Composition semi-automatique de Services Web

Nerea Arenaza

SIN
Projet de Master

Février 2006

Responsable
Dr. Denis Gillet
EPFL / LA

Assistant
Karim Zeramdini
EPFL / LA

Table des matières

Table des figures

Introduction générale

eMersion [26] est un environnement d'apprentissage permettant aux étudiants de la faculté de *Sciences et techniques de l'ingénieur* d'effectuer différents travaux pratiques par l'intermédiaire du Web. Il a été conçu dans le Laboratoire d'Automatique (LA) de l'EPFL, en collaboration avec le Laboratoire d'Informatique Théorique (LITH).

eMersion est sensé être déployé et géré par des personnes non expertes en systèmes d'information. Par conséquent, un outil auteur et un outil de publication sont mis à leur disposition pour leur faciliter la publication de nouveaux environnements d'expérimentation.

L'outil auteur à comme objective de faciliter la conception des nouveaux environnements. Pour que cette conception soit faite d'une manière intuitive l'outil auteur offre un ensemble d'outils graphiques. Cet outil va permettre de concevoir l'environnement web que l'utilisateur souhaite créer, tout en spécifiant certain nombre de contraints et de paramètres requis. Tout ces spécifications seront stockes dans un document XML que plus tard l'outil de publication emploiera pour générer l'environnement Web.

De son coté, l'outil de publication doit prendre en entrée le document XML généré par l'outil auteur et, comme dit ci-dessus, générer l'environnement web conçu. La grammaire du fichier XML définit toute l'information nécessaire dès l'emplacement spatiale des composant et la communication jusqu'à la relation entre composants. L'outil de publication doit se trouver dans un serveur web afin qu'il puisse récupérer ce document XML et produire tous les composants définies par l'auteur.

Le besoin d'eMersion comme un environnement d'expérimentation à distance doit intégrer des services qui sont capables d'aider à collecter des paramètres, commander des équipements physiques à distance, etc. Avec l'utilisation des services web composites la combinaison des différents services et l'automatisation de certains processus seraient possibles. Par exemple, il serait possible de collecter des paramètres et les exploiter pour créer des statistiques ou prendre certaines données comme entrées et les traiter pour les utiliser dans le commandement d'équipements physiques à distance.

Ce travail de Master a pour objectif de contribuer à l'extension de cet outil auteur afin qu'il prendre en charge la composition semi-automatique des Services Web. Le premier chapitre, nous permettra de poser le cadre général du projet, son architecture, ainsi que les différents outils mis à disposition. Dans le deuxième chapitre nous introduirons les services Web composite et tous les concepts théoriques nécessaires pour la compréhension du projet. Dans le troisième et quatrième chapitre, nous reviendrons sur l'outil auteur et l'outil publication qui nous sont été fournis, en présentant les changements à faire pour que l'outil auteur et l'outil de publication prennent en considération les services Web composites.

1 Présentation Générale

1.1 Définition du sujet

Les Services Web sont des composants Web qui peuvent être accessible à travers Internet. D'autres systèmes peuvent interagir avec les Services Web en utilisant des messages SOAP (Simple Object Access Protocol). Dans un framework eLearning dans lequel des enseignants veulent intégrer facilement des services Web, il est essentiel de soutenir le processus d'édition pour les non experts en technologie de l'information.

Jusqu'à ici, le Sustainable Interaction Systems Group a développé un outil auteur permettant de composer des composants multimédia classique (image, texte, documents structurés, etc.) et un outil de publication permettant de déployer ces composants sous la forme d'environnements Web d'apprentissage.

Le but de ce projet est de contribuer à l'extension de ces deux outils (outil auteur et outil de publication) afin qu'ils prennent en charge la composition semi-automatique des Services Web. D'une part, l'outil auteur permettra, à travers des mécanismes graphiques, la composition des Services Web intuitivement et sans faire recours à la programmation. D'autre part, l'outil de publication permettra de publier ces Services Web sous la forme d'environnements Web d'apprentissage.

1.2 Architecture globale

Afin de mieux comprendre l'environnement dans lequel nous avons travaillé, il est convenant de bien connaître l'architecture et le fonctionnement de toutes les entités qui la composent.

Comme décrit plus haut, l'objectif de ce travail est d'étendre les fonctionnalités de l'outil auteur afin qu'il permette la composition des Services Web. L'outil auteur permet de modéliser un environnement d'apprentissage grâce à un ensemble d'outils graphiques tout en permettant à l'utilisateur (l'enseignant) de spécifier un certain nombre de contraints et de paramètres requis. Toutes ces spécifications seront stockées dans un document XML. Ce document servira comme entrée à l'outil de publication qu'il le transformera en un environnement Web d'apprentissage.

La grammaire du document XML définit toute l'information nécessaire tel que l'emplacement spatial des composants multimédia, la communication entre Services Web

(partage de données). Le document XML sert ainsi comme format pivot entre l'outil auteur et l'outil de publication.

La figure1 suivante [Gra05], illustre l'architecture décrite ci-dessus. D'un coté nous avons l'outil auteur qui assure la conception de l'environnement d'apprentissage et de l'autre coté l'outil de publication qui permet sa publication.

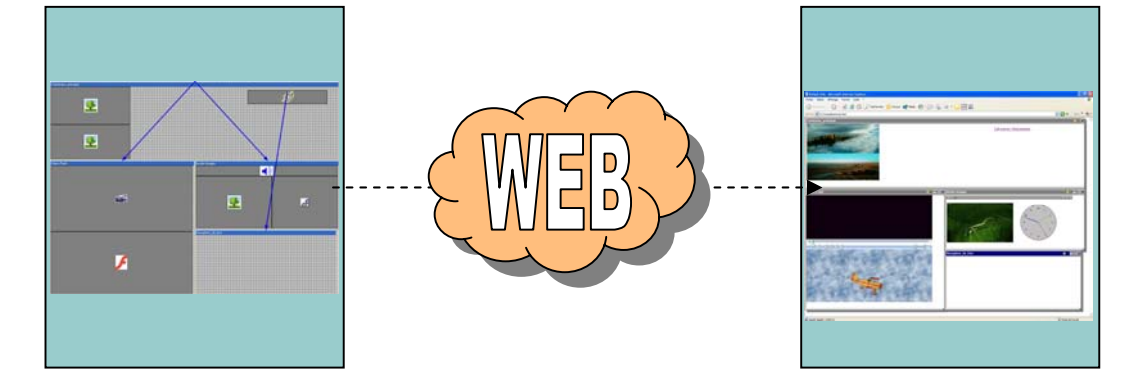


Figure 1: Architecture global

1.3 Description du fonctionnement

Dans ce qui suit, nous utiliserons le terme *cockpit* pour désigner un environnement d'expérimentation web déjà déployé. Un *cockpit* est un système de fenêtrage s'exécutant dans un navigateur d'Internet. Chaque fenêtre est indépendante des autres et contient une application différente.

Dans le cadre d'un cockpit, lorsque nous parlons de *conteneur*, nous entendons une fenêtre contenant des composants multimédia. Un conteneur peut comporter des applications différentes qui n'ont pas forcement des rapports directs entre eux. L'utilisateur peut modifier les conteneurs en les fermant, déplaçant, maximisant, etc. La communication entre différents conteneurs est aussi possible. Par exemple, depuis un conteneur on peut ouvrir un autre, ou si on le ferme, un autre conteneur se ferme aussi.

1.3.1 Architecture côté client

Pour modéliser un environnement d'expérimentation Web, un utilisateur peut utiliser l'outil auteur mis à sa disposition. Cet outil auteur lui permet de spécifier les caractéristiques souhaitées du nouvel environnement. Il peut insérer différents

composants multimédias, tel que des vidéos, des images, des applets, des Services Web, etc.

Il a aussi la possibilité d'indiquer des contraints comportementales. Par exemple, il pourrait aimer que lorsqu'un conteneur soit fermé quelque autres le soient aussi.

Une fois que toutes les caractéristiques désirées sont définies, un environnement d'apprentissage sera prêt à être publié. Pour cela, il faut envoyer le document XML qui contient toutes les caractéristiques du cockpit et tous les composants qu'il référence à l'outil de publication.

Un seul fichier sera envoyé à l'outil de publication, ce fichier est une archive ZIP contenant tous les documents concernant le cockpit (le document XML plus tous les composants référencés).

La figure 2 [Gra05], présente l'architecture décrite ci-dessous.

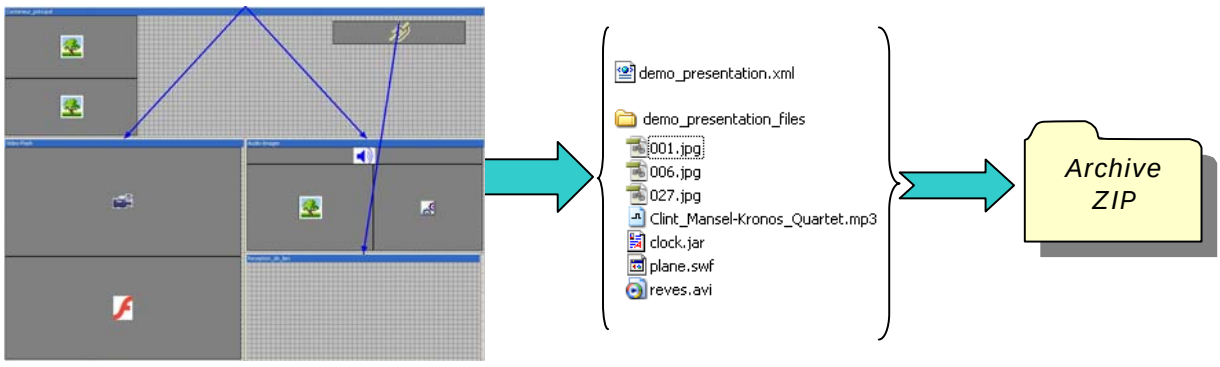


Figure 2: Architecture côté client

1.3.2 Architecture côté serveur

Une fois que l'outil auteur ait envoyé le fichier ZIP au serveur pour qu'il soit déployé, l'outil de publication va être en charge de la suite, étant sa tâche et de générer l'environnement d'expérimentation modéliser avec l'outil auteur.

Une fois le fichier ZIP reçu, l'outil de publication commence par le décompresser afin de pouvoir commencer l'analyse de son contenu. Une fois le fichier a été décompressé et la hiérarchie des documents récupéré tel quelle était dans l'outil auteur, l'analyse peut commencer. Comme nous avons spécifié auparavant, toute l'information pour récréer l'environnement se trouve dans le document XML. Il faut donc analyser

nœud à nœud tout le document afin de respecter tous les contraintes définies pendant la conception et les recréer adéquatement. Cette dernière partie se réalise par une transformation XSLT.

Finalement, une fois que tous les documents auront été générés correctement, il faudra publier l'ensemble afin que le cockpit soit accessible depuis n'importe quel navigateur.

La figure3 [Gra05], présente le processus de la publication d'un cockpit comme expliqué ci-dessus.

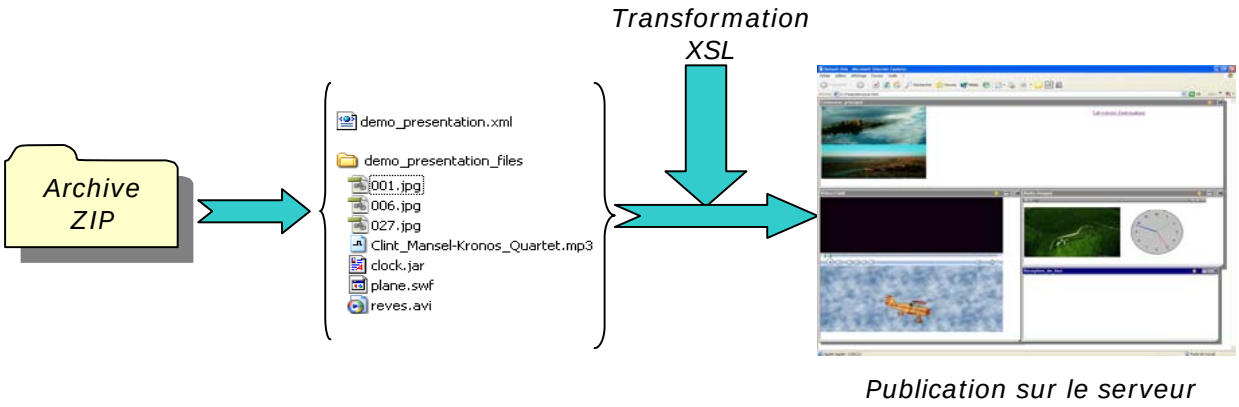


Figure 3: Architecture côté serveur

1.4 Outils mis à disposition

Comme ce projet de master vient étendre des outils qui existent déjà, nous décrivons brièvement dans ce qui suit les deux principaux outils qui ont été mis à notre disposition en présentant le cadre de leurs projets et leurs caractéristiques. Nous allons aussi présenter les modifications qu'il faudra introduire sur ces outils afin de pouvoir modéliser graphiquement des Services Web composites et les publier.

1.4.1 Outil auteur

L'outil auteur fourni a été réalisé en plusieurs étapes. L'outil de base était fait par S. Pasche dans un projet de diplôme intitulé « Outil auteur pour la production de documents actifs composites structurés » [Pas04] et plus tard modifié par O. Granges dans le projet de master « Outil de publication pour la production d'environnements d'expérimentation Web » [Gra05].

L'outil auteur est un outil graphique pour la conception d'un environnement d'expérimentation web. Il permet d'insérer différents types de composants multimédias (applets, vidéo, texte...) et il permet de préciser des règles de comportement entre ces composants et les conteneurs qu'ils contiennent. A la fin de la phase de modélisation, l'outil auteur génère un document XML comportant toutes les spécifications définies durant cette phase.

Afin que l'outil auteur puisse prendre en charge la composition semi-automatique des services web, il a fallu lui apporter quelques modifications. Une nouvelle icône a été définie dans la palette graphique de l'outil auteur pour les services web composite. Les documents décrivant le comportement du service web composite ont été liées à ses propriétés et le document XML généré a dû être mis à jour. De même, tous ces documents ont dû être introduits dans le fichier ZIP avant de l'envoyer au serveur pour la publication du cockpit.

1.4.2 Outil de publication

La toute première tentative de gestion du document XML généré lors la conception avait été faite par D. Berkovits dans le projet de semestre « Outil de publication pour la gestion de l'emplacement spatial au sein d'environnements d'expérimentation Web » [Ber04]. L'objective de cette projet était de tester la faisabilité de la génération d'un cockpit à partir d'un document XML.

Une nouvelle approche a été effectuée pendant le projet master intitulé « Outil de publication pour la production d'environnements d'expérimentation Web » [Gra05] réalisé par O. Granges.

L'outil de publication résultant se charge de la gestion et la communication entre des composants multimédia (applets, vidéos, audio ...). Il comprenait aussi la génération automatique d'une interface de communication avec des services web.

Comme pour l'outil auteur, l'outil de publication a dû être adapté afin de permettre la publication et l'invocation des Services Web composites. La publication et l'invocation des Services Web composites consistent à rendre des Services web accessibles sous la forme d'un processus métier invocable via le Web.

2 Composition de Services Web

2.1 Introduction

Dans cette section nous allons introduire tous les concepts théoriques concernant le projet. Nous allons commencer par définir brièvement les services web. Nous présentons plus en détail, par la suite, les services web composite en fournissant la description, le fonctionnement, les différents types, les langages de définition et les serveurs existants.

2.2 Définition des Services Web

Si l'on reprend la définition de Mark Colan, les Services Web sont *« des applications modulaires basées sur l'Internet qui exécutent des tâches précises et qui respectent un format spécifique »* [1]. Plus clairement, cela consiste à permettre l'utilisation d'une application à distance. En fait, les services web facilitent l'invocation de certains traitements depuis Internet.

L'avantage de ce modèle tient à présenter ces services comme des *boîtes noires* [1]. En fait, les entrées-sorties d'un Service Web sont gérées au sein de messages dont on connaît le format grâce à des interfaces clairement exposées mais sur lesquels l'implémentation interne du traitement n'influe pas au niveau de structure. Ceci permet un haut niveau de modularité et d'interopérabilité.

L'avantage du modèle de message est qu'il permet de s'abstraire de l'architecture, du langage ou encore de la plate-forme qui va supporter le service. En fait, la principale caractéristique des services web est d'être accessible depuis n'importe quelle machine par un programme écrit dans n'importe quel langage. Il suffit juste que le message respecte une structure donnée pour qu'il puisse être utilisé.

La définition des services web ne serait pas complète si l'on n'évoquait pas ses principaux standards : SOAP [2], WSDL [3] et UDDI [4].

- SOAP (Simple Object Acces Protocol) est un protocole permettant l'invocation de méthodes à distance par l'échange de messages XML [1].
- WSDL (Web Service Description Language) est une norme dérivée de XML qui permet la description de l'interface d'utilisation d'un Web Service. Il fournit les spécifications nécessaires à l'utilisation d'un Service Web en décrivant les méthodes, les paramètres et se qu'il retourne. WSDL un langage de description des services, il présente l'interface publique du service et permet la génération des stubs et skeletons des proxys [1] [Sri03].

- UDDI (Universal Description, Discovery and Integration) est une spécification définissant la manière de publier et de découvrir les services web sur un réseau [1].

2.3 Définition des Services Web Composites

2.3.1 Description et fonctionnement

Un service web est dit *composé* ou *composite* lorsque son exécution implique des interactions avec d'autres services web afin de faire appel à leurs fonctionnalités. La composition de services web spécifie quels services ont besoin d'être invoqués, dans quel ordre et comment gérer les conditions d'exception.

La composition des Services Web peut se faire de deux manières: orchestration et chorégraphie [Juric05] [Peltz03].

I) Orchestration

L'orchestration décrit l'interaction des services au niveau de messages, incluant le logique métier et l'ordre d'exécution des interactions. Les services web n'ont pas de connaissance (et n'ont pas besoin de l'avoir) d´être mêlées dans une composition et d'être partie d'un processus métier. Seulement le coordinateur de l'orchestration a besoin de cette connaissance.

La figure suivante montre le workflow dans l'orchestration des services web. Un coordinateur prend le control de tous les services web impliqués et coordonne l'exécution des différentes opérations des services web qui participent dans le processus.

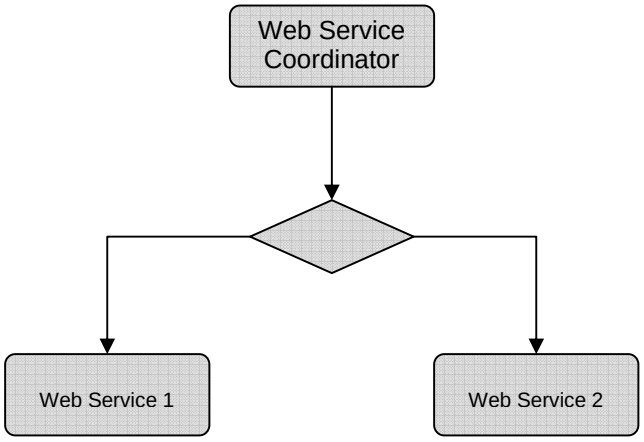


Figure 4: Web Service Orchestation

II) Chorégraphie

Contrairement à l'orchestration, la chorégraphie n'a pas un coordinateur central. Chaque service web mêlée dans la chorégraphie connaît exactement quand ses opérations doivent être exécutées et avec qui l'interaction doit avoir lieu.

La chorégraphie est un effort de collaboration dans lequel chaque participant du processus décrit l'itération qui l'appartient. Elle trace la séquence des messages qui peut impliquer plusieurs Services Web.

La collaboration dans la chorégraphie des services web peut être représenté de la manière suivant:

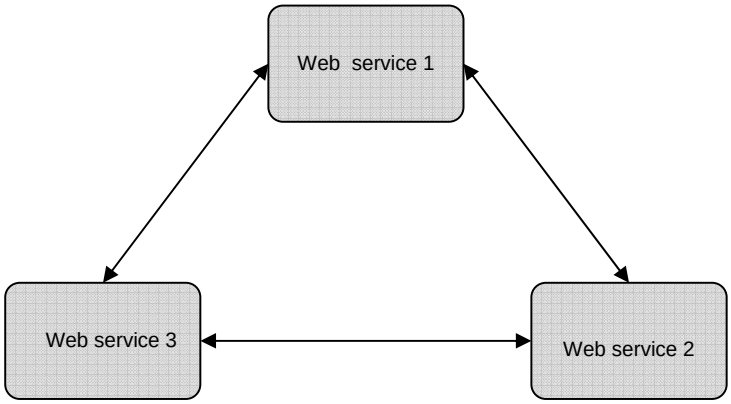


Figure 5: Web Services chorégraphie

Depuis la perspective de la composition des services web, l'orchestration est un rapprochement plus flexible que la chorégraphie:

- Le responsable ou coordinateur de tout le processus métier est connu.
- Les services web peuvent être incorporés sans soucis, parce que ils n'ont pas conscience d'appartenir à un processus métier.

2.3.2 Types de composition

Nous avons déjà expliqué qu'est ce qu'un service Web et comment fonctionnent les services web composites. Mais, il nous faudra encore comprendre comment cette composition peut s'accomplir depuis la perspective du dégrée d'automatisation. La composition peut être classifiée dans trois catégories [Fost03]:

I) Composition manuelle

La composition manuelle des services Web suppose que l'utilisateur génère la composition à la main via un éditeur de texte et sans l'aide d'outils dédiés.

II) Composition semi-automatique

Les techniques de composition semi-automatiques sont une pas en avant en comparaison avec la composition manuelle, dans le sens qu'ils font des suggestions sémantiques pour aider à la sélection des services Web dans le processus de composition.

III) Composition automatique

La composition totalement automatisée prend en charge tout le processus de composition et le réalise automatiquement, sans qu'aucune intervention de l'utilisateur ne soit requise.

La composition manuelle est, parmi les trois, la plus adaptable aux besoins de l'utilisateur, car il va tout définir à son goût depuis le début. Mais, malheureusement, elle nécessite des connaissances de bas niveau de programmation. Étant donné que l'environnement dans lequel nous allons travailler est destiné aux utilisateurs non informaticiens cette catégorie de composition doit être rejeté.

Cependant, il faut maintenir le plus grand niveau de flexibilité possible afin que l'utilisateur puisse définir son propre processus. C'est pour cette raison que la composition automatique a aussi été exclue car l'utilisateur n'intervienne pas dans la définition du processus de composition.

Il nous reste la composition semi-automatique des services web. Cette composition étant à mi-chemin entre les deux antérieures est la plus adéquate dans notre cas. L'utilisateur maintiendra certain contrôle sur le processus envisagé mais il n'aura pas besoin de connaissances de programmation et il pourra définir son propre processus en utilisant des outils graphiques pour modéliser et concevoir des services Web composites.

2.3.3 Langages de définition

Ils existent plusieurs langages de définition pour la composition de services web. Dans cette section nous allons présenter brièvement quelques uns d'entre eux [5] [6], notamment les plus connus et acceptés afin de faire le choix le plus convenant pour notre projet.

I) Business Process Modeling Language (BPML)

Le BPEL [6] est un langage de modélisation des processus métiers. Il permet de définir un modèle abstrait d'interaction entre collaborateurs participant à une activité de l'entreprise, voire entre une organisation et ses partenaires.

Les processus métiers sont représentés par un flux de données, un flux d'événements sur lesquels on peut influer en définissant des règles métier, des règles de sécurité, des règles de transactions. On peut ensuite lancer l'exécution du modèle et vérifier le fonctionnement théorique des différents processus.

II) Web Service Choreography Interface (WSCI)

WSCI [7] est un langage reposant sur XML. Il propose de se focaliser sur la représentation des services Web en tant qu'interfaces décrivant le flux de messages échangés (la chorégraphie de messages).

Il propose ainsi de décrire le comportement externe observable du service. Pour cela, WSCI propose d'exprimer les dépendances logiques et temporelles entre les messages échangés à l'aide de contrôles de séquences, corrélation, gestion de fautes et transactions. On remarque que WSDL et ses définitions abstraites sont réutilisées afin de pouvoir également décrire par la suite les modalités de concrétisation des éléments manipules pour modéliser un service.

III) Web Service Conversation Language (WSCL)

WSCL [8] propose de décrire à l'aide de documents XML les services Web en mettant l'accent sur les *conversations* de ceux-ci. En outre, les messages à échanger sont pris en compte. WSCL a été pensé pour s'employer conjointement avec WSDL.

Les définitions WSDL peuvent être manipulées par WSCL pour décrire les opérations possibles ainsi que leur chorégraphie. En retour, WSDL fournit les concrétisations vers des définitions de messages et des détails techniques pour les éléments manipulés par WSDL.

IV) XLANG

Créé par Microsoft, le XLANG [31] est une extension de WSDL. Elle fournit en même temps un modèle pour une orchestration des services et des contrats de collaboration entre celles-ci. XLANG a été conçus avec une base explicite de théorie de calcul.

Les actions sont les constituants de base d'une définition de processus de XLANG. Les quatre types d'opérations de WSDL (requête/réponse, sollicitation de la réponse, le sens unique, et la notification) peuvent être employés comme actions de XLANG. XLANG ajoute deux autres genres d'action: arrêts (date-limite et durée) et exceptions.

V) Web Service Flow Language (WSFL)

WSFL [Ley01] est un langage basé sur XML pour la description des services web composite. WSFL considère deux types de compositions de services web:

- Le premier type indique le modèle approprié d'utilisation d'une collection de services web, de telle manière que la composition résultante décrive comment réaliser le but particulier s'un business; typiquement, le résultat est une description d'un processus métier.
- Le deuxième type indique le modèle d'interaction d'une collection de services web; dans ce cas, le résultat est une description des interactions globales associés.

VI) Business Process Execution Language for Web Services (BPEL4WS)

BPEL [10] est un langage pour les processus métier basé sur XML conçu pour permettre charger/partager les données distribuées, même à travers des organismes multiples, en employant une combinaison de services web.

BPEL décrit l'interaction des processus métiers basés sur les services Web, à la fois au sein des entreprises et entre elles. Les entreprises utilisatrices du langage BPEL pourront ainsi définir leurs processus métiers et en garantir l'interopérabilité non seulement à l'échelle de l'entreprise, mais également avec leurs partenaires commerciaux, au sein d'un environnement de services Web. BPEL rend possible l'interopérabilité entre des activités commerciales basées sur des technologies différentes.

Écrit par des créateurs des systèmes de BEA, IBM, et le Microsoft, la spécification BPEL combine et remplace le WSFL d'IBM et le XLANG du Microsoft. Parfois, BPEL est appelé comme BPELWS ou BPEL4WS.

Il a fallu choisir un de ces langages de définition pour le développement de notre projet. Nous avons fait une petite analyse des différentes possibilités pour choisir celle qui nous convenait le plus.

Nous avons rejetée dès le début la possibilité d'utiliser XLANG, WSFL ou WSCL parce que l'union et l'extension de ces deux langages ont fait dériver le langage BPEL. par conséquent il n'a pas sens de les considérer comme alternatifs.

Dans le cas de WSCI, il permet d'organiser la chorégraphie des messages de services Web. Il doit travailler avec un autre langage complémentaire (tel que BPML) que permettra de décrire les processus métier. Outil complémentaire, BPML permet de décrire les processus métiers génériques en amont de l'enchaînement de messages WSCI. Dans ce cas il faudrait utiliser deux langages pour la conception d'un processus métier. Il est beaucoup plus simple d'utiliser un seul langage qui permet la conception de tout le processus, par exemple BPEL. Pour cela, WSCI et BPML ne nous conviennent pas dans notre cas.

Nous avons donc opté pour le langage BPEL. Il est le résultat de l'unification et l'évolution de trois différentes tentatives de standardisation des définitions des processus métier. Il est le standard pour la description des processus métier plus complet existant. BPEL est le plus soutenu industriellement et le mieux accepté par les développeurs. En plus, il existe plusieurs outils de développement qui peuvent nous aider dans le cadre de notre travail. Pour toutes ces raisons, BPEL est le langage que nous avons choisi pour le déroulement de ce projet.

2.3.4 BPEL

Étant donné que nous avons choisi le langage BPEL pour la définition des services Web composites, il conviendrait de le présenter avec plus de détail. BPEL a été conçu spécifiquement comme un langage pour la définition des processus métier. Il support deux types différents de processus [Juric05]:

- Les processus exécutables nous permettent de spécifier les détails du processus métier. Ils peuvent être exécutés au moyen d'un engin d'orchestration. Dans la majorité des cas, BPEL est utilisé dans ce type de processus.
- Les abstracts business protocoles nous permettent de spécifier l'échange de messages entre partenaires du processus. Ils ne sont pas exécutables.

BPEL est un langage qui se base sur XML et qui soutient la technologie des services web, tel que, SOAP [2], WSDL [3], UDDI [4], WS-Reliable Messaging [11], WS-Addressing [12], WS-Coordination [13] et WS-Transaction [13].

Le processus BPEL spécifie l'ordre exact de l'invocation des services web participants dans la composition. L'invocation peut se faire soit en parallèle soit séquentiellement. Nous pouvons conditionner le comportement, par exemple, lorsque l'invocation d'un service web peut être dépendante du résultat d'une invocation antérieur. La construction des boucles, la déclaration de variables, la copie et l'assignement des valeurs, etc. est aussi possible. De plus, il est possible combiner tous ces constructions et de définir des processus métier complexes d'une manière algorithmique [Juric05].

Comme nous venons d'indiquer, un processus métier correspond à une séquence d'opérations ou plus exactement à un flux d'activités. Ces activités peuvent faire intervenir un à plusieurs services Web. Le langage BPEL permet deux types d'activités : les activités de base et les activités structurées.

Les activités de base de ce langage permettent:

- d'invoquer une opération d'un service Web tiers (activité *invoke*),
- de présenter la composition comme un nouveau service Web avec l'activité *receive* pour d´écrire la réception d'une requête et l'activité *reply* pour générer une réponse.

Les activités structurées utilisent les activités de base pour décrire :

- des séquences ordonnées (*sequence*) et des exécutions en parallèle (*flow*),
- des branchements (*switch, if)* et des boucles (*while*),
- des chemins alternatifs (*pick*).

Il permet aussi de déclarer des variables, avec *variable,* et de définir des partner links, avec *partnerLink*. Nous pouvons définir une partner link comme un lien généré entre un service web et le processus pendant l'invocation d'un service Web ou, aussi, comme un lien créé entre le client qui invoque un processus BPEL et le processus lui même. Il faut, obligatoirement, avoir au moins un client partner link.

Ce langage intègre également un mécanisme de gestion des exceptions (*throw*, *catch*), ainsi qu'un mécanisme de compensation (*scope*) qui permet d'annuler une transaction dans son intégralité lorsque celle-ci échoue. Il constitue une couche supérieure au langage de description WSDL. Il utilise, en effet, WSDL pour définir les opérations de services Web élémentaires à appeler et pour présenter le processus métier comme un nouveau service Web.

Afin de mieux comprendre la tâche de génération d'un processus BPEL dans un environnement eLearning nous allons prendre l'exemple d'un service Web composite d'administration des permis pour accéder aux expériences de laboratoire des étudiants pendant un cours. Dans cet exemple, l'étudiant invoque le service Web d'accès aux expériences en fournissant un message de submit qui contient un ou plusieurs expériences pre-laboratoire à faire pendant le cours et le group de travail auquel il appartient. En

retour, il obtient un identificateur et un mot de passe pour qu'il puisse accéder aux expériences de laboratoire. La figure 6 présente cet exemple illustratif.

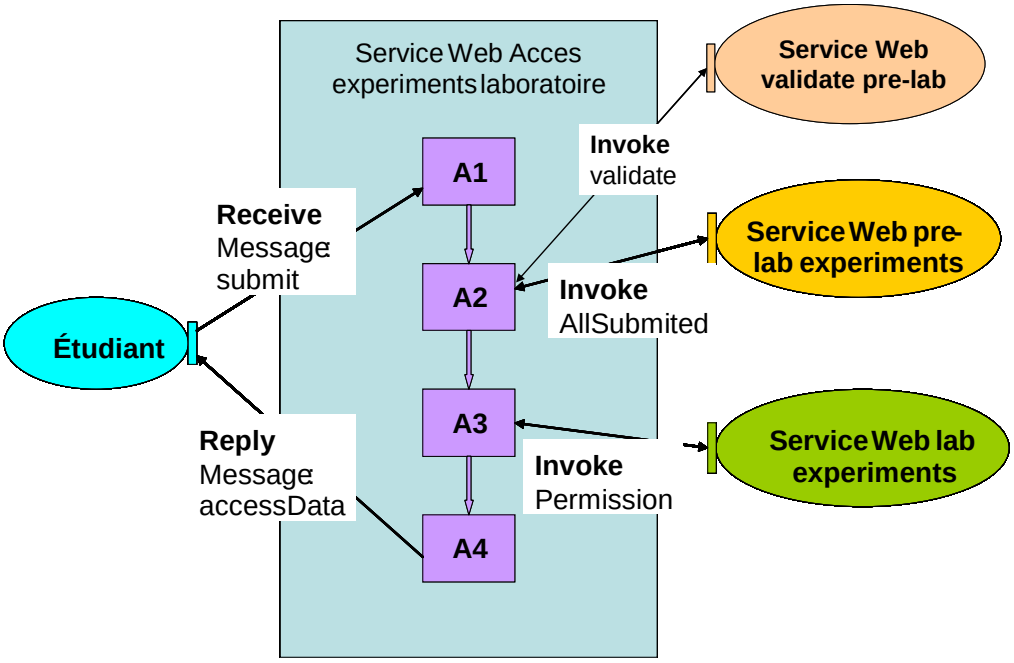


Figure 6: Service Web Lab Experiment

Ce service Web composite d'accès aux expériences du laboratoire peut être modélisé sous la forme d'un processus métier en combinant différents services Web élémentaires. Ainsi, à la réception de la demande de accès, le processus métier invoque en parallèle le service web qui permet de stocker tous les expériences pre-laboratoire dans une base de donnés et le service web qui permet à un assistant valider cette expérience. Si tous les expériences pre-laboratoire ont été stockés dans la base de donnés le premier service web retourne *true* et si l'expérience a été validé pour l'assistant le deuxième service web retourne *true*. Ensuite, si la réponse donnée pour l'ensemble de services web appelé est *true* le processus obtient l'identificateur et le mot de passe uniques auprès du service Web d'expériences de laboratoire et donne au client l'identification et le mot de passes qui vont le permettre l'accès aux expériences du laboratoire.

La description BPEL4WS correspondante se décompose en trois parties :

- définition des participants ou partner links du processus métier
- définition des types de données et des messages échangés
- définition du processus métier

Nous allons reprendre l'exemple de l'accès aux expériences du laboratoire et générer la description BPEL correspondante. La première étape sera définir des participants ou partner links de notre processus. Dans notre processus nous trouvons un client et trois services Web, ce faisant les partner links suivants:

```
<partnerlinks>
      <partnerlink name=”client”
      partnerRole=”clientAccesExperimentsLaboratoire”>
      <partnerlink name=”serviceWebPreLabExperiments”
      partnerRole=”submitPreLabExperiments”>
      <partnerlink name=”serviceWebValidatePreLab”
      partnerRole=”submitPreValidatePreLab”>
      <partnerlink name=”serviceWebLabExperiments”
      partnerRole=”fournisseurPermission”>
</partnerlinks >
```

La deuxième étape consistera à spécifier des types de données et des messages échangés. Il faudra définir tous les variables nécessaires et spécifier le type de message auquel ils sont liés. Nous le feront de la manière suivant :

```
<variables>
      <variable name=”submit” messageType=”typeSubmit”>
      <variable name=”accessData” messageType=”typeAccessData”>
      <variable name=”AllSubmited” messageType=”typeAllSubmited”>
      <variable name=”Validate” messageType=”typeValidate”>
      <variable name=”Permission” messageType=”typePermission”>
</variables>
```

La troisième et dernière étape consiste dans la définition du processus métier. Il est nécessaire de spécifier l’ordre exact d’exécution des opérations et invocations, comme montrée dans la suite:

```
<sequence name=”seqAccesExperimentsLaboratoire”>
      <receive partnerlink=”client”
      operation=”accesExperimentsLaboratoire” variable=”submit”>
      <flow>
            <invoke partnerlink=”serviceWebPreLabExperiments”
            operation=”AllSubmited”>
            <invoke partnerlink=”serviceWebValidatePreLab”
            operation=”Validate”>
      </flow>
      <switch>
            <case condition="bpws:getVariableData('AllSubmited',
            'accesExperimentsLaboratoire') == boolean(true) and
            bpws:getVariableData('Validate',
            'serviceWebValidatePreLab') == boolean(true)>
                  <invoke partnerlink=”serviceWebLabExperiments”
                  operation=”permission”>
                  <reply partnerLink="client"
                  operation="accesData"/>
            </case>
      </switch>
</sequence >
```

2.3.5 Outils de modélisation et serveurs BPEL

Comme BPEL est le langage le plus accepté pour les développeurs et les fabricants, il existe plusieurs outils de modélisation et des serveurs pour le développement des processus BPEL et leurs déploiements.

Nous allons dans ce qui suit mentionner quelques-uns parmi les outils existants pour générer et déployer les processus BPEL et choisir le plus convenant pour nos besoins.

L'outil de modélisation BPEL que nous devons choisir doit accomplir quelques caractéristiques pour nous être utile. Parmi les conditions à remplir, il faut qu'il offre un environnement graphique pour la conception du processus métier car l'outil auteur est destiné aux personnes non expertes en systèmes d'information. Il faut aussi que l'apprentissage et l'utilisation soient simples. Finalement, l'intégration dans l'outil auteur doive s'effectuer sans soucis. Étant donné que l'outil auteur est intégré dans la plate-forme d'Eclipse [14] l'outil choisi devait pouvoir s'y intégrer aussi.

D'autre coté, pour la sélection du serveur BPEL, il n'y a pas autant de conditions à remplir. Il faut tout simplement être accessible depuis Java, afin de pouvoir le déployer automatiquement sans besoin d'une plate-forme supplémentaire.

Parmi les outils trouvés nous citons les suivants :

- BPWS4J [15]
- Oracle BPEL designer [16]
- Oracle BPEL Process Manager [16]
- Twister [17]

Les deux outils de modélisation de processus BPEL les plus connues, acceptés et utilisés pour la modéliser visuellement des processus métier sont *BPWS4J* d'IBM et *Oracle BPEL Designer*. Tous les deux offrent des plug-ins des plug-ins pour Eclipse dont l'intégration dans notre outil auteur sera possible. Il a bien fallu faire le choix entre le deux. Pour cella, nous avons regardé d'un coté les fonctionnalités et par l'autre coté l'environnement graphique qu'ils offrent. Si nous observons les fonctionnalités des deux outils, nous nous rendons compte qu'il n'existe pas de différences substantielles entre eux. Nous avons donc fait le choix en dépendant de l'environnement graphique. Comme l'environnement graphique qu'offrait l'outil *Oracle BPEL designer* était plus riche et plus intuitive que celle du *BPWS4J*, nous avons choisi le premier.

En ce qui concerne les serveurs BPEL, nous avons envisagé deux possibilités: *Oracle BPEL Process Manager* et *Twister*. L'outil *Oracle BPEL Designer*, que nous avons choisi pour la modélisation, est lié au serveur *Oracle BPEL Process Manager*. Ce serveur offre une infrastructure compréhensible et facile à utiliser pour créer, déployer et gérer les processus BPEL. Malheureusement, le serveur est très lourd et prend un temps considérable dans la réalisation du processus. En plus, il est un logiciel propriétaire. Due

à ces difficultés, nous avons été contraints à rechercher un autre serveur que pourrait se charger du déploiement. *Twister* étant un choix logique.

Twister est une solution complète et totalement open source permettant la gestion des processus métiers. Il est écrit en Java en suivant le standard WS-BPEL. Il offre un outil qu'implémente les spécifications de BPEL et les principes des services web. Si le processus BPEL a été correctement spécifié et les échanges de messages ont été bien définis l'intégration de *Twister* devrait se dérouler assez facilement.

3 Adaptation de l'outil auteur

3.1 Introduction

Comme le but de ce projet est l'extension de l'outil auteur afin qu'il prenne en charge la composition semi-automatique des services web, il a fallu trouver un moyen intuitive et simple qui permet à travers une approche visuelle adéquate de le faire. Nous avons déjà analysé les possibilités existantes et maintenant il faut faire le choix.

Il faut aussi adapter l'outil auteur. La création d'une nouvelle fonctionnalité implique plusieurs changements que nous allons présenter dans cette section.

3.2 Composition des services web

3.2.1 Introduction

Nous avons présentés dans ce qui précède l'intérêt de la composition des services web. Nous avons vu aussi les différents langages qui existent pour le définir.

Nous avons aussi parlé des processus BPEL et des possibilités qu'il offre. Et la décision de composer les services web en utilisant ce standard a été prise. Nous avons aussi parlé des différents outils de modélisation que sont offert. Maintenant il faut faire le choix entre eux.

L'outil sélectionné doit remplir quelques conditions requises. Il faut qu'il offre un environnement graphique pour la conception du processus car l'outil auteur est destiné aux personnes non expertes en systèmes d'information. Il faut aussi que l'apprentissage et l'utilisation soient simples. Finalement, l'intégration dans l'outil auteur doit s'effectuer sans soucis. Étant donné que l'outil auteur est intégré dans la plate-forme d'Eclipse l'outil choisi devait pouvoir s'y intégrer aussi.

Après avoir observé les caractéristiques et les possibles des outils de modélisation existant, l'Oracle *BPEL designer* a été l'outil qui s'adapte le mieux à nos besoins.

3.2.2 Oracle BPEL designer

Le l'outil de modélisation Oracle BPEL Designer est un plug-in Eclipse que permet aux développeurs d'intégrer visuellement des services Web multiples dans un processus BPEL.

Oracle BPEL designer [16] offre un environnement graphique intuitif, simple et complet, tout en étant facile à utiliser. Il offre aussi un plug-in pour Eclipse [14], ainsi l'intégration dans la plate-forme était aussi simple que d'insérer ce plug-in dans le directoire correspondant.

Afin de mieux connaître cet outil que nous avons utilisé pour la génération des processus BPEL, nous introduisons dans ce qui suit les possibilités qu'il offre. Il offre une palette avec toutes les opérations possibles (voir section 2.3.4.). Il faut sélectionner l'opération désirée, la glisser-déposer (drag-drop) jusqu'à la représentation graphique du processus et la mettre dans l'emplacement désiré.

Figure 7: Palette BPEL

Nous allons montrer dans ce qui suit comment il est possible de modéliser graphiquement l'exemple de la figure 6 en suivant les étapes qui y sont indiquées:

- définition des participants ou partner links du processus métier
- définition des types de données et des messages échangés
- définition du processus métier

La première étape sera définir des participants ou partner links de notre processus. Dans notre processus nous trouvons un client et quatre services Web, ce faisant les partner links suivants: *client*, *serviceWebPreLabExperiments*, *serviceWebValidatePreLab* et *serviceWebLabExperiments*. Le partner link *client* et un processus de base sont construits par l'outil Oracle BPEL Designer dans l'initialisation. Il faudra, alors, définir seulement les partner links pour les services Web invoqués.

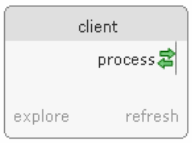
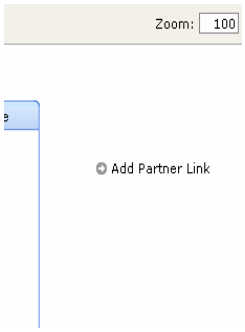


Figure 8: Partner Link client

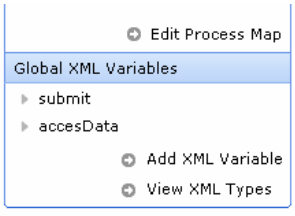
Il ne faut pas générer le parnet link pour le client, puisque il est déjà fourni.



Il faut cliquer sur « Add Partner Link » et choisir le service web désiré. Dans notre exemple il faudra répéter cette opération pour les services Web *preLabExperiments* , *validatePreLab* et *labExperiments*.

Figure 9: Ajouter des partner Links

La deuxième étape consistera à spécifier des types de données et des messages échangés. Il faudra définir tous les variables nécessaires et spécifier le type de message auquel ils sont liés. Nous le feront de la manière suivant :



Il faut ajouter tous les variables nécessaires en cliquant sur « add XML Variable ». Pour que tous les variables soient définis il faudra encore définir les variables *allSubmited*, *validate* et *permission*.

Figure 10: définir variables

La troisième et dernière étape consiste dans la définition du processus métier. Il est nécessaire de spécifier l'ordre exact d'exécution des opérations et invocations, comme montrée dans la suite:

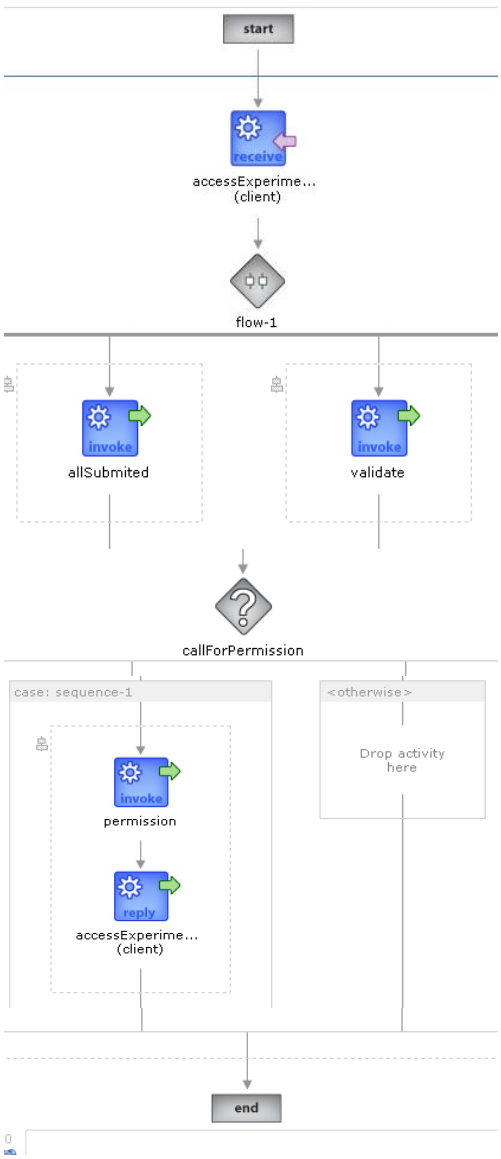


Figure 11: Description du processus BPEL

Il faut attendre que le client envoie une commande une expérience pre-laboratoire (<*receive*>). Après, le processus invoquera (<*invoke*>) en parallèle (<*flow*>) les services Web *preLabExperiments* et *validatePreLab*.

Un fois les deux services Webs appelés, il faut voir si le résultat obtenue est le désiré, cet-à-dire, si les deux services Web ont renvoyé comme résultat *true*. Pour ça, nous utiliserons une condition (<*switch*>). Si la condition est le requis, dans notre cas les deux résultats doivent être *true*, il faudra appeler (<*invoke*>) le service web *LabExperiments* et les données d'accès aux expériences du laboratoire sera renvoyé (<*reply*>) au client.

Avec cette dernière opération la description de notre processus BPEL est complète. Il faut bien sûr faire attention et remplir correctement tous les spécifications demandées dans chaque opération. Si nous avons suivi tous les instructions nous pouvons valider le processus et il serait prêt pour être déployé.

3.3 Ajustement du document XML généré

Il faut, tout d'abord, définir un nouvel élément appelé *WebServiceComposite*. Ce nouveau nœud sert à définir un service web composite, en spécifiant le nom du fichier BPEL qui lui correspond. On rappelle que ce fichier BPEL est produit par l'outil Oracle BPEL designer tel que nous l'avons déjà expliqué ci-dessus.

Pour que ce nœud puisse être généré de manière graphique nous avons ajoutée une nouvelle entrée à la palette de components de l'outil auteur. Il a fallu ajuster le fichier *CockpitPaletteFactory.java* qui se trouve dans *org\cockpitcomposer* de la manière suivant:

```
static private PaletteContainer createComponentsDrawer() {
…
…
combined = new CombinedTemplateCreationEntry(
                "WebServiceComposite",
                "Create a Web Service Composite Component", null,
                new SimpleFactory(WebServiceComposite.class),
                ImageDescriptor.createFromFile(
                        CockpitContainer.class,
                        "icons/webservicecomposite.gif"),
                ImageDescriptor.createFromFile(CockpitContainer.class,
                        "icons/webservicecomposite.gif")
        );
        entries.add(combined);
…
…
}
```

La palette résultant est:

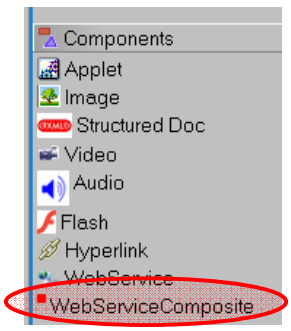


Figure 12: Nouvelle palette

Pour le traitement de génération de cette nouvelle component un nouveau class Java (*WebServiceComposite.class*) a été créé dans *org\cockpitcomposer\model*. Ce fichier java va éditer une boîte de dialogue que nous permettra d'insérer les propriétés de notre service web composite et de définir quel est le processus bpel qui le correspond.

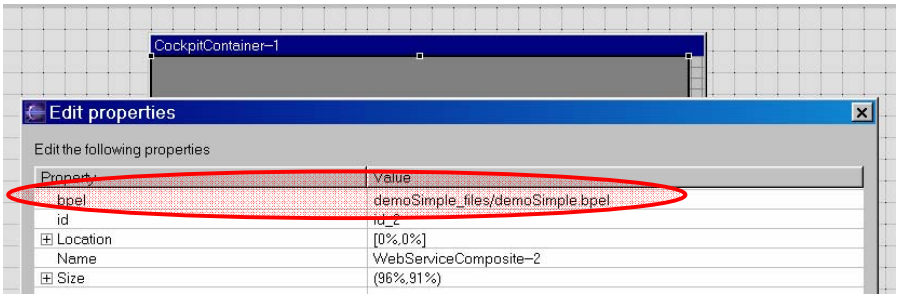


Figure 13: Propriétés du service web composite

Pour en arriver à l'édition de cette boîte de dialogue et la création d'une nouvelle entrée dans le document XML qui décrit l'environnement la class java *WebServiceComposite* doit appeler différents fichiers java. Cette communication est effectuée de la manière suivante:

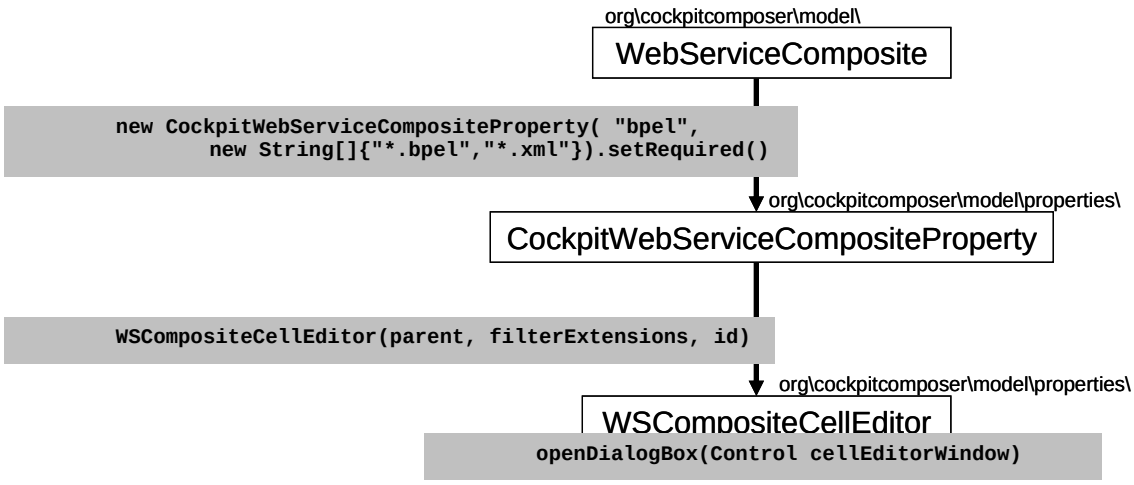


Figure 14: création du nouveau nœud

3.4 Ajustement de l'archive ZIP généré

Tous les fichiers lies à un cockpit particulier sont envoyés au serveur chargé de les déployer dans un archive ZIP. Due aux changements faites dans l'outil auteur il faut aussi vérifier si l'archive ZIP qu'on génère à besoin de quelque modifications.

Étant donne que nous avons introduit une nouvelle fonctionnalité et que celle-ci est lié au processus créé avec le plug-in BPEL d'Oracle, il est nécessaire de faire quelque changements dans la génération de l'archive zip.

Il faut introduire les fichiers crées depuis l'outil BPEL dans l'archive zip. Ces fichiers sont nécessaires pour pouvoir déployer le processus BPEL quand le cockpit est en train de se charger.

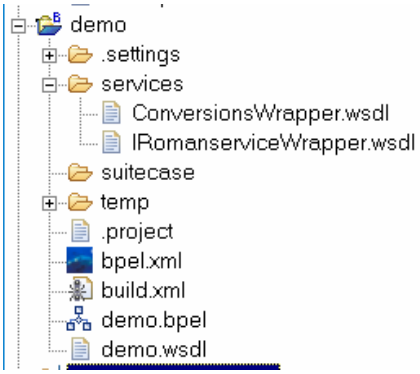


Figure 15: Fichiers à introduire dans le ZIP

Pour cela, il faut faire quelques changements dans le fichier java *zip.java* qui se trouve dans *org\cockpitcomposer\rendering*. Il faut voir si dans le cockpit_files existe un fichier bpel et, si tel est le cas, il faut prendre tous les fichiers liées au cette processus bpel comme présenté ci-dessous:

```
if(zipAddName.substring(zipAddName.indexOf(".")).compareTo(".bpel")==0){

      //Il faut monter jusqu'à le workspace d'Eclipse
      File aux =
      fileToAdd.getParentFile().getParentFile().getParentFile();

      //Un fois positionnes dans le workspace, il faut trouver le
      répertoire qui a le même nom que le fichier bpel.
      //Quand il est trouvé nous allons ajouter tout son contenue au ZIP
      if(aux.isDirectory()){
            File[] auxFiles = aux.listFiles();
            for (int i=0;i<auxFiles.length;i++){
                  if (auxFiles[i].getName().compareTo
                  (zipAddName.substring(zipAddName.indexOf("\\")+1,
                  zipAddName.indexOf(".")))==0) {
```

```
                                    if(auxFiles[i].isDirectory()){
                                          File[] aux2=auxFiles[i].listFiles();
                                          for(int j=0;j<aux2.length;j++){
                                                      zipAddFile(aux2[j]);
                                          }
                                    }
                              }
                        }
                  }
            }
}
```

Si toute l'action est développé comme étant indiqué, le déploiement se mènera correctement à sa fin et le archive ZIP aura tous les fichiers nécessaires pour que les fichiers BPEL liées au web services composite soient aussi correctement déployés.

3.5 Conclusion

Nous avons présenté dans ce chapitre les retouches effectuées à l'outil auteur mis à notre disposition. Celles-ci se sont avérées minimes mais essentielles en vue d'ajouter la fonctionnalité requise.

Nous avons aussi vu comment construire la composition des services web et nous avons présenté le plug-in Oracle BPEL Designer qu'on utilise pour la génération du processus BPEL.

Nous avons expliqué les changements qu'il a fallu faire dans le fichier XML dans lequel on a dû spécifier le nouveau composant chargé de générer la nouvelle fonctionnalité. Et finalement, nous avons commenté les changements que l'utilisation et dépendances de l'Outil BPEL ont produit dans le archive ZIP généré lors du déploiement

En conclusion, nous avons tenté de minimiser le plus possible les modifications à effectuer sur l'outil auteur en reléguant le déploiement et construction des services web composite à l'outil de publication.

4 Adaptation de l'outil publication

4.1 Introduction

Le but de l'outil publication est de récupérer le fichier XML généré par l'outil auteur, l'analyser pour reproduire dynamiquement l'environnement conçu auparavant.

En raison des changements faits dans l'outil auteur, il faut évidemment adapter l'outil de publication aux nouveaux besoins.

L'outil de publication commence par décompresser l'archive ZIP et copie son contenu sur le serveur. Bien que nous avons dû adapter la manière de l'outil auteur de générer l'archive Zip, après sa décompression, l'outil de publication ne doit entreprendre aucune action supplémentaire par rapport a ce qui a été implanté jusqu'ici.

Une fois que l'archive zip est décompressé avec succès, l'outil auteur procède a son analyse. S'il détecte la présence d'un processus BPEL (l'élément *WebServiceComposite*), il appelle déploie ce processus sur le serveur twister. Ainsi, dans le cadre de ce travail la plus part des adaptations effectuées sur l'outil de publication auront lieu à ce niveau là. Il faudra, aussi, générer des nouvelles transformations XSLT pour que l'outil de publication prenne en charge l'invocation des services web composite.

Dans la section suivante, nous allons expliquer avec plus de détailles comment le processus d'adaptation s'est effectué.

4.2 Traitement des Services Web Composite

Nous avons déjà expliqué comment le déroulement de l'outil publication est fait. Il faut maintenant spécifier comment il fait le traitement des services web composite.

D'abord, lors de l'analyse du fichier XML il faut chercher la présence d'un ou de plusieurs nœud indiquant la présence de services web composites, c'est à dire, il faut chercher le nœud appelé *WebServiceComposite*. Cette analyse se réalisera dans le fichier *StructDocFinder.java* qui se trouve dans le répertoire *WEB-INF\src\deployingTools\ structDocManager*.

Comme première pas nous avons ajouté la liste de nœuds *WebServiceComposite* trouvées dans le document XML à la liste des documents structurées:

```
	public StructDocFinder (File xmlFile) {
		this.xmlFile=xmlFile;
		this.xmlDocument=parseDocument();
		this.allStructDoc=getStructDoc();
		this.allWebService=getWebService();

		//Ajouter à la liste des documents structures les nœuds
		//WebServiceComposite trouvées
		this.allWebServiceComposite=getWebServiceComposite();
	    }


	//Cette méthode récupère l'ensemble des éléments
	//WebServiceComposite du document XML.
	private NodeList getWebServiceComposite () {
		NodeList webServiceComposite=
		xmlDocument.getElementsByTagName("WebServiceComposite");
		totalWebServiceComposite = webServiceComposite.getLength();
		return webServiceComposite;
	}
```

Ensuite, il faut voir si entre les documents structurés existent les services web composites. Cette analyse est faite dans le fichier *StructDocManager.java*. Nous allons faire cette opération dans la méthode utilisée pour effectuer la transformation XSLT des documents XML, des Services Web simples et, maintenant aussi, des Services Web Composite. Cette méthode est appelé *generateHTMLStructDocs*.

```
public boolean generateHTMLStructDocs(String contextPath) {
	…
	//s'il s'agit d'un document bpel, on génère le fichier html
	//correspondant avec la transformation XSL et tous les autres
	//fichiers nécessaire à l'interaction avec le service web
	//composite.
	if (fileType.equals("bpel") && !fileUrl.equals("")) {
	     String bpelName = bpelToHtml(structContent[i], contextPath);
	     …
	}
	…
}
```

Si nous trouvons un fichier bpel, on effectue tout un ensemble d'opérations consistant au déploiement du service web composite:

- Déploiement du processus sur Twister
- Génération de la servlet de communication
- Génération d'un formulaire html d'interaction
- Et, finalement, la génération d'un fichier web.xml pour le référencement des servlet. Si plusieurs services web ou services web composite sont compris dans le même cockpit, un seul fichier web.xml est généré, référant toutes les servlets.

Toutes les opérations indiquées seront réalisées dans la méthode *bpelToHtml* que sera expliqué en détails dans les sections suivantes. Nous parlerons du déploiement et de la communication avec le processus bpel dans la section qui traite sur Twister (section 4.3), et nous parlerons des transformations XSLT avec détail dans la section 4.4.

Pour pouvoir présenter l'architecture de la communication, nous introduirons brièvement le traitement qu'il faut réaliser. Le premier pas consiste à effectuer le déploiement du processus bpel. Une fois que le déploiement est correctement fini, une page HTML va être créée. Dans cette page HTML il faut avoir un formulaire qui permettra de générer un message qui sera envoyé au processus BPEL pour qu'il puisse agir en conséquence. Cependant, la page HTML n'est pas en mesure de rentrer directement en communication avec un processus BPEL. Il faut avoir un intermédiaire pour prendre les valeurs introduites dans le formulaire et les transformer aux valeurs acceptées pour le processus BPEL. Cet intermédiaire peut être un servlet. Toutes ces valeurs seront ensuite transmises au processus BPEL qui renverra le résultat au servlet. Lorsque la servlet reçoit le résultat, elle les reformaterait pour les insérer dans une page HTML que sera renvoyé au navigateur qu'a été à l'origine la requête initial.

Afin de pouvoir établir la connexion entre la servlet et le processus BPEL, il est obligatoire de passer par un serveur BPEL. Dans notre cas Twister est chargé d'établir cette communication entre la servlet et le processus BPEL appelé.

La figure suivant représente l'architecture de communication avec un processus BPEL, telle qu'il est implémenté dans le projet. Il montre aussi les connexions internes (partner links) que le processus BPEL génère mais qui sont transparents pour l'utilisateur.

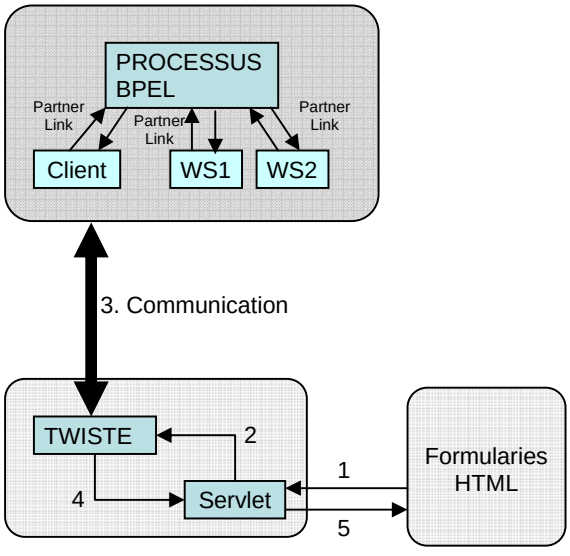


Figure 16: architecture de communication avec un service web composite

4.3 Déploiement des fichiers BPEL

4.3.1 Introduction

Dans cette section nous allons introduire le serveur BPEL qu'on utilise afin que les processus BPEL soient correctement déployés et invoqués.

Nous avons déjà observe, quand la sélection du serveur BPEL a été fait, les avantages et désavantagés de différents serveurs. Et la décision d'utiliser le Twister a été pris.

Twister est une solution complète et totalement open source du Business Process Management. Il est écrit en Java en suivant le standard WS-BPEL. Il s'est fondé sur les spécifications de BPEL et les principes des services web. Si le processus BPEL a été correctement spécifié et les échanges de messages ont été bien définis l'intégration de Twister devrait se dérouler assez facilement.

Afin de meilleur comprendre Twister, nous allons ensuite expliquer la solution que Twister apporte avec ses caractéristiques et analyser son fonctionnement.

4.3.2 Serveur BPEL: Twister

Twister [17] vise à fournir solution BPM (Business Process Management) facilement intégrable, orienté à B2B (Business to Business), écrit en Java sur les spécifications de WS-BPEL. Il support le modèle de SOA (Service Oriented Architecture) aussi bien que l'interaction humaine des participants.

La solution Twister comprendre trois bloque:

- Un *process designer* que aide aux utilisateurs à designer un processus de manière simple et produit une description suivant le standard XML (En processus d'implémentation, il n'est pas encore disponible).
- Le *deployeur* qui prend comme entrée les documents wsdl ou bpel et les déploie.
- Le *process engine* qu'exécute le processus métier.

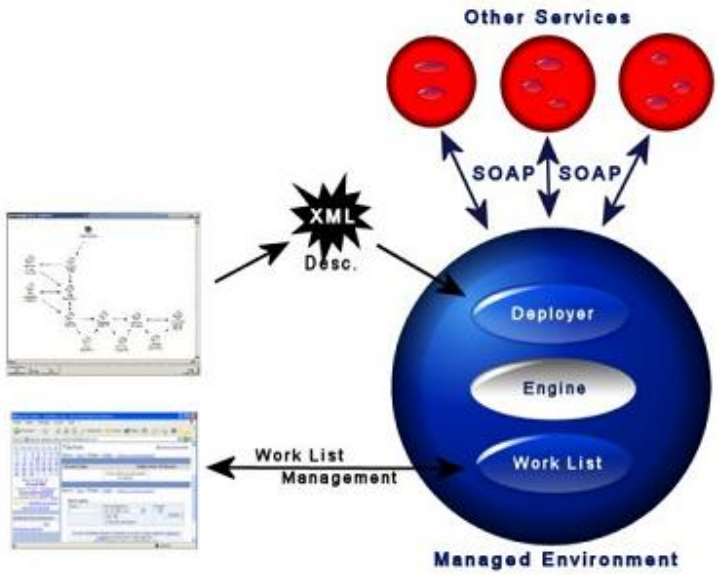


Figure 17: Twister solution

Cette figure montre les trois bloques desquelles nous venons de parler. Nous pouvons observer l'environnement d'administration qui va recevoir la description d'un processus BPEL comme entrée pour qu'il puisse le déployer. Une fois que le processus est déployé Twister doit gérer l'échange de messages avec les services web qui sera fait au moyen de SOAP. Une liste de travail aussi peut être générée pour l'administration de différents utilisateurs.

Twister fonctionne à l'intérieur d'un serveur web, tel que Tomcat [23], qui permet la connexion entre Twister et les services web requis (en utilisant Axis [18]) et serve également l'interface web de Twister.

La connexion avec un serveur de base de données est également requise pour Twister. Plusieurs serveurs de bases de données sont supportés (grâce à Hibernate [19]) et, en plus, une base de données légère et open source (HSQL [20]) est fournie avec Twister.

Une fois clarifiée la partie théorique de Twister et avoir envisagé la solution qu'il propose, nous allons continuer l'analyse en visant les différents manières de réaliser le déploiement et l'envoie de messages. En fait, Twister offre deux possibilités: la première depuis l'interface web offerte et le deuxièmes directement en utilisant l'API Java fournie.

I) Interface web Twister

Si l'on prend la première possibilité, cet à dire, utiliser l'interface utilisateur que Twister offre, nous auront une menu avec tous les opérations possibles. Ce menu offre toutes les opérations réalisables sur un processus BPEL et pour l'administration de l'outil.

Figure 18: Opérations offerts pour Twister

Il faut cliquer sur *Deployer* si l'opération à faire est de déployer un processus et/ou les fichiers wsdl de description. L'interface offert pour cet but est:

Figure 19: Interface pour le déploiement

Nous pouvons observer qu'il offre trois possibilités de déploiement. La première permet de déployer les services web importés (cette fonctionnalité n'est pas encore développé) ou les schémas XML. Le deuxième permet de déployer les fichiers wsdl qui décrivent le processus et finalement la troisième possibilité permet le déploiement d'un processus bpel. Avant de déployer un processus, il est obligatoire que sa description soit déployée.

Si nous voulons envoyer un message à un processus déjà déployé, il faut cliquer sur *send a message*. La prochaine interface est offerte par Twister:

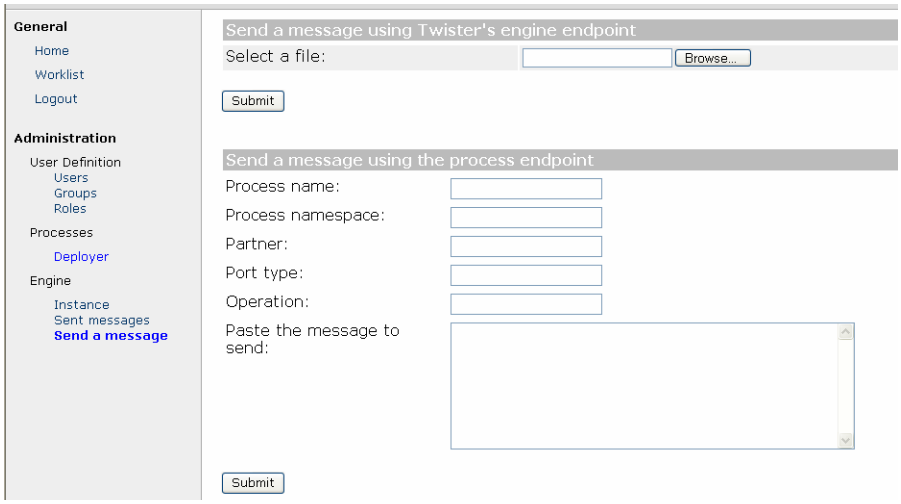


Figure 20: Interface pour envoyer les messages en Twister

Dans ce cas, deux modes d'envoie de messages est offert. La première enverra le message au moteur Twister et le deuxième l'enverra directement au processus désiré.

Bien que l'interface web soit simple et assez facile à utiliser, notre but est que le déploiement soit transparent pour l'utilisateur et qu'aucun plate-forme supplémentaire soit nécessaire par conséquence cette approche n'est pas utile dans notre cas. Nous allons donc envisager la deuxième possibilité, celle qui consiste à utiliser l'API Java fournie par Twister.

II) API Java

La deuxième alternative que Twister offre consiste en utiliser l'API Java de Twister. Les fonctionnalités fournisses sont exactement les mêmes que ceux offerts par l'interface web, cet à dire, nous pouvons administrer le serveur et réaliser des opérations (déploiement, envoie de message...) sur un processus bpel. Mais tout sera fait sans aide d'une plate-forme supplémentaire et de manière transparent pour l'utilisateur.

Pour pouvoir déployer une processus dans Twister, il faudra créer un client qui a nous permettra d'interagir avec le serveur. Pour que le client puisse déployer les

processus il doit être de type TwisterDeployer. Nous avons défini notre client de la manière suivant:

```
TwisterDeployer twisterDeployer = new TwisterDeployer();
```

Une fois le client défini, nous pouvons commencer le déploiement du processus. Avant de déployer le processus bpel il faut déployer la description du service. Twister permet de déployer un seul fichier wsdl pour chaque processus. Mais, attention, une seul fichier wsdl ne veut pas dire un seul service web sinon que tous les descriptions de tous les services web impliqués dans le processus doivent se trouver dans le même fichier wsdl.

Si nous regardons les fichiers générés depuis l'outil de modélisation Oracle BPEL designer utilisé pour créer le processus, nous nous rendons compte qu'il génère plusieurs fichiers wsdl et qu'il permet d'importer des services web. En principe, le sortie généré depuis l'outil de modélisation et l'entré de Twister ne sont pas compatibles. Il faut les rendre compatibles avant de procéder au déploiement. Nous avons crée la fonctionne *joinWSDL* dans le fichier *StructDocManager* qui va se charger de copier/coller tous les descriptions dans un seul fichier wsdl.

Cette adaptation a deux parties, la première consiste à copier les fichiers wsdl crées pendant la génération du processus. Nous trouverons un fichier pour chaque partner link définie, et ils seront dans un répertoire que s'appelle *services*. Il faut copier le nœud appelé *partnerLinkType* que se trouve dans chaque fichier et le coller dans le fichier wsdl principal.

```
if (nl.item(z).getNodeName().compareTo("plnk:partnerLinkType")==0)
{
      Node adopt = doc.importNode(nl.item(z),true);
      first.appendChild(adopt);
}
```

Nous trouverons aussi dans ce fichier wsdl un outre nœud appelé *import* et qu'indique la localisation du fichier wsdl importé pour le processus. Il faut récupérer cette fichier et copier le contenue dans le fichier wsdl principale.

```
If(nl.item(z).getNodeName().compareTo("import")==0){
      NamedNodeMap attr = nl.item(z).getAttributes();
      String str = attr.item(0).toString();
      str = str.substring(str.indexOf("/")+1);
      str = str.substring(0,str.indexOf("\""));
      Document importDoc = builder.parse(new File(str));
      NodeList wsdlDocNL = importDoc.getChildNodes()
      .item(0).getChildNodes();
      for (int t=0;t<wsdlDocNL.getLength();t++){
      Node wsdlAdopt = doc.importNode(wsdlDocNL.item(t),true);
      first.appendChild(wsdlAdopt);
      }
}
```

Nous avons maintenant un seul fichier wsdl décrivant le processus et un fichier bpel. Nous trouvons que dans le fichier bpel existent des références aux fichiers wsdl d'origine, conséquemment il faudra faire quelques ajustements dans ce fichier. Toutes les références aux fichiers wsdl d'origine doivent être changées pour les références à l'unique document wsdl. Dans ce but nous avons crée la méthode *adaptBPEL*. Pour chaque différent nœud contenant une référence le changement suivant sera appliqué :

```
if(attrs.item(k).getNodeValue().substring(0,attrs.item(k).getNode
Value().indexOf(":")).compareTo("tns")!= 0){
      String original= attrs.item(k).getNodeValue()
        .substring(0,attrs.item(k).getNodeValue().indexOf(":")+1);
      String replace = attrs.item(k).getNodeValue()
        .replaceAll(original,"tns:");
      attrs.item(k).setNodeValue(replace);
}
```

Une fois les deux fichiers adaptés aux besoins de Twister, nous pouvons poursuive avec le déploiement du processus. Il faut d'abord déployer le fichier wsdl de la manière suivante:

```
try{
      //Transferer le contenue du fichier wsdl à une buffer
      StringBuffer wsdlBuffer = getFileContent(wsdlAll);
      messages = twisterDeployer.deployWSDL
                  (wsdlBuffer.toString(),true);
}catch(ServiceException e){
      e.getStackTrace();
}catch(RemoteException e){
      e.getStackTrace();
}
```

Ensuite, si le déploiement du fichier wsdl a fini correctement nous pouvons déployer le fichier bpel :

```
try {
      bpelFile = adaptBPEL(dirs[i]);
      StringBuffer bpelBuffer = getFileContent(bpelFile);
      messages = twisterDeployer.deployBPEL
                  (bpelBuffer.toString());
} catch (ServiceException e) {
      e.getStackTrace();
} catch (RemoteException e){
      e.getStackTrace();
}
```

Si tout le processus finis sans erreurs le page suivant sera montré et nous pouvons commencer à interagir au moyen de messages avec cette processus.

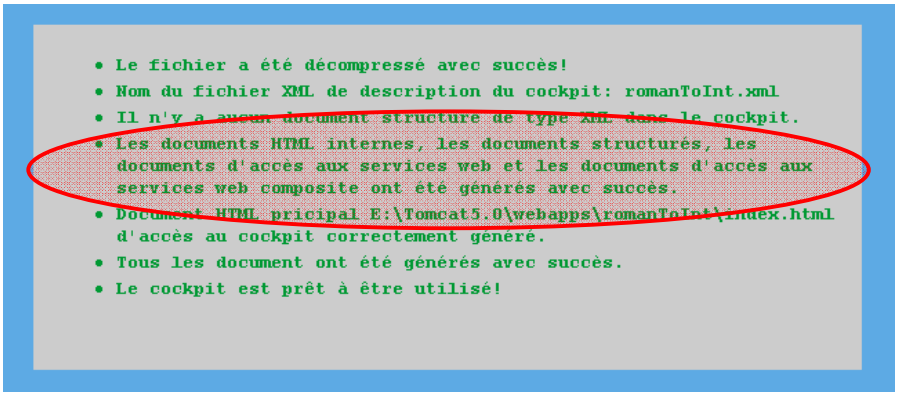


Figure 21: Le deploiement est fini sans erreurs

Twister offre deux manières d'envoyer des messages aux processus déjà déployés. Nous pouvons soit envoyer le message directement à un processus concret en spécifiant le nom du processus, soit l'envoyer au moteur Twister qui se chargera de trouver le destinataire. Cette deuxième approche permet de s'abstraire de tout le processus et de déléguer cette responsabilité au Twister. Il permet d'envoyer un message sans savoir exactement quel processus est la cible. Nous avons choisi cette dernière approche parce qu'il nous permet de simplifier le management de messages en laissant tout la responsabilité au Twister.

Le message qu'il faut envoyer au Twister doit avoir une structuration spéciale qu'est appelle *twister-message*. Un exemple de ce type de messages peut être la suivante:

```
<?xml version="1.0" encoding="UTF-8"?>
<twister-msg xmlns="http://www.smartcomps.org/twister/types/message/">
    <partner>loaner</partner>
    <port>loanServicePT</port>
    <operation>request</operation>
    <message>
        <firstName>john</firstName>
        <lastName>doe</lastName>
        <amount>5000</amount>
    </message>
</twister-msg>
```

Figure 22: Exemple d'un twister-message

L'envoie de messages est fait depuis le servlet généré au moyen d'un transformation XSLT appelé *WSC-Servlet-Generator.xsl* et que se trouve dans le répertoire *cockpitDeploy_linkedFiles\wsXslFiles*.

D'abord, il faut construire le message qu'on doit envoyer au serveur BPEL. Il faut récupérer les valeurs depuis un formulaire HTML et les paramètres envoyés depuis le fichier *StructDocManager*. Le message a été généré de la manière suivant:

```
Document doc=DocumentHelper.createDocument();
Element root = doc.addElement("twister-msg");
root.addElement("partner")
            .setText("<xsl:value-of select="$partner"/>");
root.addElement("port").setText("<xsl:value-of select="$port"/>");
root.addElement("operation")
            .setText("<xsl:value-of select="$operation"/>");

Element msgRoot = root.addElement("message");
Enumeration e=request.getParameterNames();
while(e.hasMoreElements()){
      String msgElement = (String)e.nextElement();
      msgRoot.addElement(msgElement)
            .setText(request.getParameter(msgElement));
}
```

Une fois que nous avons le message prêt à être envoyé, il faut créer de nouveau un client pour que l'interaction avec le serveur Twister soit possible. Cette fois le client doit pouvoir interagir avec le moteur donc il faut qu'il soit de type *TwisterEngine.*

```
      TwisterEngine engine = new TwisterEngine();
```

Maintenant, nous pouvons proceder avec l'envoie de notre message. Twister renverra comme response le résultat obtenue à la fin du processus bpel dû au message envoié. L''envoie est fait de la maniére suivant :

```
Document responseDoc = engine.acknowledge(
                        "<xsl:value-of select="$partner"/>",
                        "<xsl:value-of select="$port"/>",
                        "<xsl:value-of select="$operation"/>",
                        new DOMWriter().write(msg)
                        );
```

4.4 Transformations XSLT

Afin de satisfaire la réquisition de cette nouvelle approche, il s'est avéré nécessaire de générer des nouvelles transformations XSLT [21] pour le traitement des services web composite. Ces transformations ont comme base les transformations XSLT existant pour les services web simples et ils essayent de suivre la même structure et style.

Trois transformations existent pour les services web simples [Gra05] :

- La première consiste à produire la page HTML avec le formulaire d'interrogation du service Web.
- Le deuxième serve pour produire le code Java de la servlet servant de lien direct avec le service Web.
- Et le dernière est création d'une fichier nommé « Web.xml » utilisé pour référencer la servlet dans le serveur Tomcat.

Pour le traitement des services web composite il faut générer les mêmes trois transformations. La première, nommé *WSC-Html-Generator.xsl*, consistant à produire la page HTML avec le formulaire d'interrogation du processus BPEL, le deuxième, nommé *WSC-Servlet-Generator.xsl*, qui doit produit le code Java de la servlet servant de lien avec le processus BPEL et, comme avant, dû à l'utilisation de la servlet le création du fichier « Web.xml » s'avère nécessaire et il est généré par le fichier *WSC-Web_Xml-Generator.xsl*. Tous ces fichiers se trouvent dan le répertoire *cockpitDeploy_linkedFiles\wsXslFiles.*

La transformation *WSC-Html-Generator.xsl*, est chargée de produire une page html qui va nous permettre obtenir les données nécessaires pour la génération du message d'entrée au processus BPEL. Pour arriver à ce but, il faut connaitre la première opération à réaliser et obtenir le format du message correspondant. Et après, afficher les valeurs requis pour ce message dans la forme d'un formulaire html:

```
<form action="../servlet/{$servletName}" method="get">
      …
      <xsl:if test=".//*[local-name()='operation']/@name=$operation">
            <xsl:call-template name="request">
                  <xsl:with-param name="input" select="substring-after
                        (.//*[local-name()='input']/@message, ':')"/>
            </xsl:call-template>
      </xsl:if>
      …
</form>

<xsl:template name="request">
   <xsl:param name="input"/>
   <xsl:for-each select="//*[local-name()='message'][contains(@name,
$input)]">
      <xsl:if test="@name=$input">
         <xsl:for-each select="./*[local-name()='part']">
            …
            <xsl:value-of select="@name"/>
```

```
            <xsl:value-of select="substring-after(@type, ':')"/>

            <input name="{@name}" type="text"/>
            …
         </xsl:for-each>
      </xsl:if>
   </xsl:for-each>
</xsl:template>
```

Après d'avoir finie avec cette transformation, nous obtiendrons la une page html dans la forme suivant :

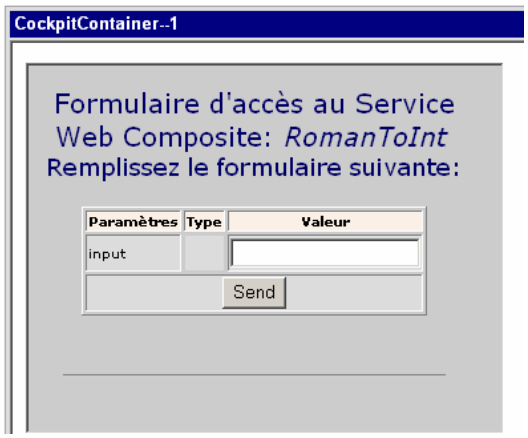


Figure 23: Formulaire html d'interrogation

La deuxième transformation, *WSC-Servlet-Generator.xsl*, va générer le servlet chargé de communiquer la page html et le serveur Twister. Cette servlet est appelé depuis le formulaire cliquant sur le bouton *Send*. La première tâche de cette sevlet est générer le message qu'il faut envoyer à Twister et l'envoyer, il est fait comme expliqué dans la section précédent (voir section 4.3.2 II) API Java).

Une fois, que le message a été envoyé au serveur BPEL et le résultat récupéré. Le servlet doit traiter le résultat obtenu depuis la requête et l'afficher dans un deuxième page HTML. Le message retourné aura la forme suivant :

```
<message>
        <return>fifty</return>
</message>
```

Figure 24: Message de response de Twister

Il faut maintenant traiter le message obtenu pour qu'il soit affiché dans une page html. On doit parcourir le message jusqu'à récupérer la valeur retourné pour le processus, nous le feront de la manière suivant :

```
NodeList nl = responseDoc.getChildNodes();
for(int i=0;i&#60;nl.getLength();i++){
      NodeList subNL = nl.item(i).getChildNodes();
      for(int j=0;j&#60;subNL.getLength();j++){
            NodeList ssubNL=subNL.item(j).getChildNodes();


      for(int k=0;k&#60;subNL.getLength();k++){
                  printHtmlResponse(out,
                        ssubNL.item(k).getNodeValue());

            }
      }
}
```

Et finalement, la page html résultant aura la forme suivante:

Figure 25: Page html qu'affiche le résultat de la rêquete

Et finalement, il faut générer la transformation *WSC-Web_Xml-Generator.xsl* chargé de créer le document *Web.xml* nécessaire pour que Tomcat prenne en charge les servlets de communication entre les pages html et Twister. Un seul fichier *web.xml* sera crée pour tous les servlets existant. Les deux actions principales seront de créer les nœuds `<servlet>` et `<servlet-mapping>` et ils sont faits en appliquant les Templates suivants :

```
<!-- Template pour imprimer le noeud <servlet> -->
<xsl:template name="print_servlet_node">
   <xsl:param name="servletName"/>
   <servlet>
      <servlet-name>
         <xsl:value-of select="$servletName"/>
      </servlet-name>
```

```
            <display-name>
               <xsl:value-of select="$servletName"/>
            </display-name>
            <servlet-class>
               <xsl:value-of select="$servletName"/>
            </servlet-class>
         </servlet>
      </xsl:template>

      <!-- Template pour imprimer le noeud <servlet-mapping> -->
      <xsl:template name="print_servlet-mapping_node">
         <xsl:param name="servletName"/>
         <servlet-mapping>
            <servlet-name>
               <xsl:value-of select="$servletName"/>
            </servlet-name>
            <url-pattern>
               /servlet/<xsl:value-of select="$servletName"/>
            </url-pattern>
         </servlet-mapping>
      </xsl:template>
```

4.5 Conclusions

Nous avons présenté, dans cette section, la manière dont nous avons géré la composition des services web dans ce projet. Nous avons décrit la manière dont nous avons pris en charge la génération automatique d'un interfaçage de communication avec un processus BPEL.

Nous avons présenté le serveur Twister, commenté les différentes possibilités qu'il offre et nous avons expliqué comment nous avons fait pour le déploiement des services Web composites.

Pour que la génération automatique d'un interfaçage de communication puisse se réaliser, nous avons créé les feuilles de style destinées à générer un document HTML, le code de la servlet de communication ainsi qu'un fichier '`Web.xml`' nécessaire au serveur Tomcat pour référencer les servlets.

Le document HTML, produit automatiquement, contient un formulaire que l'utilisateur doit remplir. Le servlet agit comme intermédiaire vers le processus BPEL et renvoie la réponse du service web.

Finalement, nous avons présenté la manière dont nous avons pu adapter l'outil de publication pour qu'il prenne correctement en charge les éléments de type 'WebServiceComposite', contenus dans le fichier XML de description du cockpit.

5 Conclusion générale

En ce qui concerne le déroulement du projet, deux étapes principales ont été nécessaires pour son développement.

La première étape de ce projet a consisté à reprendre l'outil auteur implémenté lors d'un projet précédant, afin de l'adapter aux nouvelles fonctionnalités requises. Nous avons introduit la notion des services Web composite et la façon dont nous les avons gérés. Nous avons intégré et expliqué l'outil choisi pour concevoir les processus BPEL et les changements requis dans l'outil pour que ces changements soient pris en charge.

Dans un deuxième temps, nous avons pris l'outil de publication et nous avons présenté les étapes successives pour l'intégration des services web composite. Pour cela, nous avons présenté la façon dont nous avons dû déployer et invoquer le processus métier pour intermédiaire du serveur Twister. Et finalement, nous avons présenté les transformations XSLT nécessaires pour communiquer le page HTML, généré par l'outil publication, et le processus BPEL.

Étant donné que le but de ce projet est la modélisation semi-automatique des services Web par des personnes non expertes en systèmes d'information, est-ce qu'on peut dire que ce but est atteint? Lamentablement nous n'avons pas pu y arriver. Tous les outils de modélisation existant demandent un certain niveau de connaissance en informatique. Ces outils sont vraiment intuitives pour quelqu'un avec certaines notions de programmation mais peuvent être assez compliqués pour une personne, qui, par exemple n'a jamais écouté parler des boucles et ne sait pas comment les construire.

Pendant ce projet, nous avons utilisé l'outil de modélisation le plus simple et facile à utiliser que nous avons trouvé. Ce pendant, il n'existe pas dans le marché un outil conçu pour des utilisateurs sans connaissances préalables en informatique. Une solution plausible pour ce problème peut être le développement d'une couche supérieur qui abstrait encore plus la modélisation et la rendre possible pour les utilisateurs non experts en systèmes d'information. Sinon il faudra attendre jusqu'à qu'un outil avec ces caractéristiques soit disponible sur le marché et l'intégrer dans notre environnement.

Pour le déploiement des services web composite nous avons utilisé le serveur Twister. Ce serveur est encore en vois de développement et il serait convenant d'immigrer à une nouvelle version dans le moment que l'importation des fichiers wsdl et le déploiement de plusieurs fichiers pour chaque processus bpel soit accepté. Cette fonctionnalité est annoncé pour la prochain version 3.1 que doit être accessible d'ici peu.

D’un point de vue personnel, ce projet a été stimulant par sa diversité. J’ai eu l’opportunité d’utiliser différentes technologies, outils et langages de programmation. J’ai découvert la composition des services web et les possibilités qu’il offre au de-là des services web simples que je connaissais avant.

En ce qui concerne les langages de programmation, j’ai surtout travaillé avec java. Mais j’ai fallu étudier aussi les langages de modélisation des processus métier BPEL et utiliser les transformations XSLT.

Il a été un challenge de travailler avec twister et de réaliser tout le processus de déploiement et envoi de message d’une manière transparente pour l’utilisateur, parce que étant donnée que Twister est un outil assez jeune, encore en développement et open source la documentation et les exemples sont peu abondants. J’ai dû dédier beaucoup de temps à cette tâche mais qui a été finalement une grande satisfaction d’avoir pu emmener à bon apport.

.

Annexes

Annexe A
Installation de l'outil auteur

A-I Introduction

Nous présentons brièvement dans ce qui suit le manuel d'installation de l'outil auteur. Pour plus de détails sur les procédures d'installation, prière de se référer au rapport traitant de la conception de l'outil auteur [Pas04] et qui traite les procédures d'installation de cet outil en détaille.

A-II Procédures d'installation

Dans les fichiers que nous rendons avec ce rapport, nous fournissons une archives ZIP contenant l'outil auteur que nous avons modifié. Cette archive nommée '`cockpitComposerPlugin.zip`' contient le plugin de l'outil auteur.

Pour installer ce plugin, il est nécessaire d'avoir déjà installé à l'avance eclipse [14] sur sa machine. Une fois eclipse installé, il suffit de décompresser le contenu de cette archive dans le répertoire de base d'eclipse. Les fichiers du plugin seront ainsi copiés dans le sous répertoire nommé 'plugins'.

Remarque :

Si lors du première démarrage de l'outil auteur la barre de l'outil et le menu se référant au cockpit ne sont pas affichés, comme nous pouvons le constater dans la figure 25 :

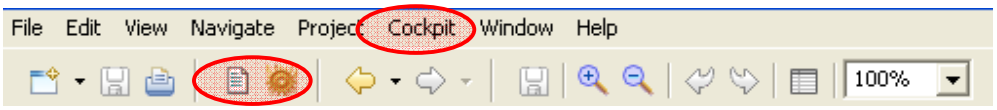


Figure 26 Menu de l'outil de publication

Dans le cas où le menu 'Cockpit' et les deux boutons ne soient pas affichés, il faut suivre les pas spécifiés ci-dessous.

- Positionner la souris à droite de la barre d'outils et cliquer sur le bouton droit. La fenêtre suivante apparaît :

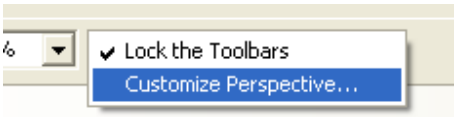


Figure 27 Cliquer sur le bouton droite

- Cliquer sur le champ '*Customize Perspective…*'. La fenêtre suivante apparaît :

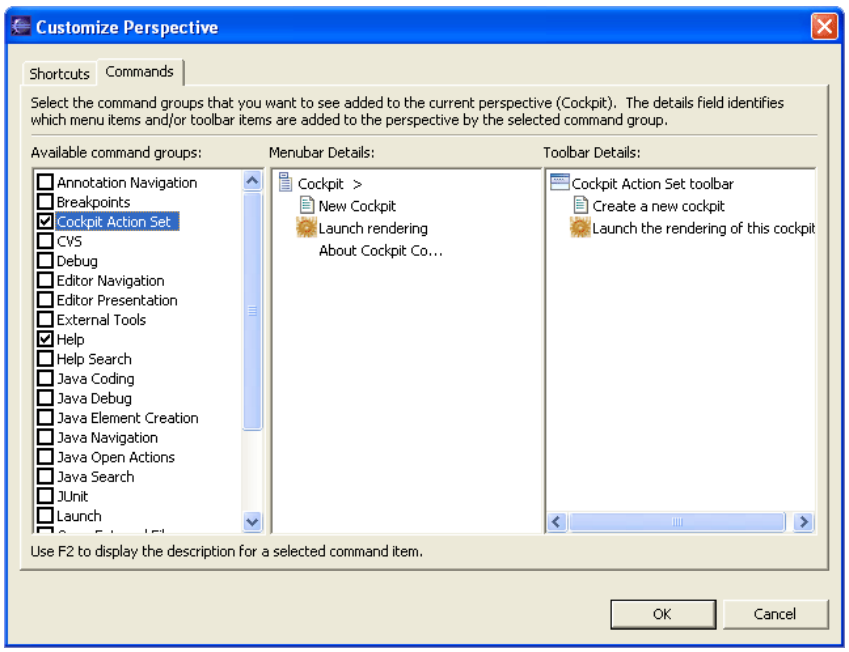


Figure 28 Fenêtre de personnalisation des barres de menus et d'outils

- Cliquer sur l'onglet '*Commands*', sélectionner '*Cockpit Action Set*' et cliquer sur le bouton '*OK*'.

La barre de l'outil auteur et le menu seront correctement affichés lorsque l'outil auteur sera relancé les fois suivantes.

A-III Manuel d'utilisation de l'outil auteur

Concernant le manuel utilisateur de l'outil auteur, prière de se référer au manuel utilisateur annexé au rapport de l'outil auteur.

Annexe B
Installation d'Oracle BPEL Designer

B-I Introduction

Dans cette annexe, le manuel d'installation de l'outil de modélisation *Oracle BPEL Designer* et un bref manuel d'utilisation seront présentes. Pour plus de détails veuillez vous diriger au manuel fournie pour *Oracle BPEL Designer* [16] et qui traite tous les possibilités de cet outil en détail.

B-II Installation d'Oracle BPEL Designer

Le BPEL Designer est un plug-in d'Eclipse que support la version 3.0GA ainsi vous devrez d'abord télécharger et installer une distribution compatible d'Eclipse. L'outil de modélisation devrait fonctionner avec des versions postérieures d'Eclipse, mais ces instructions ont été testées avec la version 3.0GA et sont soutenues pour ce version seulement.

Les pas à compléter pour la correcte installation de l'outil sont:

- Assurez vous d'avoir le JDK 1.4.1 ou postérieur.
- Assurez vous d'avoir la distribution correct d'Eclipse, dit Eclipse 3.0 GA.
- Oracle BPEL Designer peut être téléchargé depuis http://otn.oracle.com/bpel. Une fois le téléchargement réalisée il faut démarrer l'exécutable d'installation. Le répertoire d'installation par défaut BPEL est `C:\orabpel`.

Maintenant vous devrais être capable de faire fonctionner votre Oracle BPEL Designer, comme montré ci-dessous.

B-III Manuel d'utilisation d'Oracle BPEL Designer

Pour créer un nouveau projet de BPEL :

- Dans Eclipse, cliquer **File > New > Project**, choisissez **Oracle BPEL Project** et cliquer sur **Next.**

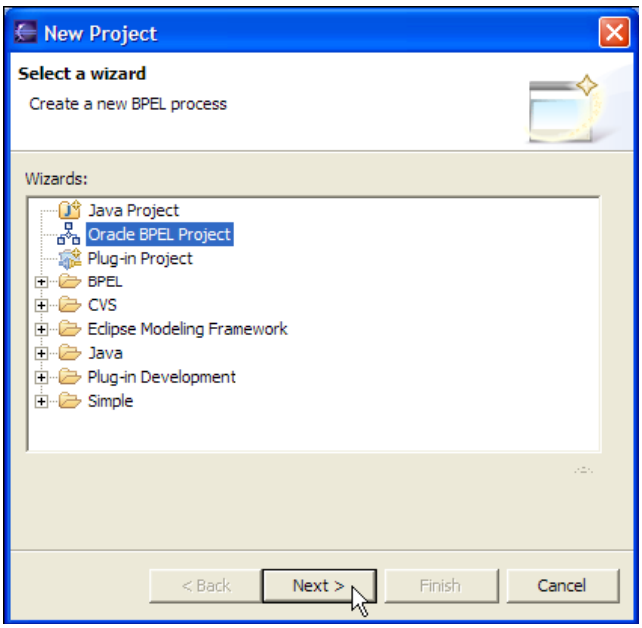


Figure 269: Oracle BPEL Projet

- Introduisez le nom pour le processus de BPEL, le namespace et choisissez si le processus est synchrone ou asynchrone. Après, cliquer sur **Finish**.

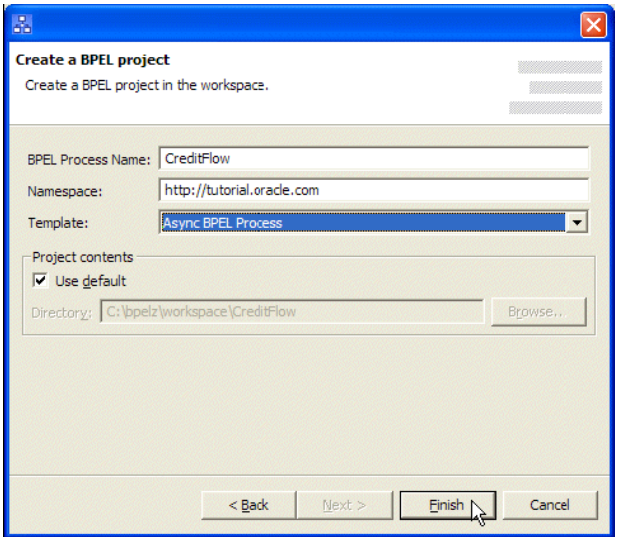


Figure 30: Oracle BPEL Projet spécifications

- Le nouveau projet sera créé dans le répertoire *C:\eclipse\workspace*. L'outil de modélisation a créé un squelette pour un nouveau processus BPEL, avec tous les fichiers source nécessaires. Les noms des fichiers sont identiques si vous créez un processus synchrone ou asynchrone, mais le contenu des fichiers bpel et wsdl différeront comme approprié.

- Dirigez vous maintenant à la vue de *Process Map*, où vous éditerez votre processus de BPEL.

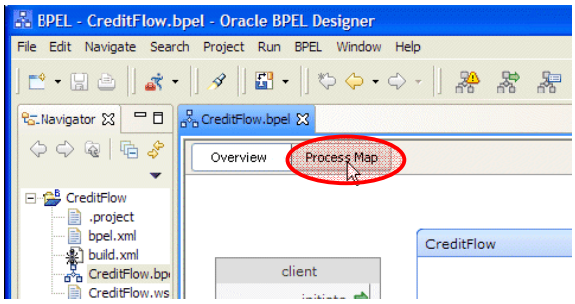


Figure 31: Process Map

- Pour ajouter différents activités au Process Map, il faut aller à la palette de BPEL et cliquer dans *More Activities* pour voir l'ensemble d'activités permis. Traînez une activité (depuis la palette de BPEL) à la flèche de transition entre l'activité initiation et la réponse du processus.

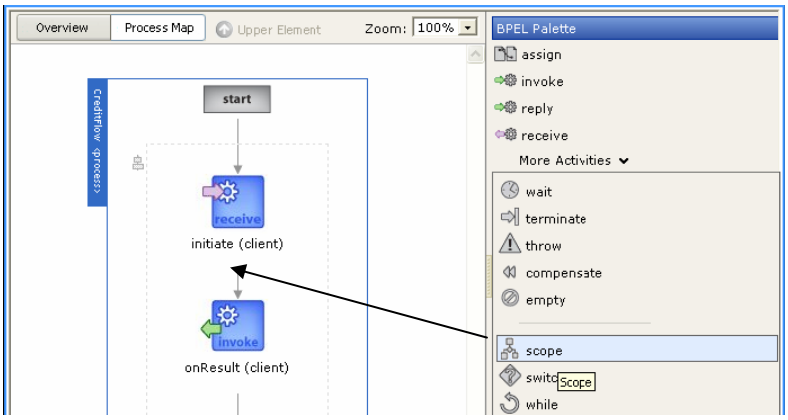


Figure 32: Ajouter activités

- Dans le BPEL designer, vous pouvez ajouter des partnerLinks à votre BPEL soit depuis la vue d'ensemble ou lorsque vous configurez l'activité.

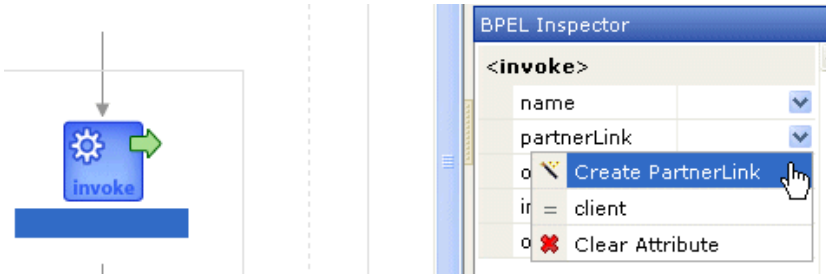


Figure 33: Création d'un Partner Link

- Pour plus d'information sur la création des processus BPEL soient asynchrones ou synchrones veuillez vous adresser au Tutorial fournie pour l'outil BPEL.

Annexe C
Installation de l'outil de publication

C-I Introduction

Ensuite, nous présentons brièvement le manuel d'installation de l'outil publication. Pour plus de détails sur les procédures d'installation, prière de se référer au rapport traitant de la conception de l'outil de publication [Gra05] et qui traite les procédures d'installation de cet outil en détaille.

C-II Installation du serveur Tomcat

Étant donne que l'outil de publication fonctionne sur un serveur Tomcat, il s'avère nécessaire que Java soit installée. Ensuite, avant d'installer l'outil de publication il faut installer Tomcat. Nous pouvons le télécharger depuis l'adresse http://jakarta.apache.org/tomcat/ ou prendre la version fournie avec les fichiers de ce projet. La version utilisée dans ce projet a été la 5.0.28.

Lors de l'installation, il existe la possibilité de configurer le port pour accéder localement au serveur. Nous avons conservé la valeur par défaut qui utilise le port 8080. Ainsi, une fois installé, l'accès en local à la page d'accueil du serveur Tomcat se fait par l'adresse 'http://localhost:8080/'.

Sous Windows, le serveur Tomcat est installé par défaut dans le répertoire '*C:\Program Files\Apache Software Foundation\Tomcat 5.0*', si cette répertoire base est changé pendant l'installation il faut actualiser la ligne de code suivante dans le fichier *unpack.jsp* qui se trouve dans le directoire *cockpitDeploy*.

```
<input type="hidden" name="realContextPath" value="C:\Program
      Files\Apache Software Foundation\Tomcat 5.0\webapps">
```

Le répertoire de base de Tomcat est nommé '`CATALINA_HOME`'.

Remarques :

- Pour le bon fonctionnement du serveur Tomcat, il faut spécifier une nouvelle variable d'environnement nommée `JAVA_HOME`. Cette variable d'environnement doit pointer sur le répertoire de base où a été installée Java.
- La configuration des différents paramètres d'exécution du serveur Tomcat est définie dans le fichier `%CATALINA_HOME%\conf\server.xml`. Dans ce fichier, un paramètre permet de gérer la mise en place dynamique de nouvelles applications Web, évitant ainsi le redémarrage du serveur. Pour cela, il est nécessaire que l'attribut '`autoDeploy`' de l'élément `<Host>` ait la valeur '`true`'. Ceci permettra le déploiement dynamique des nouveaux cockpits. Normalement la valeur de cet attribut est déjà définie à '`true`' lors de l'installation du serveur.

C-III Installation de l'outil de publication

Dans les fichiers que nous rendons avec ce rapport, nous fournissons une archives ZIP, nommée '`cockpitDeploy.zip`', qui contient l'outil de publication que nous avons modifié. Il faut décompresser cette archive dans le répertoire base du serveur Tomcat.

D'un coté, nous trouvons dans cette archive l'ensemble des librairies nécessaires à la publication de nouveaux cockpits. Ces fichiers seront copiés dans le répertoire `%CATALINA_HOME%\common\lib\`.

D'autre coté, il contient aussi tous les fichiers nécessaires pour l'exécution de l'outil de publication et des nouveaux cockpits. Ces fichiers seront copiés dans les répertoires `%CATALINA_HOME%\webapps\cockpitDeploy\` et `%CATALINA_HOME%\common\_linkedFiles\`.

Les cockpits génères à l'aide de l'outil auteur seront installés dans le répertoire `%CATALINA_HOME%\webapps\`.

Pour le démarrage de l'outil de publication, il faut démarrer un navigateur Web à l'adresse http://localhost:8080/cockpitDeploy.

C-IV Manuel d'utilisation de l'outil de publication

Concernant le manuel utilisateur de l'outil de publication, prière de se référer au manuel utilisateur annexé au rapport de l'outil de publication.

Annexe D
Installation du serveur Twister

D-I Introduction

Dans cette annexe, le manuel d'installation et configuration de Twister seront présentes. Pour plus de détails veuillez vous diriger aux manuels fournies pour Twister [17] et qui fait un traitement plus détaillé.

D-II Pre-conditions

Pour le correct fonctionnement de Twister il doit être à l'intérieur d'un serveur de servlet (Tomcat) et il doit être relié à deux types de bases de données : une base de données relationnelle et une base de données de XML. Il a besoin également de *Apache Ant*[32] pour les scripts.

Twister est liée dans l'installation à la base de données HSQL-DB, une base de données relationnelle free source écrite dans Java. Mais, il peut être intégré avec une grande variété de bases de données comme MySQL, Oracle... Twister va également fournir une base de données de XML : Apache Xindice.

Finalement, une installation fonctionnelle d'Ant sera nécessaire. Il faudra le télécharger et le décompresser dans un répertoire que sera nommé `ANT_HOME`, et devra être ajouté au classpath.

D-III La distribution de Twister

D'abord, télécharger Twister depuis la de la page du téléchargement de Sourceforge (http://sourceforge.net/project/showfiles.php?group_id=82076). Il faut décompresser le dossier téléchargé, vous devriez trouver la distribution suivante :

- bin: ant scripts et exécutables.
- conf: twister et les fichiers de configuration des dépendances, des XML schemas utilisées par Twister.
- Doc: web site documentation. Si HSQL est utilisé les données seront stockes dans cette répertoire.
- lib: librairies utilisées pout Twister et leurs licences.
- test: l'ensemble de test et de descriptions des processus BPEL.
- samples: exemples des processus et descriptions
- war: applications web de twister.
- twister.license

D-IV Configuration de la base de données HSQL

La configuration de Twister emploie HSQL par défaut, par conséquence rien de spécial doit être fait pour l'installer. La seule chose que vous devez faire est démarrer et initialiser la base de données. Twister a crée un tâche Ant que va démarrer HSQL dans une console, il faut juste écrire : `ant hsql-start.` Quand on démarre HSQL pour la première fois une nouvelle base de données vide sera crée automatiquement. Les fichiers de données sont créés dans le répertoire etc/hsql.

Une fois que HSQL est initié il faut créer les tables de Twister et initialiser la base de données. Un script fait ceci automatiquement, il faut juste exécuter le commande suivante: `ant db-create.` Une fois que cette commande a été exécutée, toutes les tables nécessaires devraient avoir été créées et quelques données initiales (comme, par exemple, l'utilisateur administrateur) devraient avoir été insérées.

D-V Installer Twister sur Tomcat

Brièvement :

- Copier tous les fichiers .war dans *twister/war* à *tomcat/webapps*
- Copier *twister/lib/hsqldb-x.x.jar* à *tomcat/common/lib*
- Copier *twister/conf/hibernate.cfg.xml* et *twister/conf/twister-configuration.xml* à *Tomcat/common/classes*.

Pour que Twister permette l'invocation de services web, il faut configure correctement le fichier *twister-configuration.xml* qui se trouve dans le répertoire *common/classes* de Tomcat. Dans le nœud *engine/implementation* vous devriez trouver quelque chose comme :

```
<message-broker
impl="org.smcp.twister.engine.priv.messaging.impl.DefaultMessageBr
okerImpl"/>
```

Pour permettre l'invocation de services web en utilisant la description WSDL déployée, remplacez juste ceci avec:

```
<message-broker
impl="org.smcp.twister.engine.priv.messaging.impl.WSMessageBroker"
/>
```

Pour utiliser le module de client de Twister, il faut installer le classpath correctement. Donc, à part de twister-client.jar, on doit inclure les librairies suivantes:

- axis-1.1.jar : librairie principal d'Axis, nécessaire pour l'invocation des services Web.
- axis-jaxrpc-1.2-alpha-1.jar : nécessaire pour Axis

- axis-saaj-1.2-alpha-1.jar : nécessaire pour Axis
- commons-discovery-SNAPSHOT.jar nécessaire pour Axis
- commons-logging-1.0.3.jar : librairie pour la notation.

6 Bibliographie

Ouvrages consultés lors de ce travail :

[Ber04] D.BERKOVITS, Projet de semestre - Outil de publication pour la gestion de l'emplacement spatial au sein d'environnements d'expérimentation Web. Lausanne :LA-EPFL, 2004.

[Gra05] O.GRANGES, Projet de master - Outil de publication pour la production d'environnements d'expérimentation Web. Lausanne :LA-EPFL, 2005.

[Pas04] S. PASCHE, Projet de diplôme - Outil auteur pour la production de documents actifs composites structurés. Lausanne :LA-EPFL, 2004.

[Juric05] M. JURIC, BPEL and JAVA. April 2005

[Sri03] B. SRIVASTAVA, J. KOEHLER. Web Service Composition – Current Solutions and Open Problems. ICAPS 2003 (http://citeseer.ist.psu.edu/srivastava03web.html)

[Gar05] I. GARCIA, M. POLO, F.RUIZ, M. PIATTINI. Servicios Web. Universidad de Castilla-La Mancha, Espagne. Informe técnico UCLM DIAB 05/01/1

[Aal03] W.M.P. VAN DER AALST, M. DUMAS, A.H.M. TER HOFSTEDE. Web Service Composition Languages: Old Wine in New Bottles?, 2003

[Ari05] J.ARIAS, C.DELGADO, L.SANCHEZ. Verificacion de Composiciones de Servicios Web : Aplicacion de Model Checking a BPEL4WS. Mars 2005.

[Fost03] H.FOSTER, S.UCHITEL, J.MAGEE, J.KRAMER. Model-based Verification of Web Service Composition, Oct. 2003.

[Peltz03] C.PELTZ. Web Service Orchestration and Choreography – A look at WSCI and BPEL4WS. July 2003.

[Ley01] Prof. Dr. F.LEYMANN. Web Service Flow Language (WSFL 1.0). IBM, may2001

[Tha01] S.THATTE. XLANG :Web Services for Business Processes Design. Microsoft, 2001.

Sites Internet consultés lors de ce travail :

[1] Web Services
http://www.3ie.org/nouvelles_technologies/fiches.php?techno_id=74

[2] SOAP Specifications
http://www.w3.org/TR/soap/

[3] Web Service Definition Language (WSDL)
http://www.w3.org/TR/wsdl

[4] UDDI
http://www.uddi.org

[5] Web Service Composition Standards
http://lsdis.cs.uga.edu/proj/meteor/mwscf/standards.html

[6] Cover Pages
http://www.bpms.info/lexique_detail.asp?ref=154

[7] Web Service Choreography Interface (WSCI)
http://www.w3.org/TR/wsci
http://www.isima.fr/ponge/dea/rapport-dea.pdf

[8] Web Service Conversation Language (WSCL)
http://www.w3.org/TR/wscl10
http://www.isima.fr/ponge/dea/rapport-dea.pdf

[9] Search Web Services related Definitions
http://searchwebservices.techtarget.com

[10] Business Process Execution Language for Web Services (BPEL4WS)
http://www-128.ibm.com/developerworks/library/specification/ws-bpel
http://www.vnunet.fr/tpepme_-_business/entreprises/20020829007

[11] Web Service Reliable Messaging
http://www.oasis-open.org/committes/wsrm

[12] WS – Addressing
http://www.w3.org/submission/ws-addressing

[13] WS-Transaction specifications
http://www-128.ibm.com/developerworks/library/specification/ws-tx

[14] The Eclipse Project
http://www.eclipse.org

[15] BPW4J
http://www.alphaworks.ibm.com/tech/bpws4j

[16] Oracle BPEL Process Manager
http://www.oracle.com/technology/products/ias/bpel

[17] Twister
http://www.smartcomps.org/twister

[18] Axis (Apache <Web Services/> Project):
http://ws.apache.org/axis/

[19] Hibernate
http://www.hibernate.org
http://sourceforge.net/projects/hibernate

[20] HSQL
http://www.hsqldb.org
http://sourceforge.net/projects/hsqldb

[21] XSLT
http://www.w3.org/TR/xslt

[22] Web The Extensible Stylesheet Language Family (XSL)
http://www.w3.org/Style/XSL/

[23] The Apache Software Foundation
http://www.apache.org/
http://www.apachefrance.com/

[24] Mozilla Firefox:
http://www.mozilla-europe.org/fr/products/firefox/

[25] Java Technology :
http://java.sun.com/

[26] Le projet eMersion
http://emersion.epfl.ch/

[27] Description de l'outil auteur actuellement utilisé
http://emersion.epfl.ch/ cockpit.html#cockpit

[28] World Wide Web Consortium
http://www.w3.org/

[29] Extensible Markup Language (XML)
http://www.w3.org/XML/

[30] Laboratoire d'Automatique (LA)
http://lawww.epfl.ch/

[31] XLANG
http://www.alaide.com/dico.php?q=XLANG

[32] Apache Ant
http://ant.apache.org/