Business Process Execution Language

Rapport du projet de systèmes distribués d'information

Markus Lindström

6 mai 2009

Motivation personnelle

Le sujet que j'ai retenu et présenté dans le cadre du cours de systèmes distribués d'information de M. Zimányi m'est venu lors d'une réunion en tant que représentant étudiant. Pour remettre cela en contexte, il faut savoir que depuis une grosse année maintenant, l'Université Libre de Bruxelles mène un grand projet de refonte de toute l'infrastructure informatique de son administration en implémentant un système logiciel de gestion intégré (appelé *ERP* pour *Enterprise Resource Planning*), projet répondant au nom de SMILEY (`http://www.ulb.ac.be/smiley/`).

Dans ce cadre, l'Université a dû documenter les processus divers et variés mais surtout très compliqués de son administration, chose qui n'avait pas été faite auparavant. C'est alors que le terme BPEL a surgi lors d'une réunion du comité de pilotage SMILEY et m'a intrigué, et j'ai profité de l'occasion pour explorer le sujet.

J'espère que les lecteurs éventuels trouveront ici une introduction satisfaisante à certains aspects qui deviennent de plus en plus importants dans le monde de l'entreprise, monde vers lequel se destine la majorité des étudiants en sciences informatiques.

Bonne lecture !

1 Introduction

Le monde de l'entreprise se retrouve de plus en plus souvent confronté au problème d'intégration et d'interconnexion de ses applications [4]. En effet, on retrouve souvent au sein d'une entreprise une multitude de logiciels qui n'ont pas à la base été conçus avec un objectif d'intégration à long terme avec des applications futures, ne serait-ce que parce que les outils de développement évoluent avec le temps. Ceci pose un problème fondamental puisqu'il arrive tôt ou tard un moment où l'entreprise se retrouve mise à mal à cause de son infrastructure mal intégrée parce qu'il est impossible de la faire évoluer.

Pour illustrer ce fait, je prendrai l'exemple de l'ULB. Elle a été poussée à lancer le projet SMILEY essentiellement parce qu'il était devenu impossible de gérer tous les systèmes disparates se trouvant au niveau central et dans les Facultés, voire dans des entités plus petites encore. Plusieurs bases de données stockant des informations sur les étudiants et les programmes de cours existaient, souvent en plusieurs copies ; on se retrouvait ainsi trop souvent avec des informations incohérentes de part et d'autre. Par ailleurs, il devenait virtuellement impossible de greffer de nouvelles fonctionnalités aux systèmes existants. Bref, l'administration n'était plus en état de fonctionner correctement, et encore moins d'améliorer ses processus.

1.1 Architecture orientée services

Pour tenter de trouver une solution à ce problème central d'intégration des applications, une idée fondamentale a été avancée : l'encapsulation de celles-ci sous forme de *services*. Fondamentalement, un service est une entité logicielle *réutilisable* et *découplée le plus possible* de son implémentation, ce qui n'est bien sûr pas sans rappeler le credo de la programmation orientée objet.

On introduit alors la notion *d'architecture orientée services* ou SOA pour *service-oriented architecture* qui est simplement un environnement fournissant des outils adéquats pour *décrire*, *rechercher*, et *exploiter* des services [6].

Il est important de noter que, tout comme en programmation orientée objet, un même service peut disposer de plusieurs implémentations possibles. Une des forces des SOA en théorie est le fait que les applications ne doivent s'intéresser qu'à la *nature* (en d'autres termes, la description) du service dont elles ont besoin, et ne doivent pas *a priori* choisir une implémentation à utiliser. Au moment de l'exécution, l'application peut choisir *dynamiquement* une implémentation à utiliser selon ce qui est disponible.

Par exemple, on pourrait imaginer qu'une application ait besoin d'un service de débit de carte de crédit. On peut envisager que plusieurs entreprises offrent ce service ; l'application peut alors à l'exécution faire un choix de quel service utiliser sans intervention manuelle.

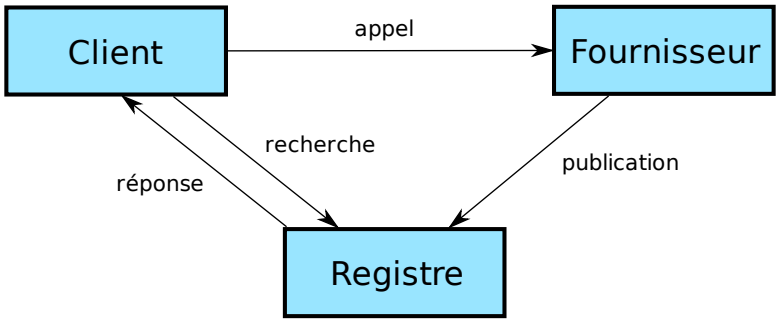

FIG. 1 – Composants d'une architecture orientée services [6]

La figure 1 montre les trois composants fondamentaux d'une SOA et leurs interactions mutuelles. Un *fournisseur de services* publie dans un *registre de services* ceux qu'il propose. Quand une application cliente veut utiliser un service (elle sait bien sûr de quel type de service elle a besoin), elle fait appel au registre qui lui renvoie les implémentations disponibles du type de service dont elle a besoin, et elle peut alors effectivement faire appel à un de ceux-ci.

Il est clair que pour qu'une SOA fonctionne, il est nécessaire de rendre les services interopérables et donc d'avoir des standards qui gouvernent la manière d'effectuer toutes les actions reprises dans la figure 1.

1.2 Web services

L'implémentation actuelle des SOA est ce qu'on appelle les *Web services* [6]. En d'autres termes, il existe une série de normes qui ont été définies pour mettre en place des SOA. Le nom *Web* provient simplement du fait que l'infrastructure du Web est utilisée (TCP/IP et HTTP notamment).

Trois grands standards sont utilisés dans les Web services :

- SOAP (signifiait à l'origine *Simple Object Access Protocol* mais l'acronyme a été abandonné depuis la mouture 1.2) est un protocole d'échange de messages XML via HTTP permettant d'invoquer un service et de recevoir le résultat de son travail.
- WSDL (*Web Service Description Language*) est un langage XML permettant de décrire les services. Un document WSDL est en réalité à la fois un contrat *logique* (dans le sens où il décrit essentielle-

ment l'interface du service ; le format de ce qu'il faut lui fournir et de ce qu'il renvoie) mais aussi *physique* (des informations plus pratiques liées à l'implémentation, telle qu'à quelle adresse le trouver) [6].
– UDDI (*Universal Description, Discovery and Integration* permettant de mettre en place des registres de services.

Pour qu'une application cliente puisse utiliser un Web service, il ne lui faut que deux choses : la description WSDL du service (qui peut être obtenue dans un registre ou que le développeur peut aller chercher directement chez le fournisseur) et la capacité d'utiliser le protocole SOAP. Le client construit alors d'abord un message SOAP (appelé *enveloppe*) avec les informations dont a besoin le service (le format XML précis du message est stipulé dans la description WSDL) et l'envoie via SOAP. Le fournisseur du service effectue ensuite le traitement et renvoie une enveloppe à son tour au client selon un format spécifique, toujours défini via WSDL.

1.3 Composition de services

On peut les composer des services entre eux pour ensuite exposer ce « méta-service » sous forme d'un nouveau service, et ainsi de suite récursivement. On peut ainsi voir émerger une hiérarchie de services arbitrairement complexe [6].

On distingue deux paradigmes dans la composition de services [2]. D'une part, *l'orchestration* qui consiste à avoir un service qui joue le rôle de « chef d'orchestre » et qui connaît seul la logique de composition ; les services auquel il fait appel n'ont pas besoin de savoir qu'ils font partie d'un processus plus gros. D'autre part, la *chorégraphie* dispense de ce rôle de chef d'orchestre. Dans ce paradigme, chaque service doit être au courant de la logique du (ou des) processus auquel il appartient, et doit par exemple savoir que « *quand je reçois un message du service un tel, je dois attendre 50 secondes puis envoyer un message à tel autre* ». Pour des raisons évidentes, la chorégraphie de services n'est pas la plus facile à mettre en œuvre.

2 Business Process Execution Language

BPEL est le langage standard en matière de composition de Web services ; c'est un langage XML. Il est né de la fusion entre deux langages développés indépendamment dans ce but : WSFL (*Web Service Flow Language*) développé par IBM et étant une extension de FL, leur langage de modélisation de processus ; et *XLANG* de Microsoft qui était une extension de WSDL.

En 2003, leur collaboration a mené à la publication de la norme BPEL4WS 1.1 (*BPEL for Web Services*) ; celle-ci a ensuite été soumise au consortium OASIS (responsable notamment de la norme OpenDocument poussée par OpenOffice.org), et ce nouveau travail a abouti en 2007 à la norme WS-BPEL 2.0 qui apporte quelques nouveautés. Le terme « BPEL » désigne de manière globale ces deux langages. Dans le cadre de ce projet, suite au choix de l'outil utilisé pour illustrer les concepts, seul BPEL4WS a été examiné.

BPEL a été conçu dans le seul but de composer des services, et n'est donc pas préconisé pour construire des services élémentaires « de travail ». En effet, le langage dispose de très peu de moyens dans ce domaine, on est essentiellement limité aux fonctions fournies par XPath 1.0 avec quelques extensions [5].

Plus fondamentalement, BPEL permet aux entreprises de modéliser et ainsi *formaliser* leurs processus de fonctionnement, ce qui permet de les optimiser [2].

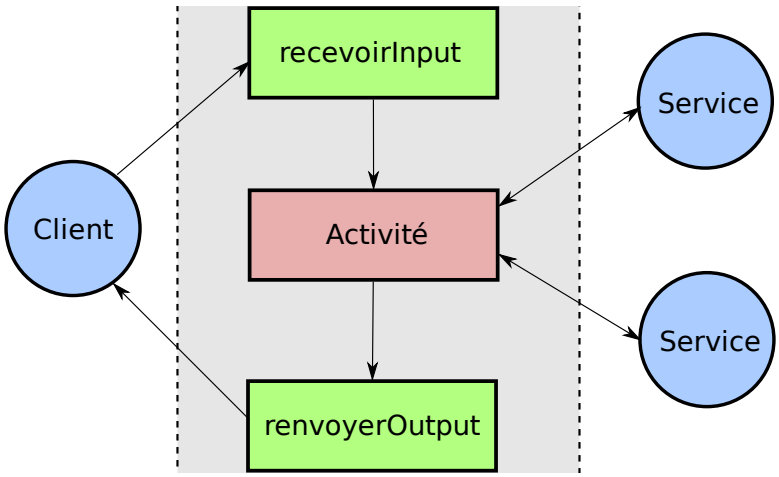

FIG. 2 – Structure d'un processus BPEL synchrone

2.1 Anatomie d'un processus BPEL

BPEL permet de concevoir des processus *synchrones* ou *asynchrones*. Quand un client se connecte à un service synchrone, il attend patiemment que ce dernier se termine avant de continuer son travail (c'est donc essentiellement un appel de fonction comme on l'entend d'habitude). Souvent toutefois, un service peut prendre beaucoup de temps pour se terminer, et il peut donc être utile d'effectuer des appels asynchrones. Le client demande alors au service d'effectuer un certain travail, mais n'attend pas que celui-ci soit effectué. Le service préviendra lui-même le client (un procédé appelé *callback*) quand il a terminé.

La figure 2 schématise le fonctionnement d'un processus BPEL synchrone. Le processus en tant que tel est représenté dans la zone grisée. Le client initie le processus (ce qui cause une *instanciation* de ce dernier), et l'instance récupère l'input fourni. Il effectue ensuite son traitement qui consistera essentiellement à invoquer d'autres services à son tour. Finalement, l'instance du processus renvoie l'output au client.

Un processus BPEL est constitué *d'activités* mises bout-à-bout par des structures de *contrôle de flux*. Les activités possibles [1] pour un processus sont les suivantes :

- *Invoke* permet d'appeler un service ;
- *Receive* permet de recevoir un message (utilisé notamment pour initier le processus) ;
- *Reply* permet de renvoyer un message ;
- *Throw* permet d'émettre une exception au même titre que dans d'autres langages de programmation. En effet, WSDL permet également de décrire des messages d'erreur qui peuvent transiter, et BPEL permet d'exploiter cela facilement (il y a des constructions de type *catch*).
- *Terminate* permet de terminer le processus ;
- *Wait* permet de suspendre le processus pendant une durée déterminée ;
- *Assign* permet d'effectivement manipuler des données, comme par exemple copier des valeurs entre variables ou utiliser des expressions XPath 1.0 pour effectuer des traitements comme manipuler des strings, etc.

Les structures de contrôle de flux BPEL [1] sont les suivantes :

- *Sequence* est la structure la plus simple, consistant simplement à exécuter une activité séquentiellement après une autre ;
- *Switch* est une structure de branchement qui se comporte à peu près comme celle rencontrée en C ; les conditions booléennes sont écrites sous forme d'expressions XPath 1.0. Si les conditions booléennes de plusieurs branches sont vraies, BPEL n'exécutera que la branche correspondant à la première de celles-ci.

– *While* est une structure de boucle ;
– *Pick* permet au processus d'attendre simultanément plusieurs types de messages, ainsi que de fixer un délai maximum d'attente, et d'effectuer un traitement différent selon le cas ;
– *Flow* permet de paralléliser des activités et se comporte comme les *fork* et *join* qu'on retrouve dans les diagrammes d'activités UML.

2.1.1 Interactions homme-processus

Il est possible de faire interagir des humains dans un processus BPEL. Ceci n'est pas une fonctionnalité offerte en tant que telle par le langage, mais Oracle permet notamment ceci en offrant un service prédéfini *TaskManager* qui permet à un processus de créer des tâches humaines. Une API Java existe pour se connecter au service et pour facilement créer des pages Web JSP permettant d'afficher et exécuter des tâches. Fondamentalement, le processus attend simplement qu'un humain lui envoie un message XML particulier et reprend une fois celui-ci arrivé ; *TaskManager* permet donc en quelque sorte d'abstraire l'humain sous forme de service.

2.2 Conception et déploiement d'un processus BPEL sur Oracle

Comme un processus BPEL n'est rien de plus qu'un document XML, on peut tout à fait l'écrire soi-même, mais c'est très fastidieux. En pratique, on utilisera un environnement graphique pour dessiner le processus et le laisser se charger de créer le fichier BPEL correspondant. De plus, il faut également une plate-forme sur laquelle déployer et exécuter le processus.

2.2.1 Oracle BPEL Process Manager

La plate-forme utilisée pour le déploiement et l'exécution des processus était *Oracle BPEL Process Manager*, fourni notamment dans *Oracle SOA Suite* (la version 10.1.3.4.0 a été utilisée lors de ce projet). À la base de la SOA Suite, il y a *Oracle Application Server* qui est un serveur d'applications Java[1]. Il est à noter que BPEL Process Manager peut fonctionner sur d'autres serveurs d'applications, dont WebSphere et JBoss.

BPEL Process Manager offre une boîte à outils considérable pour tester les processus construits en permettant notamment pour une instance donnée voir visuellement la séquence activités qu'elle a faite et les messages qu'elle a échangé.

Il y aussi une base de données sous-jacente qui permet de *déshydrater* des instances. Quand une instance effectue un appel asynchrone à un autre service, le Process Manager va en fait archiver l'instance dans une base de données prévue à cet effet ; l'instance ne sera récupérée (*réhydratée*) que lorsque le service appelé effectue son *callback* [3]. Ceci permet notamment de libérer de la mémoire consommée par des instances alors qu'elles attendent parfois longuement la réponse d'un service, mais cela permet aussi de se prémunir contre des pertes de données. En effet, si l'Application Server ayant déshydraté l'instance souffre d'une panne, un autre Server peut recevoir le message de callback et réhydrater l'instance. Ainsi, le client final ne prend même pas conscience de la panne et reçoit le résultat escompté.

BPEL Process Manager expose le processus BPEL sous forme d'un Web service, et sert donc aussi à communiquer avec des clients via le protocole SOAP.

[1] En découvrant cette infrastructure, j'ai enfin compris quel intérêt Oracle avait en achetant récemment Sun Microsystems.

2.2.2 Oracle JDeveloper

JDeveloper est l'environnement de développement attitré d'Oracle qui dispose notamment d'un éditeur graphique pour des processus BPEL (la version 10.1.3.4.0 a été utilisée dans le cadre de ce projet). Au-delà du fait qu'il génère automatiquement le code BPEL correspondant, il génère également le fichier WSDL correspondant au processus. Un éditeur graphique WSDL et XML Schema existe également ce qui permet d'assez facilement définir l'interface du processus. BPEL4WS 1.1 est supporté, avec certaines extensions tirées d'un brouillon de la spécification WS-BPEL 2.0 [4]. Enfin, JDeveloper livre une petite collection de macros BPEL pour effectuer certaines tâches, comme envoyer des e-mails ou interagir avec des humains.

En outre, JDeveloper fournit une famille *d'adaptateurs* qui permettent d'encapsuler des ressources physiques (fichiers, bases de données, serveurs FTP, etc.) sous forme de services. Pour accéder à une base de données par exemple, il faut dire quel genre d'opération on souhaite faire (par exemple `SELECT * FROM Table`), et JDeveloper construit alors automatiquement les types de messages (la sortie sera typiquement une collection d'éléments de la base de données) et la description WSDL de l'adaptateur. Le processus BPEL peut dès lors s'y greffer facilement.

2.3 Exemples de processus BPEL

2.3.1 Exemple implémenté et illustré lors de l'exposé

Puisque BPEL n'est, comme dit précédemment, destiné qu'à l'orchestration de services, il était assez difficile de construire un exemple très complexe faute d'avoir des services élémentaires utiles. Le processus d'exemple créé dans le cadre de ce projet est synchrone et prend en entrée deux entiers et renvoie leur somme, leur produit, ainsi qu'une liste de pays puisée dans une base de données Oracle Express Edition.

La figure 3 montre à quoi ressemble le processus dans JDeveloper. La première activité du processus est de recevoir l'input de client ; elle lance ensuite un *flow* composé de trois branches : l'une calcule la somme, la seconde le produit, et la dernière effectue la requête à la base de données (il est important de noter que le service « HR » à droite est un adaptateur, pas la base de données proprement dite). Une fois les trois branches terminées, le *flow* se termine et le processus renvoie l'output au client, et se termine ainsi. Rajouter des activités dans le processus revient à faire du glisser-déposer à partir d'une palette.

Il a été montré lors de l'exposé oral qu'il était possible d'invoquer le processus de diverses manières : premièrement, en ayant recours à l'interface fournie par BPEL Process Manager, ce qui permet de suivre l'évolution de l'instance par des outils visuels. D'autre part, il a également été montré par le biais d'une petite application programmée en PHP et utilisant la librairie NuSOAP qu'il était possible de se connecter assez facilement au service ainsi que d'extraire et afficher les résultats.

2.3.2 Réinscription d'un étudiant

Pour revenir à un contexte universitaire, on peut considérer le processus relatif à la réinscription d'un étudiant au terme d'une année académique. Voici le genre de processus simplifié qu'on pourrait imaginer.

L'étudiant invoque (via une page Web sur le site de l'ULB par exemple) le processus asynchrone de réinscription en lui fournissant son matricule et l'année d'études dans laquelle il souhaite s'inscrire. Le processus peut alors être modélisé par un *flow* qui va exécuter deux tâches en parallèle : d'une part, la vérification des résultats des délibérations (qu'on peut supposer être totalement automatisée) et d'autre part la gestion du paiement du minerval (où il y aura notamment la tâche humaine d'imprimer et d'envoyer le virement par courrier). Une fois ces tâches terminées, le *flow* s'achève et on

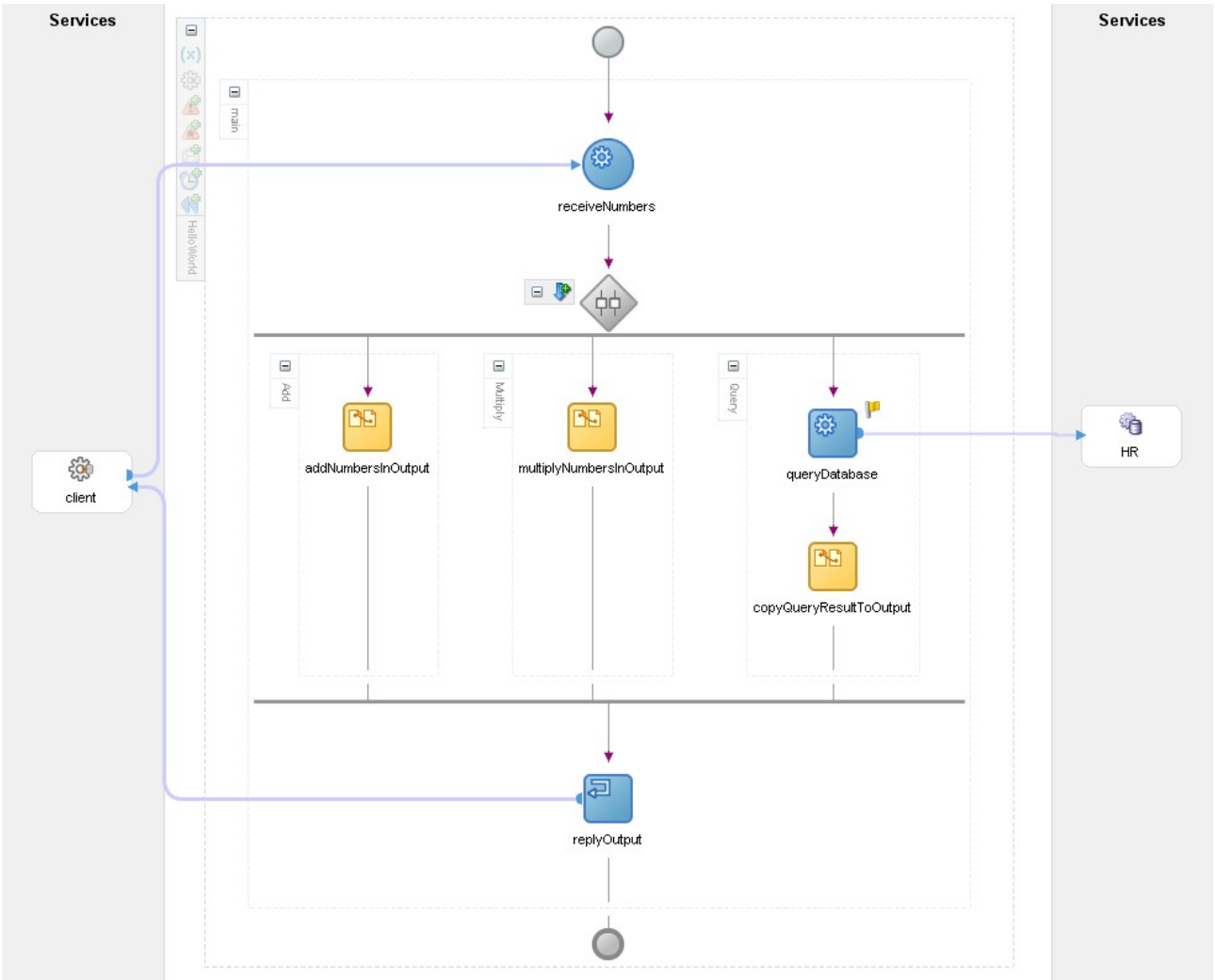

FIG. 3 – Capture d'écran illustrant le processus d'exemple dans JDeveloper.

procède à l'envoi d'un e-mail à l'étudiant pour lui signifier si oui ou non la demande de réinscription a été acceptée et réglée.

3 Conclusion

Les *architectures orientées services* gagnent en popularité dans le monde de l'entreprise suite à leur besoin de pouvoir plus facilement optimiser et faire évoluer leur infrastructure. Les *Web services* sont l'implémentation actuelle de ces architectures. Enfin, *BPEL* a pour rôle central d'orchestrer ces services entre eux et de modéliser les processus employés par l'entreprise, notamment dans le but de les optimiser.

Références

[1] IBM et al. Standard BPEL4WS 1.1.

[2] Matjaz B. Juric. A hands-on introduction to BPEL. Oracle Technology Network.

[3] Oracle. BPEL 101 tutorial.

[4] Oracle. BPEL and Oracle BPEL Process Manager FAQ.

[5] Oracle. BPEL tutorial 3 : manipulating XML documents in BPEL.

[6] Mathias Weske. *Business Process Management : concepts, languages, architectures*. Springer, 2007.