l'appelé et caller pour l'appelant.

II.6.2.1.7 Rôles

Ces entités sont en fait les points de branchement auxquels peuvent se raccorder les composants. Ils identifient le type d'interaction qu'un connecteur peut établir et le type de composant avec lequel ils peuvent être connectés.

II.6.2.1.8 Syntaxe

Comme pour les composants, nous retrouvons une notation syntaxique en deux parties, le protocole et l'implémentation. La partie protocole se compose d'une liste de rôles. A noter que contrairement aux composants, UNICON-2 n'offre pas la possibilité d'implémenter des connecteurs composites.

II.6.2.1.9 Style

Les styles architecturaux dans UNICON-2 sont représentés par des « duty » [DeLine-1996]. Un duty décrit les informations fournies par un type donné de rôle ou de protocole. Le duty se compose

- d'une clause *requires* qui inclut une liste de canevas à respecter ou prohibés,
- d'une clause optionnelle *provides* pour spécifier des paramètres.

```
interface duty Filter
        requires
                ( Stream-In player $ | Stream-Out player $ ) +
        provides
                gui-icon-size = (120, 80)
                gui-icon = [ …omitted… ]
end Filter
```

Notons que pour chaque clause requires, le canevas donné par le duty doit être respecté par la composition de l'architecture.

II.6.2.1.10 Atouts

- Le mécanisme de communication n'est pas inclus dans le code des composants mais tient une place particulière au niveau du langage de description architecturale. Par conséquent, une même définition d'un connecteur peut gérer les échanges entre plusieurs composants, à partir du moment ou ils sont autorisés à interagir via ce type de communication.

- La symétrie entre les composants et les connecteurs facilite la conception d'architectures.

II.6.2.1.11 Limites

- UNICON-2 ne permet qu'une description statique de l'architecture qu'il fournit. Une architecture UNICON-2 est définie par une énumération statique de composants et de connecteurs. Aucun aspect dynamique n'apparaît donc. La création et la suppression de nouveaux composants/connecteurs pendant l'exécution ne peut pas être décrite dans ce langage.

- UNICON-2 ne permet pas de définir des propriétés d'analyse.

II.6.2.1.12 Bilan

UNICON-2 est un langage de description architecturale qui semble intéressant pour la séparation entre le composant et le connecteur, et la description du style grâce à la fonctionnalité duty qui semble appropriée à nos travaux.

Cependant, une architecture UNICON-2 ne présente pas d'abstraction au niveau du langage pour décrire la dynamique de l'application. Aucun ajout ou suppression de composants et connecteurs n'est possible à l'exécution. Ce manque d'évolution est une limite très contraignante dans le cadre d'une modélisation d'architecture appelée à évoluer dans le temps.

II.6.2.2 AESOP

Aesop est issu des travaux de recherche effectués par l'équipe de David Garlan à l'université de Carnegie Mellon. Comme tous les ADL, l'objectif est de réduire les coûts de développement en exploitant les points communs des familles de systèmes et en permettant la réutilisation de code existant.

La particularité d'Aesop est d'être un modèle objet générique pour la conception. Les styles architecturaux se définissent par la spécialisation de ce modèle objet.

En fait, Aesop offre une boîte à outils pour la création d'un environnement de conception ouvert à partir de la description d'un style architectural spécifique.

L'idée étant d'offrir un environnement de développement rapide pour la spécification de styles architecturaux.

Aesop est donc un système pour le développement des environnements spécifiques à un style.

Chacun de ces environnements admet:

- une palette d'éléments de conception (composants et connecteurs spécifiques au style),
- la composition d'éléments de conception qui satisfait les contraintes topologiques du style,
- une interface qui permet aux outils externes d'analyser et de manipuler des représentations architecturales,
- de multiples visualisations d'informations, spécifiques au style avec un éditeur graphique pour les manipuler.

L'idée fondatrice d' Aesop est de déterminer une forme explicite du style architectural, qui peut être utilisée par des outils de conception pour construire des systèmes dans ce style.

Ainsi, des nouveaux styles peuvent être construits à coût réduit par la spécialisation incrémentale des styles existants Aesop adopte une ontologie générique de 7 entités:

4. **composant** : le lieu de computation.
5. **connecteur** : interaction entre composants.
6. **configuration** : topologie de composants et connecteurs.
7. **port** : point d'interaction d'un composant avec son environnement ; l'ensemble des ports d'un composant forme son interface.
8. **rôle** : identifie les participants dans une interaction ; l'ensemble des rôles d'un connecteur forme son interface.
9. **Représentation** : le "contenu" d'un composant ou connecteur lié à la représentation hiérarchique des architectures.

10. **lien** (binding) - définie la correspondance entre les éléments de la configuration interne et l'interface externe d'un composant ou connecteur lie un port/rôle interne avec un port/rôle externe

La Figure 17 donne une représentation graphique d'une architecture Aesop.

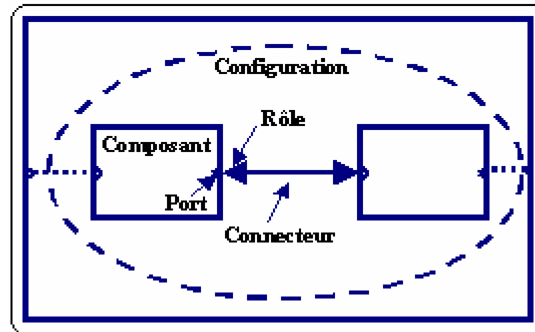


Figure 17 Architecture Aesop

Basé sur l'approche objet, Aesop détermine un ensemble fixe de définitions de classes abstraites. Ces classes supportent des opérations et permettent l'ajout et la suppression des ports au composant ainsi que l'attachement d'un connecteur entre les composants.

Un composant peut avoir une représentation qui spécifie sa fonctionnalité et un connecteur peut avoir une représentation qui spécifie son protocole

Aesop fournit un sous-type de représentation appelé external_rep qui à son tour a des sous-types, par exemple: text_file_rep, oracle_rep, … ces références sont interprétées par des outils d'analyse qui les accèdent.

Le modèle de définition des styles est basé sur le principe du sous-typage ainsi que sur un vocabulaire spécifique au style. Les styles peuvent être organisés dans des relations de sous-typage. De plus, le principe de substitution s'applique non seulement aux éléments individuels, mais aussi aux conceptions (une conception architecturale dans un sous-style peut être substituée par une conception architecturale dans le super-style)

II.6.2.2.1 Atouts

- lors de l'implémentation, les entités architecturales peuvent avoir de multiples représentations,
- Aesop fournit des outils d'analyse et de suivie de la modélisation.

II.6.2.2.2 Limites

- Aesop manque de représentation explicite des contraintes de style,
- Aesop utilise les méthodes des classes du style pour coder les contraintes de style, ce qui implique que les contraintes soient liées au code des méthodes,
- Aesop est limité en matière de règles de configurations,
- peu dynamique dans le sens où il est impossible d'intégrer de nouveaux styles au moment de l'exécution.

II.6.2.2.3 Bilan

Aesop est un langage de description architecturale qui semble intéressant pour les outils d'analyse mis à la disposition du concepteur de l'architecture, mais son système de typage lié au code des méthodes reste trop contraignant. De plus, une architecture Aesop n'offre pas beaucoup de dynamicité puisque aucune modification du style n'est possible à l'exécution.

II.6.2.3 ARMANI

Armani est un langage né dans les laboratoires de l'université de Carnegie Mellon aux Etats-Unis. Emanation direct du langage ACME qui avait pour objectif de fédérer différents ADL. Basé sur la logique des prédicats, Armani est un langage riche pour décrire la conception d'architectures logicielles, définir des contraintes et valider des spécifications.

Armani introduit sept éléments de conception architecturale : composants, connecteurs, ports, rôles, composites, propriétés et représentations. La Figure 18 représente un system. Un system représente la configuration de l'architecture ; il décrit la relation entre les composants et les connecteurs via les ports et les rôles.

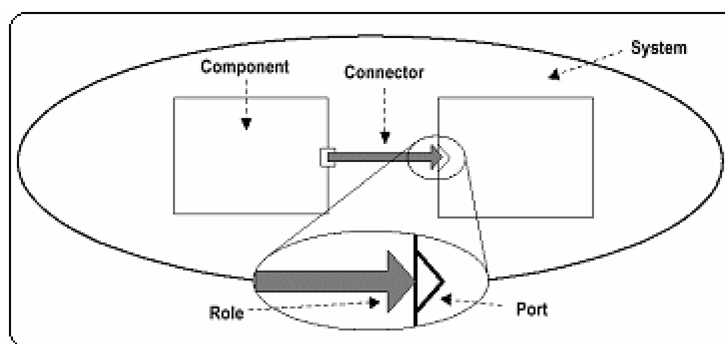


Figure 18 Représentation d'un System Armani

II.6.2.3.1 Style

Armani intègre parfaitement la notion de style telle que nous l'avons décrite dans la section I.4.3. Un style de spécification architectural avec Armani se compose de :

- un ensemble de type de définitions,
- un ensemble de règles de conception (invariants et heuristiques),
- un ensemble d'analyses de conceptions,
- un ensemble de structures minimum.

Les styles architecturaux prennent en compte à la fois la description du système par les éléments (composants, connecteurs,…) pour décrire par exemple un style client-serveur mais aussi les règles de conception (invariants, contraintes).

```
Style <style-name> = {<style-element>};
<style-element> : :=<sequence of : required structure and values
| required properties
| explicit invariants
| type definitions >
```

D'autre part, Armani accepte la notion de sous-style par extension d'un style en utilisant des types et des règles qui définissent un style existant. L'exemple qui suit montre la création d'un style « sub » extension d'un style existant appelé « super ».

```
Style super = {…} ;
Style sub extends super with {
        Component type new-component = {…} ;
        Invariant forall x in self.components ;
};
```

II.6.2.3.2 Propriétés :

Armani introduit cette notion de propriété dans les architectures pour apporter des informations supplémentaires au système et détailler les éléments qui le composent, généralement des informations d'ordre non comportemental.

L'exemple qui suit de l'architecture d'un client-serveur avec Armani montre cette notion de propriété qui permet à l'architecte d'exprimer des contraintes sur les éléments assemblés.

```
System simple_cs = {
        Component client = {
                Port send-request ;
                Properties { request-rate : float = 17;
                        source-code : external-file =
« lib/client.class »}
                }
        Component server = {
                Port receive-request ;
                Properties { idempotent  : boolean = true;
                        max-concurrent-clients : integer = 1 ;
                        source-code : external-file =
« lib/server.class »}
                }
        Connector rpc = {
                Role caller ;
                Role callee ;
                Properties { synchronous : boolean = true ;
                        pax-roles : integer = 2 ;
                        protocole : wright = « … »}
                }

Attachements { client.send-request to rpc.caller ;
                server.receive-request to rpc.calle }
```

II.6.2.3.3 Extensions des propriétés

Nous l'avons vu, Armani permet de constituer des systèmes par assemblage de composants et de connecteurs spécifiés par des propriétés (properties) mais il va

au-delà en offrant la possibilité d'étendre ces propriétés en ajoutant des clauses. Par exemple :

> **Component c = {Port p;} extended with {Property rate : int = 100 ;} ;**

II.6.2.3.4 Atout

- Permet la définition de propriétés topologiques,
- Permet de définir des contraintes architecturales,
- Admet la notion de style qui favorise l'analyse de la conception,
- Offre des outils pour définir, analyser et vérifier les spécifications des architectures.

II.6.2.3.5 Limites

- Armani n'offre pas de propriétés de comportement,
- Le langage n'intègre pas de notion de dynamicité (ajout et suppression d'éléments lors de l'exécution impossible).

II.6.2.3.6 Bilan

Dédié à la capture de l'expertise lors de la conception d'architectures logicielles, Armani est aujourd'hui un des langages de description architecturale les plus performant en matière d'analyse de propriétés.

Cependant, Armani ne permet pas de décrire le comportement des composants à l'exécution. Cette limite lui confère un désavantage certain dans le choix du langage envisagé dans le cadre de nos travaux.

II.6.2.4 π-ADL

ArchWare (π-ADL) est un Langage de Description d'Architecture développé, dans le cadre du projet européen : European RTD Project, par :

- l'équipe de recherche IPG –Informatics Process Group- de l'université de Manchester,
- l'équipe de recherche PPG -Persistent Programming Group de l'université de St Andrews,
- et l'équipe de recherche SEG de l' université de Savoie.

π-ADL est défini comme un langage formel de description d'architecture basé sur le π-calcul. Il permet la composition de plusieurs composants et offre un ensemble d'opérations permettant la manipulation de ces composants. De plus, le langage comporte un mécanisme d'extension basé sur les définitions de styles.

Fondé sur une algèbre de processus, le π-calcul [Oquando-2003]. Ce langage permet de décrire des architectures selon un haut niveau par l'utilisation de mécanisme de style [Alloui-2003]

Comme les ADL précédents, π-ADL fournit des éléments de base: composants, connecteurs, ports, canaux et composites comme le montre la Figure 19.

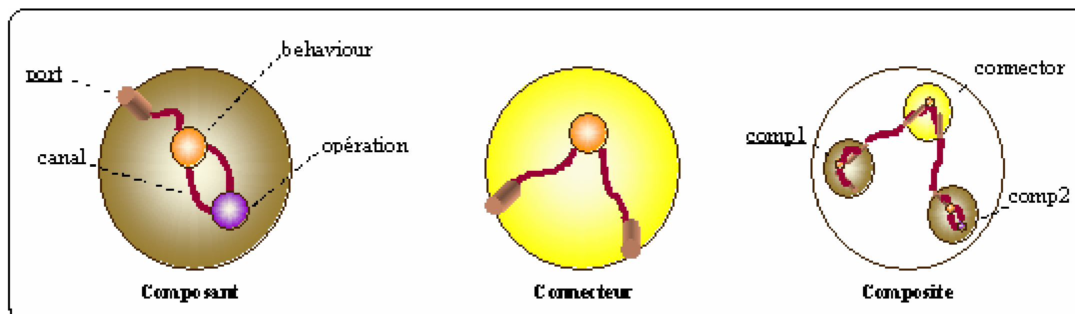


Figure 19 Eléments π-ADL

π-ADL introduit la notion de style, dans le sens ou il permet à l'architecte de réutiliser plusieurs fois la même définition d'un élément au sein de l'architecture. Ainsi, π-ADL ne définit pas nécessairement les éléments d'une architecture mais des styles d'éléments. Compare un style au plan de construction d'une maison où le style représente toutes les caractéristiques propres à la construction de cette maison.

```
System_Base_COTS is style where
Types......
Ports......
Elements......
Constraints......
Templates......
Analysis
```

Le style prend en compte à la fois la description du système par les éléments (composants, connecteurs,…) pour décrire par exemple un style client-serveur mais aussi les règles de conception (comportements, contraintes).

Dans π-ADL, le style apporte un niveau d'abstraction plus élevé que les types. Le style est un mécanisme qui permet d'instancier des types. Ainsi, après avoir choisi d'appliquer un style, l'architecte décrit son type en précisant des contraintes spécifiques à son application mais qui ne doivent pas aller à l'encontre de celles imposés par le style.

Le style prend en compte à la fois la description du système par les éléments (composants, connecteurs,…) pour décrire par exemple un style client-serveur mais aussi les règles de conception (comportements, contraintes).

Dans π-ADL, le style apporte un niveau d'abstraction plus élevé que les types. Le style est un mécanisme qui permet d'instancier des types [Leymonerie et al-2001]. Ainsi, après avoir choisi d'appliquer un style, l'architecte décrit son type en précisant des contraintes spécifiques à son application mais qui ne doivent pas aller à l'encontre de celles imposés par le style.

II.6.2.4.1 Comportement

La spécificité de π-ADL réside dans la notion de comportement exhibé par l'introduction d'une entité behaviour qui spécifie le comportement interne du composant. Le behaviour lie le composant à des opérations[1] qui contiennent les algorithmes de la logique de ce composant. Le comportement est décrit par une suite d'actions coordonnées entre les éléments.

```
behaviour {value x is connection(Any).
compose {
behaviour {value v is any(). unobservable. via x send v. done}
and
behaviour {via x receive y. unobservable. done}
}
}
```

II.6.2.4.2 Evolution

La prise en compte de l'évolution dynamique de l'architecture est une des particularités de π-ADL. Le langage autorise des éléments dynamiques (composants, connecteurs et ports). Ces éléments dynamiques peuvent être ajoutés par l'environnement. Dans l'exemple classique du client-serveur, on peut imaginer qu'une multitude de clients se connectent au serveur sans que ce nombre de clients soit spécifié lors de la conception de l'application client-serveur.

[1] Une opération n'est autre qu'un composant (primitif ou composite) muni des ports libres qui peuvent être composés avec un type de comportement.

π-ADL permet de faire évoluer la composition au cours de l'exécution en décomposant puis en recomposant le composite après apport de nouveaux éléments.

II.6.2.4.3 Atouts

Puisque le langage π-ADL est basé sur π-Calcul (nous détaillerons ce langage plus tard). Il a hérité de ce dernier certain concept, tel que la mobilité, le comportement, synchronisation, parallélisme. Aussi :

- le langage permet de décrire des styles instanciables par des types (favorise la réutilisation de modèles architecturaux),
- d'offrir la possibilité de spécifier le comportement des composants et des connecteurs,
- permer de faire évoluer dynamiquement le système en changeant à la volée la topologie du système décrit,

II.6.2.4.4 Bilan

π-ADL est un langage riche pour la description du comportement de l'architecture. La possibilité de faire évoluer (ajout ou suppression de composants) le système en cours d'exécution lui confère un atout majeur aux autres ADL précédemment étudiés.

II.6.3 Bilan des ADL

Nous venons de présenter quelques langages de description architecturale. Tous ne présentent pas les mêmes caractéristiques. Il s'agit maintenant de définir des critères afin de sélectionner le langage le plus approprié pour la suite de nos travaux.

Ce langage doit satisfaire les critères suivants :

- Permettre une modélisation topologique,
- Etre en mesure de décrire des styles architecturaux,
- Permettre de spécifier des contraintes et des propriétés,
- Etre capable d'exprimer le comportement des composants,
- Permettre l'évolution (ajout ou suppression de composants) durant l'exécution.

Résumons, sous la forme d'un tableau, les différentes caractéristiques de chacun des ADL étudiés :

	UNICON	**AESOP**	**ARMANI**	**π-ADL**
Communication	Symétrique	Symétrique	symétrique	Symétrique
Architecture	Statique	Dynamique	Dynamique	Dynamique

Style	OUI	OUI	OUI	OUI
Contraintes	-	OUI	OUI	OUI
Comportement	-	-	OUI	OUI
Propriétés	-	-	OUI	OUI
Evolution	-	-	-	OUI

Au vu de ce tableau et des critères précédemment établis, notre choix se porte sur le langage π-ADL.

II.6.4 Le langage formel π-CALCUL

II.6.4.1 Introduction :

Les algèbres de processus sont des cadres adéquats pour la spécification et la vérification de systèmes réactifs. Ce domaine a connu différentes approches : CSP [Hoare-1985], CCS [Milner-1989], π-Calcul [Milner-1989]. Dans ces algèbres, tout terme bien formé dénote un processus. L'abstraction fondamentale est qu'on ne s'intéresse au comportement d'un processus qu'à travers un certain nombre de points d'interaction appelés «canaux».

La synchronisation et la communication sont exprimées par des lois de composition internes et externes.

Il faut noter que tant en CCS qu'en π-Calcul, on ne fait aucune hypothèse sur les vitesses d'exécution relatives des différents processus, qui sont donc présumés progresser chacun à leur rythme.

Le π-Calcul est un langage formel, pour raisonner sur les systèmes distribués communicants. Il permet de plus de décrire des systèmes de processus mobiles, c'est-à-dire des systèmes dont les processus ainsi que liens de communication entre processus peuvent changer d'endroit au cours du temps.

Le but recherché par le π-Calcul est l'introduction du concept de mobilité, qui était inexistant dans CCS. L'objectif est de faire évoluer dynamiquement la topologie des applications.

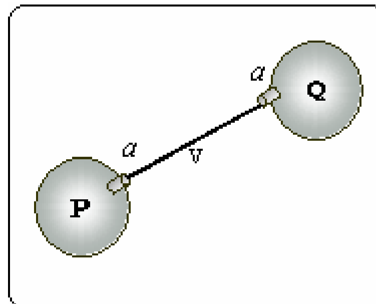


Figure 20 transmission de valeur dans π-calcul

$$(\bar{a}v.P' | a(x).Q')\backslash\{a\} \xrightarrow{\tau} (P' | Q'\{v/x\})\backslash\{a\}$$

Le processus P transmet la valeur v au processus Q, le long du canal a. Q reçoit cette valeur comme une autre variable x, ensuite une substitution sera effectuée en changeant x par v

II.6.4.2 Concept de Mobilité

Le processus P délègue au processus Q la tâche de transmettre v à R.

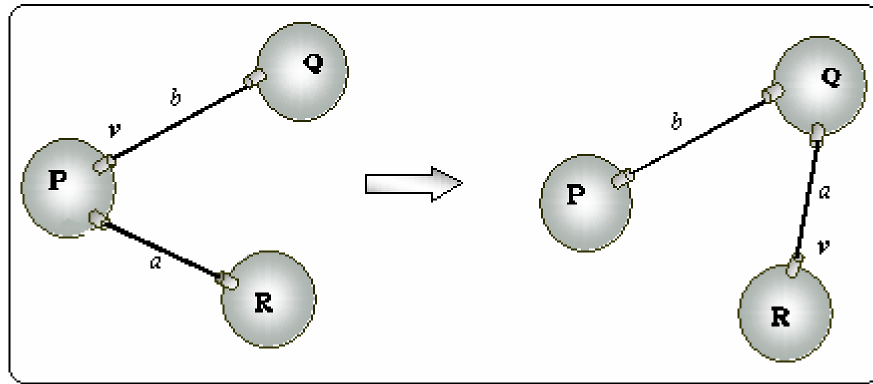


Figure 21 mobilité dans π-calcul

$$(\overline{b}a.\overline{b}v.P' \mid b(y).b(z).\overline{y}z.Q' \mid a(x).R')\backslash\{a,b\}$$

Cette expression correspond au processus de délégation de tâche, par transmission de variable. Le processus P délègue au processus Q la tâche de transmettre v à R. Pour cela P doit d'abord envoyer le canal a ensuite la valeur v au processus Q. Après deux communications (première communication entre P et Q et une deuxième communication entre Q et R), le système s'écrit :

$$(P' \mid \overline{a}v.Q' \mid a(x).R')\backslash\{a,b\}$$

Le processus Q a ainsi acquis dynamiquement le lien de communication a.

Ce mécanisme va au-delà des algèbres de processus antérieur.

II.6.4.3 Syntaxe

Le π-Calcul fournit des opérateurs pour définir un système logiciel en termes de processus. Par exemple, supposons qu'on a deux processus π-calcul : P,Q, les processus suivants sont aussi des processus π-calcul :

- 0 | Processus Inactif (qui correspond au nil du CCS).
- p.P | Préfixer le processus P par une action p.
- P|Q | Mise en parallèle des deux processus P et Q.
- P + Q | Choix indéterministe de l'un des deux processus.
- [x = y] P | Matching – instruction gardée (condition)
- (vx)P | Restriction (le processus P ne peut pas communiquer avec l'extérieur sur le canal x.
- A(x1, x2, …) | Définition d'un processus.
- !P | Réplication (nombre infini de P s'exécute en parallèle).

Le préfixe p peut être :

- **x(y) :** est appelé « préfixe positif ». Il dénote la réception de la variable y sur le canal x.
- $\overline{a}y$ **:** est appelée « préfixe négatif ». Il dénote l'émission de la variable y sur le canal x.
- l'action inobservable τ

Dans tout processus de la forme x(y).P et (y)P, l'occurrence de y entre parenthèses est liée, c'est-à-dire que sa portée est P, une occurrence est dite libre si elle n'est pas liée.

- On note fn(P) « free names », l'ensemble des variables libres de P.
- On note bn(P) « bound names », l'ensemble des variables libres de P
- De plus, « names of P » : n(P)=fn(P)$\cup$bn(P)

II.6.4.4 Sémantique

- Règle d'émission d'une donnée (OUTPUT) : $\overline{x}y.P \xrightarrow{\overline{x}y} P$
- Règle de réception d'une donnée (INPUT) : $x(y).P \xrightarrow{x(z)} P\{z/y\}$
- Somme de deux processus (SUM) : $\dfrac{P \xrightarrow{\alpha} P'}{P+Q \xrightarrow{\alpha} P'}$
- Matching (Match) : $\dfrac{P \xrightarrow{\alpha} P'}{[x=x]P \xrightarrow{\alpha} P'}$
- Communication entre deux processus (COM) : $\dfrac{P \xrightarrow{\overline{x}y} P', Q \xrightarrow{x(y)} Q'}{P|Q \xrightarrow{\tau} P'|Q'}$
- Restriction (RES) : $\dfrac{P \xrightarrow{\alpha} P'}{(x)P \xrightarrow{\alpha} P'} \; x \notin n(\alpha)$
- Mise en parallèle de deux processus (PAR) : $\dfrac{P \xrightarrow{\alpha} P'}{P|Q \xrightarrow{\alpha} P'|Q'} bn(\alpha) \cap fn(Q) = \phi$
- Réplicateur « bang » (REP) : $\dfrac{P|!P \xrightarrow{\alpha} P'}{!P \xrightarrow{\alpha} P'}$

II.6.4.5 Le π-calcul asynchrone

L'idée fondamentale consiste à considérer que l'émission n'est plus une action préfixe mais un processus. On considère ainsi que l'émission asynchrone

$\overline{x}y$ suivie du processus P : $\overline{x}y.P$ équivaut à la mise en parallèle suivante : $\overline{x}y|\ P$. Ainsi, au niveau de la sémantique transitionnelle, on introduit la règle : OUT $\overline{x}y \xrightarrow{\overline{x}y} 0$

II.6.4.6 Le π-calcul polyadique

La spécificité du π-calcul polyadique réside dans le fait que l'on peut transmettre un tuple de valeurs lors d'une interaction $\overline{x}\langle\vec{y}\rangle.P \xrightarrow{\overline{x}\langle\vec{y}\rangle} P$

II.6.4.7 Le π-calcule d'ordre supérieur

Dans le π-calcul d'ordre supérieur, on transmet non seulement des valeurs de « noms » mais aussi des processus. Le π-calcul d'ordre supérieur se conçoit en général sous forme polyadique. $A::=(\lambda\vec{\eta}).P$

II.6.4.8 Bilan

Nous avons essayé dans cette section de donner une présentation générale du π-calcul, puis de quelque versions plus complexe.

Nous trouvons ce langage moins complexe, mieux maîtrisable que d'autre langages formelles. son architecture procédurale, plus proche d'algorithmique. En plus, l'envoi de messages avec de paramètre, le favorise d'être la base d'un langage architecturale tel que π-ADL.

CHAPITRE III. PROPOSITIONS

III.1 Rappel de la problématique

A ce jour la construction d'applications à base de COTS a un intérêt économique évident. Les COTS :

- Sont souvent de bonne qualité.
- Couvrent un spectre très large de besoins.
- Il n'est pas nécessaire de les maintenir en interne.

Une application à base de COTS est moins chère à développer et à maintenir qu'une application développée en interne.

Rappelons que l'enjeu des systèmes à base de COTS est de permettre, à un architecte de construire une application logicielle à partir de COTS existants, d'en défini le fonctionnement qui devra être cohérent pour l'utilisateur final.

Cependant, les travaux dans ce domaine n'offrent pas de formalisme, pour la conception d'une application à base de COTS, qui peuvent regrouper des COTS pour accomplir des tâches bien définies. Tout on respecte certains préceptes [Verjus-2001] :

- **d'hétérogénéité** : les COTS sont hétérogènes,
- **d'autonomie** : les COTS doivent participer dans le système, mais doivent pouvoir, si le besoin s'en fait sentir, rester disponibles dans leurs environnements respectifs (autonomie locale) de même que l'information qu'ils gèrent localement peut être à caractère confidentiel,
- **d'évolution** : les COTS peuvent être sollicités différemment. Ils peuvent quitter le système ou y participer à n'importe quel moment selon les exigences et les besoins du processus,
- **de coopération** : les COTS sont amenés à coopérer de manière à atteindre les buts du système,
- **de contrôle** : le mode de fonctionnement du système (la manière de gérer la coopération) doit pouvoir être contrôlé et être flexible,
- **de distribution :** les COTS peuvent être distribué sur une large zone géographique.

Dans le but de satisfaire la cohérence des clauses citées ci-dessus, nous avons proposé les contraintes suivantes :

- Les COTS sont considérés comme des boites noires qui, ne partagent aucune ressource externe.
- Les COTS sont considérés comme des fournisseurs de services[Estublier-2001]. Cela, implique l'utilisation des services Web comme des facettes de COTS, ce qui leurs permet d'interopérer (Figure 23).

Comme nous l'avons vu précédemment (voir section I.4), l'approche centrée architecture permet dans un premier temps d'établir un modèle hiérarchique par la définition de styles et de sous styles suite, à une série de raffinements, d'arriver à une solution implémentée.

Rappelons que l'enjeu d'une approche centrée architecture est de modéliser un système dont les propriétés, au fur et à mesure du raffinement, sont prouvées par des algèbres mathématiques.

Nous souhaitons fournir un modèle qui répond aux règles mentionnées ci-dessus, dont la solution implémentée reposera sur l'architecture des services WEB

III.2 Proposition d'une architecture de référence

La description formelle d'un style à l'aide de langages de description architecturale n'est pas instinctive. Elle requiert une certaine représentation à la fois de la structure que l'on souhaite décrire et à la fois de ce que le langage nous permet de décrire.

Ainsi, selon notre étude (voir section II.1):

- Les COTS sont des unités de traitement qui ne possèdent pas d'information sur leur environnement.
- Les COTS ne possèdent pas d'information sur les COTS qui peuvent répondre à leurs demandes (qui possède le service, et s'il est libre). Donc il faut qu'il adresse leur demande à un organisme bien précis.
- Le COTS ne sait pas l'objectif général pour le quel il va participer, et comment les tâches doivent s'organiser [Estublier-2001]. Donc il faut une unité qui coordonne les tâches dans le système à base de COTS.

Un système à base de COTS peut être représenter selon plusieurs architectures (architecture d'assemblage des objets, architecture hiérarchique, anarchique, centraliser, etc.). Mais si nous revenons sur les points (2, 3), un organisme que tous les COTS connaissent, et un organisme qui contrôle tous les COTS. Ces deux points nous orientent vers une architecture client/serveur

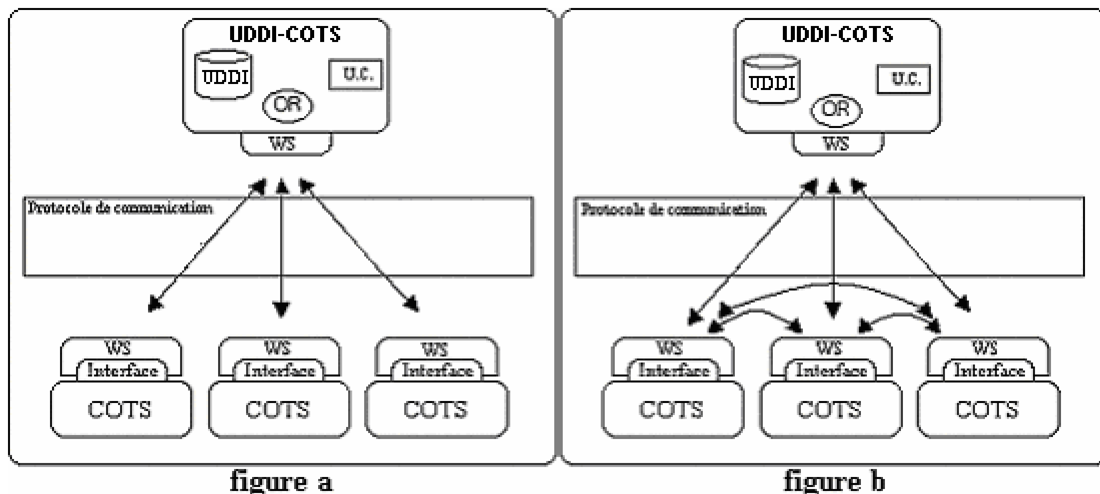


Figure 22 les deux modèles proposés

Dans notre architecture (Figure 22), Nous proposons un serveur[1] que nos nommons UDDI-COTS qui prend en charge les points présidant(2, 3). C'est-à-dire :

- Il a une structure dynamique pour sauvegarder des informations sur les COTS. Dans notre cas cette structure est nommée UDDI, inspirer du concept UDDI des services WEB.
- Il a une structure qui gère le flot de contrôle de tout le système. Dons notre cas cette structure est nommée unité de contrôle (U.C).
- De plus, plusieurs COTS peuvent avoir le même service, mais avec des interfaces et des mécanismes différents pour l'invoquer. Pour cette contrainte, nous avons créé une structure qui oriente les communications. On la nomme unité d'orientation (OR).

En plus, notre perspective est la création de nouveaux services ou COTS par la composition d'autres services ou COTS (voir une composition comme un COTS qui offre des services), le UDDI-COTS peut représenter une composition.

Dans la Figure 22, nous avons proposé deux architectures client/serveur. Dans la première (figure a), la communication se fait à travers le UDDI-COTS, dans ce cas les COTS clients ne se connaissent pas (aucun interaction directe entre les COTS). La deuxième, les COTS peuvent se communiquer directement.

L'avantage de l'architecture de la Figure 22.b est de diminuer la charge sur le UDDI-COTS, cela est dû aux interactions directes entre les COTS. Mais d'après les contraintes présenter par les COTS. Sur tout le cas des interfaces différents, que les COTS publier pour chaque service. Cette contrainte nous oblige à formater la requête selon le destinateur. Et reformater la réponse. Donc il faut un intermédiaire entre le demandeur et le fournisseur. Ce qui donne l'avantage a la première proposition (Figure 22.a). De plus, il y a d'autre avantage tel que:

- Par fois, on base sur une stratégie d'appelle d'offre, on envoie une demande a plusieurs COTS en parallèle, et on attend la première réponse ou le meilleur service.
- Cacher les COTS fils dans une composition de composition.
- Gestion des échanges entre les COTS (approche EAI).

III.3 Définition des services principaux fournis par UDDI-COTS

Le UDDI-COTS doit assurer les fonctions suivantes

- Service publication (UDDI) : il offre certaines fonction pour chercher et localiser des services. Nous nous sommes inspirés, le nom de ce service, du service UDDI [Donsez-2002] fourni par les technologies de service WEB. Nous décrivant leur spécification plus tard.

- Service Contrôle de processus (U.C): il contrôle les activités principales du système (les activités secondaires sont contrôlées au niveau de chaque service)

[1] Nous utiliserons l'abréviation UDDI-COTS pour désigner un serveur (contrôleur) de système dans notre architecture

- Service orientation (OR) : dirige les requêtes vers leur destinateur effectif ou redirige les requêtes en cas d'erreur ou d'absence du COTS.

III.4 Description des éléments de l'architecture

Dans cette partie, nous spécifions les composants principaux de notre architecture selon le style composant / connecteur défini par Fabien et Sorana [Leymonerie-2001], et les décrire en π-ADL.

III.4.1 Service de publication (UDDI)

Ce service stocke les informations de tous les services offerts par les COTS, d'une manière ordonnée. Il peut avoir des services qui réalisent la même tâche, dans ce cas on dit que les services sont semblables, on note S1≡S2. Par fois les services sont semblables mais avec une re-formalisation de la requête, EX : le service SQR(A) ≡ MULT(A,A) (pas d'information externe).Ou le service INC(A) ≡ ADD(A,1). Avec le deuxième paramètre est une constante.

Par fois, il faut composer des services pour avoir le service semblable, EX : MOD (A,B) ≡ IDEM(SUB(A,MULT(B, DIV(A,B)))) donc le service soit le résultat du dernier service. Pour ça, on définit un service Identité, qui rend les même paramètres (IDEM(X)=X).

Par fois on définit dans l'annuaire des services abstraits, c'est-à-dire des services qui n'existent pas réellement, mais ils ont une représentation semblable. Donc pour représenter ces services, on respecte les contraintes suivantes:

- Il ne faut jamais représenter un service abstrait par un autre service abstrait, pour ne pas tomber dans le cas de la récursivité.

- Les valeurs de la réponse doivent être des résultats de service, des constants ou les paramètres initiaux.

Dons ce cas, l'appelle des services semblables, le système crée un nouveau processus qui rend le résultat. Donc une création dynamique des UC. Dans le cadre de ce travail, nous ne traitons pas ce point.

Aussi notre solution ne traite pas les problèmes de la re-formalisation des requêtes aux niveaux de UDDI-COTS, mais il les unifie dans le service UDDI, avec un seul format. Et au niveau de chaque COTS on ajoute une couche pour transformer les paramètres [Estublier-2001].

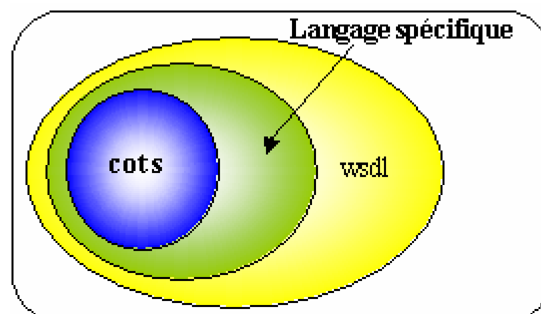


Figure 23 Encapsulation d'un COTS

Bien qu'il existe quelques langages qui peuvent automatiser cette couche tel que FTML[Verjus-2001] , les développeurs de systèmes préfèrent de l'écrire à la main cette couche avec le même langage dons lequel le COTS a été écrit ou un langage proche, puisque :

- Les COTS dépendent des machines.
- Les COTS manipulent des structures spécifiques (Ex : entier sur 4Bit).
- Les COTS sont écrits avec des langages spécifiques.

On définit deux fonctions principales pour UDDI, Demande_Inf_Service et liberer_service.

La première fonction sert à la recherche, et la localisation des services. Elle retourne le COTS qui le fournit, et les paramètres de service. Elle les envoie en format XML.

> **Exemple : < Demande_Inf_Service > SQRT </ Demande_Inf_Service >**
> **Dans cet exemple on cherche une fonction qui s'appelle SQRT. La réponse de UDDI-COTS est la suivant**
>
> ```
> < Inf_service >
>         <Name_service> SQRT
>                 <COTS name="ID_COTS_1"/>
>                 <partIn name="param1" type="xsd:int"/>
>                 <partOut name="param1" type="xsd:int >
>         </Name_service>
> </ Inf_service >
> ```

La deuxième fonction sert comme notification de la libération de service, a la récupération, le UDDI pourra allouer ce service à un autre demandeur[1].

Pour des besoins spécifiques tel que la gestion des erreurs, nous avons ajouté une troisième fonction qui sert à notifier l'unité OR.

Nous cumulons tous ces spécifications, nous décrivons l'architecture de UDDI suivant le style composant/connecteur

> ```
> UDDI_component(Annuaire : Type_annuaire) is style extending (
> component ) where
>         Cammen
>         Ports
>                 port_notification : In_out_P_notification,
>                 port_Demande_Inf_Service :
> In_P_Demande_Inf_Service,
>                 port_Inf_service : Out_P_Demande_Inf_Service,
>                 port_liberer_service  :In_P_UDDI_liberer_service,
>         attributes
>         constituents
> ```

[1] Dans notre architecture, nous ne traitons pas les services épuisables.

```
constraints
attachments
```

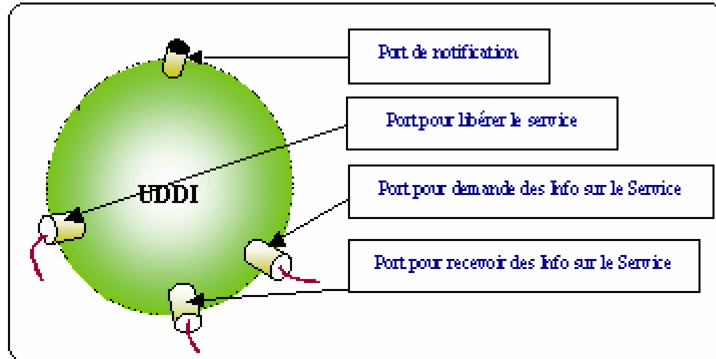


Figure 24 Représentation graphique de UDDI

Nous venons de définir un Composant générique pour UDDI. Nous allons, dans les parties qui suivent, décrire ce composant en π-ADL.

```
Recursive Value UDDI is abstraction(Annuaire is Type_annuaire)
{
notification is Type_notification.
demande_service is Type_demande_offre_service

existe is boolean.
port_notification  is connection(notification).
port_Demande_Inf_Service is
connection(Type_demande_offre_service).
port_Inf_service is connection(Type_ID_fourniseur_service).
port_liberer_service is connection(Type_liberer_service).

Choose{
        {via port_notification receive notification.
        ...-- ! traitement de la notification
        }
Or
        {
                via port_Demande_Inf_Service receve
demande_service .
                existe=(existe and not(existe)).
                iterate Annuaire by nom : view[service :
Type_ID_service, set_COTS:set(Type_fournisseur)]do .
                {
                        …-- ! localiser le service--
                        …--! Choisir selon une stratégie quelles COTS
réalisent le service.
                }
```

```
                    If existe do -- ! s'il y a un service disponible
                    {
                            Cots::Capacite_service =
COTS::Capacite_service -1.
                            via port_Inf_service send Cots.
                    }else -- ! s'il n'y a un service disponible
                    {
                    via port_notification send "pas_disponible"
                    }
        }
Or
        {
                via port_liberer_service receve liberer_service:
Type_liberer_service .
                iterate Annuaire by nom : view[service :
Type_ID_service, set_COTS:set(Type_fournisseur)]do .
                {
                    ...-- ! localiser le service--
                    ...-- ! localiser le COTS dans une deuxième
boucle--
                }
                Cots::Capacite_service = COTS::Capacite_service -1.
        }.
        }.
        service_UDDI (Annuaire ).}
```

III.4.2 Service Contrôle de processus (UC)

L'unité de contrôle est l'entité qui exécute les tâches dans un ordre bien défini, pour aboutir à objectif au quel le système doit accomplir.

Durant sa vie, ce processus a besoin des services pour compléter ses tâches, à ces fins, il fait des appels au service orientation à travers les canaux définis pour ces fins.

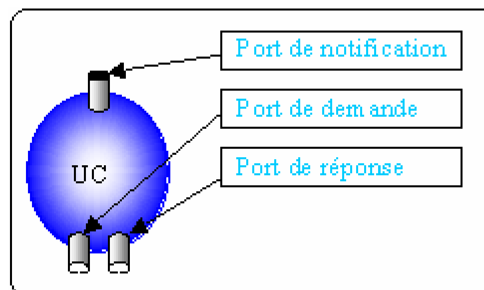


Figure 25 Représentation graphique de UC

Selon la Figure 25, UC a trois ports :

- Port de demande de service : au besoin d'un service, UC fait une demande au près de l'unité d'orientation.

- Port de réponse : l'UC reçoit les réponses de ses demandes a travers ce port.
- Port de notification : en cas d'incident, tel que un long temps d'attente ou une plantation de système, l'UC notifie le système.

L'architecture de UC selon le style composant connecteur est la suivante :

```
UC_ component is style extending ( component ) where
        Cammen
        Ports
                port_OR_UC_demande_service :
Out_P_demande_service
                port_OR_UC_reponse_service : In_P_reponse_service
                port_OR_UC_notification : In_out_P_notification
        attributes
        constituents
        constraints
                forall(p |p in type Out_P_demande_service; q |q in type
In_P_reponse_service
                        implies
                        every sequence {true*.<via p send any>.(not<
via q resive any >)*.<via p send any>.true*} – il faut pas avoir une
réponse avent une demande de service
                        leads to state{false})
        attachments
```

L'architecture ci-dessous représente l'architecture de l'unité UC en π-ADL.

```
Value UC is behaviour {
--! UC a deux ports pour demander et recevoir les services et un port
pour les notifications --
        value port_OR_UC_demande_service is
connection(Type_demande_service).
value port_OR_UC_reponse_service is
connection(Type_Repense_service).
value port_OR_UC_notification is connection(Type_notification)


Value notification is Type_notification.
value demande_service is Type_demande_service.
value Repense_service is Type_Repense_service.

....
unobservable.
....
...--! Preparer les parameters et le service à demander--
via port_OR_UC_demande_service send demande_service.
Choose{
                {via port_notification receive notification.
                -- ! traitement de notification --
```

```
                  -- ! dans notre étude, on a simplifier le cas de
notification a une seule signification (service non disponible)--

                  }
Or
                  {via port_OR_UC_reponse_service receive
Repense_service
                  }
}
....
unobservable.
....
. done
}
```

III.4.3 Service orientation (OR)

L'unité d'orientation encapsule les interactions entre le fournisseur et le demandeur de service, à la demande, il localise le service adéquat, en coopération avec UDDI. Puis il envoie la requête au COTS fournisseur, en attendant la réponse.

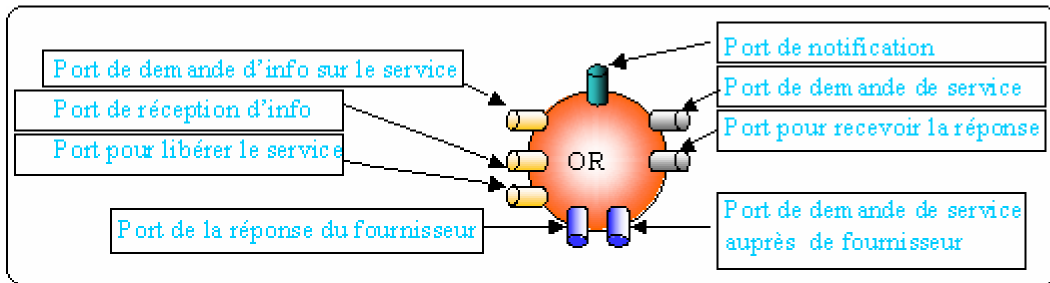


Figure 26 Représentation graphique de OR

L'unité OR joue un rôle très important dans le système, puisqu'elle a des interactions avec tous les éléments du système, donc elle doit assurer le transport de messages à leurs destinataires. Ce problème dépasse notre architecture, puisqu'il sera résolu par le protocole de transmission de message tel que SMTP, HTTP.

L'architecture de l'unité OR selon le style composant/connecteur est la suivante :

```
OR_component is style extending ( component ) where
         Cammen --! Tous les ports sont nécessaires
         Ports
                  port_demande_service : In_P_demande_service
                  port_reponse_service : Out_P_reponse_service
                  port_demande_auFournisseur_service :
Out_P_demande_service
                  port_reponse_duFournisseur_service :
in_P_reponse_service
                  port_Demande_Inf_Service :
Out_P_Demande_Inf_Service
```

port_Inf_service : In_P_Inf_service

port_liberer_service :Out_P_OR_liberer_service
port_notification : In_out_P_notification
attributes
constituents
constraints

forall(**p** |**p** in type **P_demande_service; q** |**q** in type **P_reponse_service**
implies
every sequence {true*.<via p send any>.(not< via q resive any >)*.<via p send any>.true*} – il faut pas avoire une réponse avant une demande de service
leads to state{false})
forall(**p** |**p** in type **P_demande_service; q** | **q** in type **Out_P_Demande_Inf_Service**
implies
every sequence {true*.(not<via p send any>).< via q send any >.true*} – il fault allouer un service sans demande.
leads to state{false})
attachments

III.4.4 Représentation externe du COTS

Le COTS, dans notre étude, n'est qu'une boite noire qui offre des services ou, par fois, demande des services s'il en a besoin.

Un COTS n'est connu que par son interface, qui diffère d'un COTS à un autre. Tous nos COTS sont reliés à l'unité d'orientation (OR), cette dernière doit s'adapter avec chaque interface du COTS. Ce la signifier qu'il faut un connecteur adéquat pour chaque COTS. D'autre part, l'ajout d'un nouveau COTS nécessite un nouveau connecteur adéquat, et une adaptation de l'unité d'orientation pour ce nouveau COTS. Cet inconvénient dégrade considérablement notre solution, surtout du point de vu de la dynamique du système. Pour cela nous avons pensé à unifier toutes les interfaces des COTS. L'approche optée par les interfaces WSDL été la plus proche a notre cas, donc nos l'avons adopté sans trop réfléchi. De plus, cette architecture a un intérêt à la génération du code WSDL, qui sert comme enveloppe du COTS, puisqu'elle est proche de la technologie des services WEB, d'une part. D'autre part, les interfaces prennent le même modèle, c'est-à-dire les même fonctions à publier.

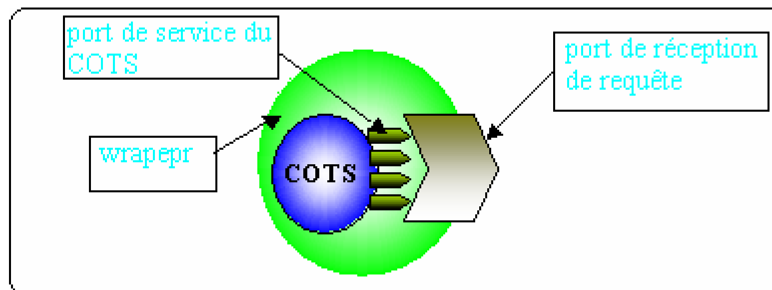


Figure 27 Unification de l'interface de COTS

Le code suivant représenté l'interface du COTS en WSDL :

```
<? xml version="1.0"?>

<definitions name="Offre_service"
          targetNamespace="urn: serveur_orchestration "
          xmlns:typens="urn: serveur_orchestration "
          xmlns:xsd="http://www.w3.org/2001/XMLSchema"
          xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
          xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
          xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
          xmlns="http://schemas.xmlsoap.org/wsdl/">

  <!—Declaration de parametre de message -->

  <types>
   <xsd:schema xmlns="http://www.w3.org/2001/XMLSchema"
            targetNamespace="urn:serveur_orchestration">

    <xsd:element name="Nom_cervice"      type="xsd: string "/>

    <xsd:complexType name="Param">
       <xsd:element name="Nom_param"      type="xsd: string "/>
       <xsd:element name="Type_param"                type="xsd: string
"/>
       <xsd:element name="Valeur_param"         type="xsd:string"/>
    </xsd:complexType>

    <xsd:complexType name="Table_param">
       <xsd:element name="Set_param"      WSDL:arrayType="xsd:
Param[]"/>
    </xsd:complexType>

    <xsd:Type name="Reponse_pret"     type="xsd:boolean"/>

  </types>

  <!—assemblage de parametre, pour notre cas nous avons un seul
parametre -->

  <message name="demande_service">
   <part name="Nom_cervice"           type="xsd: Nom_cervice "/>
   <part name="Param"           type="xsd: Table_param"/>
  </message>

  <message name="reponse_service">
   <part name="Param"        type="xsd: Table_param "/>
```

```
</message>

<!—décrire l’operation demande_pret" -->

<portType name="Offre_service_port">
  <operation name="Offre_service_op">
    <input message="typens: demande_service "/>
    <output message="typens: reponse_service "/>
  </operation>
</portType>



<!-- Spécifie la liaison d’un <porttype> à aux protocole (SOAP, http
GET/POST -->

<binding name="Offre_service_bind" type="typens:
Offre_service_port ">
  <soap:binding style="rpc"
transport="http://schemas.xmlsoap.org/soap/http"/>

  <operation name=" Offre_service_op ">
    <input>
      <soap:body use="encoded"

encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
    </input>
    <output>
      <soap:body use="encoded"

encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
    </output>
  </operation>
</binding>

<service name=" Offre_service">
  <port name=" Offre_service_op " binding="typens:
Offre_service_Bind">
    <soap:address location="http:// Local_COTS.com "/>
                                   < !—URL comme exemple -- >
  </port>
</service>

</definitions>
```

Dans cette interface nous voyons que le service offert par un COTS est devenu un paramètre de la fonction Offre_service. Donc pour appeler un service de COTS, on doit faire référence à l’URL du COTS mais pas au service lui-même. De plus, les services n’ont pas le même nombre de paramètres, ni le même type,

nous avons choisi d'utiliser un tableau de trois champs pour représenter les paramètres nécessaires pour l'appel d'un service. Il va de même pour la réponse (Figure 28).

Nom	Type	Valeur
Param1	Boolean	"False"
Param2	Real	"3.14"
Param3	Integer	"10"
Param4	Stringe	"Cercle"

Figure 28 le tableau de paramètre utiliser par une interface WSDL

Maintenant, nous pouvons rédiger le style de notre COTS comme suite :

- Le port offre_service prend la forme suivante

```
Type Type_demande_service is  view(
Nom_Service : string
Param : Sequence (view(nom : string; type: string; valeur : string)))

Donc le port sera

In_P_demande_service is port with
                    Connections
                              In: connection[Type_demande_service];
                    Protocol
                              Via In receive.
```

Le COTS prend le style composant/connecteur suivant

```
COTS_component is style extending ( component ) where
          Cammen
          Ports
-- !pour les besoin des autres demandeur.
                    port_OR_COTS_Fournisseur_demande_service :
in_P_demande_service
                    port_OR_COTS_Fournisseur_reponse_service :
out_P_reponse_service
-- !pour les besoins du COTS.
                    port_COTS_OR_demande_service :
out_P_demande_service
                    port_COTS_OR_reponse_service :
in_P_reponse_service
-- !pour les notifications
                    port_OR_UC_notification :
in_out_In_out_P_notification
```

```
constraints

exists(port_OR_COTS_Fournisseur_demande_service); --! Les
deux ports (fournisseur)sont nécessaire, si
        exists(port_OR_COTS_Fournisseur_reponse_service);
-- !est passif non le COTS, mais les deux autre s'ils n'existent pas , on
dit que le COTS est autonome

        every sequence {true*.<via
port_OR_COTS_Fournisseur_demande_service  resive any>. (not<via
port_OR_COTS_Fournisseur_reponse_service  send any >)*.<via
port_OR_COTS_Fournisseur_demande_service resive any >.true*} —-!
Il ne faut pas avoir une réponse avant une demande de service
leads to state{false})

        forall(p |p in type P_demande_service;
            implies
            {via p resive D : Type_demande_service > and
D~ ID_repondeur_service != e.name} —-! il faut recevoir les requêtes
qui lui sont adressé
            leads to state{false})
```

III.4.5 L'attachement des composants

Nous avons décrit chacun des éléments qui constituent notre architecture. Nous pouvons maintenant les assembler en une architecture cohérente. Dans le style composant / connecteur les attachements s'effectuent dans la partie Templates.

Notre template, intitulé Attache_SBC, responsable de tous les attachements. Elle prend en entrée l'UC, les COTS et leurs descriptions. Sa structure est la suivante :

```
Attache_SBC is template with
    Parameters as
        Set_COTS : set (view[Cots :COTS_component,
set_service : set (Type_fournisseur)]),
        UC : CU_component.
    Configuration
        Annuaire : Type_annuaire
        attach UC~ port_OR_UC_demande_service
        to OR~port_demande_service
        attach UC~ port_OR_UC_reponse_service
        to OR~port_OR_UC_reponse_service
        attach UC~port_notification
        attach OR~port_notification


        Iterate Set_COTS By C
            Do {
```

```
                         attach
C~port_OR_COTS_Fournisseur_demande_service

                         to
OR~port_demande_auFournisseur_service.
                         attach C~
port_OR_COTS_Fournisseur_reponse_service
                         to
port_reponse_duFournisseur_service.
                         attach C~
port_COTS_OR_demande_service
                         to OR~port_demande_service.
                         attach
C~port_COTS_OR_reponse_service
                         to OR~port_reponse_service
                         attach C~port_notification
                         to OR~port_notification
                         --! parcourir set_service pour générer
l'annuaire --
                         Iterate C~ set_service By S
                               Do {
                                     …….
                                     -- ! Organiser dans la
variable annuaire les service avec
                                     les COTS qui fourni
se service.
}

}
            UDDI= new(UDDI_component(Annuaire)).

            attach OR~port_notification
            to UDDI~port_notification.

            attach OR~ port_Demande_Inf_Service
            to UDDI~ port_Demande_Inf_Service.

            attach OR~ port_liberer_service
            to UDDI~ port_liberer_service.

            attach OR~ port_Inf_service
            to UDDI~ port_Inf_service ;
```

Nous voyons que les liens entre nos composants sont des liens statiques, cela signifier que la vérification du système sera très facile.

III.5 Génération de code (π-ADL à XLANG)

Une description du π-ADL (une spécification) ne peut pas être exécuté directement. Comme nous l'avons présenté dans les sections I.4.2.2 et II.6, une

telle spécification doit être raffinée à une architecture de mise en oeuvre (Figure 1).

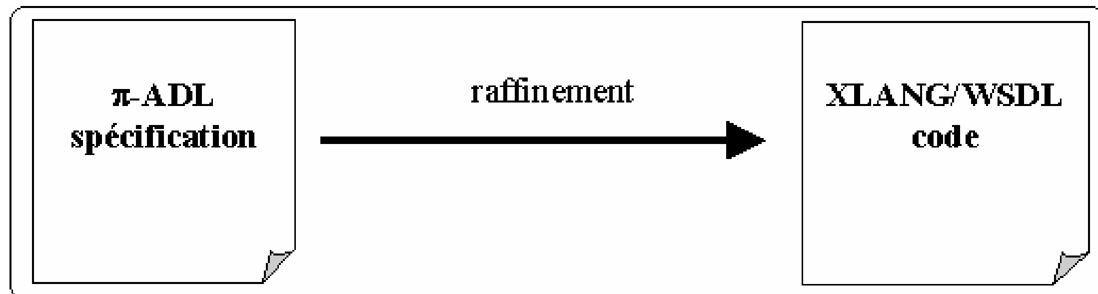


Figure 29 raffinement

Dans cette partie, Nous avons essayé de générer les règles syntaxiques de transformation de π-ADL en XLANG, et WSDL. Nous présentons ci-dessous quelque règle:

- Règles de contrôle de flux : ces règles permettent la transformation du code de l'unité UC écrit en π-ADL en XLANG

Done

Ex:{done}

<empty/>

Ex :<empty/>

{(action.)*action}

Ex :

```
{
décrémentationDeCompteur .

Ajouter_au_file
}
```

sequence ::= sequence [action | process]*

Ex :

```
<xlang:sequence>
  <xlang:action
operation="décrémentationDeCompteur"
activation="true"/>
  <xlang:action operation="Ajouter_au_file"/>
</xlang:sequence>
```

case { project_list }

project_list ::= variant_project_list

| union_project_list

| any_project_list

variant_project_list ::= identifier do clause [or identifier do clause]* or default do clause

union_project_list ::= type do clause [or type do clause]* or default do clause

Ex :

```
behaviour {via x receive y : String.
case { y= "A" do {….}
or y=B do {…}
}
```

switch ::= switch branch* default?

default ::= default process

branch ::= branch case process

case ::= case Qname

la clause If est similaire a case (cas spéciale)

Ex :

```
<switch xmlns:y="http://…/y">
  <branch>
    <case>      y:A    </case>
    <sequence>     …   </sequence>
  </branch>
  <branch>
    <case>      y:B    </case>
    <sequence>     …   </sequence>
  </branch>
</switch>
```

ADL a un comportement récursive

C'est un peut complexe de modeliser ce

while ::= while case processExample

comportement avec la boucle while, mais on va donnée un exemple.

Ex :

```
Recurcive value c(x) is behaviour {if  x > 0
                {x=x-1.c(x)}
}
```

```
<while xmlns:X="http://.../X/">
  <case>
    X:retryEnabled
  </case>

  <sequence>
    ...X
    <!-- retry some behavior -->
  </sequence>
</while>
```

compose { [composition_list]* }

```
compose {
behaviour X1{}
and
behaviour X2{}
                  }
```

all ::= all process*

```
<all>
  <sequence>
    <!--  comportemant X1 -->
  </sequence>

  <sequence>
    <!-- comportemant X2 -->
  </sequence>
</all>
```

- Règles de génération d'interface : ces règles permettent la transformation du code de représentation de COTS en WSDL :

Value ID behaviour { [clause] }	**--! Nous considérons que tous le comportement représente un seul service.** **<definition>** **[Clause]** **<service name = "ID">** **[Clause]** **</service>** **</definition>**
Type x is y	**<Types>** **<xsd : element name =x type="xsd : y"** **name = "x"** **</Types>**
--!π-ADL n'a pas une notion de fonction, donc tous les connexions sont de type One-way. **Value x is connection (y)**	**-- ! dans nos traductions, on considère que chaque connexion représente un seul port.** **<porttype>** **<operation name = "x">** **<input message="y">** **</operation>** **</porttype>**

III.6 Bilan

Dans cette partie, nous avons proposé une topologie, ainsi qu'une modélisation d'un style architectural pour cette topologie.

Nous avons proposé un code π-ADL simultanément avec le style, pour expliquer les partie principale du code.

Nous avons aussi donné les règles générales pour la génération du code XIANG, WSDL.

Revenons à l'architecture, elle répond aux contrainte demander dans la problématique tel que :

- Hétérogénéité : les COTS peuvent être hétérogènes.
- Evolutivité : chaque COTS peut changer ses compétences (augmenter la capacité d'un COTS pour exécuter un service, absence d'un service).
- Autonomie : Chaque COTS est responsable de son travail et de la gestion de ses ressources.
- Ouverture : Arrivée/départ d'un COTS, n'influé pas sur l'architecture globale de notre système.
- Coopération : le COTS lui-même, peut demander des services.

Malgré la richesse de cette architecture, son système d'interaction est le point faible, puisqu'il ne permet que de véhiculer des objets simples entre les COTS. Cela est dû au protocole SOAP utilisé comme protocole de communication.

Nous expliquons dans le paragraphe suivant, comment ce jeu de style doit être mis en œuvre pour concevoir une application.

III.7 Formalisation d'un exemple avec nos styles architecturaux

Nous allons appliquer notre approche pour modéliser une infrastructure industrielle[1]. Dont, une société de construction de voiture (SV) procure des pièces, des fournitures, et des services auprès d'un certain nombre de fournisseurs. Nous voulons passer par le WEB pour communiquer avec les fournisseurs.

III.7.1 Description de l'exemple

Un événement déclenche la nécessité d'une transaction commerciale, par exemple une personne de la société qui cherche à acheter un produit auprès d'un fournisseur. Cet événement particulier demande l'émission d'un bon de commande.

Disons que la société SV veut se procurer dix moteurs de voiture. La première procédure que doit suivre la société SV consiste à créer un formulaire de demande d'achat où figurent les champs qui aideront les personnes appropriées à traiter la demande. Il s'agit d'un document interne qui permet aux différents services de la société SV de communiquer entre eux.

Une fois la demande d'achat remplie, elle est adressée à la personne responsable du traitement des demandes d'achat (s'appelle PRTDA) et de la création des bons de commande ; PRTDA connaît les règles de gestion à appliquer au traitement de ces documents et va les respecter pour traiter la demande.

La première tâche de PRTDA consiste à s'assurer que le rédacteur de la demande est autorisé à dépenser l'argent de la société. Ensuite, il doit s'assurer que l'article demandé n'existe pas en stock.

Dès que PRTDA est convaincue de bien fondé de la demande, elle la place dans la file d'attente des demandes d'achat. D'autre employés de la société SV veulent également acheter des produits auprès de la société de fournitures (SF1). Aux termes d'un accord passé entre la société SV et la société SF1, il a été convenu que, sauf en cas d'urgence, aucun bon de commande ne doit être émis tant que les articles de votre société ne font pas l'objet d'au moins trois demandes d'achat.

Dès que PRTDA aperçoit deux autre demande, il rédige un formulaire de bon de commande, contient les articles velus et leurs quantité.

III.7.2 Création de l'architecteur du système

III.7.2.1 Projection du système selon la topologie

Selon la description ci-dessus, la société SV se compose de divisions. Chaque division a son propre système de gestion (des COTS propre à chaque division)

Chaque COTS-SD peut générer un document BizTalk, a la réception de ce document par le COTS-PRTDA, il vérifie la conformité de document utilisant les règles de gestion présentée ci-dessus. Dès que le nombre d'articles demandés dépassent trois d'articles, il envoie une commande au COTS-SF. La Figure 30 présente le diagramme de séquence du système.

[1] La description du système a été prise du livre [Brian-2001]

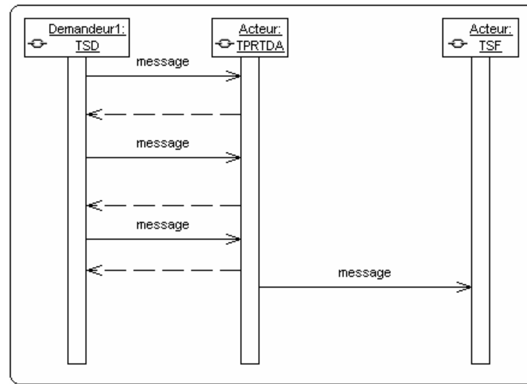


Figure 30 diagramme de séquence

Après cette brève description du système, on va suivre notre démarche pour développer un tel système. Commençons par donner un rôle a chaque acteur du système selon notre topologie.

Le SD, a chaque envoi de message, il demande un service. Donc il présente un simple COTS dans notre topologie.

Le PRTDA, on recevant le message, il l'analyse ensuite, répond positivement ou négativement. Dès que le nombre d'article demandé dépasse trois, le PRTDA envoi un message au SF. Donc ce COTS est un fournisseur de service pour le COTS SD, et demandeur de service pour SF.

De même, le SF est un simple COTS dons la topologie.

De plus, on observe le système réel, un rôle omniprésent du facteur, qui achemine les documents. L'UC dans notre système peut jouer le même rôle. D'où le système prend la forme suivante.

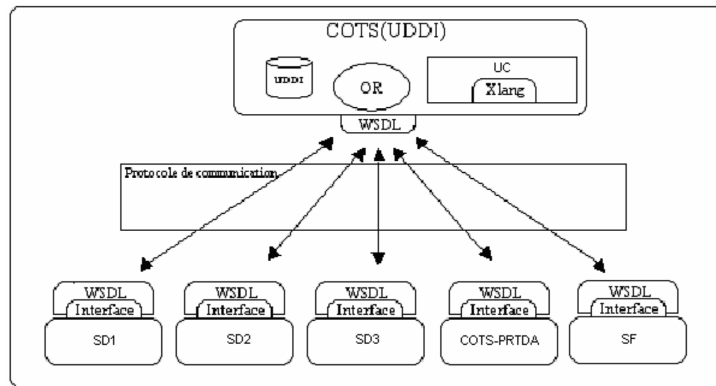


Figure 31 Représentation du système selon notre architecture

III.7.2.2 Description du système en π-ADL

Pour décrire le système, on va reprendre notre style en raffinant certain partie. Dans ce paragraphe nous allons raffiner le COTS-PRTDA. Dans la partie de gestion des contraintes, on va ajouter par exemple, la contrainte « on ne peut envoyer la commande que si on reçoit plus de trois messages de type demande »

Cette contrainte sera ajouter dans la partie « Constraints » dans le nouveau style

```
COTS-PRTDA is COTS_component where
        element
                --! pas d'élément, puisque l'architecture interne du
COTS est inconnu
        attributes
                        NBMessageSD:integer default value is 0
        Constraints
                        forall(p |p in type P_demande_service; q |q in
type P_reponse_service
                        implies
                        every sequence Not {true*.<via q resive
MessageDemande>.(not< via p send MessageCommand >)*.
                        <via q resive MessageDemande>.(not< via p
send MessageCommand >)*.<via q resive MessageDemande>.true*}
                        --! il faut pas commander sauf s'il y a plus de
trois commande.
                        leads to state{false})
        attachments
```

La variable NBMessageSD est déclarée dans le but d'être utiliser comme compteur de message des demandeurs.

Pour l'UC, le cerveau du style, dans ce système il joue un rôle passif. Il achemine seulement les documents de l'expéditeur au destinateur.

III.7.2.3 Raffinement du système

Après cette brève description du système, on passe à l'étape de raffinement.

Puisque l'automatisation de l'étape de raffinement dépasse le cadre de notre étude, passons directement à la programmation.

Simplifions les choses. Supposons que le COTS-SD est une page HTML, où l'employé peut saisir sa commande dessus.

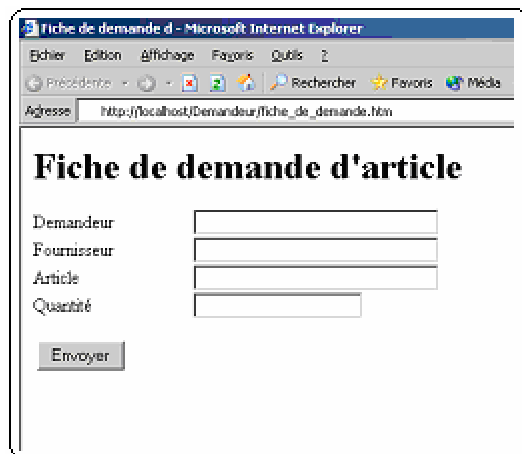


Figure 32 Fiche de demande d'article

Cliquant sur le bouton « Envoyer », une requête de type SOAP envoi un événement au serveur BizTalk. Le corps de l'enveloppe contient les informations

de formulaire. Dans le code ci-dessous nous présentons la fonction principale de la page web.

```
<SCRIPT LANGUAGE="JScript">
            <!--créer une instance de l'objet XMLHTTP-->
var xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");

            <!--//une instance contiendera le document de requete SOAP-->
var SOAPRequest = new ActiveXObject("MSXML2.DOMDocument");
SOAPRequest.async = false;
            <!--//une instance recevra la réponse SOAP-->
var SOAPResponse = new
ActiveXObject("MSXML2.DOMDocument");
SOAPResponse.async = false;

Function EnvoiDemade()
{
xmlhttp.Open("POST", "http://localhost/BizTalk", false);
             strXML =
              "<SOAP:Envelope " +
              "   xmlns:SOAP='urn:schemas-xmlsoap-org:soap.v1'>" +
              "   <SOAP:Body>" +
              "       <ServiceDemandeArticle" +
              "         xmlns='urn:schemas-Demendeur'>" +
              "           <Demandeur>" + Demandeur.value +
"</Demandeur>" +
              "           <Fournisseur>" + Fournisseur.value +
"</Fournisseur>" +
              "           <Article>" + Article.value + "</Article>" +
              "           <Qte>" + Qte.value + "</Qte>" +
                "       </ ServiceDemandeArticle >" +
                "   </SOAP:Body>" +
                "</SOAP:Envelope>"

             SOAPRequest.loadXML(strXML)
             alert (SOAPRequest.xml);
             xmlhttp.setRequestHeader("SOAPMethodName", "
ServiceDemandeArticle ")
             xmlhttp.Send(SOAPRequest.xml);
             alert(xmlhttp.responseText)
             SOAPResponse.loadXML(xmlhttp.responseXML.xml)
             alert (SOAPResponse.xml)
             return
      </SCRIPT>
```

A la réception de la requête par le serveur BizTalk, il l'analyse. Conformément au flux de contrôle suivi par le serveur BizTalk (voir la Figure 33), le serveur envoi le corps de la requête au serveur PRTDA.

Le serveur PRTDA attend les données sous forme d'une requête http de type Post. A la réception, il extrait les informations, ensuite il les charge dans la base de données.

Nous présentant les partie principale de ce programme CGI qui était rédigé en Pascal.

```
If UpperCase(GetVar('REQUEST_METHOD')) <> 'POST' then
  WriteLn('Pas de méthode Post en appelle </body>');

  if getvar('REQUEST_METHOD')='POST' then begin
   parmstring:=getvar('CONTENT_LENGTH');
   if parmstring<>'' then begin
    size:=strtoint(parmstring);
    setlength(parmstring,size);
    for i:=1 to size do read(parmstring[i]);
   end;
  end else
  parmstring:=getvar('QUERY_STRING');
  XMLDocument:=TXMLDocument.Create(parmstring);

  ADOTable.Insert;
  ADOTable.FieldByName('Demandeur').AsString:=
        XMLDocument.ChildNodes.FindNode('Demandeur').NodeValue;
  ADOTable.FieldByName('Fournisseur').AsString:=
        XMLDocument.ChildNodes.FindNode('Fournisseur').NodeValue;
  ADOTable.FieldByName('Article').AsString:=
        XMLDocument.ChildNodes.FindNode('Article').NodeValue;
  ADOTable.FieldByName('QTE').AsString:=
        XMLDocument.ChildNodes.FindNode('QTE').NodeValue;
  ADOTable.Post;
```

Dans ce code, le CGI enregistre les données interceptées dans une base de données Access.

Dès que l'agent PRTDA juge qu'il peut envoyer les commande, il lance un programme qui construit et envoie les requêtes. Ces requêtes seront intercepter par le serveur BizTalk, et rediriger vers le serveur SF de la même manière que la première requête.

Maintenant nous passons au serveur BizTalk, selon la description, il prend la forme de la Figure 33. Pour créer le document Xlang, nous avons utilisé l'EDI « BizTalk Orchestration Designer ». Le code générait par cet EDI est présenté intégralement dans l'annexe B « L'UC de l'exemple en Xlang ». Ici on va présenter les partie principale.

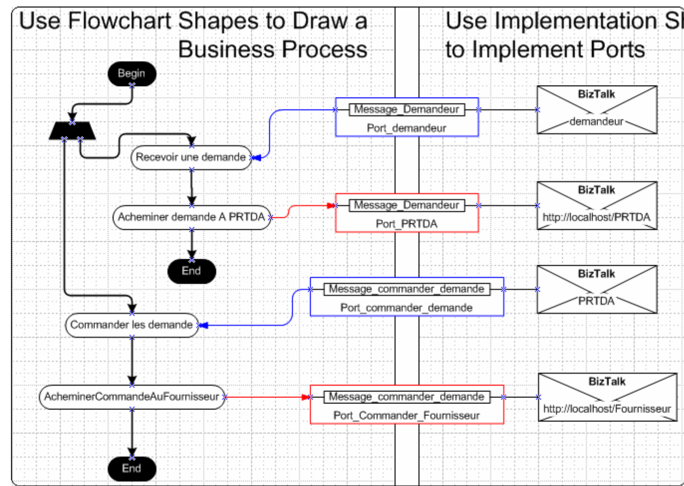


Figure 33 Flux de control suivi par le serveur BizTalk

Selon la norme d'un document Xlang déjà présenté dans la section II.4.2, nous commençons par déclarer les ports a utilisé. Les ports sont présentés comme des enveloppes à gauche dans le diagramme. Ces ports sont utilisés pour la communication entre le serveur BizTalk et les autres serveurs.

```
<portList>
        <port tag="0!38" name="Port_commander_demande"/>
        <port tag="0!64" name="Port_Commander_Fournisseur"/>
        <port tag="0!13" name="Port_demandeur"/>
        <port tag="0!25" name="Port_PRTDA"/>
</portList>
```

Ensuite, on déclare les types de messages communiqués avec le serveur

```
<messageList>
        <message tag="4!3" name="Constants"/>
        <message tag="4!37"
name="Message_commander_demande"/>
        <message tag="4!17" name="Message_Demandeur"/>
</messageList>
```

Puis, on déclare les actions. Mais selon le diagramme, nous avons deus flux de contrôle qui s'exécutent, l'un ou l'autre ou les deux en parallèle. Donc on déclare deux blocs d'action qui s'exécutent, entre eu un choix conditionnel « pick », le premier avènement qui arrive, nous exécutons leurs actions.

```
<pick>
<eventHandler>
<action operation=" Recevoir une demande "/>
       port=" Port_demandeur" activation="true"/>
        <sequence>…</sequence>

</eventHandler >
<eventHandler >
<action operation=" Commander les demande "/>
```

```
        port=" Port_Commander_Fournisseur " activation="true"/>

         <sequence>…</sequence>
</eventHandler >
</pick>
```

III.7.3 Bilan

Nous avons par cet exemple, montré qu'il était possible d'utiliser le style issu de nos travaux pour formaliser un système EAI. Nous avons raffiné ce style pour arriver à une solution concrète.

CHAPITRE IV. BILAN GENERALE

IV.1 Bilan de travail

Dans ce mémoire, nous avons traité le problème des systèmes à base de COTS. L'objectif de ces travaux est de fournir une approche permettant de concevoir des systèmes à base de COTS et de proposer un cadre pour leurs mises en oeuvre.

A travers l'approche proposée, plusieurs problèmes qui n'étaient pas traités simultanément, ont pu être pris en compte et gérés selon une même approche.

Notre proposition et l'implémentation réalisée au travers d'un prototype permettent :

- d'établir une description d'un système à base de COTS;
- de mettre en place une architecture supportant les systèmes à base de COTS et dont le fonctionnement est conforme à la description fournie.

Il est alors possible, pour le concepteur, de construire un système cohérent avec des COTS hétérogènes, autonomes, souvent issus du marché, qui vont être distribués sur le réseau.

Les expériences et les travaux menés jusqu'alors pour atteindre ces objectifs soient ne prennent pas simultanément en compte l'ensemble des contraintes (hétérogénéité, distribution, autonomie, etc.), soient abordent le sujet selon une approche bas niveau(CORBA, COM, EJB) ; cela constitue un frein à l'émergence de solution adaptée[Estublier-2001], attendue par des concepteurs et des entreprises, répondant à une problématique concrète.

Nos travaux couvrent essentiellement trois facettes :

- la description d'une topologie architecturale. et un ensemble de caractéristiques à prendre en compte ;
- la mise en place d'un style architectural, basé sur π-ADL, et quelques règles de contrôle ;
- une implémentation du système selon une approche service WEB.

Nous avons, dans le Chapitre III, proposé une architecture d'un système à base de COTS. Ainsi, une description de cette architecture dans le langage π-ADL.

Nos objectifs couvrent à la fois la proposition d'une architecture flexible et modulable, mais également un ensemble de concepts permettant de définir facilement un système à base de COTS.

Concernant la coopération, elle n'est gérée (et définie) qu'au travers le serveur BizTalk (l'architecture service WEB). Ce serveur, qu'on a jugé apte (voir la section II.5) a gérer explicitement les interactions directes entre les composants, qui constitue le système, basant sur des technologies simples : SOAP, UDDI, WSDL, etc. Ces technologies permettent la distribution, la découverte et l'interopérabilité des services Web.

IV.2 Conception de système

Le concepteur de système à base de COTS dispose d'un ensemble de concepts. L'utilisation de ces concepts permet de modéliser des systèmes à base de COTS selon :

- les concepts communs de l'application à réaliser ;
- les COTS participants ;
- le fonctionnement de système à base de COTS, le type de contrôle et la cohérence entre les COTS.

Nous retrouvons les axes décrits dans[Bolusset et al-2000], c'est-à-dire, décrire le quoi, le qui et le comment.

Le premier point est couvert par le choix de notre architecture et la méthode d'interaction entre les éléments de l'architecture (service WEB comme exemple).

Le second point est couvert par la description des modèles des COTS utilisés.

Le troisième point comprend deux axes : (1) le contrôle de système par l'unité UC, et (2) l'adaptation de système à base de COTS, prise en charge par la description de ce dernier par un langage formelle.

IV.3 Le raffinement

La faible distance qui existe entre l'architecture conceptuelle (architecture de référence) et l'architecture d'implémentation, du fait de la prise en compte des services Web, permet d'alléger le processus de raffinement inhérent au développement de logiciels selon l'approche centrée architecture logicielle. Nous proposerons quelques règles permettant, à partir de l'architecture de référence et de sa formalisation en π-ADL, de projeter un système de COTS vers une architecture concrète (architecture d'implémentation) qui permet la mise en œuvre et l'exécution de ce dernier.

IV.4 Conclusion

Enfin, notre objectif, est de fournir un environnement complet permettant non seulement la modélisation et la mise en oeuvre d'un système à base de COTS mais également d'offrir un support pour facilité leurs développement et leurs évolutions. Nous excusons de ne pas arriver à notre but à cause de plusieurs contraintes tel que le temps, la non-stabilité de langage π-ADL (π-ADL est en phase de conception).

Espérons qu'une prochaine étape accompagnant ce travail, sera pour but de créer des outils pour automatiser cette approche. Ces outils permettront d'analyser la syntaxe d'une description, de vérifier les comportements décrits ou encore de simuler ces comportements de manière visuelle. Un outil permettra en dernier lieu de créer automatiquement une partie du programme informatique à partir de l'architecture.

En fin ce travail m'a permis d'étudier des problèmes du développement des logiciels de demain. A présent, je dispose de connaissances approfondies sur les technologies de service Web et sur leur mise en place. En effet, j'ai utilisé les outils de développement a fin de déployer le petit exemple qui a servi à la validation de mon architecture.

Ce travail aussi, ma donné une bonne expérience sur les architectures logicielles et sur les notions qui y ont trait.

REFERENCES BIBLIOGRAPHIQUES

Références bibliographiques

[Abd-Allah-1996] A. Abd-Allah
Composing Heterogeneous Software Architectures (out 1996)
Doctoral Dissertation, Center for Software Engineering, University of Southern California.
http://sunset.usc.edu/TechRpts/dissertation.html

[Adolphe-2002] Adolphe Francois, Julien Marmel, Dominique Perlat, Olivier Printemps
SOAP Simple Object Access Protocol(01/2002).
http://etna.int-vry.fr/cours/middleware/enseignement/projetsEtudiant/0102/rapportSOAP.pdf.

[Allen-1996] R. Allen et D. Garlan
The Wright Architectural Specification Language(24/09/1996)
http://www.cs.cmu.edu/afs/cs/project/able/ftp/wright-tr.ps

[Alloui-2003] Ilham Alloui, Radu Matuscuo et Flavio Oquendo
The ArchWare : Architecture Analysis Language (01/2003)
(Project Deliverable D3.1)

[Avignon-2000] L. Avignon, D. Joguet, P. Pezziardi.
Intégration d'applications: l'EAI au coeur du e-business. Collection "Solutions d'entreprise",
Edition Eyrolles, novembre 2000.

[BEA et al-2002] BEA systems, Intalio, SAP,Sun Microsystems
Web Services Choregraphy Interface 1.0, Juin 2002.

[Blanc-1999] Benjamin Blanc. « Modélisation et spécification d’architectures logicielles ».
Mémoire de DEA, ENS Cachan, juin 1999.

[Boehm-1986] B.W. Boehm « A Spiral Model for Software Development and Enhancement ».
ACM SIGSOFT Software Engineering Notes, vol. 11, No. 4, août 1986.

[Bolusset et al-2000] Thomas Bolusset, Luciane Da Silva and Flavio OQUENDO.
"Développement à base de composants : une approche centrée architecture pour le
raffinement et l’implémentation de logiciels en Java Beans"

[Brian-2001] Brian E. Travis,
“Programming XML and SOAP for BizTalk serveurs”, ISBN 0-7356-1126-2

[Carney-1997] D Carney.
Assembling Large Systems from COTS Components:
Opportunities, Cautions, and Complexities.(06/1997)
http://www.sei.cmu.edu/cbs/papers/monographs/assembling-
systems/assembling.systems.pdf

[Chauvet-2000] Jean-Marie Chauvet
Services WEB avec SOAP, WSDL, UDDI, ebXML(2002).
Edition Eyrolles

[Christopher-1997] Christopher M. Abts
COTS Software Integration Cost Modeling Study (29/06/1997).
http://sunset.usc.edu/research/COCOTS/docs/USAFReport.pdf

[Cristina-1999] Cristina Gacek and Barry Boehm
Composing Components: How Does One Detect Potential Architectural Mismatches?
University of Southern California

[Davis-1997] Davis, M.J., Williams, R.B

Software Architecture Characterization(1997)
http://delivery.acm.org/10.1145/260000/258380/p30-davis.pdf

[DeLine-1996] R. DeLine. « Toward User-Defined Element Types and Architectural Styles », *Second International Software Architecture Workshop,* pp 47-49, San Francisco 1996.

[Donsez-2002] Didier Donsez
Description et Annuaire pour les WEB Services WSDL & UDDI(16/02/2002)
www-adele.imag.fr/~donsez/cours/wsdluddi.pdf

[Estublier-2001] J.Estublier, H.Verjus, P.Y.Cunin, « Building Software Federation », Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'2001), Las-Vegas, US, juin 2001.

[Garlan-1995] David Garlan. « What is Style?", Proceedings of Dagshtul Workshop on Software Architecture », February 1995.

[Hoare-1985] C.A.R , Hoare, Communicating Sequential Processes. Prentice Hall, 1985.

[Levy-2002] David Levy, « Coordination de Web Services :
langages de description et plate-formes d'exécution » , Septembre 2002
Centre de recherche Xerox Meylan

[Leyman-2001] Leyman F., « Web Services Flow Language », IBM Software Group specification, Mai 2001.

[Leymonerie-2001] F. Leymonerie, « Les architectures logicielles », Mémoire de DEA, Université de Savoie, Annecy – Juin 2001.

[Leymonerie et al-2001] F. Leymonerie, S. Cîmpan, F. Oquendo, « Extension d'un langage de description architecturale pour la prise en compte des styles architecturaux : Application à J2EE », 14emes Journées Internationales Génie Logiciel & Ingénierie de Systèmes et leurs Applications, Paris, 4-6 décembre 2001.

[Leymonerie-2003] Leymonerie F., Cîmpan S., Oquendo F., «ADL Foundation Style», IST-2001-32360, mai 2003.

[Lomme et Martin-2001] S.Lomme et O.Martin « Les Services Web, Implémentation sous WebLogic Server 6.1, Implémentation par Apache SOAP » ,p6,ver 1.0,10 déc 2001.

[Luckman et al-1995] C. Luckman, J. Vera et S. Meldal, «Three Concept of system Architecture», Juillet 1995

[Maurizio-2000] Maurizio M., Sunderhaft N.
Commercial-Off-The-Shelf (COTS): A Survey
A DACS State-of-the-Art Report (12/2000).
http://www.dacs.dtic.mil/techs/cots/cots.pdf

[Medvicovic et al-1997] N. Medvidovic et R. N. Taylor, «A framework for classifying and comparing architecture description languages», In Sixth European Software Engineering Conference, Zurich, Suisse, Septembre 1997.

[Milner-1989] R. Milner, Communication and Concurrency Prentice Hall, 1989.

[Nicolle-2002] Christophe NICOLLE
Définition des services WEB (11/10/2002)
http://www.iai-france.org/projets/batinterop/RT_Nicolle_0210_c.pdf

[Oquando-2003] Flavio Oquendo
π-ADL Tutorial(01/2003)

(Project Report R1.1-1)

[Orfali et al-1999] Orfali R., Harkey D., Edwards J., Client/Serveur : Guide de survie. Troisième édition. Traduction de François Leroy et Jean-Pierre Gout. Edit. Vuibert, Paris 1999, 782 pages

[Patzer-2000] A. Patzer
Programmation Java côté serveur", traduit de l'anglais (2000)
Edition Eyrolles, ISBN 2-212-09109-5, Paris.

[Riveill-2000] Michel Riveill et Philippe Merle
Programmation par composants, 11/2000
http://www.techniques-ingenieur.fr/affichage/DispPdf.asp?pdfID=784960&nid=92819

[Shaw-1996] Mary Shaw et David Garlan
Software Architecture: Perspectives on an Emerging Discipline(1996)
Prentice Hall Publishing, Copyright 1996, Paper (0-13-182957-2).

[Sun-1999] Sun Microsystem
Enterprise Java Beans Specification (1999)
http://java.sun.com/products/ejb.

[Telisson-2002] David TELISSON
Formalisation de styles architecturaux pour les fédérations d'outils logiciels(Juillet 2002)
Mémoire de DEA informatique (université de Savoie).

[Verjus-2001] Hervé VERJUS
Thèse de doctorat Conception et construction de fédérations de progiciels (24/09/2001)
http://www-adele.imag.fr/Les.Publications/reports/PHD2001Ver.pdf

ANNEXE A : DESCRIPTION DU STYLE

System_Base_COTS **is style where**

Types

```
Type_ID_demandeur_service is string,      --! Identificateur de demandeur de service.--
Type_ID_fourniseur_service is string,     --! Identificateur de fournisseur de service(Ex nom de COTS).--
Type_ID_service is string                 --! Identificateur de service(Ex nom de la méthode).--
Type_Param_service is view[               --! Les paramètres nécessaires a la demande d'un service--
Type : string,
Valeur : string
],
Type_Repense_service is any,              --la réponse d'un service--
Type_demande_service is view              --!Pour identifier le demandeur
[                                         Pour identifier le répondeur
ID_demandeur_service :                    Si le demandeur n'est pas connu
Type_ID_demandeur_service,                au début le champ répondeur
ID_repondeur_service:Type_ID_fourniss     reçoit NIL
eur_service                               De plus les services WEB précise
ID_service : Type_ID_service,             expéditeur et le destinateur dans
Param_service : Type_Param_service        leurs requête
        ],
                                          pour ne pas demander au COTS le
                                          services que lui même entrain de
                                          le demander(pb deadlock)--
Type_fournisseur                          --! La capacité d'un COTS de
is view[                                  répondre a plusieurs demandes de
COTS: Type_ID_fourniseur_service,         même service en même temps.--
Capacite_service: integer,
]

Type_description_service is viw           -- !déclaration des service et les
[ID_service : Type_ID_service,            COTS qui les offerts
Param_service : Type_Param_service,
Repense_service :
Type_Repense_service,
COTS_offre is set[Fournisseur]
]
Type_demande_offre_service is view
[
ID_service : Type_ID_service,
Param_service : Type_Param_service,
Repense_service :
Type_Repense_service
],

Type_notification is string               -- ! pour véhiculer certain
                                          message(Ex : les erreurs)--
```

```
Type_liberer_service  is view[
ID_service : Type_ID_service,
ID_COTS : Type_ID_fournisseur_service;
]
```

-- !Notification de libération de COTS--

```
Type_Set_COTS :                          set
(view[Cots :COTS_component,
set_service :                            set
(Type_fournisseur)])
```

Les COTS qui décrivent le système, et leurs caractéristique.

```
Type_Annuaire :  set  (view[service :
Type_ID_service,
set_COTS:set(Type_fournisseur)])
```

-- ! sert a chargée la base de donnée de l'annuaire

Ports

Out_P_demande_service **is port with**
Connections
Out**: connection[**Type_demande_service**];**
Protocol
Via out **send**.
In_P_demande_service **is port with**
Connections
In**: connection[**Type_demande_service**];**
Protocol
Via In receive.

port_demande_auFournisseur_service is port with …. -- ! le même protocole, mais cette demande sera destiner au fournisseur de service (COTS)

Out_P_reponse_service **is port with**
Connections
Out: connection[],
Protocol
Via Out send.

In_P_reponse_service **is port with**
Connections
In: connection[],
Protocol
Via in receive.

port_reponse_duFournisseur_service is port with …. -- ! le même protocole, mais cette demande sera destiner au OR

-- !pour demandes les info de l'annuaire
Out_P_Demande_inf_Service **is port with**
Connections
Out**: connection[**Type_demande_offre_service**];**
Protocol
Via out **send**.
In_P_Demande_inf_Service -- ! de même, mais pour l'autre côté

```
-- ! pour recevoir les info
In_P_inf_Service is port with
        Connections
            In: connection[Type_ID_fourniseur_service],
        Protocol
            Via in receive.
Out_P_inf_Service -- ! de même, mais pour l'autre côté

-- ! Le port de notification bidirectionnel

In_out_P_notification is port with
        Connections
            In_Out: connection[],
        Protocol
            Chose{
                Via In_Out receive.
                    or
                Via In_Out send.
                }

P_OR_liberer_service  is port with
        Connections
            Out: connection[Type_liberer_service ],
        Protocol
            Via Out receive.


P_UDDI_liberer_service  is port with
        Connections
            In: connection[Type_liberer_service ],
        Protocol
            Via in receive.
```

Elements

```
--!Le composent COTS
COTS_component is style extending ( component ) where
    Cammen
    Ports
-- !pour les besoin des autres demandeur.
        port_OR_COTS_Fournisseur_demande_service :
in_P_demande_service
        port_OR_COTS_Fournisseur_reponse_service :
out_P_reponse_service
-- !pour les besoins du COTS.
        port_COTS_OR_demande_service : out_P_demande_service
        port_COTS_OR_reponse_service : in_P_reponse_service
-- !pour les notifications
        port_OR_UC_notification : in_out_In_out_P_notification

    constraints
        exists(port_OR_COTS_Fournisseur_demande_service); --! Les
deux ports (fournisseur)sont nécessaire, si
        exists(port_OR_COTS_Fournisseur_reponse_service); -- !est
passif non le COTS, mais les deux autre s'ils n'existent pas , on dit que le
COTS est autonome
```

```
            every sequence {true*.<via
port_OR_COTS_Fournisseur_demande_service  resive any>.(not<via
port_OR_COTS_Fournisseur_reponse_service  send any >)*.<via
port_OR_COTS_Fournisseur_demande_service resive any >.true*} –-! il
faut pas avoir une réponse avant une demande de service
leads to state{false})

            forall(p |p in type P_demande_service;
                    implies
                    {via p resive D : Type_demande_service > and  D~
ID_repondeur_service != e.name} –-! il faut recevoir les requête qui lui sont
adressé
                     leads to state{false})

-- !Le composent annuaire UDDI
UDDI_component(Annuaire: Type_Annuaire) is style extending ( component
) where
    Cammen
    Ports
            port_notification : In_out_P_notification,
            port_Demande_Inf_Service : In_P_Demande_Inf_Service,
            port_liberer_service  :In_P_UDDI_liberer_service
            port_Inf_service : Out_P_Demande_Inf_Service

    attributes
    constituents
    attachments

-- !Le composent d'orientation
OR_component is style extending ( component ) where
    Cammen
    Ports
            port_demande_service : In_P_demande_service
            port_reponse_service : Out_P_reponse_service
            port_demande_auFournisseur_service : Out_P_demande_service
            port_reponse_duFournisseur_service : in_P_reponse_service
            port_Demande_Inf_Service : Out_P_Demande_Inf_Service
            port_Inf_service : In_P_Inf_service
            port_liberer_service  :Out_P_OR_liberer_service
            port_notification : In_out_P_notification

    attributes
    constraints
            exists(port_demande_service ); --! Tous les ports sont
nécessaires
            exists(port_reponse_service); -- !
            exists(port_demande_auFournisseur_service); -- !
            exists(port_reponse_duFournisseur_service); -- !
            exists(port_Demande_Inf_Service); -- !
            exists(port_Inf_service ); -- !
            exists(port_liberer_service); -- !


            forall(p |p in type P_demande_service; q |q in type
P_reponse_service
                    implies
                    every sequence {true*.<via p send any>.true*.<via p send
any>.true*.<via q
                     resive any>} – il faut pas avoire une réponse avent une
demande.de service
```

```
                    leads to state{false})
    attachments

-- !Le composent UC
UC_ component is style extending ( component ) where
    Cammen
    Ports
            port_OR_UC_demande_service : Out_P_demande_service
            port_OR_UC_reponse_service : In_P_reponse_service
            port_OR_UC_notification : In_out_P_notification
    attributes
    constituents
            exists(port_OR_UC_demande_service); --! Les deux  ports sont
nécessaires
            exists(port_OR_UC_reponse_service ); -- !
            forall(p |p in type Out_P_demande_service; q |q in type
In_P_reponse_service
                    implies
                    every sequence {true*.<via p send any>.true*.<via p send
any>.true*.<via q
                     resive any>} – il faut pas avoir une réponse avent une
demande de service
                    leads to state{false})

    attachments
```

Constraints

Templates

```
Attache_SBC is template with
    Parameters as
            Set_COTS : set (view[Cots :COTS_component, set_service : set
(Type_fournisseur)]),
            UC : CU_component.
    Configuration
            Annuaire : Type_annuaire
            attach UC~ port_OR_UC_demande_service
            to OR~port_demande_service
            attach UC~ port_OR_UC_reponse_service
            to OR~port_OR_UC_reponse_service
            attach UC~port_notification
            attach OR~port_notification


            Iterate Set_COTS By C
                    Do {
                            attach
C~port_OR_COTS_Fournisseur_demande_service
                            to OR~port_demande_auFournisseur_service.
                            attach C~
port_OR_COTS_Fournisseur_reponse_service
                            to port_reponse_duFournisseur_service.
                            attach C~ port_COTS_OR_demande_service
                            to OR~port_demande_service.
                            attach C~port_COTS_OR_reponse_service
                            to OR~port_reponse_service
                            attach C~port_notification
                            to OR~port_notification
```

```
                    -- ! parcourir set_service pour générer l'annuaire--
                    Iterate C~ set_service By S
                          Do {
                                …….
                                -- ! Organiser dans la variable
annuaire les service avec
                                les COTS qui fourni se service.
}

}
          UDDI= new(UDDI_component(Annuaire)).

          attach OR~port_notification
          to UDDI~port_notification.

          attach OR~ port_Demande_Inf_Service
          to UDDI~ port_Demande_Inf_Service.

          attach OR~ port_liberer_service
          to UDDI~ port_liberer_service.

          attach OR~ port_Inf_service
          to UDDI~ port_Inf_service ;
```

ANNEXE B : L'UC DE L'EXEMPLE EN XLANG

Nous présentons ci-dessous le code intégrale de l’unité UC en Xlang. Ce code a été générer par le compilateur du diagramme de flux de l’EDI BizTalk.

```
<?xml version="1.0"?>
<!--
   File created by XLANG Scheduler Engine version 1.0
   at Sun Sep 28 15:18:29 2003

   '==========================
   ' Starting the XLANG schedule
   Set oSked =
GetObject("sked://localhost/C:\Inetpub\application\diagramXlang.skx"
)
   sName = oSked.FullyQualifiedName

   '==========================
   ' Getting proxy for COM port
   ' Set oPort = oSked.Port("yourPortNameBoundToCOM")
   ' Call oPort.yourMethod(args ...)


   '==========================
   ' Get full queue name (MSMQ port)
   ' or moniker (COM port)
   ' sName = oSked.FullPortName("yourPortName")
-->
<module xmlns="urn:schemas-microsoft-com:scheduler"
xmlns:com="urn:schemas-microsoft-com:comscheduler"
xmlns:msmq="urn:schemas-microsoft-com:msmqscheduler">
		<module name="diagramXlang" identity="F25CEC92-F3BF-
45BB-8DCB-D26DFAC2E0C7">
			<schedule>
				<header>
					<portList>
						<port tag="0!38"
name="Port_commander_demande"/>
						<port tag="0!64"
name="Port_Commander_Fournisseur"/>
						<port tag="0!13"
name="Port_demandeur"/>
						<port tag="0!25"
name="Port_PRTDA"/>
					</portList>
					<messageList>
						<message tag="4!3"
name="Constants"/>
						<message tag="4!37"
name="Message_commander_demande"/>
						<message tag="4!17"
name="Message_Demandeur"/>
```

```
</messageList>
<ruleList/>
<contextParameterList/>
</header>
<partition tag="0!43">
<sequence tag="0!48">
<block>
<sink tag="0!28"
comment="Commander les demande">
<portRef
location="Port_commander_demande"/>
<messageRef
location="Message_commander_demande"/>
</sink>
<source tag="0!48"
comment="AcheminerCommandeAuFournisseur">
<portRef
location="Port_Commander_Fournisseur"/>
<messageRef
location="Message_commander_demande"/>
</source>
</block>
<zero tag="0!47"/>
</sequence>
<sequence tag="0!7">
<block>
<sink tag="0!5"
comment="Recevoir une demande">
<portRef
location="Port_demandeur"/>
<messageRef
location="Message_Demandeur"/>
</sink>
<source tag="0!7"
comment="Acheminer demande A PRTDA">
<portRef
location="Port_PRTDA"/>
<messageRef
location="Message_Demandeur"/>
</source>
</block>
<zero tag="0!33"/>
</sequence>
</partition>
</schedule>
<binding>
<moduleRef location="ancestor::*[local-
name()='module' and *[local-name()='schedule']]"/>
<translationHeaderList>
<translationHeader>
```

```
<com:header
txnsupport="DISABLED"/>

</translationHeader>
<translationHeader>
<msmq:header/>
</translationHeader>
</translationHeaderList>
<schemaList>
<Schema xmlns="urn:schemas-
microsoft-com:xml-data" xmlns:dt="urn:schemas-microsoft-
com:datatypes" name="Schema_Constants">
<ElementType
name="ElementType_Constants">
<element
type="__Instance_Id__"/>
</ElementType>
<ElementType
name="__Instance_Id__" dt:type="string"/>
</Schema>
<Schema xmlns="urn:schemas-
microsoft-com:xml-data" xmlns:dt="urn:schemas-microsoft-
com:datatypes" name="Schema_Message_commander_demande">
<ElementType
name="ElementType_Message_commander_demande">
<element
type="Document"/>
</ElementType>
<ElementType
name="Document" dt:type="string"/>
</Schema>
<Schema xmlns="urn:schemas-
microsoft-com:xml-data" xmlns:dt="urn:schemas-microsoft-
com:datatypes" name="Schema_Message_Demandeur">
<ElementType
name="ElementType_Message_Demandeur">
<element
type="Document"/>
</ElementType>
<ElementType
name="Document" dt:type="string"/>
</Schema>
</schemaList>
<messageDeclList>
<messageDecl>
<messageRef
location="Constants"/>
<messageTypeRef
location="ancestor::*[local-name()='binding']/*[local-
name()='schemaList']/*[local-name()='Schema' and
@name='Schema_Constants']/*[local-name()='ElementType' and
@name='ElementType_Constants']"/>
```

```
                                        <ElementType_Constants
xmlns="x-schema:#Schema_Constants" xmlns:sched="urn:schemas-
microsoft-com:scheduler">

                                                <__Instance_Id__
sched:tag="4!9">__Instance_Id__</__Instance_Id__>
                                        </ElementType_Constants>
                                </messageDecl>
                                <messageDecl>
                                        <messageRef
location="Message_commander_demande"/>
                                        <messageTypeRef
location="ancestor::*[local-name()='binding']/*[local-
name()='schemaList']/*[local-name()='Schema' and
@name='Schema_Message_commander_demande']/*[local-
name()='ElementType' and
@name='ElementType_Message_commander_demande']"/>
                                </messageDecl>
                                <messageDecl>
                                        <messageRef
location="Message_Demandeur"/>
                                        <messageTypeRef
location="ancestor::*[local-name()='binding']/*[local-
name()='schemaList']/*[local-name()='Schema' and
@name='Schema_Message_Demandeur']/*[local-
name()='ElementType' and
@name='ElementType_Message_Demandeur']"/>
                                </messageDecl>
                        </messageDeclList>
                        <portBindingList>
                                <portBinding tag="0!38">
                                        <portRef
location="Port_commander_demande"/>
                                        <portTranslation>
                                                <msmq:queue
tag="0!32"
queueName="http://localhost/PRTDA?Channel=PRTDA&amp;QPath
=.\private$\PRTDA" wellKnown="0" queueInfoClsid="27f4275d-707c-
4047-a77b-56a5d0eb034f" messageClsid="167c7c8e-8300-434c-90f1-
d35bb39538fd"/>
                                        </portTranslation>
                                        <messageBindingList>
                                                <messageBinding
tag="0!57">
                                                        <messageRef
location="Message_commander_demande"/>

        <messageTranslation>

        <msmq:message messageType="commanderDemandeType"/>

        </messageTranslation>

        <fieldBindingList>
```

```
<fieldBinding tag="4!44">

<fieldRef location="Document"/>

<fieldTranslation>

<msmq:parameter tag="4!44"/>

</fieldTranslation>

</fieldBinding>

</fieldBindingList>
                                        </messageBinding>
                                </messageBindingList>
                        </portBinding>
                        <portBinding tag="0!64">
                                <portRef
location="Port_Commander_Fournisseur"/>
                                <portTranslation>
                                        <msmq:queue
tag="0!55" queueName="bts://&amp;http://localhost/Fournisseur"
wellKnown="1" queueInfoClsid="27f4275d-707c-4047-a77b-
56a5d0eb034f" messageClsid="167c7c8e-8300-434c-90f1-
d35bb39538fd"/>
                                </portTranslation>
                                <messageBindingList>
                                        <messageBinding
tag="0!68">
                                                <messageRef
location="Message_commander_demande"/>

<messageTranslation>

<msmq:message messageType="commanderDemandeType"/>

</messageTranslation>

<fieldBindingList>

<fieldBinding>

<fieldRef location="Document"/>

<fieldTranslation>

<msmq:parameter tag="4!44"/>

</fieldTranslation>

</fieldBinding>
```

```
        </fieldBindingList>
                                        </messageBinding>

                                </messageBindingList>
                        </portBinding>
                        <portBinding tag="0!13">
                                <portRef
location="Port_demandeur"/>
                                <portTranslation>
                                        <msmq:queue
tag="0!8"
queueName="http://localhost/demandeur?Channel=demandeur&amp;
QPath=.\private$\demandeur" wellKnown="0"
queueInfoClsid="27f4275d-707c-4047-a77b-56a5d0eb034f"
messageClsid="167c7c8e-8300-434c-90f1-d35bb39538fd"/>
                                </portTranslation>
                                <messageBindingList>
                                        <messageBinding
tag="0!18">
                                                <messageRef
location="Message_Demandeur"/>

        <messageTranslation>

        <msmq:message messageType="__Instance_Id__"
labelIsInstanceID="1"/>

        </messageTranslation>

        <fieldBindingList>

        <fieldBinding tag="4!24">

        <fieldRef location="Document"/>

        <fieldTranslation>

                <msmq:parameter tag="4!24"/>

        </fieldTranslation>

        </fieldBinding>

        </fieldBindingList>
                                        </messageBinding>
                                </messageBindingList>
                        </portBinding>
                        <portBinding tag="0!25">
                                <portRef
location="Port_PRTDA"/>
                                <portTranslation>
```

```
                                        <msmq:queue
tag="0!16" queueName="bts://&amp;http://localhost/PRTDA"
wellKnown="1" queueInfoClsid="27f4275d-707c-4047-a77b-
56a5d0eb034f" messageClsid="167c7c8e-8300-434c-90f1-
d35bb39538fd"/>
```

```
                                    </portTranslation>
                                    <messageBindingList>
                                        <messageBinding
tag="0!29">
                                            <messageRef
location="Message_Demandeur"/>

    <messageTranslation>

    <msmq:message messageType="__Instance_Id__"
labelIsInstanceID="1"/>

    </messageTranslation>

    <fieldBindingList>

    <fieldBinding>

    <fieldRef location="Document"/>

    <fieldTranslation>

        <msmq:parameter tag="4!24"/>

    </fieldTranslation>

    </fieldBinding>

    </fieldBindingList>
                                        </messageBinding>
                                    </messageBindingList>
                                </portBinding>
                            </portBindingList>
                            <contextBindingList/>
                            <ruleBindingList/>
                            <callBindingList/>
                        </binding>
                    </module>
</module>
```

ANNEXE D : PUBLICATION REALISEE

FROM ARCHITECTURAL STYLES TO SERVICES ORIENTED ARCHITECTURE: A FIRST ATTEMPT IN DESIGNING AND BUILDING COTS-BASED SYSTEMS

Hervé VERJUS *, Kamel MANSOURI **, Mohammed Salah KHIREDDINE **
* : LISTIC - ESIA Lab., University of Savoie, France.
herve.verjus@esia.univ-savoie.fr
**: Electronic Lab., University of Batna, Algeria.
Mansouri_kl@yahoo.fr
mskhireddine@caramail.com

Keywords : COTS-based system, software architecture, styles, services oriented architecture.

ABSTRACT

The development of big software applications is oriented toward the integration or interoperation of existing software components (like COTS and legacy systems) [11]. This tendency is accompanied by a certain number of drawbacks for which classical approaches in software composition cannot be applied and fail. COTS-based systems are built in ad-hoc manner and it is not possible to reason on them no more it is possible to demonstrate if such systems satisfy important properties like Quality Of Service and Quality Attributes.

The recent works issued in web field allow the definition and the use of a complex web service architecture [12].

Languages such as WSFL [13], XLANG [14] and BPEL4WS [15] support these architectures called Services Oriented Architectures.

The definition of software systems using these languages benefits some existing technical solutions such as SOAP [16], UDDI [17], etc., that permit the distribution, the discovery and the interoperability of web services.

However, these languages do not have any formal foundation. One cannot reason on such architectures expressed using such languages: properties cannot be expressed and the system dynamic evolution is not supported.

On the other hand, software architecture domain aims at providing formal languages for the description of software systems allowing to check properties (formal analyses) and to reason about software architecture models.

The paper proposes a formalisation of COTS-based system (their structure, their behaviours) using architectural styles. The ADL used is π-ADL (based on the π-calculus, supporting style description). The paper will also present our approach consisting in refining an abstract architecture to an executable and services-oriented one.

INTRODUCTION

Information systems are now based on aggregation of existing components that have to cooperate in a precise manner in order to satisfy user needs and software functionalities.

Information systems are more and more complex, need more and more functionality provided by several software applications that already exist (COTS or legacy systems). Reusing and assembling existing components (COTS or/and legacy systems) are questions that cope with some difficulties that are not covered by classical component-based programming solutions like EJB, COM+, CCM, etc. As these are specifications for components development, they do not address the case of COTS-based systems, where source code is not available or/and has been previously developed with other specifications and programming languages. The EAI (Enterprise Application Integration) domain provides integration models and techniques for assembling heterogeneous software applications in a pragmatic way. EAI emerging solutions encompass (1) a distributed architecture using web services and (2) a description of the web services centric architecture, expressed using a web services orchestration/choreography language (i.e. XLANG, WSFL, BPEL4WS, etc.). Information systems based on such technology integrate heterogeneous software components, COTS, using a process-based integration approach, where the process description has to insure the execution correctness of the system. Such information systems, building from COTS, will be called COTS-based systems in the following.

In such context, an issue is still open: the adequation between the information system provided (i.e. its composition) and the functionalities it would able to provide (i.e. to the end user). Because EAI solutions fail in insuring that the information systems provided, succeeds in end-user needs satisfaction.

This paper presents a first attempt in formally describe an information systems building from COTS (or legacy systems). The approach used is based on a architecture-centric development process where the system description is the heart of the process. Using such approach, the (abstract) description can be checked, refined in order to obtain more concrete descriptions that will be executed. In our case, the concrete description (the one that has to be executed) is expressed in a commonly used language dedicated to web services systems description. We assume in this paper that COTS can interoperate as web services.

The paper will first present our approach. In section 3, we introduced our reference architecture for building COTS-based systems. This architecture is based on web services. Then, we will introduce part of the formal description for describing such systems in section 4. We will briefly present the code generation (section 5) and we will then conclude in section 6.

FROM A SOFTWARE ARCHITECTURE SPECIFICATION TO A SERVICES ORIENTED ARCHITECTURE

The architecture of a software system defines the elements that compose the system, and how they interact. The software architecture definition can be made informally, or by using a dedicated language. Different abstraction levels are considered for describing the software architecture. The use of formal architecture refinement guarantees the preservation of properties specified at abstract levels all the way towards architecture implementation.

The architecture centric development process (see figure 1) aims at providing means for defining software systems at a very abstract level. Such descriptions can be then validated in order to check systems properties and are refined in a more concrete description (that allows to deploy the system in a concrete environment).

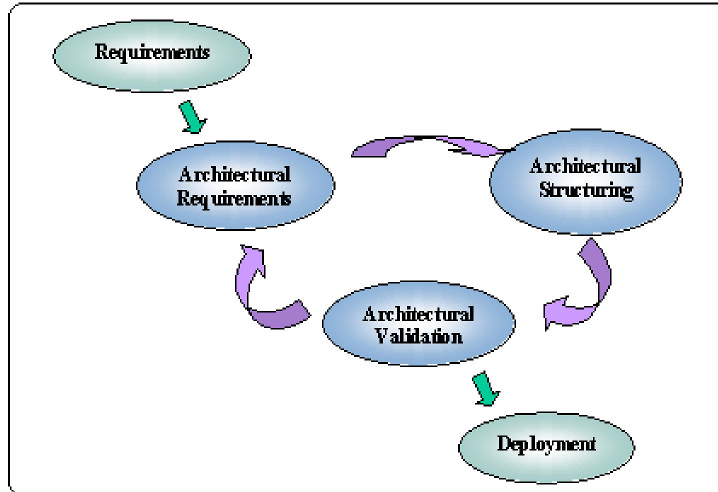


Figure 34 Architecture centric development process

We decided to describe information systems using a formal language (a textual language). The formal description can then be refined in order to obtain a concrete representation. We have to manage the concrete representation generation starting from a formal one. During the generation step, all of the system properties have to be preserved. The targeted representation use web services as integration technology among our components: the COTS.

In our concrete architecture, web services are used as COTS facets, which allow them to interoperate (figure 2). In such concrete context, all well-known languages (WSFL, XLANG, BPEL4WS, etc.) and technologies (WSDL, SOAP, etc.) may be candidates for supporting the deployment and the execution of our systems using web services.

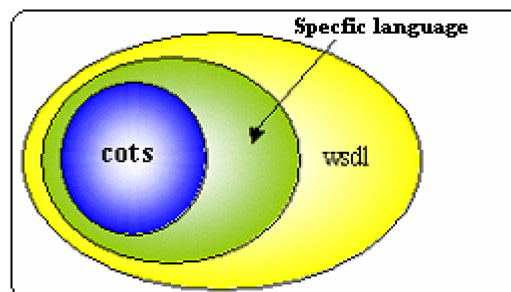


Figure 35 COTS wrappers

In order to (1) support COTS-based information systems (particularly required control when building a system from COTS [24]) and (2) to take into account web services as implementation technology, we defined a particular services oriented architecture (SOA) that will be presented in the next section.

3. A SERVICE ORIENTED ARCHITECTURE FOR BUILDING COTS-BASED SYSTEMS

The architecture of a software system defines the elements that compose the system (i.e. often called “components”), and how they interact in order to satisfy the system requirements [18].

Research in this domain does not offer architecture that can be taken as a reference for a designed COTS based system, especially when dealing with several constraints such as [9]:

- **heterogeneity**: COTS are heterogeneous;
- **autonomy**: COTS must participate in the system and still remaining in their environments (local autonomy) if needed. Even, information managed by COTS can be locally confidential;
- **evolution**: COTS can be solicited in different ways. They can leave the system, or participate in at any moment according to the process requirements;
- **cooperation**: COTS are brought to cooperate in order to reach the system’s goals;
- **control**: operational mode of the system (the manner for managing cooperation) has to be controlled and flexible;
- **distribution**: COTS can be distributed on a large scale.

For satisfying the above-stated clauses consistency, we propose the following assumptions:

- COTS are considered as black boxes that do not share any external resource.
- COTS are considered as services suppliers [4].
- COTS are autonomous and have no knowledge of other COTS availability, nor they have information on their environment.
- COTS have no idea about the objective for which they are going to participate.

Dealing with our objectives as well as with the COTS features, we define a reference architecture. This architecture is a services oriented architecture that is deducted from a reference architecture for building COTS-based federations (in which the control among COTS can be tuned) [10, 24].

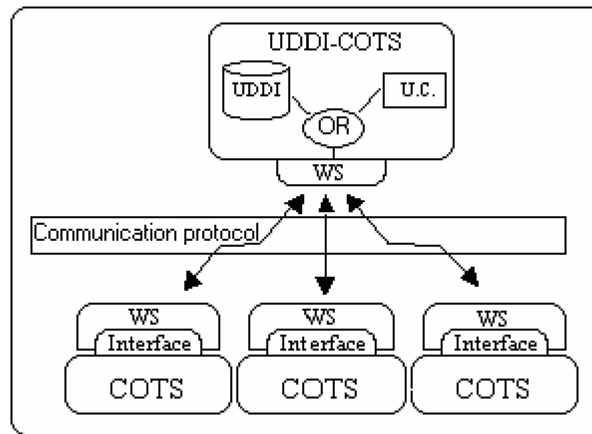


Figure 36 The reference architecture

From the implementation point of view (i.e. the concrete representation), the architecture is basically a set of web services interacting together. From the abstract point of view, the previously shown architecture is inherited from the one presented in [24] for which COTS orchestration (i.e. choreography in SOA) is an important topic.

We restricted the architecture presented in [4][9][23] in order to simply our study (i.e. the part called Control Foundation is the one that we reuse in the architecture shown in figure 3, with some extensions taking into account web services control needs).

In this architecture we propose a COTS-UDDI, that is composed of:

- **Publish policy** service **(UDDI)**: stores the information of all services offered by the COTS, in a neat manner, and permits, by calling functions, to get their positions. It may have services that achieve the same goal, in this case we assume that services are similar and we note S1=S2, these services are stocked in even level in UDDI.

- **Orientation service (OR)** : the unit of orientation is a process that manages interactions between the supplier and the requester of services; on the request, orientation service gets the adequate service, in cooperation with the UDDI. It sends then the request to the supplier and waits for its response. On error, the request is redirected.

- **Controls process service (UC)**: it schedules the main activities of the system (other activities are controlled at the level of every COTS). It is the services orchestration unit.

When a client (in terms of classical client-server architecture) needs a service, it sends its request to the O.R. which gets information about the service from UDDI (information that consists on supplier and the supplier's capabilities). On the response, the O.R. unit can either suspend or send its request to the supplier. At the end of the process, it even sends the answer to the client.

SYSTEM ARCHITECTURAL DESCRIPTION USING STYLES

In the case of complex COTS-based systems (like information systems) classical approaches fail:

- Industrial need systems that are adapted to their requirements: the design (including properties) of such systems is a crucial step but systems designs/models have to be validated before implemented.

- COTS are specific software components with which components classical integration patterns or idioms are not relevant: COTS have to be characterized as well as their integration (the "glue" has also to be formalized).

COTS-based system models (when existing) cannot be checked nor validated. That is, one cannot reason on models nor analysis can be made on such models. This lack of formalization has following consequences:

- design (of complex systems) expertise cannot be caught nor maintained;

- there is a gap and discrepancies between the design and the execution. It is impossible to guarantee that the execution will be conformant to the design;

- the COTS-based systems evolution (replacement, deletion, addition of COTS, changing system behaviour, etc.) is not well supported nor it can be validated;

- crucial properties (safety, completeness, consistency, etc.) of the systems are not taken into account.

The work on the software architectures formalisation proposes solutions that might meet the needs issued by the identified limitations.

The architecture-centric development process we propose (see figure 4) is quite different to the classical software development process [20]: if the system behaviour does not fit the requirements, the architecture description can be modified without restarting entirely the development process. Representations (architectural descriptions) are also checked at every stage of the process before generating the code.

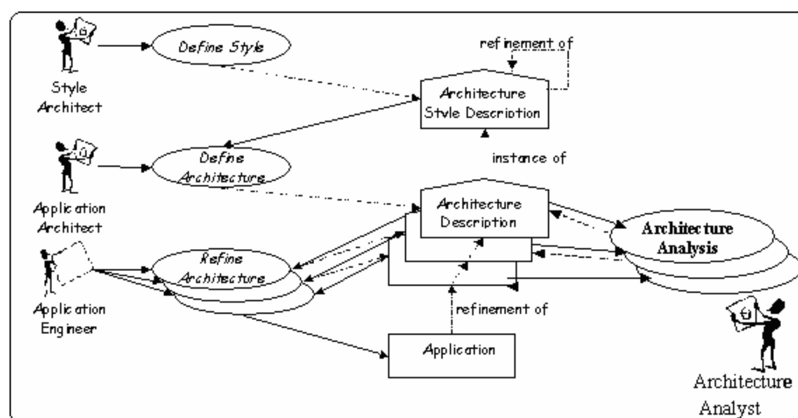


Figure 37 Architecture centric development process

The work on architecture centric approaches for software development has been very fruitful during the past years, leading, among other results, to the proposition of a variety of Architecture Description Languages (ADLs) [27], usually accompanied by analysis tools. The enthusiasm around the development of formal languages for architecture description comes from the fact that such

formalisms are suitable for automated handling. These languages are used to formalize the architecture description as well as its refinement. The benefits of using such an approach are manifold. They rank from the increment of architecture comprehension among the persons involved in a project (due to the use of an unambiguous language), to the reuse at the design phase (design elements are reused) and to the property description and analysis (properties of the future system can be specified and the architecture analyzed to check their verification).

The different ADLs proposed share some common concepts, especially on the way the structural aspects are treated. Thus components entail the functionality of the future systems and interact and communicate via connectors. Interfaces of components and connectors are structured as sets of ports. An architecture description is a configuration of such interacting components and connectors. Of course there are variations from one language to another, due to their historic evolution and to their purpose. Thus some languages are general purpose (ACME [7], π-Space [24]) while others are dedicated to a specific domain (like META-H [26], which is dedicated to the real-time multiprocessor avionics system architecture).

If the structural aspect are covered by all the ADLs, the behavior is handled by only some of them (like Wright [1], π-Space [24], CHAM [2]). Wright and π-Space are based on process algebra (CSP for Wright and π-calculus for π-Space), which allows the behavior description. The use of π-calculus in the case of π-Space leads to the possibility of describing dynamic architectures. A CHAM allows to specify behavior as a succession of chemical reactions in a chemical solution.

A central aspect of architectural design is the use of recurring organizational patterns and idioms – or architectural styles [5]. A number of benefits in using architectural styles are identified [5], such as design and code reuse, ease on system understanding, interoperability improvement, style-specific analyses an visualization.

Architectural styles are means for intensive design reuse. They provide a design framework to software architects. They are also means for guaranteeing that the architecture will exhibit certain properties. Thus knowing that a component or an architecture follows a particular architectural styles, induces that the considered component or architecture has all the properties ensured by the style.

Styles can range from very generic ones to very specific ones: the expression of a specific style is an architecture. The range of generality is achieved by building hierarchies of styles. Thus, a style can have sub-styles that enforce the constraints imposed by it, making the definition more specific.

A style represents a family of architectures that are compliant to the style. The more specific a style definition, the smaller the family of architectures it represents. In the case of completely specific styles this set has cardinality 1, i.e. the style represents a family of architectures with only one element.

Some of the existing ADLs propose mechanisms for style definition. It is the case for Aesop [6], Armani [8], Acme [7], Unicon-2 [3], σπ-Space [25] and π-ADL [20].

Among these languages, π-ADL is the only one that :

- allows the architecture structural modeling as well as the behavioral description (as an extension of π-calculus);

- supports properties/constraints definition;
- supports dynamic evolution of the systems;
- proposes styles mechanism.

Basically, a style is instantiated by architecture descriptions; at instantiation all the constraints have to be verified. Sub-styles can be defined, by adding constraints to an existing one. For formalizing the particular SOA (our reference architecture) we propose, we defined dedicated styles inherited from the component/connector style already defined 0. The following figure (figure 5) presents an abstract of our styles library definition. It concerns the orientation unit introduced in the previous section.

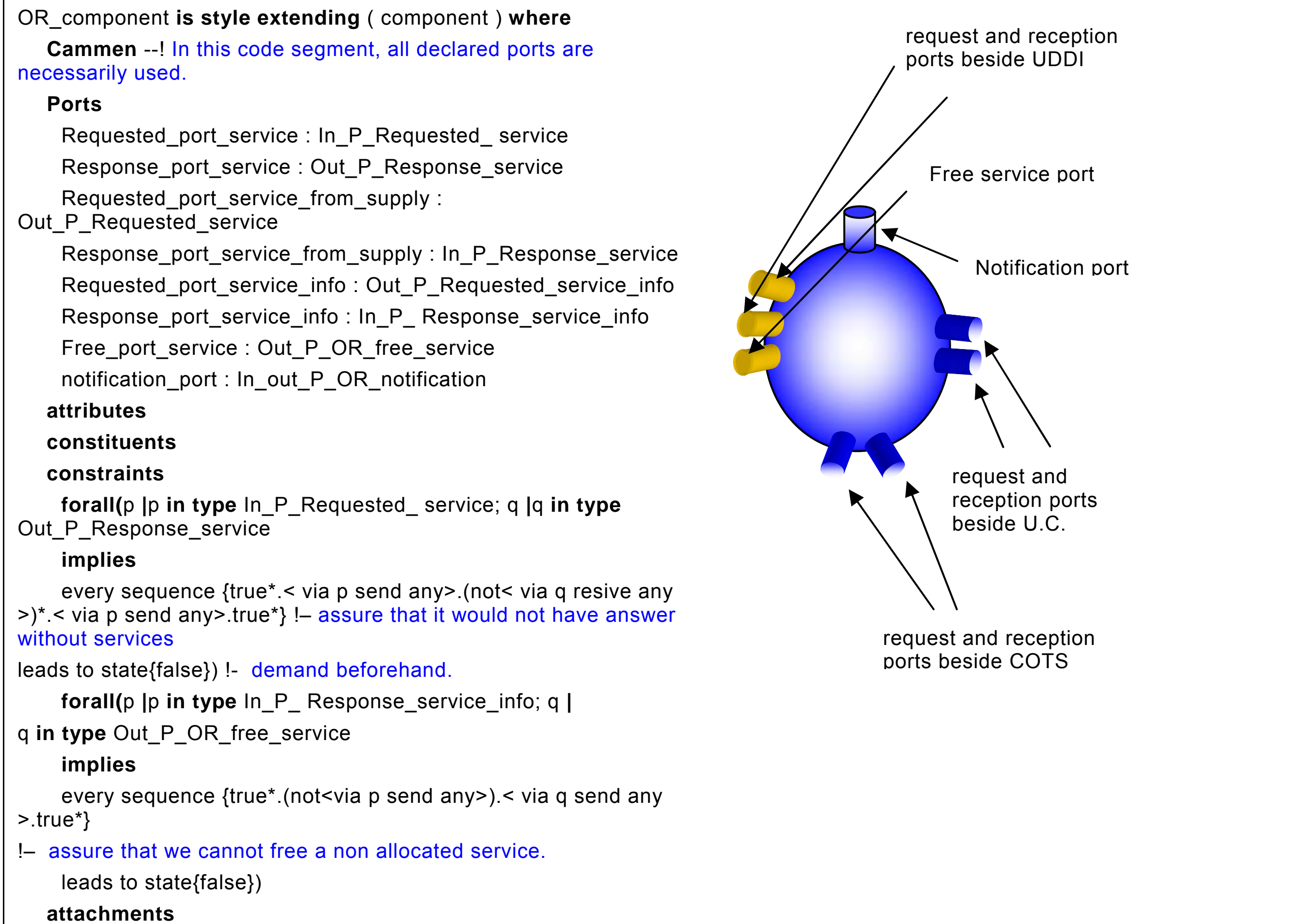


Figure 5 - Orientation unit description

In the π-ADL code fragment above,

- in *cammen* section, static ports are defined for OR unit (whereas, port types are declared somewhere in the code);
- in *constraints* section: a set of properties for the OR unit (such properties will then be checked). Such property expresses that OR provides responses to a client only if it has previously sent requests. This property is expressed in AAL [22] (based on temporal logic) that is encapsulated in π-ADL styles mechanisms for properties expression.

The figure 5 is an extract of the complete π-ADL styles library for describing COTS-based systems according to the architecture presented in section 3.

Starting from the described styles, software designer is now able to define architectures following these styles. It means that an architecture referencing our architectural styles inherits all properties defined in the styles; such architecture can be checked according the styles definition and then may be instantiated.

Styles may be also specialized in sub-styles. In such case, sub-styles inherit again all of the properties of their super-styles.

The next section will show the refinement step consisting in generating implementation code (in order to execute instantiated architectures) from a π-ADL specification (a formal description of a COTS-based architecture using our architectural styles).

FROM π-ADL DESCRIPTION TO XLANG CODE GENERATION

A π-ADL description (a specification) cannot be directly executed. As we presented in sections 2 and 4, such specification has to be refined to an implementation architecture (figure 6). The targeted and executable architecture is a specific services oriented architecture for building COTS-bases systems. Such web services architecture can also be described using classical web-centric languages such as WSFL, XLANG, BPEL4WS, etc.

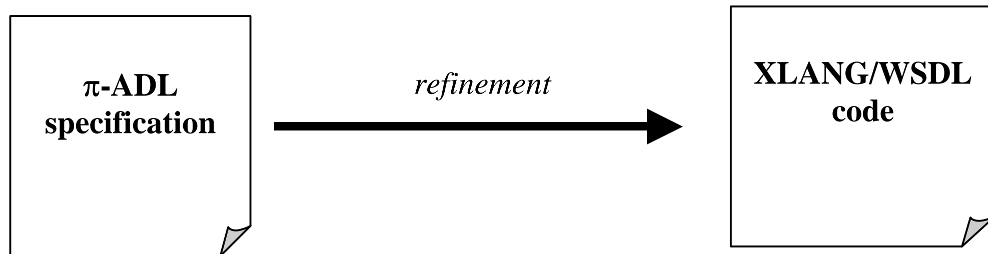


Figure 6 – Refinement step

We aim at generating XLANG and WSDL code using rewriting rules. Such rules support the transformation from a π-ADL specification to XLANG and WSDL code. As example, we focus on the Unit Control. The rewriting rules are presented:

XLANG generation code rules:

Inaction behaviour: the empty process does not contain any actions. It plays a role rather like the null statement of ordinary programming languages. The rule of transformation from π-ADL to XLANG is:

{done} → <empty/>

Ex :<empty/> **Ex**:{done}

Sequence action: A sequence contains zero or more actions or processes which are executed sequentially. The sequence concludes when its final action or process is terminated. The rule of transformation from π-ADL to XLANG is:

{(action.)*action} → sequence ::= sequence [action | process]*

Ex :

```
{
dec_Speedometer .

Sleep
}
```

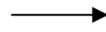
Ex :

```
<xlang:sequence>
  <xlang:action operation="
dec_Speedometer "
activation="true"/>
  <xlang:action
operation="Sleep"/>
```

</xlang:sequence>

Switch action : The switch process supports conditional behaviour. The branches of the switch are considered in the order in which they appear. The first branch whose condition holds true provides the process for the switch. If no branch with a rule is taken, then the default branch will be proceeded. If the default branch does not exist, a default branch with an empty process is deemed to be present. The switch is completed when the process form of the selected branch completes.

Case { project_list }
project_list ::=
variant_project_list
| union_project_list
| any_project_list
variant_project_list ::=
identifier do clause [or identifier do clause]* or default do clause
union_project_list ::= type do clause [or type do clause]* or default do clause



switch ::= switch branch* default?
default ::= default process
branch ::= branch case process
case ::= case Qname

Ex :
behaviour {via x receive y : String.
case { y= "A" do {….}
or y=B do {…}
}

Ex :
<switch xmlns:y="http://…/y">
<branch> <case> y:A </case>
<sequence> ... </sequence>
</branch>
<branch> <case> y:B </case>
<sequence> … </sequence>
</branch>
</switch>

Composition process : The Composition process executes each of the specified sub-processes concurrently.

compose {
[composition_list]* }

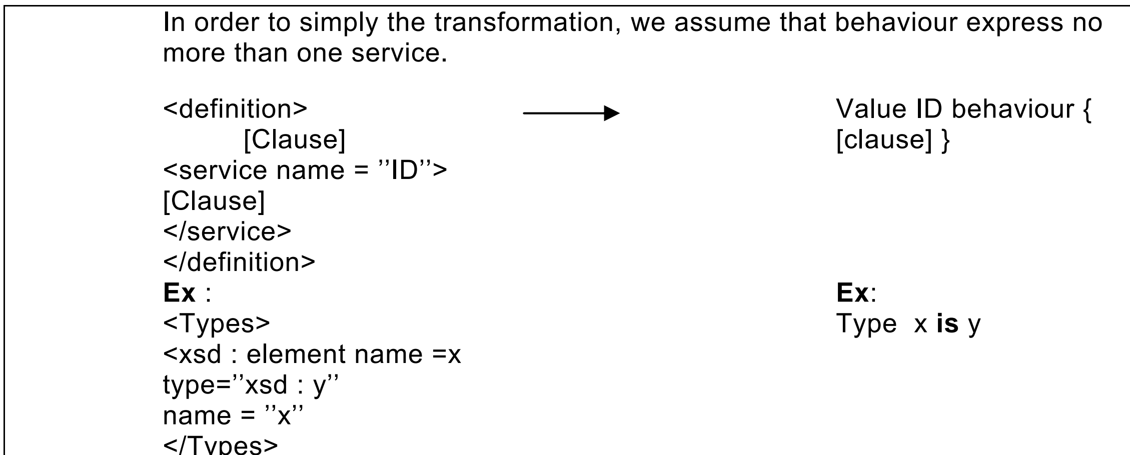
all ::= all process*

Ex :
compose {
behaviour X1{}
and
behaviour X2{}
}

Ex :
<all>
<sequence> <!-- behaviour X1 --> </sequence>
<sequence> <!-- behaviour X2 --> </sequence>
</all>

As XLANG code mainly expresses services orchestration (i.e. basically control flow in our previous transformation rules), WDSL code states for service interfaces definition. Some rules for generating WSDL code are shown in the following.

WSDL generation code rules:



In order to simply the transformation, we assume that behaviour express no more than one service.

<definition> [Clause] <service name = ''ID''> [Clause] </service> </definition>	⟶	Value ID behaviour { [clause] }
Ex : <Types> <xsd : element name =x type=''xsd : y'' name = ''x'' </Types>		**Ex**: Type x **is** y

When XLANG and WSDL pieces of code have been generated, the concrete architecture can now be executed by deploying it, using a specific web-services application server (like BizTalk, etc.). From the COTS side point of view, each COTS has to be integrated with their corresponding web services.

CONCLUSION AND ONGOING WORK

Building COTS-based system generally fails due to non formal approaches (often ad-hoc solutions like EAI) used. In [4], [23]and [10] we claim that designing and building COTS-based systems addresses lots of issues: the gap between the design level and the implementation one is one of them. Because COTS (as well as legacy systems) already exist, it is not possible to generate all of the software system code but we have to deal with the “glue” between such software components (COTS, etc.). Instead of managing an ad-hoc approach (i.e. rewriting approach, etc.) consisting in refinement steps from specification to implementation code generation, we have to focus on the “glue” that have to guarantee the properties of the COTS-based system the designer is interested in. Our approach is divided in two parts:

the definition of an architecture that is convenient for the design of COTS-based systems as well as it is also closed to a concrete architecture (in our case, a Services-Oriented Architecture);

an architecture-centric development process using a formal ADL as a specification language. Such language allows us to define our reference architecture in architectural styles. COTS-based systems architectures are expressed using these styles and can be validated against properties (both structural and behavioural). The (abstract) architecture of a COTS-based system is then refined in order to produce an implementable SOA that is entirely compliant and checked with the abstract architecture that is expressed in XLANG and WSDL piece of code. But the behavior of COTS-based system largely depends of each COTS involving in the system; that is, it is not possible to reason on each of the architecture components and one cannot demonstrate all of the properties a designer is interested in (completeness, safety, security, vivacity, etc.) but only a sub set of them.

We will focus on services composition at a high level of abstraction (composition properties will be formally studied) that would be permit to formally and entirely

specify SOA and to generate an executable architecture. In such case, we will be able to reason on each of web service as well as on their respecting properties (QoS) and on their composition. We also will to apply our approach in an industrial context like an information system deployed with an EAI solution.

REFERENCES

[1] Allen, R. and Garlan, G. "A formal basis for architectural connection", *ACM Transaction on Software Engineering and Methodology*, 1997.

[2] Wermelinger M., "Towards a Chemical Model for Software Architecture Reconfiguration", *Proceedings of the 4th International Conference on Configurable Distributed Systems*, 1998.

[3] DeLine, R.: "Toward User-Defined Element Types and Architectural Styles", *Second International Software Architecture Workshop*, San Francisco, 1996.

[4] Estublier, J., Verjus, H., and Cunin, P.-Y., "Building Software Federation", *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'2001)*, Las-Vegas, USA, June 2001.

[5] Garlan, D. "What is Style?", *Proceedings of Dagshtul Workshop on Software Architecture*, February 1995.

[6] Garlan, D., Monroe, R. and Wile, D., "Exploiting style in architectural design environments", *Proceedings of SIGSOFT'94 Symposium on the Foundations of Software Engineering*, ACM Press, December 1994.

[7] Garlan, D., Monroe, R. and Wile, D., "Acme: an Architecture Description Interchange Language", *Proceedings of CASCON'97*, November 1997.

[8] Monroe, R., "Capturing Software architecture Design Expertise with Armani", *School of Computer Science Carnegie Mellon University*, Pittsburgh, January 2001.

[9] Verjus, H., "Conception et construction de fédérations de progiciels", *PhD thesis* (in french), LLP-ESIA Lab., University of Savoie, Annecy, France, September 2001.

[10]Verjus, H., Cîmpan, S., Telisson, D., "Formalising COTS-based federations using software architectural styles", in *Proceedings of the 15th International Conference Software & Systems Engineering and their Applications*, December 2-5, 2002, Paris, France.

[11]Abts C.,"COTS Software, Integration and Usage Issues", Center for Software Engineering, USC 27 September 2000.

[12]IBM Web Services Architecture Team, 'Web Services Architecture Overview', IBM, Septembre 2000.

[13]Leyman F., 'Web Services Flow Language', IBM Software Group specification, Mai 2001.

[14]Satish T., 'XLANG, Web Services for business process design', Microsoft, Mai 2001.

[15]Curbera F. and All, "Business Process Execution Language for Web Services", Ver1.0, IBM 31 July 2002.

[16]Ehnebuske, D. and All, Simple Object Access Protocol (SOAP), Ver1.1, 08/05/2000, http://www.w3.org/TR/2000/NOTE-SOAP-20000508.

[17]Bellwood, T. and all, UDDI, Ver 3.0, 19/07/2002, http://uddi.org/pubs/uddi-v3.00-published-20020719.htm.

[18]Luckman, C., Vera, J., and Meldal, S., « Three Concept of system Architecture », 07/1995.

[19]B.W. Boehm « A Spiral Model for Software Development and Enhancement ». ACM SIGSOFT Software Engineering Notes, vol. 11, No. 4/ 8/ 1986.

[20]Oquendo F., Alloui I., Cîmpan S., Verjus H., The ArchWare ADL: Definition of the Abstract Syntax and Formal Semantics, ArchWare European RTD Project IST-2001-32360, Deliverable D1.1b, 12/ 2002.

[21]Leymonerie F., Cîmpan S., Oquendo F., «ADL Foundation Style», 5/ 2003.

[22]Oquendo F., Alloui I., Mateescu R.; Architecture Analysis Language, Project Deliverable D3.1, Ver 1.0, 31/1/2003

[23]J. Estublier, H. Verjus, P.Y. Cunin, « Designing and Building Software Federations », *Proceedings of 1st Conference on Component Based Software Engineering, (CBSE)*, Warsaw, Poland, September 2001.

[24]Chaudet, C. and Oquendo, F., "π-SPACE: A Formal Architecture Description Language Based on Process Algebra For Evolving Software Systems", *Proceedings of the 15th IEEE International Conference on Automated Software Engineering (ASE'00)*, Grenoble, France, September 2000.

[25]Leymonerie, F., Cîmpan, S. and Oquendo, F., "Extension d'un langage de description architecturale pour la prise en compte des styles architecturaux : Application à J2EE", *14emes Journées Internationales Génie Logiciel & Ingénierie de Systèmes et leurs Applications*, Paris, December 2001.

[26]Binns, P., Engelhart, M., Jackson, M. and Vestal, S., "Domain Specific Software Architectures for Guidance, Navigation, and Control", International *Journal of Software Engineering and Knowledge Engineering*, 1996.

[27]Medvidovic, N., and Taylor, R.N., "A Classification and Comparison Framework for Software Architecture Description Languages", *Technical Report UCI-ICS-97-02*, Department of Information and Computer Science, University of California, Irvine, February 1997.