République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université de Djilali BOUNAÂMA Khemis Miliana

Faculté des Sciences et de la Technologie

Département des Mathématiques Informatique

Mémoire Présenté

Pour l'obtention de diplôme

Master En Informatique

Spécialité : « Ingénierie De Logiciel »

Titre :

Une approche dirigée par les modèles basée sur UML pour la mise en œuvre de services web composés

Réalisé par :

Kirli Asma.

Bouhini Meryem.

Encadré par :

Dr. Hachichi Hiba.

Membre du jury :

Mr. Bahloul. D.

Mr.Harbouch.O.

Mr. Khalfi. A.

Année Universitaire 2016/2017

SOMMAIRE

Liste Des Figures

Listes des Figures

Remerciements

Nous remercions le grand dieu le tout puissant, de nous avoir donné le bon sens et la grande volonté pour réaliser ce travail.

Nous remercions du fonds du cœur, notre encadreur : Mme. Hachichi Hiba pour nous avoir aidés et guidés en toute sincérité tout au long de notre préparation du projet.

Nous remercions tous nos professeurs de l'université pour leur contribution a notre formation scientifique.

Nous remercions tous ceux qui nous ont aidés, de près ou de loin, à réaliser ce travail.

A Tous Merci.

Dédicace

A celle qui m'a donné la vie, symbole de tendresse, qui d'est sacrifiée pour mon bonheur et ma réussite, à ma mère.

A mon père, école de mon enfance, qui a été mon ombre durant toutes les années des études, et qui a veillé tout au long de ma vie à m'encourager, à m'aider et à me protéger.

A ma sœur et à mes frères en témoignage de mon attachement affectif.

A ma grande et chaleureuse famille.

A mon binôme : Meryem, sans toi je ne serai pas arrivé ici.
A tous mes ami(e)s et collègues.
A toute la promotion d'informatique 2016-2017.

Asma.

Dédicace

À mon Père,

"L'épaule solide, l'œil attentif compréhensif et la personne la plus digne de mon estime et de mon respect. Aucune dédicace ne saurait exprimer mes sentiments, que Dieu te Préserve et te procure santé et longue vie. "

À ma Mère,

"Tu m'as donné la vie, la tendresse et le courage pour réussir. Tout ce que je peux t'offrir ne pourra exprimer l'amour et la reconnaissance que je te porte. En témoignage, je t'offre ce modeste travail pour te remercier pour tes sacrifices et pour l'affection dont tu m'as toujours entourée."

À mes très chers frères Mohamed, Hocine et le petit Rafik.

À mes très chères sœurs Fatma Zohra, Hayet et la petite Razika.

À toute ma famille BOUHINI et AIT CHIKHOUNE.

A mon binôme : Asma, qui m'encourage souvent ainsi que toute sa famille.

À mes amies : Asmaa et sa petite fille meriouma, Amel, Soumia, Razika, Meriem.CH, Zoula et Habiba.

A toute la promotion d'informatique 2016-2017.

Meryem.

مـلـخـص

مـلـخـص

الهدف من هذا البحث هو اقتراح منهاج لنمذجة لغات تكوين خدمات الويب ولغة BPEL4SWعلى وجه الخصوص. هذه اللغة هي واحدة من المعايير الأكثر شعبية لتنسيق خدمات الويب. ويتحقق هذا المنهاج عن طريق تحويل مخططات الأنشطة الخاصة بلغة النمذجة الموحدة إلى BPEL استنادا على تحويل منحنيات محققة بالأداة ATOM³. منهاجنا يتضمن اقتراح نموذج الفوقية لمخططات النشاط بالإضافة إلى قواعد للغة.

الكلمات الدليلية: مخطط النشاط، BPEL، تحويل المنحنيات، ATOM³، خدمة ويب، تكوين خدمة ويب، تنسيق.

Résumé

Résumé

L'objectif visé dans ce mémoire est de proposer une approche pour la modélisation de langages de compositions de Services Web et du langage BPEL4SW en particulier. Ce langage est l'un des standards les plus populaires pour les orchestrations de Services Web. Cet approche est concrétiser par la transformation des diagrammes d'activité d'UML vers BPEL, basée sur la transformation de graphes, et réalisée à l'aide de l'outil ATOM³. Notre approche consiste à proposer un méta-modèle des diagrammes d'activité et une grammaire de graphes.

Mots clés : Diagrammes d'activité d'UML, BPEL, Transformation de graphes, ATOM³, Composition des services web, Orchestration.

Abstract

Abstract

The aim of this paper is to propose an approach for modeling a Web services composition languages and the BPEL4SW language in particular. This language is one of the most popular standards for web services orchestrations. This approach is realized by transforming the activity diagrams of UML to BPEL, based on the transformation of graphs, and realized with the tool ATOM³. Our approach consists in proposing a meta-model of the activity diagrams and a grammar.

keywords : Activity diagram, BPEL, graph transformation, ATOM³, web service, web service composition, Orchestration.

INTRODUCTION

INTRODUCTION

De nos jours, les systèmes informatiques sont devenus de plus en plus complexes et hétérogènes avec un marché toujours plus exigeant et évolué en matière de performance, d'efficacité, d'innovation, de compétitivité et de productivité. Pour survivre dans un tel contexte, les systèmes informatiques se doivent d'être évolutifs et adaptables. L'architecture orientée services permet d'être une solution très efficace pour ces systèmes, en améliorant leur qualité et en simplifiant leur intégration dans l'infrastructure informatique de l'entreprise, via l'utilisation de composants réutilisables et distribués. Ces composants, basés sur l'infrastructure d'internet, sont appelé services Web. Un service Web est une technologie qui permet aux entreprises de faire communiquer leurs systèmes d'informations avec ceux de leurs clients ou partenaires, via internet. Cette technologie basée sur le standard XML est totalement indépendante du matériel, ou du langage de programmation utilisé ce qui permet de facilement la mettre en œuvre. D'ailleurs, le nombre de services Web utilisés par les entreprises est en constante évolution.

Les services Web individuels restant limités par leurs capacités, l'entreprise est amenée à composer un ensemble de services afin de créer des services plus complexes. On parle ainsi de la composition des services Web. Cette composition se présente comme un paradigme fondamental de la technologie des services Web. Elle permet de résoudre des problèmes complexes en combinant des services de base disponibles pour satisfaire un but initial. Ce paradigme demeure l'un des axes de recherche les plus actifs des services Web, durant ces dix dernières années, vue la complexité du processus de composition et l'évolution rapide des normes et des standards de cette technologie.

UML est un langage qui permet de modéliser non seulement des applications informatiques ou des structures de données, mais également les activités d'un domaine : mécanique, biologie, processus métier. Plus précisément, UML permet d'offrir des outils d'analyse, de conception et d'implémentation des systèmes logiciels, ainsi que pour la modélisation d'entreprise et des systèmes non logiciels. Un diagramme d'activité est une variante des diagrammes d'états-transitions. Il permet de représenter graphiquement le comportement d'une méthode ou le déroulement d'un cas d'utilisation.

La technologie MDA (Model Driven Architecture) est un standard issu de l'OMG qui permet d'exploiter et de transformer les modèles UML. Ceci permet notamment de :
- Produire d'autres modèles.

- Générer automatiquement du code à partir du modèle.

Une fonctionnalité essentielle de la MDA est la notion de transformation. Une transformation regroupe un ensemble de règles et de techniques pour transformer un modèle vers un autre. Plusieurs outils de modélisation ont été proposés pour effectuer la transformation des modèles soit transformation modèle vers modèle ou modèle vers code (génération du code), par exemple AGG, AToM3, VIATRA2, et VMTS.

ATOM3 supporte la métamodélisation et les transformations des modèles en se basant sur une grammaire de transformation. Cet outil crée le méta-modèle et édite leur apparence visuelle, puis il génère l'outil de modélisation et il permet également de spécifier les règles de transformation (Une règle de transformation : est une description de comment un concept ou plus dans un langage source peut être transformé en un concept ou plus d'un langage cible).
En plus ATOM3 assume plusieurs types de transformation : La transformation endogène, la transformation exogène, la transformation PIM vers PIM, la transformation PIM vers PSM, la transformation PIM vers PSM, les transformations inverses PIM vers PSM et PSM vers code. La transformation explorée est la transformation PSM vers code (La transformation de PSM vers l'implémentation (le code)) est une transformation de type modèle à texte.

Au cours de notre travail, nous nous sommes basés sur la thèse de Christophe DUMEZ [30] qui est intitulé : Approche dirigée par les modèles pour la spécification, la vérification formelle et la mise en œuvre de services Web composés. En utilisant le langage C comme langage de programmation, il a créé un logiciel qui permet d'importer des fichiers WSDL (Web Service Description Language) et de les transformer en un diagramme de classe via un ensemble de règles définit. Puis, pour modéliser l'aspect de composition, il ajoute manuellement une nouvelle classe qui décrit le nouveau service composite qui va gérer les différentes interactions entre les services web présents. Ce nouveau service va bien évidemment contenir une méthode, un diagramme d'activité lui va être générer (le diagramme d'activité ne va pas être générer complétement par le logiciel le scénario de composition va être définit par le développeur lui-même). Une fois le scénario de composition prêt, un fichier BPEL va

être générer automatiquement à partir de ce dernier en suivant des règles bien définit. Ainsi, l'approche que nous proposons, vise à transformer les diagrammes d'activité vers le langage d'orchestration BPEL pour modéliser une composition de plusieurs services web. Dans cette approche, ATOM3 est utilisé pour créer un méta-modèle pour le diagramme d'activité, éditer les apparences visuelles des constituants du méta-modèle (les classes et les associations), générer automatiquement l'outil de modélisation des diagrammes d'activité, spécifier les règles de la grammaire de transformation puis exécuter la grammaire qui génère le code BPEL

Plan de travail

Ce mémoire est divisé en quatre chapitres :

- **Le chapitre I** : présente les services web, leur fonctionnement, leur caractéristique ainsi que les technologies qui leur sont liés. Aborde le problème de composition de services ains que son objectif et ses types. Présente un des les langages d'orchestration, BPEL : qui est un langage exécutable standardisé par l'OASIS et basé sur XML.
- **Le chapitre II** : présente le domaine d'ingénierie des modèles et son approche MDA ainsi que son processus de transformation de modèles, une classification des approches de transformation de modèles, ainsi qu'une introduction à la transformation de graphes et une brève présentation de ses approches. Présente AToM³, l'outil visuel de modélisation et de métamodélisation multi-formalismes, utilisé dans l'implémentation de notre travail.

- **Le chapitre III** : présente l'UML ainsi que ses différents diagrammes, détaille le diagramme d'activité puisqu'il constitue la base de notre transformation au fichier Bpel qui définit comment vas se produire l'orchestration de plusieurs services web pour un nouveau service dit composite. Définit c'est quoi un profil UML et introduit le profil UML-S proposé dans la thèse qui constitue notre référence au cours de notre travail ainsi que son rôle dans le cycle de développement de d'développement.

- **Le chapitre IV :** propose une approche de transformation de diagrammes d'activité vers BPEL en exploitant l'outil de méta-modélisation ATOM3.

CHAPITRE 1 :

Services Web

I.1. Introduction

Les services web constituent un phénomène assez récent, il a eu un grand intérêt de la part des développeurs des systèmes d'information. De cela, il est considéré comme étant la solution idéale intégrant des fonctionnalités déjà implémentées afin de construire et développer des systèmes d'information. D'une part, et malgré le succès de cette technologie, les services Web ont confronté le problème de l'interopérabilité, et pour s'en débarrasser, une jungle de standards a été élaborée, ce qui nous invite à aborder les technologies reliées telles qu'eXtensible Markup Language (XML) [1], Simple Object Acces Protocol (SOAP) [2], Web Services Description Language (WSDL) [3] et Universal Description Discovery and Integration (UDDI) [4]. Dans ce chapitre nous allons présenter les services web, leur fonctionnement, leur caractéristique ainsi que les technologies qui leur sont liés. Puis nous aborderont le problème de composition de services.

I.2. Service Web

Le groupe de W3C qui travaille sur les services web, a utilisé dans un document appelé « Web Services Architecture » la définition suivante : « Un Service Web est un système logiciel destiné à supporter l'interaction ordinateur à ordinateur sur le réseau. Il a une interface décrite en un format traitable par l'ordinateur WSDL. Autres systèmes réagissent réciproquement avec le Service Web d'une façon prescrite par sa description en utilisant des messages SOAP, typiquement transmis avec le protocole http (HyperText Transfert Protocol) et une sérialisation XML, en conjonction avec d'autres standards relatifs au web » [5].

I.3 Principes de fonctionnement des Service Web

Le schéma de la Figure 1.1 ci-dessous, concrétise les grandes phases de la mise en œuvre d'un Service Web. Au préambule, le fournisseur réalise un Service Web ainsi que son interface au format WSDL [6].

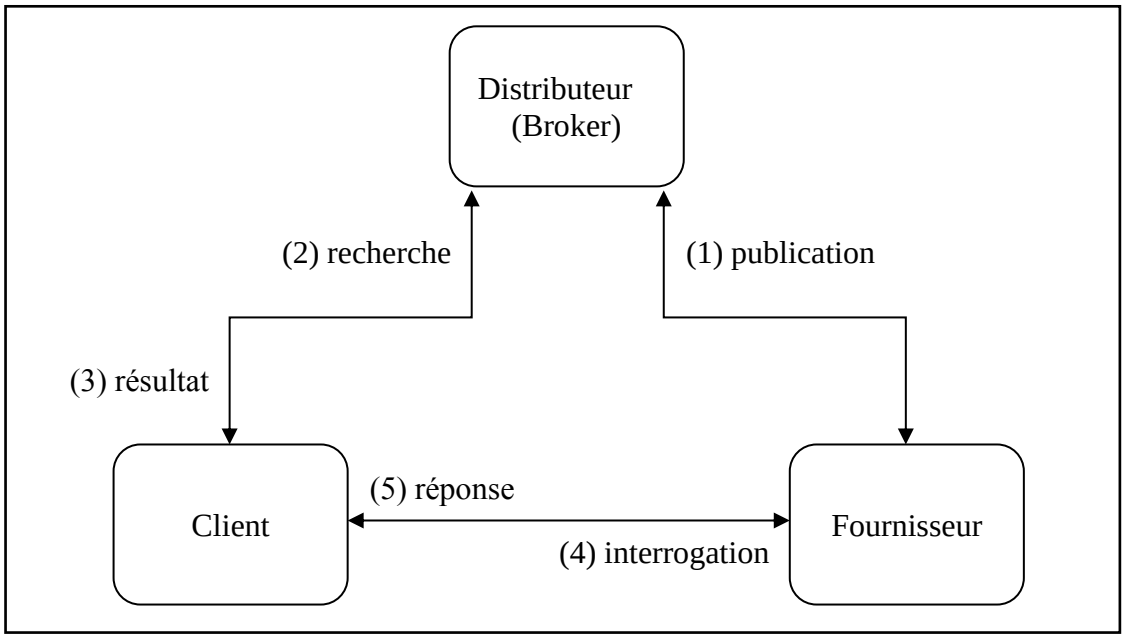

Fig 1.1 : Principe général du fonctionnement des services web.

(Les numéros se réfèrent au Figure ci-dessus) :

- (1) Publication d'un Service Web : le fournisseur commence par enregistrer son Service Web auprès d'un distributeur (sur un annuaire UDDI). Cette opération se fait en envoyant directement un message SOAP via un protocole de transport.
- (2) Recherche d'un Service Web : le client envoie un message encapsulé dans une enveloppe SOAP via un protocole de transport dont le but est la recherche d'un Service Web auprès du distributeur (annuaire UDDI).
- (3) Résultat de la recherche : le client recevra la réponse sous forme d'un message WSDL encapsulé dans une enveloppe.
- (4) Interrogation du service au près du fournisseur de service : Selon le message reçu, le client accèdera directement au Service Web chez le fournisseur. La demande de service s'effectue à l'aide d'un message SOAP via un protocole de transport.
- (5) Réponse et exécution du Service Web : le client reçoit une réponse (via un protocole de transport) du Service Web sous la forme d'un message SOAP. Le client exploitera directement la réponse.

I.4. Caractéristiques d'un service web

I.4.1 Interopérabilité

L'interopérabilité présente la caractéristique principale des Services Web. En effet, quels que soient la plate-forme utilisée (Windows, Unix ou autres) et le langage de développement employé, les Services Web se basent sur des standards XML pour simplifier la construction des systèmes distribués et la coopération entre ces derniers.

I.4.2 Simplicité d'utilisation

L'utilisation des standards tels que XML et HTTP a mis en valeur la simplicité de la manipulation des Services Web et a encouragé les acteurs forts du marché tels que Microsoft et IBM pour intégrer de nouveaux produits.

I.4.3 Couplage souple des applications

Les Services Web constituent un support d'échange des documents structurés qui traversent les contrôles d'accès dans un environnement hétérogène. La collaboration entre différentes applications afin d'échanger des documents se fait d'une manière directe entre objets.

I.5. Technologies liées aux services web

Ce point nous permet de traiter les différents standards liés aux Services Web. Pour les utiliser, il faut savoir où les trouver, comment y accéder et les utiliser correctement. Pour ce faire, une jungle de standards a été proposée illustrés dans la figure ci-dessous(fig1.2) :

Acronyme	Désignation	Fonction dans le contexte des services Web
XML	eXtensible Markup Language	Edition des documents
XML Schema	Schema XML	Définition des types de données lors de la description des services web
WSDL	Web Services Description Language	Description des services web
UDDI	Universal Description, Discovery, and Integration	Publication des services web
SOAP	Simple Object Acces Protocol	Communication entre les services web
OWL-S	Web Ontology Language for Services	Description sémantique des services web
RDF	Ressource Description Framework	Représentation des métas donnés pour les services web
DL	Description Logics	Description formelle des services web

Fig 1.2 : Les technologies liées aux services web.

I.5.1 XML : eXtensible Markup Language

XML « eXtensible Markup Language » est un format de texte simple, très flexible. A l'origine conçue pour la publication électronique à grande échelle, XML joue aussi un rôle de plus en plus important dans l'échange d'une large variété de données sur le web et ailleurs.

W3C recommande depuis 1998 XML en tant que standard de description de données. XML est un méta Language permettant d'identifier la structure d'un document. Un document XML est composé d'une structure et d'un contenu. La structure d'un document XML est souvent représentée graphiquement comme un arbre. La racine du document constitue le sujet du document, et les feuilles sont les éléments de ce sujet. De ce fait, XML est alors flexible et extensible, et est devenue rapidement le standard d'échange de données sur le web. [1]

XML a été conçu pour des documents arbitrairement complexes, tout en s'appuyant sur cinq grands principes simples et clairs :

- Lisibilité à la fois par les machines et par les utilisateurs ;
- Définition sans ambiguïté du contenu d'un document ;
- Définition sans ambiguïté de la structure d'un document ;
- Séparation entre documents et relations entre documents ;
- Séparation entre structure du document et présentation du document.

I.5.2 SOAP/WSDL/UDDI

Le Langage WSDL, le protocole SOAP et l'annuaire UDDI représentent les trois éléments clés pour le développement des services web. De ce fait la figure ci-dessous(fig1.3) montre les interactions directes et indirectes des services web avec ces 3 standards.

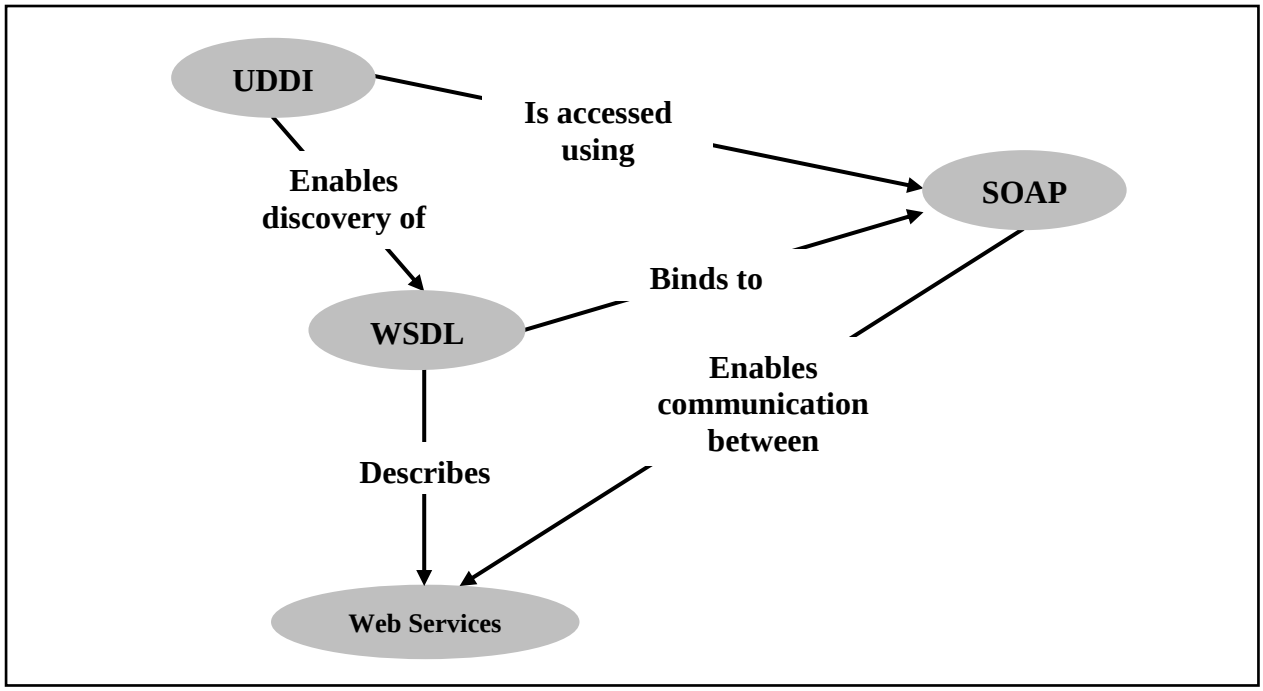

Fig 1.3 : les relations entre WSDL, SOAP, UDDI.

Les relations observées au niveau de la Fig1.3 montrent d'une part que la description des Services Web est assurée via le langage WSDL [3]. D'autre part, la communication entre les Services Web est garantie par le protocole SOAP [2] et les descriptions des services sont liées via SOAP. En effet, l'annuaire UDDI [4] permet la découverte des Services Web via leurs descriptions et l'accès au catalogue UDDI est assuré par le protocole SOAP.

- SOAP : Simple Object Access Protocol : Le World Wide Web Consortium a défini SOAP comme étant : " un protocole léger destiné à l'échange d'informations structurées dans un environnement décentralisé, distribué. Il utilise des technologies XML pour définir une structure d'échange de messages fournissant une construction de messages pouvant être échangés sur divers protocoles sous-jacents. La structure a été conçue pour être indépendante de tout modèle de programmation et autres sémantiques spécifiques d'implémentation. Ce protocole de transmission de messages a été employé pour les services web afin d'offrir aux utilisateurs du web des fonctionnalités pratique et afin de rendre l'échange d'information entre des systèmes hétérogènes une opération très

pratique. La communication entre les services web se fait par l'échange des paquets de données via le protocole SOAP. Un message SOAP n'est en effet pas qu'un fichier XML classique. Le document décrivant un message de ce type est en réalité une déclaration très structurée, appelé enveloppe, et contenant le nécessaire à l'exécution de l'opération qu'elle contienne.

```
<?xml version="1.0" ?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/evelope/">
  <env:Header />
  <env:Body ... >
    [...]
  </env:Body>
</env:Envelope>
```

Fig 1.4 : Format général d'une requête SOAP.

L'enveloppe : C'est l'élément supérieur du document : il englobe entête et corps. Il est obligatoire - sans enveloppe, le message ne peut pas être transporté -, et doit répondre qualifié, c'est-à-dire répondre à l'espace de nom définissant SOAP, comme présenté dans l'exemple ci-dessus

L'entête : Placé au sein de l'enveloppe avant même le corps, l'entête peut être utilisé pour compléter les informations nécessaires à une requête. Le plus souvent, y sont précisés des extensions SOAP (prédéfinies ou propre à l'application), des identifiants de cibles SOAP intermédiaires, ou plus généralement des métadonnées relatives au message.

L'entête reconnaît plusieurs attributs spécifiques :

- Actor, qui indique le destinataire de l'information indiqué par l'entête. Cela permet de viser une application intermédiaire spécifique, via une URL.
- MustUnderstand, qui prend une valeur booléenne et indique si le traitement de l'élément est obligatoire ou non.
- EncodingStyle, qui spécifie les règles d'encodage s'appliquant à l'élément.

Le corps : Cette section contient les données transportées par les messages SOAP. Il doit contenir en envoie, le nom de la méthode appelée, ainsi que les paramètres appliqués à cette méthode. En réponse, il contiendra soit un appel méthode, soit une réponse à sens unique, ou finalement un message d'erreur détaillé.

Comme on peut le voir dans l'exemple ci-dessous (Fig1.5), la première ligne du message SOAP contient une déclaration XML qui n'est pas obligatoire. L'enveloppe SOAP est ensuite déclarée via la balise <SOAP-ENV : Envelope>. Cette dernière est composée d'un corps <SOAP-ENV : Body>. Dans le corps de ce message, il y'a la méthode « GetNombre » et son paramètre qui est égale à 10.

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<SOAP-ENV:Envelope SOAPENV:=""

encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
              xmlns:SOAP-ENV="http:"//schemas.xmlsoap.org/soap/envelope/
              xmlns:SOAP-ENC=http://schemas.xmlsoap.org/soap/encoding/
              xmlns:xsi=http://www.w3.org/1999/XMLSchema-instance
              xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Body>
    <ns1:GetNombre
      xmlns:ns1="urn:MySoapServices">
      <param1 xsi:type="xsd:int">10</param1>
    </ns1:GetNombre>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Fig 1.5 : Exemple de requête SOAP.

Le message de réponse(Fig1.6) a la même structure que celle du message envoyé. Ceci veut dire qu'il contient une déclaration XML, ainsi qu'une enveloppe SOAP composée d'un corps.

```
<?xml version="1.0" encoding="UTF-8" ?>
<SOAP-ENV:Envelope
            xmlns:SOAP-ENV="http:"//schemas.xmlsoap.org/soap/envelope/
            xmlns:xsi=http://www.w3.org/1999/XMLSchema-instance
            xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Body>
      <ns1:GetNombreResponse
        xmlns:ns1="urn:MySoapServices"
        SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
        <return xsi:type="xsd:int">10</return>
      </ns1:GetNombreResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Fig 1.6 : Exemple de requête de réponse SOAP.

Dans le corps du message de réponse, nous pouvons toujours voir notre méthode «GetNombre» : le mot « Response » a été rajouté à la fin du nom de la méthode pour bien préciser qu'il s'agit du retour d'une requête sur la méthode. La valeur renvoyée par la méthode « GetNombre » est égale à 10.

La spécification SOAP se divise en quatre parties :

- L'enveloppe SOAP, qui définit le contexte d'un message, son destinataire, son contenu et différentes options ;
- Les règles de codage SOAP, définissant la représentation des données d'une application dans le corps d'un message SOAP (en particulier leur structure) ;
- Un protocole de RPC définissant la succession des requêtes et des réponses ;
- La définition de l'utilisation de HTTP comme couche de transport des messages SOAP. Les règles de codage utilisent abondamment XML Schema pour décrire la structure des données constitutives des messages SOAP.

- WSDL : Web Service Description Language : Le World Wide Web Consortium a défini WSDL comme étant : " un langage XML pour la description des services sur le réseau à travers une collection de communication entre les terminaux par l'échange de messages. » [3]

  Nous tenons à présenter un exemple de description d'un Service Web "TemperatureService.wsdl" permettant de retourner la température(Fig1.7).

```
<?xml version="1.0"?>
<definitions name="TemperatureService"

targetNamespace="http://www.xmethods.net/sd/TemperatureService.wsdl"

xmlns:tns="http://www.xmethods.net/sd/TemperatureService.wsdl"
              xmlns:xsd="http://www.w3.org/2001/XMLSchema"
              xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
              xmlns="http://schemas.xmlsoap.org/wsdl/">
  <message name="getTempRequest">
    <part name="zipcode" type="xsd:string"/>
  </message>
  <message name="getTempResponse">
    <part name="return" type="xsd:float"/>
  </message>
  <portType name="TemperaturePortType">
    <operation name="getTemp">
      <input message="tns:getTempRequest"/>
      <output message="tns:getTempResponse"/>
    </operation>
  </portType>
  <binding name="TemperatureBinding" type="tns:TemperaturePortType">
    <soap:binding style="rpc"
transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="getTemp">
      <soap:operation soapAction=""/>
      <input>
        <soap:body use="encoded" namespace="urn:xmethods-Temperature"

encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </input>
      <output>
        <soap:body use="encoded" namespace="urn:xmethods-Temperature"

encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </output>
    </operation>
  </binding>
  <service name="TemperatureService">
    <documentation>Returns current temperature in a given U.S. zipcode
</documentation>
    <port name="TemperaturePort" binding="tns:TemperatureBinding">
      <soap:address
location="http://services.xmethods.net:80/soap/servlet/rpcrouter"/>
    </port>
  </service>
</definitions>
```

Fig 1.7 : Exemple d'une description d'un service web.

A travers le code source du fichier WSDL [7] : Présenté par Fig1.7, on distingue que la structure est composée de six éléments essentiels et un septième optionnel, à savoir :

- Définitions : élément racine du document, il inscrit le nom du service, les espaces de noms utilisés, et contient les éléments du service.
- Types : décrit tous les types de données utilisées entre le client et le serveur. WSDL n'est pas exclusivement lié a un système spécifique de typage, mais Schéma [8].
- Message : décrit les données échangées, autrement dit, les paramètres des opérations.
- PortType : définit les messages pour composer une opération. Chaque opération se réfère à un message en entrée et à des messages en sortie.
- Binding : décrit les spécifications concrètes de la manière dont le service sera implémenté. Protocole de communication et format des données
- Service : définit les adresses permettant d'invoquer le service donné, ce qui sert à regrouper un ensemble de ports reliés.
- Documentation : contient la documentation lisible. (Optionnel)

- UDDI : Universal Description, Discovery and Integration : Le Consortium international [4] a défini UDDI comme étant : “ le membre clé du groupe interdépendant des standards qui comporte l’empileur des services web. Il définit une méthode standard pour publier et découvrir au niveau du réseau les composants logiciel d’une architecture orientée service SOA. »

  Le modèle d’un catalogue UDDI est un élément central de l’approche orientée service pour la conception de logiciel. Via la distribution à base de règles et la gestion des services web de l’entreprise, le registre de l’UDDI délivre une valeur significative [9].
  UDDI est un registre public conçu pour héberger des informations sur les entreprises et leurs services de façon structurée. Au travers d'UDDI, il est possible de publier et de découvrir des informations sur une entreprise et ses services Web. Ces données peuvent être classifiées à l'aide de taxinomies standard. Ces informations peuvent donc être recherchées par catégories. Enfin et surtout, UDDI contient des informations sur les interfaces techniques des services d'une entreprise. Grâce à un jeu d'API XML basées sur SOAP, il est possible d'interagir avec UDDI au moment de la conception et de l'exécution pour découvrir des données techniques, de sorte que ces services peuvent être invoqués et utilisés. De cette façon, UDDI sert d'infrastructure à un paysage logiciel reposant sur des services Web [10].

UDDI fonctionne à partir de pages :

• Blanches : informations sur les fournisseurs de services (par nom)

• Jaunes : informations sur les fournisseurs de services (par catégorie)

• Vertes (spécificité de UDDI) : définition des services fournis en WSDL.

Les entreprises publient les descriptions de leurs services web en UDDI, sous la forme de fichiers WSDL. Ainsi, les clients peuvent plus facilement rechercher les services web dont ils ont besoin en interrogeant le registre UDDI. Lorsqu'un client trouve dans UDDI une description de service web qui lui convient, il télécharge son fichier WSDL depuis le registre UDDI. Ensuite, à partir des informations inscrites dans le fichier WSDL, notamment la référence vers le service web, le client peut invoquer le service web et lui demander d'exécuter certaines de ses fonctionnalités. La Figure ci-dessous (Fig1.8) présente le schéma général du protocole UDDI. L'entreprise B a publié le service web S et l'entreprise A est client de ce service [11].

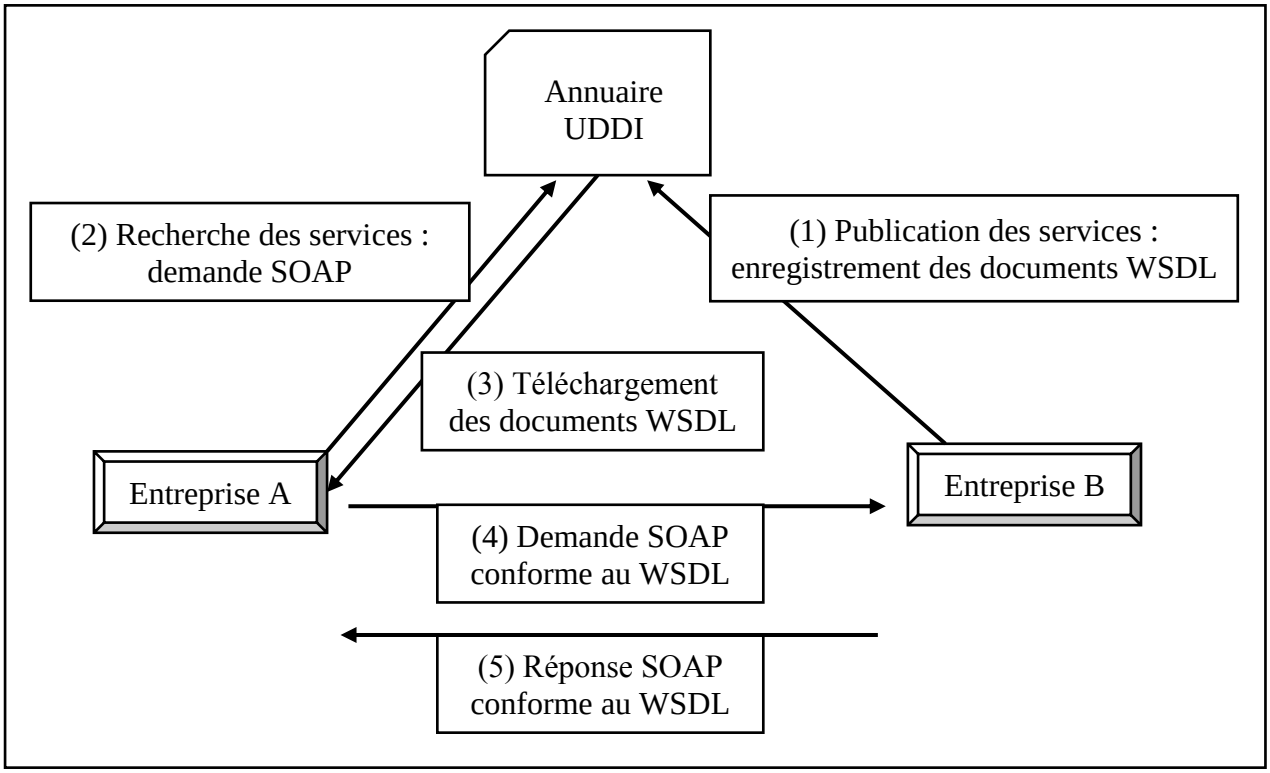

Fig 1.8 : Schéma général de UDDI.

I.6. Composition de services web

Dans l'industrie les termes workflow et document management systèmes sont utilisés pour décrire comment les composants sont connectés pour construire des procédés métiers complexes. Ils décrivent l'enchaînement du travail entre plusieurs logiciels d'une organisation. Ces logiciels peuvent inclure des applications partenaires (legacy systems) des logiciels locaux ou distants construits dans différentes technologies et même des personnes devant réaliser certaines tâches. Avec l'introduction des services web des termes tels que "web services composition" ou "web services flow" ont été utilisés pour décrire la composition de services web dans un flux de procédé. Cela a relancé les travaux de recherche sur les problèmes d'interopérabilité et coordination de services web. La solution proposée par la plupart de ces travaux consiste à modéliser le logiciel par un procédé. Ce procédé gère la coordination des tâches et des 19 opérations de ce logiciel. Ces travaux de recherche autour de la coordination de services web ont donné naissance à deux nouveaux termes : la Chorégraphie et l'Orchestration des services web. Nous allons présenter dans cette section la définition de ces termes. Nous allons surtout nous concentrer sur l'orchestration qui est la base de ce travail de recherche [12].

I.6.1. Définition

La composition de services, c'est à dire la combinaison de plusieurs services pour obtenir de nouvelles fonctionnalités. Les approches pour la composition de service peuvent être classifiées en deux catégories, selon qu'elles soient basées sur l'orchestration ou la chorégraphie.

I.6.2. Chorégraphie

La chorégraphie modélise la séquence des échanges de messages entre services web et conditions dans lesquelles ces messages sont échangés entre des clients, des fournisseurs et des partenaires. La chorégraphie est typiquement associée à l'échange de messages publics entre les services web, alors qu'un procédé métier est exécuté de manière centralisée.

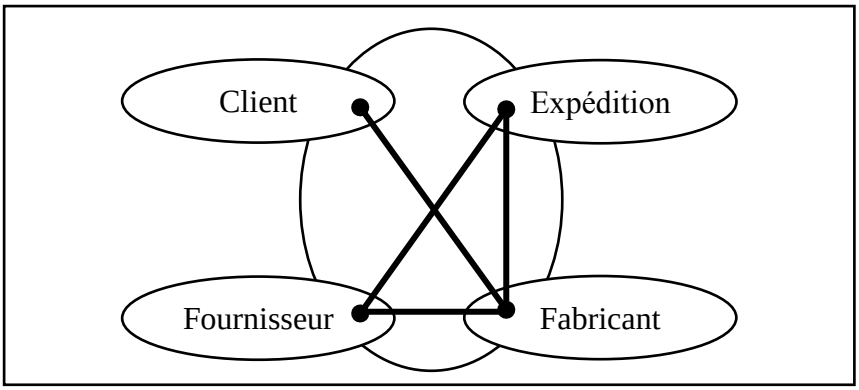

Fig 1.9 : Vue global de la chorégraphie.

La Figure ci-dessus (Fig1.9) présente une vue générale de la chorégraphie. Cet exemple illustre l'échange de messages entre 4 partenaires : Client, Expédition, Fournisseur et Fabricant, ces partenaires sont des services web. Dans les travaux récents sur la chorégraphie, nous trouvons le langage WS-CDL (Web Services Choreography Description Language) qui est un langage basé sur XML pour décrire les collaborations paires à paires entre les participants et l'échange de messages entre ces participants dans l'objectif d'accomplir un but commun dans la logique métier. [12]

I.6.3. Orchestration

L'orchestration de services permet de définir l'enchaînement des services selon un canevas prédéfini, et de les exécuter à travers des "scripts d'orchestration". Ces scripts sont souvent représentés par des procédés métier ou des workflows inter/intra-entreprise. Ils décrivent les interactions entre applications en identifiant les messages et en branchant la logique et les séquences d'invocation [13].

"L'orchestration décrit, du point de vue d'un service, les interactions de celui-ci ainsi que les ´étapes internes (ex. transformations de données, invocations a des modules internes) entre ses interactions." Dans une orchestration, le chef d'orchestre est le service dont les interactions sont dépeintes. Celles-ci comprennent l'envoi et la réception de messages à/de la part de partenaires sélectionnés. L'orchestration permet ainsi a un service d'être enchaîné à d'autres d'une manière prédéfinie. Elle est ensuite exécutée par des scripts d'orchestration qui décrivent les interactions entre les applications en identifiant les messages, la logique et les s´séquences d'invocations. Le composant exécutant les scripts d'orchestration est appelé moteur d'orchestration. Celui-ci agit comme une entité centralisée pour coordonner les interactions entre les services. Un ensemble de commandes centralise la logique du workflow ce qui facilite l'interopérabilité

entre deux ou plusieurs applications différentes. Une mise en œuvre courante de l'orchestration est un modèle qui permet à des participants de l'extérieur d'interagir avec un moteur d'orchestration central [14] comme illustré dans la figure ci-dessous :

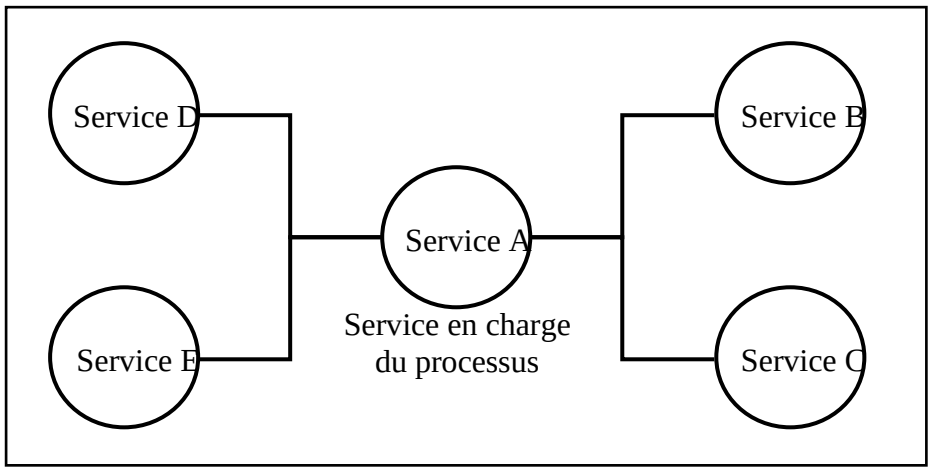

Fig 1.10 : Une orchestration contrôle presque toutes les facettes d'une activité complexe.

L'orchestration décrit la manière dans laquelle les services web peuvent interagir ensemble au niveau des messages, incluant la logique métier et l'ordre d'exécution des interactions. Ces interactions peuvent couvrir des applications et/ou des organisations, et le résultat peut être un modèle de procédé de longue durée, transactionnel, et multi étapes. Une différence importante entre l'orchestration et la chorégraphie est que l'orchestration offre une vision centralisée, c'est-à-dire que le procédé est toujours contrôle de la perspective d'un des partenaires métier. En revanche, la chorégraphie offre une vision globale et plus collaborative de la coordination. Elle décrit le rôle qui joue chaque participant impliqué dans l'application. [11] Une orchestration de Services Web est vue comme un orchestre, où un processus particulier joue le rôle de chef d'orchestre. Celui-ci coordonne l'exécution des autres services. Il s'agit d'un point de vue individualiste : les services, à l'exception de l'orchestrateur, n'ont pas besoin de savoir qu'ils font partie d'une plus grande partie. Il y a un seul document de haut niveau représentant les ´étapes du processus et ce document est seulement connu et traité par l'orchestrateur. Dans une chorégraphie, en revanche, les services sont vus comme des danseurs qui savent exactement quoi faire et de quelle manière interagir avec les autres parties. Il s'agit d'une approche collaborative et chacun des participants a besoin d'un document dans lequel l'interaction est décrite. Ces documents mettent l'accent sur l'échange de message. Un des langages pour représenter l'orchestration est BPEL4WS (Business Process Execution Language For Web Services). [14]

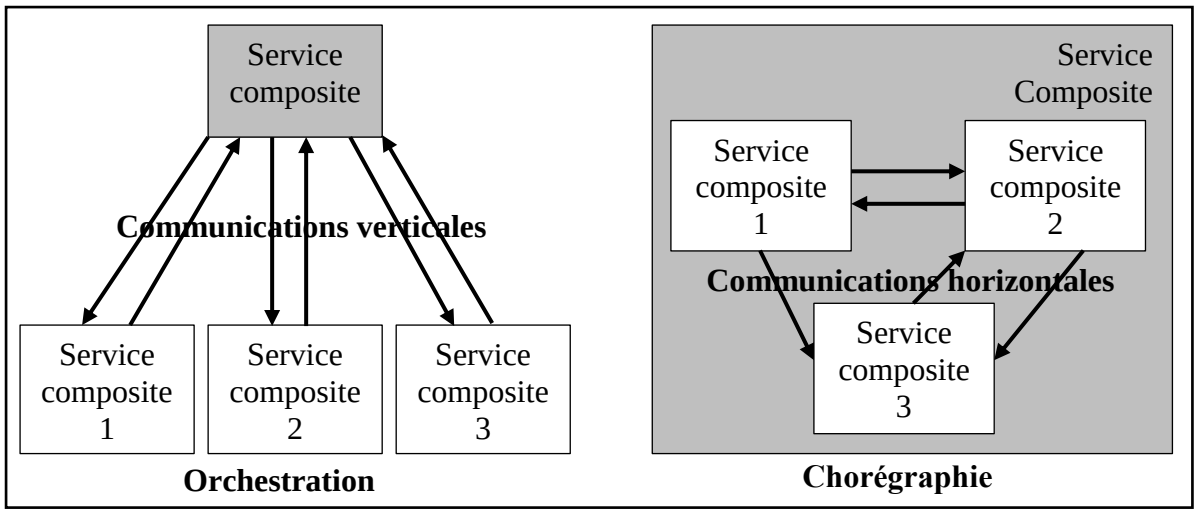

Fig 1.11 : Orchestration VS Chorégraphie de services.

I.7. BPEL

"L'intégration des systèmes exige plus que la capacité de gérer des interactions simples [comme permis par SOAP et WSDL] ... Le plein potentiel des Services Web comme une plate-forme d'intégration ne sera réalisé que lorsque les applications et les processus métiers seront en mesure d'intégrer leurs interactions complexes en utilisant un modèle standardisé." "WS-BPEL définit un modèle et une grammaire pour décrire le comportement d'un processus métier basé sur les interactions entre le processus et ses partenaires. L'interaction avec chaque partenaire se fait par des interfaces de Service Web" (Les deux citations sont tirées de la spécification WS-BPEL, Introduction, p. 8-10, Oasis (2007)).

Le modèle de procédé BPEL forme une couche au-dessus de WSDL. Il définit la coordination des interactions entre l'instance de procédé et ses partenaires. Les procédés dans BPEL exportent et importent les fonctionnalités en utilisant des interfaces de services web uniquement. BPEL contient les caractéristiques d'un langage structuré en blocs (block structured language) du XLANG, ainsi que les caractéristiques d'un graphe direct de WSFL. [13] BPEL permet de modéliser deux types de procédé :

- Le procédé abstrait : spécifie les échanges de messages entre les différentes parties, sans spécifier le comportement interne de chacun d'eux.
- Le procédé exécutable : spécifie l'ordre d'exécution des activités constituant le procédé, des partenaires impliqués dans le procédé, des messages échangés entre ces partenaires, et le traitement de fautes et d'exceptions spécifiant le comportement dans les cas d'erreurs ou d'exceptions.

Le grand avantage de BPEL est la possibilité de décrire les interactions entre les logiques métiers des différentes entreprises à travers services web. Les éléments du procédé BPEL sont : les liens de partenaires, les activités et les données.

- Les liens de partenaires : Un lien de partenaire (partnerLink) correspond au service avec lequel le procédé échange des informations. Le lien de partenaire représente la relation de conversation entre deux procédés partenaires. Chaque lien de partenaire est typé par un partnerLinkType, il est chargé de définir le rôle que joue chacun des deux partenaires dans une conversation. L'attribut myRole ou partnerRole définit si c'est une action qui appelle le processus ou si c'est une action appelée par le processus.

```
<partnerLinks>
  <partnerLink name="PartnerLink1" partnerLinkType="tns:examplePL" myRole="exampleRole"
/>
</partnerLinks>
```

Fig 1.12 : La balise PartnerLinks.

- Les activités : Le procédé dans BPEL est constitué d'activités liées par un flot de contrôle. Ces activités peuvent être basiques ou structurées.

  Les activités basiques sont :

  <invoke>, pour invoquer une opération dans un service web.

```
<invoke name="invoke1" PartnerLink="PartnerLink1" operation="exampleOperation"
portType="examplePortType" inputVariable="varIn" outputVariable="varOut"/>
```

Fig 1.13 : La balise invoke.

<recieve>, pour attendre un message d'une source externe.

```
<receive name="Receive1" createInstance="yes" PartnerLink="PartnerLink1"
operation="exampleOperation" portType="examplePortType" variable="var1In"/>
```

Fig 1.14 : La balise receive.

<reply>, pour répondre à une source externe.

```
<reply name="Reply1" PartnerLink="PartnerLink1" operation="exampleOperation"
portType="examplePortType" variable="var1Out"/>
```

Fig 1.15 : La balise reply.

<wait>, pour attendre un certain temps.

<assign>, pour copier les données d'une place à l'autre.

<throw>, pour lancer une erreur d'exécution.

<terminate>, pour terminer l'instance de service en entier.

<empty>, qui ne fait rien (utile pour la synchronisation des activités parallèles).

Les activités structurées sont composées d'autres activités basiques et structurées. Les types d'activités structurées sont :

<sequence>, pour définir un ordre d'exécution.

```
<sequence name="Main">
  [Actions]
</sequence>
```

Fig 1.16 : La balise sequence.

<switch>, pour l'acheminement conditionnel.

<while>, pour les boucles.

<pick>, pour attendre l'arrivée d'un événement

<flow>, pour l'acheminement parallèle.

<scope>, pour regrouper les activités afin qu'elles soient traitées par le même gestionnaire d'erreur.

<componsate>, pour invoquer les activités de compensation par le gestionnaire d'erreur, pour défaire l'exécution déjà complétée d'un regroupement d'activité.

- Les données : Le procédé dans BPEL a un état, cet état est maintenu par des variables contenant des données. Ces données sont combinées afin de contrôler le comportement du procédé. Elles sont utilisées dans les expressions et les opérations d'affectation. Les expressions permettent d'ajouter des conditions de transition ou de jointure au flot de contrôle. L'affectation (assignement) permet de mettre à jour l'état du procédé, en copiant les données d'une variable à une autre ou en introduisant de nouvelles données en utilisant

les expressions. Dans BPEL il n'y a pas de flot de données, BPEL se sert des variables pour passer une donnée d'une activité à une autre, à l'aide de l'affectation.

```
<variables>
  <variable name="var" messageType="tns:exampleMessage" />
</variables>
```

Fig 1.17 : La balise variables.

I.8. Conclusion

Dans ce chapitre nous avons présenté les services web, leur fonctionnement, leur caractéristique ainsi que les technologies qui leur sont liés. Puis nous avons abordé le problème de composition de services. L'objectif de la composition de services Web est de créer de nouvelles fonctionnalités en combinant les fonctionnalités offertes par des services existants. On distingue deux types de compositions : l'orchestration et la chorégraphie.

De l'orchestration de services résulte un nouveau service dit composé qui peut être défini comme l'agrégation de plusieurs autres services atomiques ou composés. Ce service composé contrôle la collaboration entre les services, tel un chef d'orchestre. Parmi les langages d'orchestration, BPEL : qui est un langage exécutable standardisé par l'OASIS et basé sur XML. La chorégraphie de services est une généralisation de l'orchestration qui consiste à concevoir une coordination décentralisée des applications.

D'autres approches pour la composition de services se basent sur le principe de l'ingénierie dirigée par les modèles (MDE) et se basent sur des modèles pour spécifier la composition et servir de base à l'implémentation. Un état de l'art de ces approches est fourni dans le chapitre 2.

CHAPITRE 2 :

Ingénierie des modèles (IDM)

II.1. Introduction

L'Ingénierie Dirigée par les Modèles (IDM) est une discipline récente du génie logiciel qui promeut les modèles en entités de première classe dans le développement logiciel. Elle fait l'objet d'un grand intérêt aussi bien de la part des équipes de recherche académiques (Action IDM-website) que des laboratoires industriels. Persuadés que le modèle est devenu le paradigme majeur par lequel l'industrie du logiciel pourra lever le verrou de l'automatisation du développement, des organismes tels que l'OMG (Object Management Group) et les chercheurs en génie logiciel, cherchent à enrichir les modèles qui sont utilisés dans la conception d'applications ou à en définir de nouveaux, à faciliter la création de nouveaux espaces technologiques plus adaptés aux besoins des utilisateurs, et à faciliter les différentes étapes de modélisation nécessaires à l'élaboration d'un produit fini. Ainsi, pour obtenir un produit fini qui réponde aux attentes des utilisateurs, il est nécessaire de pouvoir transformer des modèles d'un niveau d'abstraction à un autre ou d'un espace technologique à un autre. Dans ce chapitre, nous allons présenter le domaine d'ingénierie des modèles, une classification des approches de transformation de modèles, ainsi qu'une introduction à la transformation de graphes et une brève présentation de ses approches.

II.2. Ingénierie dirigée des modèles

IDM est une forme d'ingénierie générative par laquelle tout ou partie d'une application informatique est générée à partir de modèles. Dans cette nouvelle perspective, les modèles occupent une place de premier plan parmi les artéfacts de développement des systèmes et doivent en contrepartie être suffisamment précis et riches afin de pouvoir être interprétés ou transformés par des machines. Le processus de développement des systèmes peut alors être vu comme une séquence de transformations de modèles partiellement ordonnée, chaque transformation prenant un ou des modèles en entrée et produisant un ou des modèles en sortie, jusqu'à l'obtention d'artéfacts exécutables. Cette transformation des modèles n'est bien sûr pas une tâche aisée. Il est donc nécessaire de disposer d'outils flexibles pour la gestion des modèles et de langages dédiés pour leurs transformations et leur manipulation tout au long de leur cycle de vie. Pour donner aux modèles cette dimension opérationnelle, il est essentiel de spécifier leur sémantique et donc aussi de décrire de manière précise les langages utilisés pour les représenter. On parle alors de métamodélisation. L'intérêt pour l'IDM a été fortement amplifié, en novembre 2000, lorsque l'OMG a rendu publique son initiative MDA qui vise la définition d'un cadre normatif pour l'IDM [15].

II.3. Standards de l'OMG

L'OMG est un consortium regroupant des industriels, des fournisseurs d'outils et des académiques dont l'objectif principal est d'établir des standards pour résoudre les problèmes d'interopérabilité entre les systèmes d'information. Ces standards sur lesquels repose l'IDM sont centrés sur les notions de métamodèles et de métamétamodèles.

- Meta Object Facility (MOF) : se situe au sommet dans l'architecture à quatre niveaux de l'OMG (voir figure 2.1 ci-après). MOF est un standard de l'Object Management Group s'intéressant à la représentation des métamodèles et leur manipulation. c'est-à-dire un formalisme pour établir des langages de modélisation permettant eux-mêmes d'exprimer des modèles. Dans cette architecture, le monde réel est représenté à la base de la pyramide (niveau M0). Les modèles représentant cette réalité constituent le niveau M1. Les métamodèles permettant la définition de ces modèles (UML, SPEM, etc.) constituent le niveau M2. Enfin, le métamétamodèle (MOF), unique et méta circulaire, est représenté au sommet de la pyramide (niveau M3) [16].

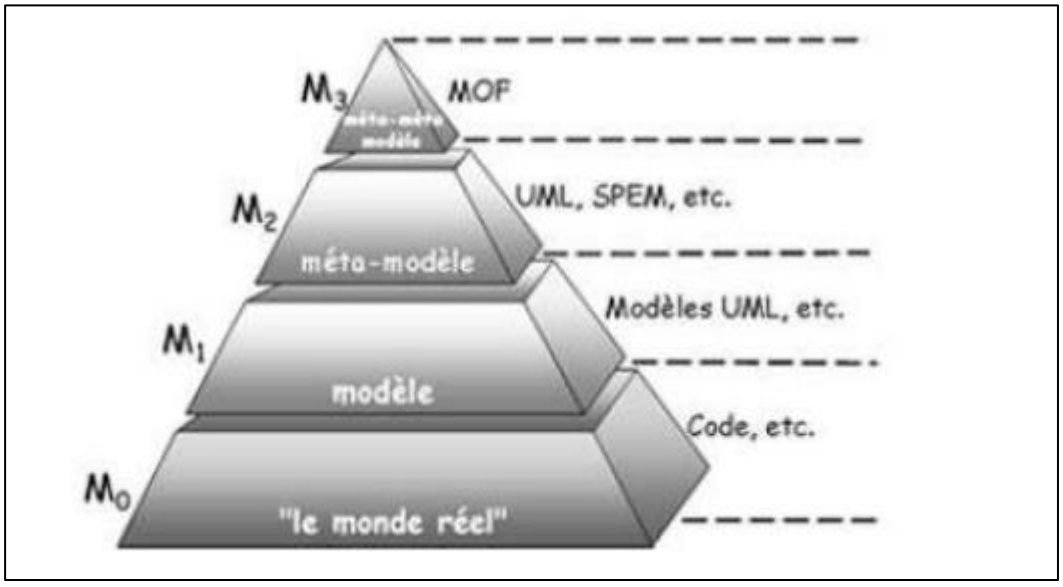

Fig 2.1 : Pyramide de modélisation de l'OMG (Bézivin, 2003).

Unified Modeling Language (UML) [17] : est un langage de modélisation graphique à base de pictogrammes conçu pour fournir une méthode normalisée pour visualiser la conception d'un système. Il est couramment utilisé en développement logiciel et en conception orientée objet. L'UML est le résultat de la fusion de précédents langages de modélisation objet : Booch, OMT, OOSE. Principalement issu des travaux de Grady Booch, James Rumbaugh et Ivar Jacobson, UML est à présent un standard adopté par l'Object

Management Group (OMG). UML est utilisé pour spécifier, visualiser, modifier et construire les documents nécessaires au bon développement d'un logiciel orienté objet. UML offre un standard de modélisation, pour représenter l'architecture logicielle. Les différents éléments représentables sont :

- Activité d'un objet/logiciel
- Acteurs
- Processus
- Schéma de base de données
- Composants logiciels
- Réutilisation de composants

- Object Constraint Lanaguage (OCL) [18] : est un langage informatique d'expression des contraintes utilisé par UML. C'est une contribution d'IBM à UML 1.1. Ce langage formel est volontairement simple d'accès et représente un juste milieu entre langage naturel et langage mathématique. Il permet ainsi de limiter les ambigüités dans la spécification des contraintes logicielles. Sa grammaire simple lui permet d'être interprété par des outils logiciels pour faire de la programmation par contrat et vérifier qu'un logiciel répond à ses spécifications techniques.

  OCL permet de décrire des invariants dans un modèle, sous forme de pseudo-code :

- Pré et post-condition pour une opération.
- Expression de navigation.
- Expression booléenne.
- Etc.

- XML Metadata Interchange (XMI) : est un standard pour l'échange d'informations de métadonnées UML basé sur XML , qui offre une représentation concrète des modèles sous forme de documents XML. Cette représentation concrète des modèles se fait par des mécanismes appelés sérialisation et génération. La génération consiste à transformer un métamodèle en un DTD (Document Type Definition) alors que la sérialisation permet de représenter les modèles sous forme de document XML (voir figure 1.2 ci-dessous) [15].

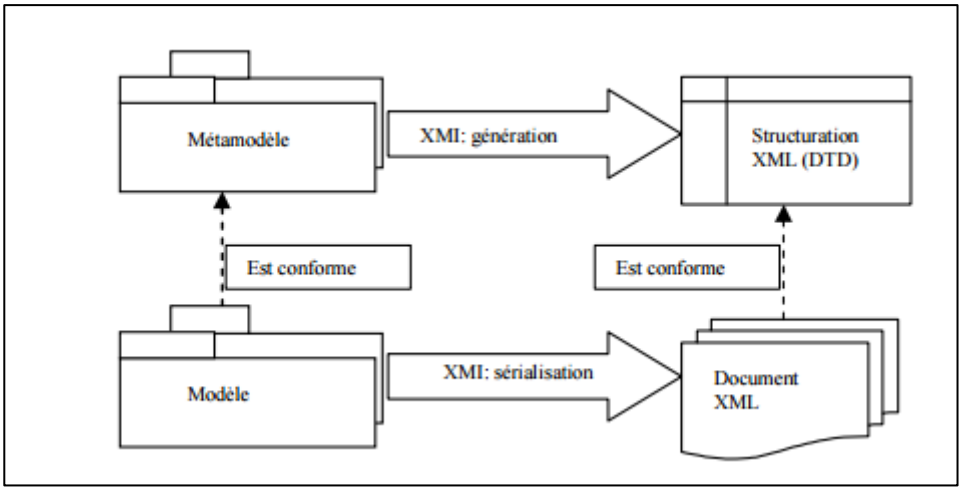

Fig 2.2 : XMI et la structuration de balises XML (BLANC, 2005).

II.4 Architecture dirigée par les modèles

II.4.1 Les modèles

Les modèles offrent de nombreux avantages. L'avantage le plus important qu'ils procurent est de spécifier différents niveaux d'abstraction, facilitant la gestion de la complexité inhérente aux applications. Les modèles très abstraits sont utilisés pour présenter l'architecture générale d'une application ou sa place dans une organisation, tandis que les modèles très concrets permettent de spécifier précisément des protocoles de communication réseau ou des algorithmes de synchronisation. Même si les modèles se situent à des niveaux d'abstraction différents, il est possible d'exprimer des relations de raffinement entre eux. Véritables liens de traçabilité, ces relations sont garantes de la cohérence d'un ensemble de modèles représentant une même application. Partageant des intérêts communs, sans qu'ils soient pour autant entravés par les anciennes Contraintes géographiques et sociales.

L'OMG a défini MDA (Model Driven Architecture) en 2000 dans cet objectif. L'approche MDA préconise l'utilisation massive des modèles et offre de premières réponses aux comment, quand, quoi et pourquoi modéliser. Sans prétendre être une Bible de la modélisation, répertoriant toutes les bonnes pratiques, elle vise à mettre en valeur les qualités intrinsèques des modèles, telles que pérennité, productivité et prise en compte des plateformes d'exécution. MDA inclut la définition de plusieurs standards, notamment UML, MOF et XMI.

Le principe clé de MDA consiste en l'utilisation de modèles aux différentes phases du cycle de développement d'une application. Plus précisément, MDA préconise l'élaboration de modèles d'exigences (CIM), d'analyse et de conception (PIM) et de code (PSM). L'objectif majeur de MDA est l'élaboration de modèles pérennes, indépendants des détails techniques des plates-formes d'exécution (J2EE, .Net, PHP ou autres), afin de permettre la génération automatique de la totalité du code des applications et d'obtenir un gain significatif de productivité [19].

- Modèle CIM- Computational Independant Model : Les modèles d'exigence CIM décrivent les besoins fonctionnels de l'application, aussi bien les services qu'elle offre que les entités avec lesquelles elle interagit. Leur rôle est de décrire l'application indépendamment des détails liés à son implémentation. Les CIM peuvent servir de référence pour s'assurer que l'application finie correspond aux demandes des clients [20].

-Modèle PIM- Platform Independant Model : Les modèles PIM sont les modèles d'analyse et de conception de l'application. La phase de conception à cette étape du processus suppose l'application de Design pattern, le découpage de l'application en modules et sous-modules, etc. Le rôle des PIM est de donner une vision structurelle et dynamique de l'application, toujours indépendamment de la conception technique de l'application [20].

-Modèle PM- Platform Model : Rarement utilisé, un PM décrit la structure, et les fonctions techniques relatives à une plateforme d'exécution (systèmes de fichiers, de mémoire, de BDD…) et précise comment les utiliser. Le PM est associé au PIM pour obtenir le PSM [20].

-Modèle PSM- Platform Specific Model : Le PSM est le modèle qui se rapproche le plus du code final de l'application. Un PSM est un modèle de code qui décrit l'implémentation d'une application sur une plateforme particulière, il est donc lié à une plateforme d'exécution [20].

-Code source : Représente le résultat final du processus MDA, le code source est obtenu par génération automatique (partielle ou totale) du code de l'application à partir du PSM. Le code source obtenu peut toujours être enrichi ou modifié manuellement [20].

II.4.2 Un langage de modélisation

La notion de modèle fait explicitement référence à la notion de formalisme ou de langage de modélisation bien défini. Un langage de modélisation est défini par une syntaxe abstraite, une syntaxe concrète et une sémantique. La syntaxe abstraite définit les concepts de base du langage. La syntaxe concrète définit le type de notation qui sera utilisé pour chaque concept abstrait qui peut être graphique, textuelle ou mixte. Enfin, la sémantique définit comment les concepts du langage doivent être interprétés par les concepteurs mais surtout par les machines.

II.4.3 Un métamodèle

Un métamodèle est un modèle qui définit le langage de modélisation d'un modèle. Autrement dit, le métamodèle représente (modélise) les entités d'un langage, leurs relations ainsi que leurs contraintes, c'est-à-dire une spécification de la syntaxe du langage.
Le Métamodèle à son tour est exprimé dans un langage de métamodélisation spécifié par le Méta-Méta-modèle. Le langage utilisé au niveau du méta-méta-modèle doit être suffisamment puissant pour spécifier sa propre syntaxe abstraite et ce niveau d'abstraction demeure largement suffisant (méta-circulaire). Chaque élément du modèle est une instance d'un élément du métamodèle.

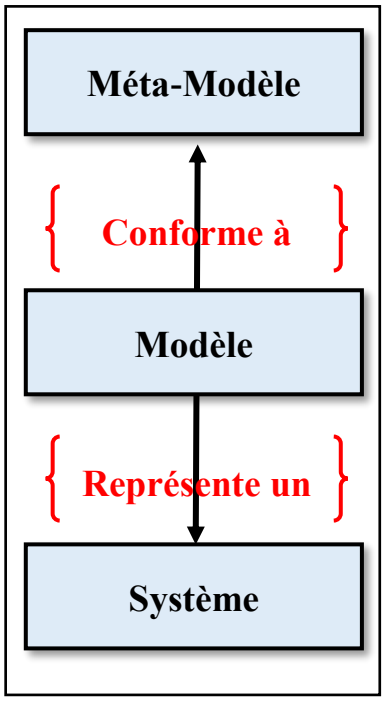

Fig 2.3 : Relation entre système.

II.4.4 Transformation des modèles

Les modèles CIM, PIM et PSM constituent les étapes principales de l'approche MDA. Chacun de ces modèles contient des informations nécessaires à la génération du code source de l'application. Le code est obtenu par génération automatique à partir du PSM, le PSM est obtenu par transformations successives des modèles CIM vers PIM et des modèles PIM vers PSM. La transformation de modèles est une étape importante du processus MDA, c'est grâce aux transformations que les modèles deviennent des éléments productifs de MDA. L'exécution des transformations permet d'assurer un lien de traçabilité entre les différents modèles du processus MDA. Ces liens sont un gage de qualité du processus de développement logiciel dans MDA.

- Transformation de PIM vers PIM, ou raffinement, consiste à ajouter des informations (non liées à une plate-forme) sous forme d'annotations
- Transformation de PIM vers PSM consiste à ajouter au PIM des informations propres à une plate-forme technique Les plates-formes visées (J2EE, .NET...) sont décrites dans un PDM Les règles de transformation sont généralisées et capitalisées pour une réutilisation future
- Transformation de PSM vers PSM (raffinement), souvent nécessaire pour générer un code, se fait par l'utilisation de formalismes intermédiaires comme SDL (Specification and Description Language).
- Transformation de PSM (ou du code) vers PIM, ou rétro ingénierie (reverse engineering), indispensable pour permettre l'intégration d'applications existantes [20].

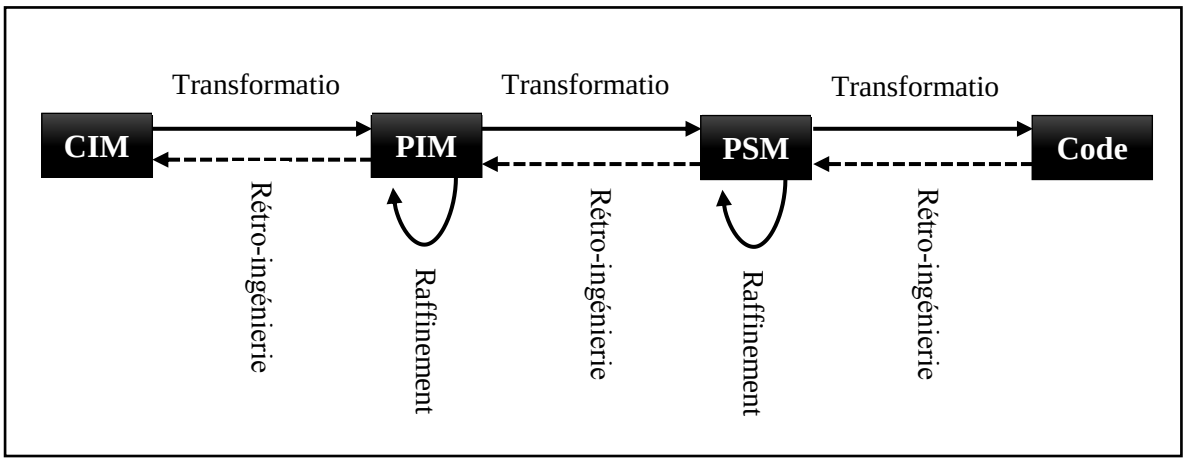

Fig 2.4 : Types de transformations de modèles.

II.4.5 Classification des approches de transformation

Les approches de transformation de modèle ont été classées selon plusieurs axes. Chaque axe mène à une classification particulière. La classification des approches de transformation proposée par **Czarnecki** et **Helsen** se base sur les techniques de transformation utilisées dans les approches et les facettes qui les caractérisent. Selon cette classification on peut distinguer deux types de transformation de modèles : les transformations de type *modèle vers code* et les transformations de type *modèle vers modèles*. Généralement, le premier type de transformation peut être vu comme un cas particulier du deuxième type, nous avons seulement besoin de fournir un méta-modèle pour le langage de programmation cible. Cependant, pour des raisons pratiques de réutilisation de la technologie des compilateurs existants, souvent, le code est simplement généré en tant que texte, qui est ensuite introduit dans un compilateur. Pour cette raison, nous distinguons entre la transformation de type *modèle vers code* et la transformation de type *modèle vers modèle* [21].

- Transformation de type modèle vers code : Dans cette catégorie, on distingue entre les approches basées sur le principe du *visiteur (Visitor-based approach) et* celles basées sur le principe des *patrons (Template-based approach*).
- Approche basée sur le visiteur (*Visitor-based*) : approche de base pour la génération de code, elle consiste à fournir un mécanisme de visiteur pour traverser la représentation interne d'un modèle et créer le code. On peut citer comme exemple le Framework *Jamda* qui fournit un ensemble de classes pour représenter les modèles UML, une API pour manipuler les modèles, et un mécanisme de visiteur pour générer le code.
- Approche basée sur les Templates (*Template-based*) : Actuellement, la majorité des outils MDA disponibles supportent cette approche. La structure d'un *Template* ressemble au code à générer, dans un Template il n'y a pas de séparation syntaxique entre le LHS et le RHS. Le LHS utilise une logique exécutable pour accéder au modèle source, le RHS combine des patrons non typés et une logique exécutable (la logique : code ou requêtes déclaratives).
- Transformation de type modèle vers modèle : Les transformations de type modèle vers modèle consistent à transformer un modèle source en un modèle cible, ces modèles peuvent être des instances de différents métamodèles. Elles offrent des transformations plus modulaires et faciles à maintenir. Dans les cas où on trouve un grand espace d'abstraction entre PIMs et PSMs, il est plus facile de générer des modèles intermédiaires qu'aller directement vers le PSM cible. Les modèles intermédiaires

peuvent être utiles pour l'optimisation ou bien pour des fins de débogage. De plus, les transformations de type *modèle vers modèle* sont utiles pour le calcul des différentes vues du système et leurs synchronisations.

II.4.6 Les étapes de développement MDA

Le développement d'une application débute toujours par le recensement des besoins et exigences de l'utilisateur. Comme dans l'approche MDA, l'étude d'un problème débute par la spécification des besoins métier qui sont ensuite modélisés par un ou plusieurs modèles CIM.

Une fois les fonctions de l'application définies, vient ensuite l'étape d'analyse et de conception abstraite de l'application. Les PIM sont générés à partir des modèles CIM (transformation CIM vers PIM). Les CIM et les PIM sont des modèles pérennes puisqu'ils sont indépendants des plateformes d'exécution. La définition des modèles CIM et PIM constitue l'analyse fonctionnelle de l'application.

L'étape suivante est l'étude technique du développement de l'application. L'application est décrite dans son environnement d'exécution, le PSM est obtenu à partir des modèles PIM (transformation PIM vers PSM) associés aux spécifications techniques de la plateforme cible. Un PSM est spécifique à une plateforme particulière et n'est donc pas pérenne.

Le code source de l'application est obtenu par génération automatique à partir du PSM, c'est le rôle du générateur de code. La génération du code peut être partielle ou totale, cela dépend du niveau de détail de l'information véhiculée par le PSM.

Les besoins de l'utilisateur sont initialement modélisés dans les CIM. Tout au long du processus MDA ces besoins sont retranscrits vers les modèles PIM et PSM, jusqu'au code à travers les diverses étapes de transformation de modèles et de génération de code. Ce lien de traçabilité qui existe, du modèle CIM jusqu'au code, est assuré grâce aux transformations de modèle, et permet d'assurer que l'application à l'arrivée respecte bien les besoins de l'utilisateur [20].

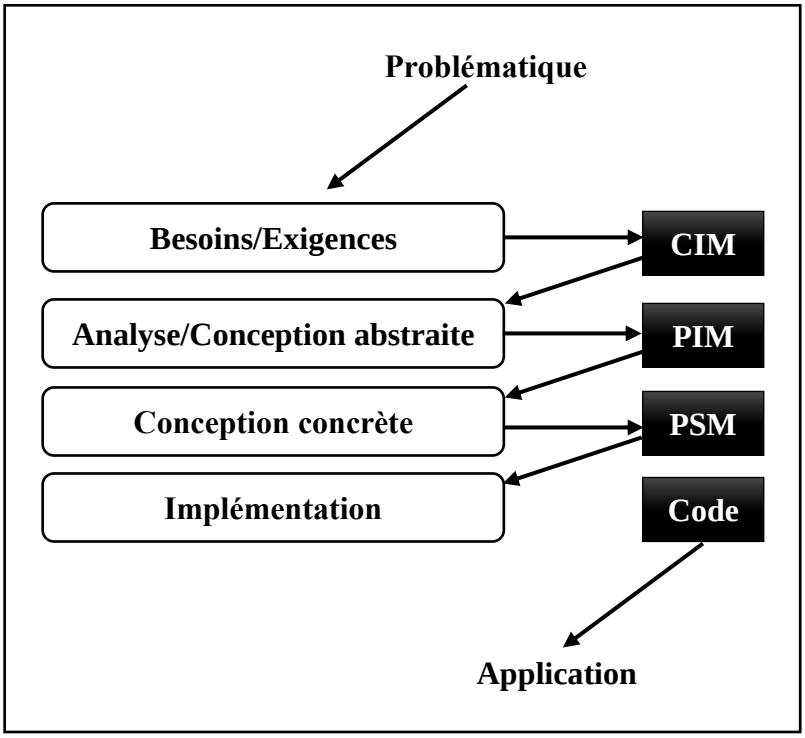

Fig 2.5 : Étapes de développement MDA.

II.5 Outils de transformation de graphes

Plusieurs outils ont été développés pour faire de manière efficace des transformations de Modèles à l'aide des transformations de graphes, parmi ces outils on peut par exemple citer :

- AGG [AGG]: The **A**ttributed **G**raph **G**rammar System.
- FUJABA [Fujaba]: **F**rom **U**ML to **J**ava **a**nd **b**ack **a**gain.
- ATOM3 [Atom3]: **A T**ool for **M**ulti-formalism and **M**eta-**M**odelling.
- VIATRA [Viatra]: **VI**sual **A**utomated model **TRA**nsformations.
- GreAT [Great]: The **G**raph **Re**write **A**nd **T**ransformation tool suite.
- booggie [booggie]: **b**rings **o**bject-**o**riented **g**raph **g**rammars **i**nto **e**ngineering.
- Et d'autre outils comme : DiaGen, GenGED, PROGRES, *GrGen.NET* [Blomer10], MoTMoT, GROOVE, etc.

II.6 AToM^3

*AToM*3 *(A Tool for Multi-formalism and Meta-Modelling*) est un outil pour la modélisation multi-paradigme développé dans le laboratoire MSDL *(Modelling, Simulation and Design Lab)* de l'institut d'informatique à l'université de McGill Montréal, Canada.

Il est développé avec le langage *Python* 5 en collaboration avec le professeur Juan de Lara de l'université Autónoma de Madrid *(UAM)*, Espagne [22] April 2010.

*AToM*3 est développé pour satisfaire deux fonctionnalités principales qui sont :

1. La métamodélisation.
2. La transformation des modèles.

Les formalismes et les modèles dans *AToM*3 sont décrits graphiquement. À partir d'une méta-spécification (exemple : dans le formalisme Entité-relation) d'un formalisme, *AToM*3 génère un outil pour manipuler visuellement (créer et modifier) les modèles décrits dans le formalisme spécifié. Les transformations des modèles sont réalisées par la réécriture des graphes, qui peuvent être exprimées d'une manière déclarative comme un modèle de grammaire de graphes [23].

Les méta-modèles permettent de construire des modèles valides dans un certain formalisme et les méta-méta-modèles sont utilisés pour décrire les formalismes eux-mêmes. *AToM*3 traite les modèles de la même manière dans n'importe quel méta-niveau. L'idée principale d'*AToM*3 est: *"tout est un modèle"* [24].

Certains méta-modèles sont disponibles avec *AToM*3 :

- Entité-relation (Entity-Relationship).
- Réseaux de Petri (Petri Nets).
- Diagrammes de Classes (Class Diagrams).
- ...etc.

Dans notre approche nous utilisons le formalisme *“ diagramme de classe ”*. La figure 2.6 illustre l’interface d’*AToM*3 avec le formalisme de diagramme de classe chargé.

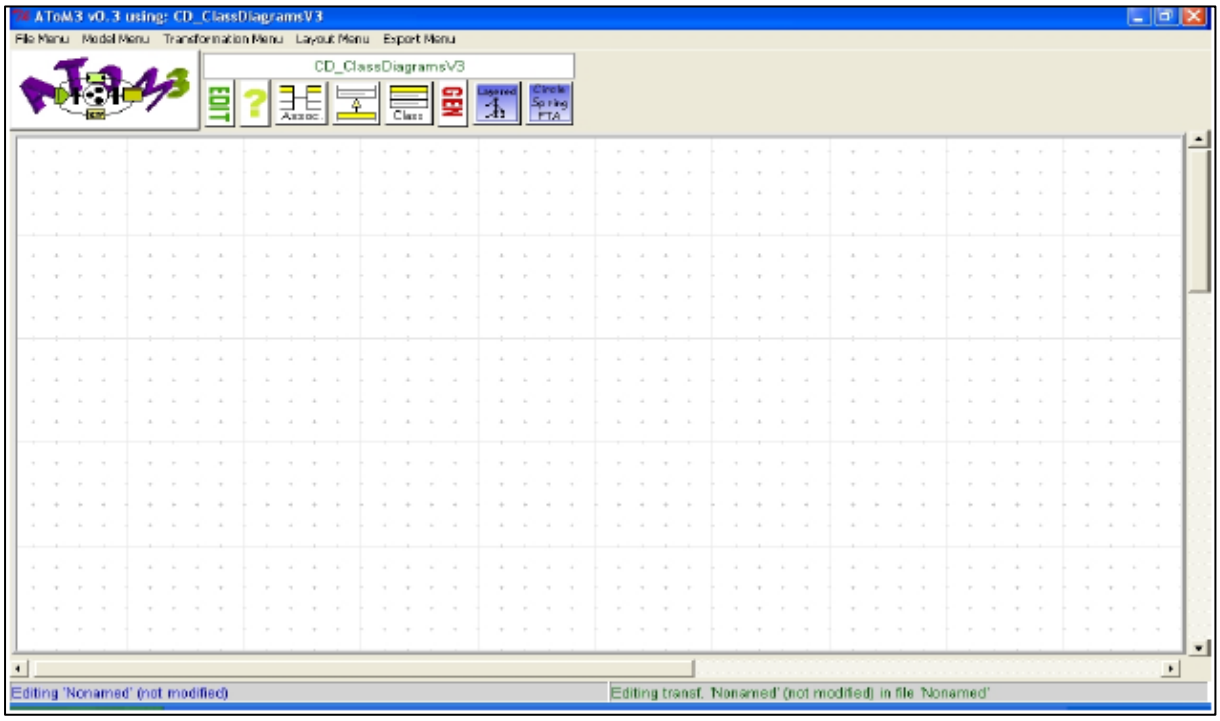

Fig 2.6 : Interface d’AToM^3.

II.6.1 Formalisme diagramme de classe dans AToM^3

Dans *AToM*3 les méta-modèles peuvent être construites à partir des Classes et des relations. La description des classes et des relations d’associations se compose de :

- Nom
- Attributs
- Contraintes
- Action
- Cardinalités
- Apparence.

L’éditeur illustré par la figure 2.7 permet de manipuler ces propriétés.

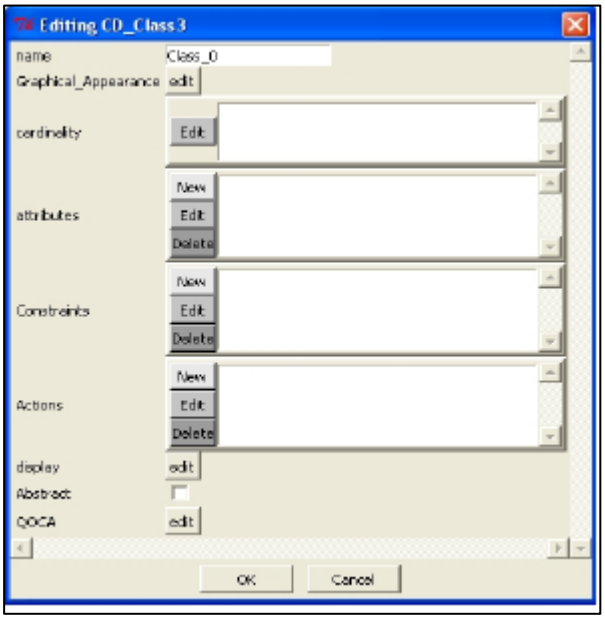

Fig 2.7 : Editeur des propriétés.

- **Contraintes :** Les contraintes peuvent être spécifiées comme des expressions OCL (*Object Constraint Language*) ou Python [23]. Elles peuvent êtres locales associées à une entité, ou globales. Elles ont les propriétés suivantes :

- Un nom de contrainte
- Un événement déclencheur : il peut être soit : sémantique tel que la sauvegarde d'un modèle, etc. graphique ou structurel, tel que le déplacement ou la sélection d'une entité.
- L'évaluation soit : avant l'événement (pré-condition) ou après (post-condition).
- Le code (soit en OCL ou Python). La figure 2.8 illustre l'éditeur des contraintes dans *AToM*3.

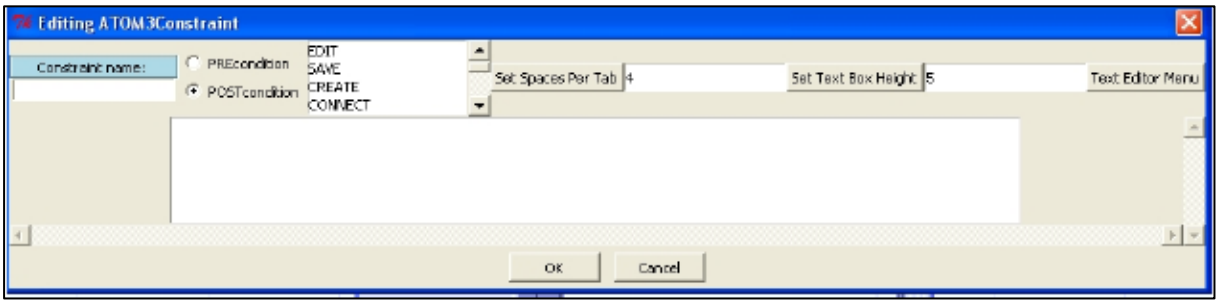

Fig 2.8 : Editeur de contraintes.

- **Action :** Une action est similaire à une contrainte sauf qu'elle a d'autres effets et elle est un code en Python seulement [23]. Elles ont les propriétés suivantes :

- Un nom d'action.
- Un événement déclencheur : Il peut être soit
- Sémantique tel que la sauvegarde d'un modèle, etc.
- Graphique ou structurel, tel que le déplacement ou la sélection d'une entité.
- L'exécution soit : Avant l'événement (pré-condition) ou Après (post-condition)
- Le code (soit en OCL ou Python).
- L'éditeur des actions est similaire à l'éditeur des contraintes illustré par la figure 2.8.

- **Attributs :** Les entités (classes et relation d'association) qui doivent apparaître sur les modèles sont spécifiées ensemble avec leurs attributs et leurs apparences graphiques. Par exemple, pour définir le formalisme de réseau de Petri, il est nécessaire de définir à la fois les places et les transitions. En outre, pour les places nous avons besoin d'ajouter l'attribut nom et le nombre de jetons. Pour les transitions, nous avons besoin de spécifier le nom [23]. Deux types d'attributs existent dans *AToM*3 :

- ***Attributs réguliers* :** Ils sont utilisés pour identifier les caractéristiques d'une entité.
- ***Attributs générateurs* :** Ils permettent de générer d'autres propriétés.

Il y a deux types de base dans *AToM*3 :

- Type régulier tel que String, Integer, Float, Boolean. . . etc.
- Type générateur qui permet de générer des attributs, des contraintes, des attributs graphiques. La figure 2.9 illustre l'éditeur des attributs.

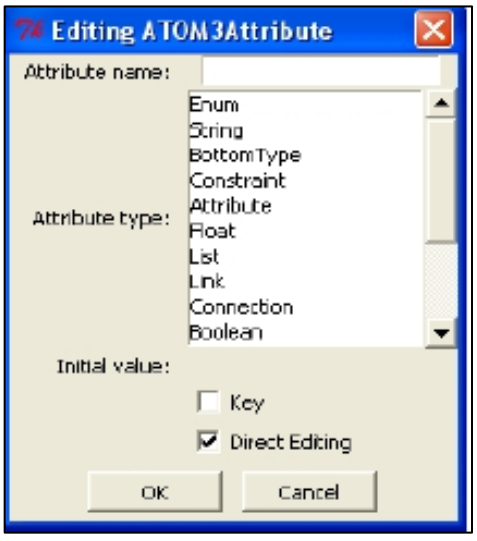

Fig 2.9 : Editeur des attributs.

II.6.2 Transformation des graphes

Dans *AToM*3 la grammaire est un modèle caractérisé par :

- Une action initiale.
- Une action finale.
- L'ensemble des règles.

L'éditeur de grammaire (figure 2.10) permet l'édition, la génération et l'exécution de la grammaire. Chaque règle est constituée de :

- Un nom spécifique pour la règle.
- Une priorité indiquant l'ordre dans lequel la règle est appliquée.
- Une partie gauche *(Left Hand Side : LHS)* qui est un graphe.

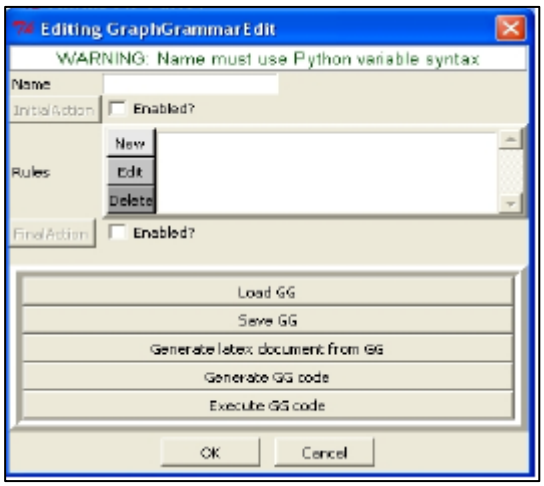

Fig 2.10 : Editeur de grammaire.

Une partie droite *(Right Hand Side : RHS)* qui peut être un graphe.

- Une condition *(Un code en Pyton)* qui doit être vérifiée avant que la règle soit appliquée.
- Une action *(Un code en Pyton)* qui doit être exécutée une fois que la règle soit appliquée.

L'éditeur de règle (figure 2.11) permet l'édition des différentes parties de la règle ainsi que l'action initiale et finale de chaque règle. L'éditeur de condition et l'éditeur d'action d'une règle sont similaires à l'éditeur des contraintes présenté par la figure 2.8.

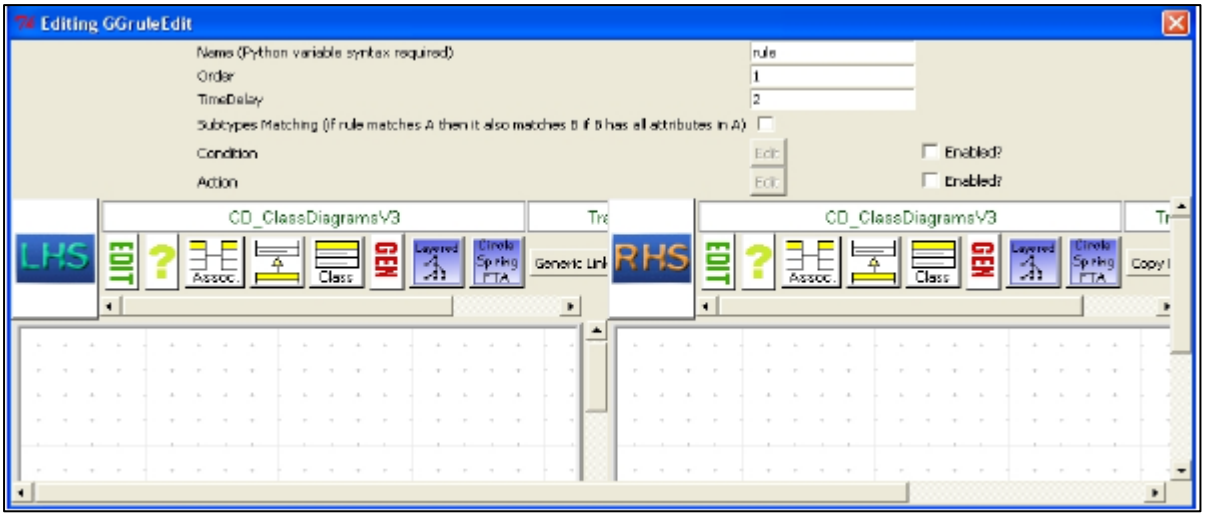

Fig 2.11 : Editeur de règles.

II.7 Conclusion

Dans ce chapitre, nous avons présenté le domaine d'ingénierie des modèles et son approche MDA ainsi que son processus de transformation de modèles, une classification des approches de transformation de modèles, ainsi qu'une introduction à la transformation de graphes et une brève présentation de ses approches. Nous avons également cité quelques outils de transformation de graphes et présenter AToM³, l'outil visuel de modélisation et de métamodélisation multi-formalismes, utilisé dans l'implémentation de notre travail. Les concepts présentés dans ce chapitre constituent un background nécessaire pour la compréhension de notre travail dans le cadre de ce mémoire.

CHAPITRE 3 :

UML pour

l'ingénierie des services

III.1. Introduction

Le langage de modélisation objet unifié UML (*Unified Modeling Language*), est né de la fusion des trois méthodes objet : OMT (*Object Modeling Technique*) de James Rumbaugh, OOSE (*Object Oriented Software Engineering*) de Ivar Jacobson, et la méthode de Grady Booch. Puis il est normalisé par l'OMG en 1997 dans sa version 1.1 comme langage de modélisation des systèmes d'information à objets. UML est rapidement devenu un standard incontournable [25].

Au niveau de *Unified Modeling Language*, deux éléments importants sont à noter. Le terme *unified* et le terme *langage.* Le premier terme signifie que les auteurs [26] ont essayé de regrouper les éléments importants des concepts objets, alors que le deuxième montre qu'il s'agit d'un langage de modélisation et non d'une méthode. UML est un langage qui permet de modéliser non seulement des applications informatiques ou des structures de données, mais également les activités d'un domaine : mécanique, biologie, processus métier... [26].

Plus précisément, UML permet d'offrir des outils d'analyse, de conception et d'implémentation des systèmes logiciels, ainsi que pour la modélisation d'entreprise et des systèmes non logiciels [27].

Ce langage de modélisation unifié repose sur deux concepts essentiels :

1- La modélisation du monde réel au moyen de l'approche orientée objet.

2- L'élaboration d'une série de diagrammes facilitant l'analyse et la conception des systèmes, et permettant de représenter les aspects statiques et dynamiques du domaine à modéliser ou à informatiser.

Le but de ce chapitre est de donner une description détaillée de l'un des diagrammes fondamentaux d'UML, qui est le diagramme d'activité.

III.2. Diagrammes UML

UML 2.0 propose treize types de diagrammes (9 en UML 1.3) pour représenter les différents points de vue de modélisation. Ils se répartissent en deux grands groupes :

1- Diagrammes structurels (*Structure Diagram*).

2- Diagrammes comportementaux (Behavior Diagram).

III.2.1. Diagrammes structurels ou diagrammes statiques (*Structure Diagram*)

Ces diagrammes permettent de visualiser, spécifier, construire et documenter l'aspect statique ou structurel du système informatisé.

- ***Diagramme de classes (Class diagram) :*** Le but d'un diagramme de classes est d'exprimer de manière générale la structure statique d'un système, en termes de classes et de relations entre ces classes. Une classe a des attributs, des opérations et des relations avec d'autres classes.
- ***Diagramme d'objets (Object diagram) :*** Le diagramme d'objet permet d'éclairer un diagramme de classe en l'illustrant par des exemples. Il montre des objets et des liens entre ces objets (les objets sont des instances de classes dans un état particulier).
- ***Diagramme de composants (Component diagram) :*** il montre les composants du système d'un point de vue physique, tels qu'ils sont mis en œuvre (fichiers, bibliothèques, bases de données...). Il montre la mise en œuvre physique des modèles de la vue logique avec l'environnement de développement.
- ***Diagramme de déploiement (Deployment diagram) :*** Ce type de diagramme UML montre la disposition physique des matériels qui composent le système (ordinateurs, périphériques, réseaux...) et la répartition des composants sur ces matériels. Les ressources matérielles sont représentées sous forme de nœuds, connectés par un support de communication.
- ***Diagramme des paquetages (Package Diagram) :*** un paquetage est un conteneur logique permettant de regrouper et d'organiser les éléments dans le modèle UML, il sert à représenter les dépendances entre paquetages.
- ***Diagramme de structure composite (Composite Structure Diagram) :*** Le diagramme de structure composite permet de décrire sous forme de boîte blanche les relations entre les composants d'une seule classe.

III.2.2. Diagrammes comportementaux ou diagrammes dynamiques (*Behavior Diagram*)

Les diagrammes comportementaux modélisent les aspects dynamiques du système. Ces aspects incluent les interactions entre le système et ses différents acteurs, ainsi que la façon dont les différents objets contenus dans le système communiquent entre eux.

- ***Diagramme des cas d'utilisation (Use Case Diagram) :*** Les cas d'utilisation sont une technique de description du système étudié selon le point de vue de l'utilisateur. Ils décrivent sous la forme d'actions et de réactions le comportement d'un système. Donc, le diagramme des cas d'utilisation, permet d'identifier les possibilités d'interaction entre le système et les acteurs. Il permet de clarifier, filtrer et organiser les besoins.
- ***Diagramme d'activité (Activity Diagram) :*** Un diagramme d'activité est une variante des diagrammes d'états-transitions. Il permet de représenter graphiquement le comportement d'une méthode ou le déroulement d'un cas d'utilisation. dans un diagramme d'activité les états correspondent à l'exécution d'actions ou d'activités et les transitions sont automatiques.
- ***Diagramme états-transitions (State Machine Diagram) :*** permet de décrire sous forme de machine à états finis le comportement du système ou de ses composants. Il est composé d'un ensemble d'états, reliés par des arcs orientés qui décrivent les transitions.
- ***Diagramme de séquence (Sequence Diagram) :*** Il représente séquentiellement le déroulement des traitements et des interactions entre les éléments du système et/ou de ses acteurs. Le diagramme de séquence peut servir à illustrer un cas d'utilisation.
- ***Diagramme de communication (Communication Diagram) :*** C'est une représentation simplifiée d'un diagramme de séquence, en se concentrant sur les échanges de messages entre les objets.
- ***Diagramme global d'interaction (Interaction Overview Diagram) :*** permet de décrire les enchaînements possibles entre les scénarios préalablement identifiés sous forme de diagrammes de séquences (variante du diagramme d'activité).
- ***Diagramme de temps (Timing Diagram) :*** Le diagramme de temps permet de décrire les variations d'une donnée au cours du temps.

La figure 3.1, montre la hiérarchie des diagrammes d'UML 2.0.

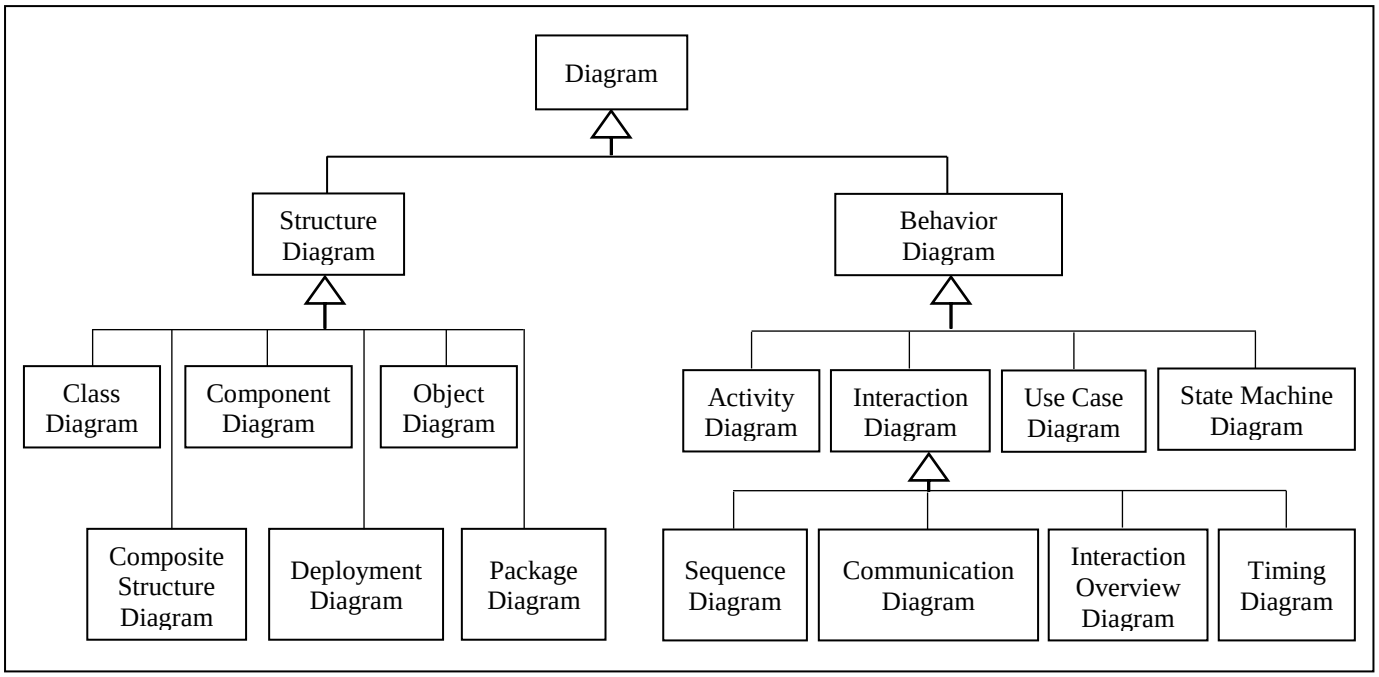

Fig 3.1 : Hiérarchie des diagrammes UML 2.0.

III.3. Diagramme d'activité

III.3.1. Définition

UML permet de représenter graphiquement les aspects dynamiques des systèmes à l'aide de plusieurs diagrammes comportementaux. Parmi eux, on distingue les diagrammes d'activité qui permettent de mettre l'accent sur les traitements. Ils sont utilisés pour représenter le comportement d'une méthode ou le déroulement d'un cas d'utilisation [25].

Un diagramme d'activité est une variante des diagrammes d'états-transitions, dans lequel les états correspondent à l'exécution d'actions ou d'activités, et les transitions sont automatiques (les transitions sont déclenchées par la fin d'une activité et provoquent le début immédiat d'une autre). Un diagramme d'activité peut être attaché à n'importe quel élément de modélisation afin de visualiser, spécifier, construire ou documenter le comportement de cet élément [28].

Les diagrammes d'activité d'UML constituent un outil de modélisation des systèmes workflows, des modèles orientés service et des processus métiers (ils montrent l'enchaînement des activités qui concourent au processus). Un modèle d'activité consiste en activités liées par des flux de données et de contrôle. Une activité peut varier d'une tâche humaine à une tâche complètement automatisée. La différence principale entre les diagrammes d'interaction et les diagrammes d'activité, est que les premiers modélisent le flot de contrôle entre *objets*, alors que les seconds sont utilisés pour modéliser le flot de contrôle entre *activités***.**

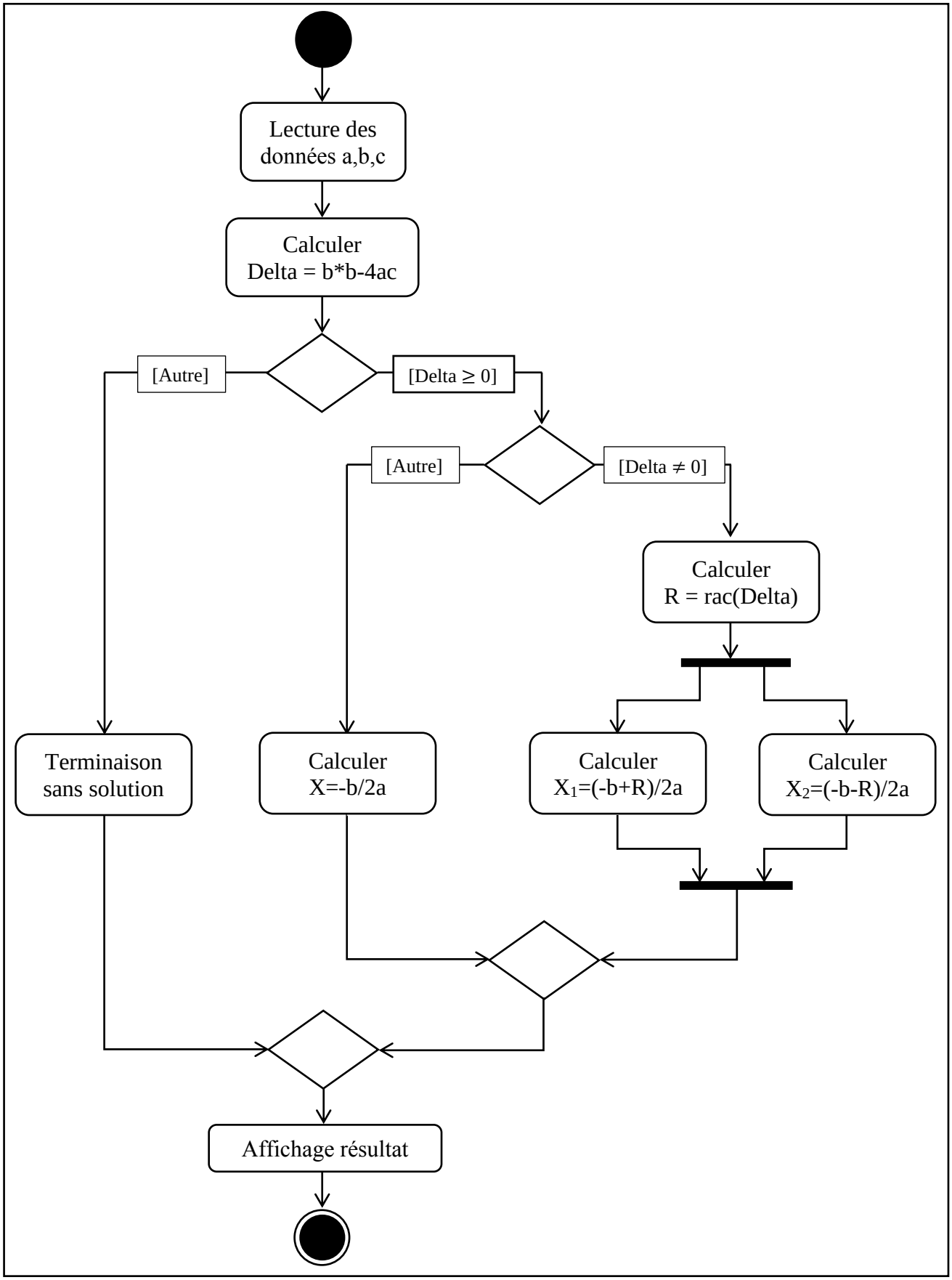

Fig 3.2 : Exemple de diagramme d'activité.

III.3.2. Composition d'un diagramme d'activité

Les définitions de cette partie ont été inspirées essentiellement de L'OMG [27]. Les éléments qui peuvent être contenus dans un diagramme d'activité sont :

1. **Les nœuds :**
   - A- Nœud d'activité : Un nœud d'activité est une classe abstraite permettant de représenter les étapes le long du flux d'une activité. Un nœud d'activité peut être l'exécution d'un comportement subordonné, comme un calcul arithmétique, un appel à une opération, ou la manipulation du contenu d'un objet. Les nœuds d'activité comprennent également le flux de contrôle des constructions, tel que la synchronisation, la décision et la concurrence. Il existe trois types de nœuds d'activités :

- Les nœuds d'exécutions (executable node).
- Les nœuds objets (object node).
- Les nœuds de contrôle (control nodes).

La figure ci-dessus représente graphiquement les nœuds d'activité : On trouve de gauche vers la droite : le nœud d'action, un nœud objet, un nœud de décision ou de fusion, un nœud de bifurcation ou d'union, un nœud initial, un nœud final et un nœud final de flux.

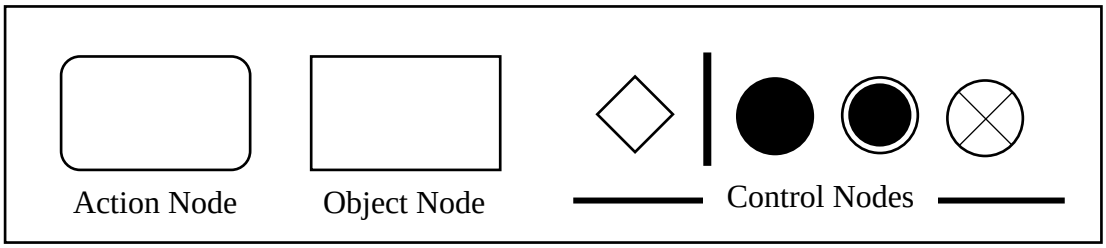

Fig 3.3 : Notation nœuds d'activité.

- B- Nœud de contrôle : Un nœud de contrôle est un nœud d'activités abstrait utilisé pour coordonner les flux entre les nœuds d'une activité. La figure ci-dessous présente l'arbre de spécialisation des nœuds de contrôle.

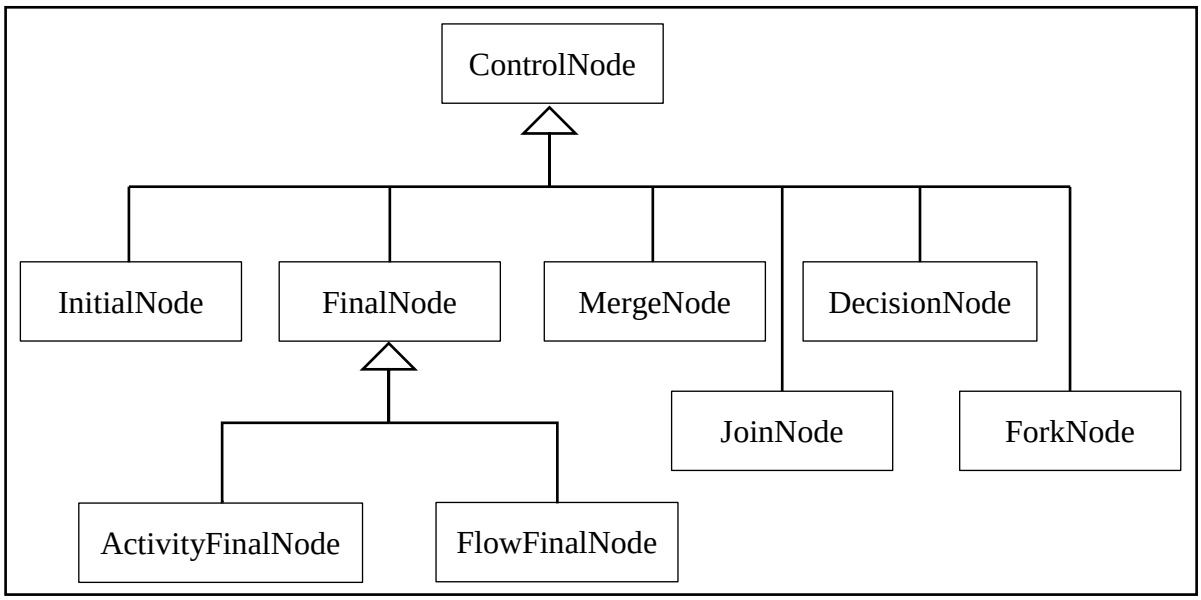

Fig 3.4 : Arbre de spécialisation des nœuds de contrôle.

Graphiquement, les nœuds de contrôle sont présentés comme suit :

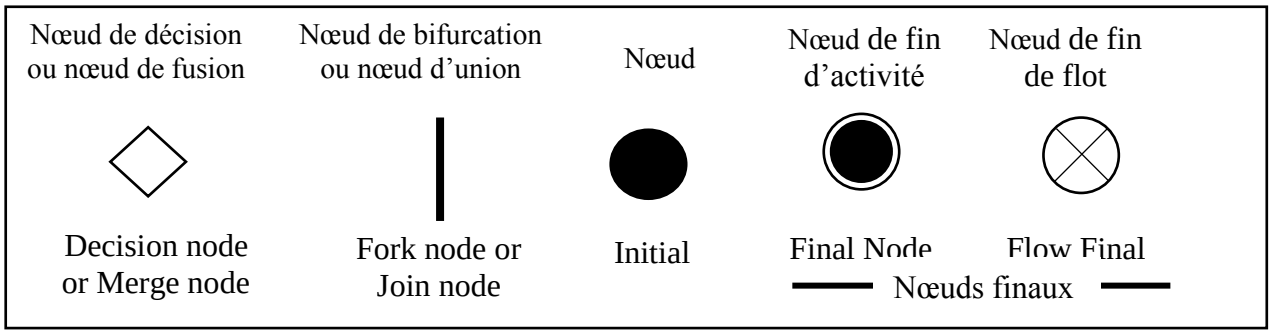

Fig 3.5 : Représentation graphique des nœuds de contrôles.

B.1. Noeud initial (*initial node*) : Un nœud initial est un nœud de contrôle à partir duquel le flot débute. Il possède un arc sortant et pas d'arc entrant. Dans une activité on peut avoir plusieurs nœuds initiaux.

B.2.Un nœud final (*final node*) : Un nœud final est un nœud de contrôle dans lequel le flux d'activité s'arrête. Un nœud final peut avoir un ou plusieurs arcs entrants et aucun arc sortant.

B.3. Noeud de fusion (*merge node) :* Un nœud de fusion est un nœud de contrôle, il rassemble plusieurs flots alternatifs entrants en un seul flot sortant. L'utilité de ce nœud n'est pas pour synchroniser des flux concurrents mais pour accepter un flux (en sortie) parmi plusieurs flux entrants.

B.4. Noeud de décision (*decision node*) : Un nœud de décision est un nœud de contrôle, il permet de faire un choix entre plusieurs flux sortants. Les flux sortants sont sélectionnés en fonction de la condition de garde qui est associée à chaque arc sortant.

(Possibilité d'existence du problème du choix indéterministe). Si aucun arc en sortie n'est franchissable, le modèle est mal formé, et l'utilisation d'une garde [else] est recommandée.

B.5. Noeud de bifurcation (*fork node) :* Un nœud de bifurcation est un nœud de contrôle qui sépare un flux d'entrée en plusieurs flots *concurrents* en sortie.

B.6. Noeud d'union (*join node) :* Un nœud d'union (nœud de jointure) est un nœud de contrôle qui synchronise des flots multiples. Il possède plusieurs arcs entrants et un seul arc sortant. Ce dernier ne peut être activé que lorsque tous les arcs entrants sont activés. L'exemple suivant illustre l'utilisation des nœuds de contrôle.

- C- Nœud exécutable : Un nœud exécutable est une classe abstraite pour les nœuds d'activité qui peuvent être exécutés. Il possède un gestionnaire d'exception qui peut capturer les exceptions levées par le nœud, ou par l'un de ses nœuds imbriqués.
- Action *:* Une action est un nœud d'activité exécutable, c'est le plus petit traitement qui puisse être exprimé en UML. L'exécution d'une action peut être une transformation ou un calcul dans le système modélisé (affectation de valeur à des attributs, création d'un nouvel objet, calcul arithmétique, émission ou réception d'un signal…).

III.4. Profil UML :

Le profil UML est un diagramme de structure qui décrit un mécanisme d'extension léger à l'UML en définissant des stéréotypes personnalisés, des valeurs marquées et des contraintes. Les profils permettent l'adaptation du métamodèle UML pour différents :

- Plateformes, comme java plateforme, Entreprise Edition (java EE) ou microsoft.net.
- Domaines, comme la modélisation des processus métiers, architecture orientée services et les applications médicales…etc

Le mécanisme des profils n'est pas un mécanisme d'extension de première classe. Il ne permet pas de modifier les métamodèles existants ou de créer un nouveau métamodèle tel que MOF le permet. Le profil ne permet que l'adaptation ou la personnalisation d'un métamodèle existant avec des constructions spécifiques à un domaine, une plate-forme ou une méthode particuliers. Il n'est pas possible de supprimer les contraintes qui s'appliquent à un métamodèle, mais il est possible d'ajouter de nouvelles contraintes spécifiques au profil [29].

Selon l'OMG, un profil UML fournit un mécanisme d'extension générique pour adapter les modèles UML à un domaine ou à une problématique spécifique. Les profils permettent aux spécialistes des différents domaines de décrire au mieux leurs systèmes à l'aide d'un ensemble d'extensions clairement définies. Un profil n'ajoute pas réellement de nouveaux concepts à UML mais se contente plutôt de spécialiser les concepts existants. Un profil peut également définir de nouvelles contraintes relatives aux concepts ou aux relations entre ceux-ci.
Comme pour toute approche MDA, l'approche de développement utilisée repose sur l'utilisation de modèles décrit dans un langage clair et précis en se basant sur le mécanisme de profil. Pour cet effet, un profil UML a été proposé spécialement pour permettre au métamodèle de s'adapter à l'architecture orientée service et qui est nommé : UML-S (UML pour l'ingénierie des services).

III.5. UML pour l'ingénierie des services

UML-S est défini comme étant une personnalisation du métamodèle UML. Cette personnalisation est réalisée par l'ajout de stéréotypes, de valeurs étiquetées ainsi que de contraintes. Les stéréotypes sont représentés par des noms entre guillemets tel que «Webservice» et placés au-dessus des noms des éléments tels que les classes. Ils permettent aux développeurs d'étendre le vocabulaire d'UML en ajoutant de nouveaux types d'éléments de modélisation, dérivés des éléments existants. A chaque stéréotype, il est possible d'assigner des propriétés nommées les valeurs étiquetées. La personne chargée de la modélisation pourra alors attribuer une valeur à chacune de ces propriétés au sein de son modèle. Le dernier mécanisme d'extension réside dans l'ajout de contraintes. Ces contraintes permettent de raffiner la sémantique d'un élément du modèle en exprimant textuellement une condition ou une restriction à laquelle l'élément doit se conformer. Il est par exemple possible de définir une contrainte concernant la valeur d'une propriété d'un élément. Pour éviter les ambiguïtés au niveau de ces contraintes, l'OMG a défini l'OCL qui est un langage standard pour l'expression de contraintes. Le langage de modélisation défini par le profil UML-S joue un rôle important au sein de notre approche de développement puisqu'il est utilisé pour la spécification des modèles décrivant la composition de services. Ce langage permet ainsi au développeur de spécifier le comportement du service composé d'une manière abstraite, indépendante de la technologie d'implémentation.

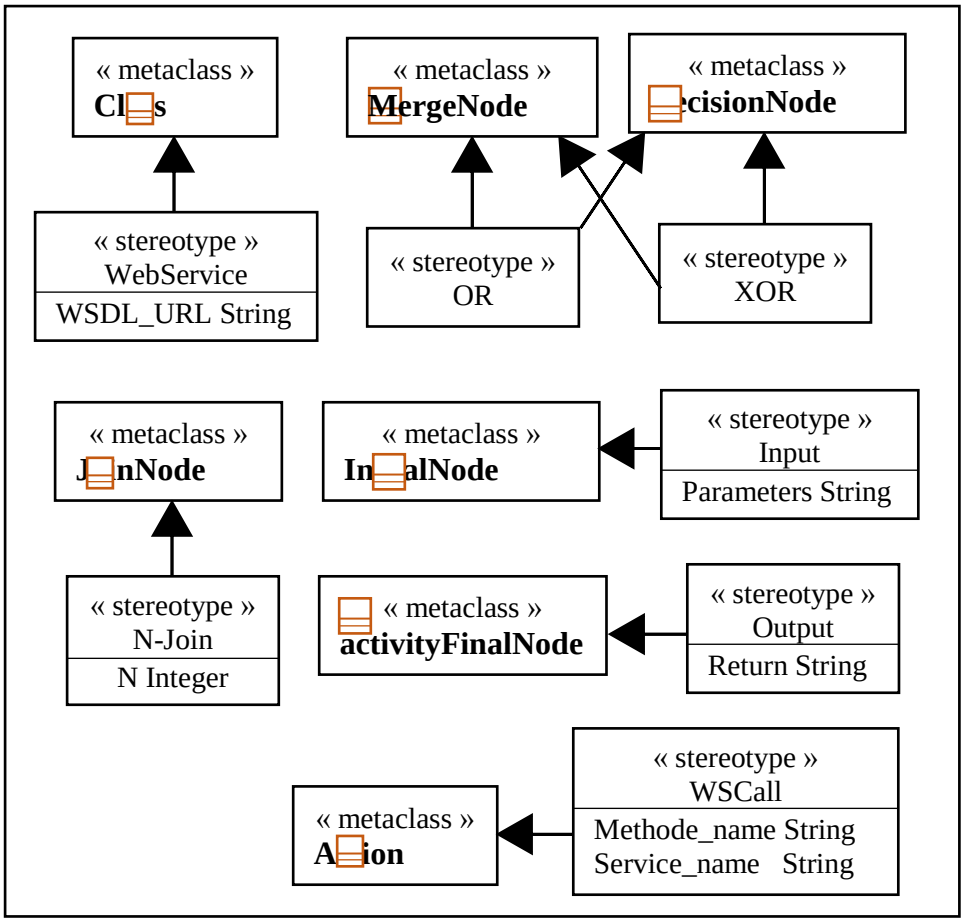

Fig 3.6 : Définition du profil UML-S sous forme de diagramme de classe.

Comme indiqué dans la figure 3.6, le profil UML-S étend des éléments UML appartenant à deux types de diagrammes : le diagramme de classes et le diagramme d'activité. Plus précisément, la métaclasse nommée Class appartient au diagramme de classes alors que toutes les autres métaclasses font partie du diagramme d'activité. Parmi ces autres métaclasses, on remarque l'Action qui est une étape de l'activité (représentée par un rectangle aux bords arrondis dans un diagramme d'activité) ; InitialNode qui est un nœud initial d'activité représenté par un rond noir ; le ActivityFinalNode qui est un nœud final d'activité représenté par un rond noir entouré d'un cercle plus large ; DecisionNode qui est un nœud de décision permettant la séparation en plusieurs branches d'exécution.

III.6. Rôle d'UML-S dans le cycle de développement

La place d'UML-S dans le cycle de développement est illustrée dans la figure 3.7. Deux types de diagrammes UML-S existent afin de permettre la modélisation d'un service composé selon différents points de vue. Le diagramme de classes permet de modéliser l'aspect statique du système, c'est à dire les interfaces des services et les types de données manipulés. Le diagramme d'activité modélise quant à lui l'aspect dynamique de la composition, c'est à dire

les interactions entre les services. Le diagramme d'activité donne ainsi une représentation claire et précise du scénario de composition sous la forme d'un processus métier.

III.7. Description du cycle de développement

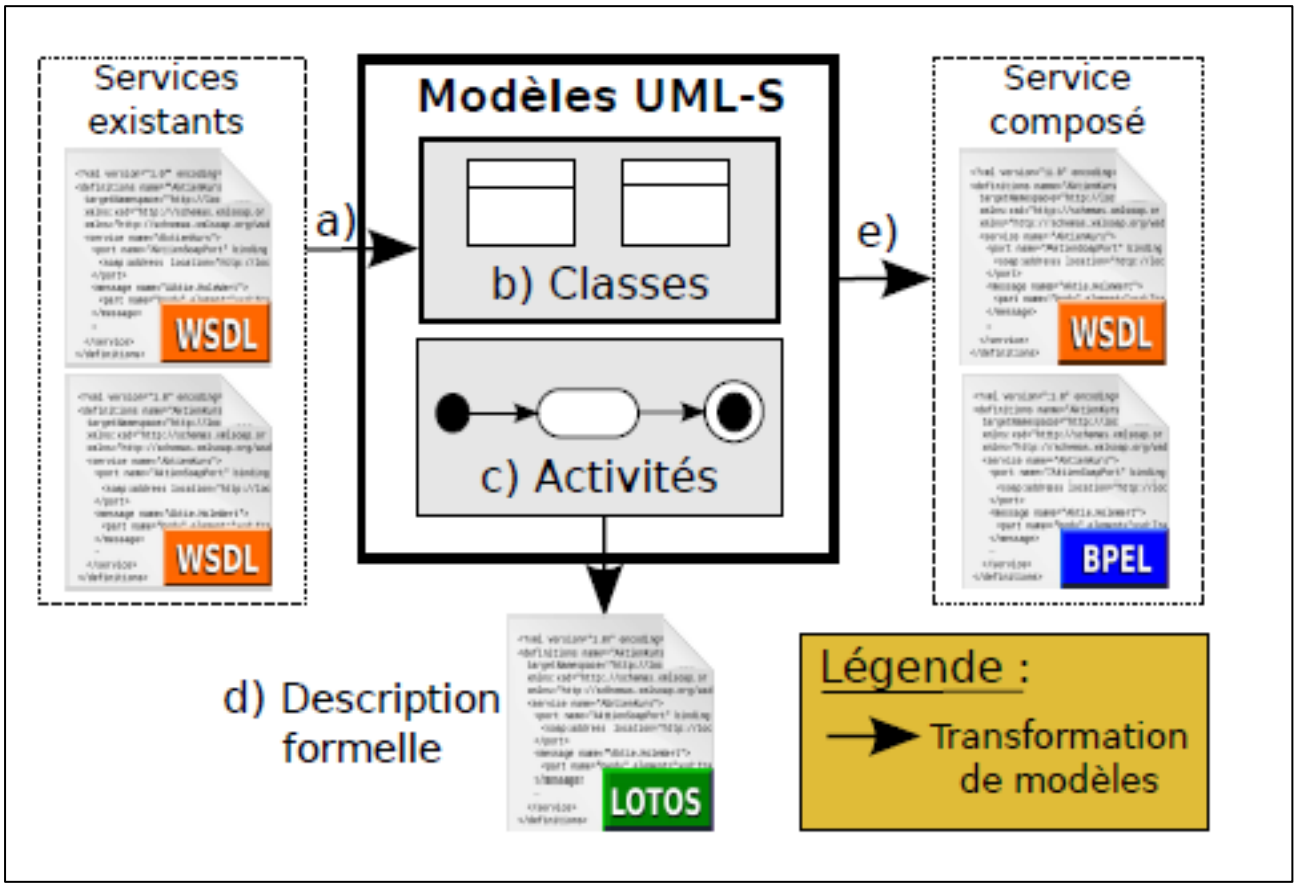

Fig 3.7 : Place UML-S dans le cycle de développement.

A. Import des descriptions WSDL des services existants

- Le développeur sélectionne les services existants qu'il désire composer en récupérant les adresses de description de chaque service au format WSDL.
- Les documents WSDL décrivent principalement l'interface de service : les opérations publiquement disponibles et les types de données manipulés.
- La méthodologie MDA met en œuvre la transformation automatique des modèles ce qui va permettre de transformer le code WSDL en un diagramme UML-S.
- Le modèle en question consiste en un diagramme de classe représentant graphiquement les interfaces de services sélectionnés pour la composition.
- Chaque service est représenté par une classe marqué du stéréotype <<Web Service>> qui a le même nom tel que le service et dont les méthodes correspondent aux opération mise à disposition par le service en question.

B. Définir l'interface du service composé

- Dans le cas d'orchestration, un nouveau service dit composé résulte de la composition des services existants : c'est ce dernier qui va diriger la composition.
- La description de l'interface du service composé passe principalement par l'ajout d'une classe stéréotypé <<Web Service>> : le développeur lui assigne un nom et y ajoute des méthodes.

C. Définition du comportement de chaque opération

- Les méthodes qu'a défini le développeur dans l'étape précédente correspond a un scénario de composition distinct dont le comportement doit être spécifié.
- La conversation entre les services est généralement modélisée par un workflow réalisant un processus métier via diagramme d'activité.
- Une fois le profil UML-S appliqué le diagramme d'activité devient un outil BPM efficace pour permettre aux développeurs de spécifié le comportement de la composition.

D. Vérification formelle de la composition

- Lors de l'étape précédente, le développeur a construit un ou plusieurs modèles dynamiques de composition. Certains scénarios de composition sont complexes et il est parfois difficile de s'assurer manuellement que la spécification est conforme vis à vis du comportement attendu.

E Transformation des modèles en code

- Obtenir à la fois la description du service composé WSDL et son code exécutable BPEL.

III.8 Conclusion

Dans ce chapitre nous avons parlés d'UML et de ses différents diagrammes nous avons détaillé le diagramme d'activité puisqu'il constitue la base de notre transformation au fichier BPEL qui définit comment vas se produire l'orchestration de plusieurs services web pour un nouveau service dit composite. Ensuit on a défini c'est quoi un profil UML et nous avons introduit le profil UML-S proposé dans la thèse qui constitue notre référence au cours de notre travail ainsi que son rôle dans le cycle de développement.

CHAPITRE 4 :

Contribution

IV.1. Introduction

Dans ce chapitre on va présenter l'approche proposée par l'auteur [30] ainsi que notre travail qui consiste à implémenter l'approche proposée en utilisant le logiciel de transformation Atom3 qui a déjà était décrit en détail dans le chapitre 2.

IV.2. Utilisation du diagramme d'activité [30]

Le diagramme d'activité est utilisé pour décrire le comportement interne d'une opération fournie par le service composé. En effet, chaque opération réalise un scénario de composition distinct et il est donc nécessaire de construire un diagramme par opération. L'appel à l'opération est alors modélisé par le nœud initial de l'activité et son retour par le nœud final. Dans le cas où l'opération prend des paramètres, ils ont assigné le stéréotype « Input » au nœud initial. De la même manière, si l'opération retourne une valeur, ils ont assigné le stéréotype « Output » au nœud final. Les autres nœuds du diagramme correspondent soit à des actions, soit à des structures de contrôle. Dans le contexte de la composition de services avec UML-S, une action symbolise un appel à un service tiers et on lui assigne le stéréotype « WSCall », comme illustré dans la figure 4.1.

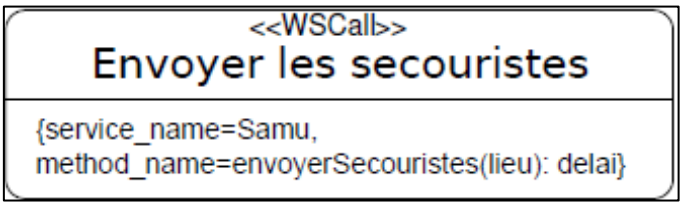

Fig 4.1 : UML-S Action.

Comme il est possible de le constater sur la figure, deux valeurs étiquetées sont associées au stéréotype « WSCall » : service_name et method_name. La propriété service_name indique le nom du service à appeler, ici Samu, et fait obligatoirement référence au nom d'une classe «Webservice » dans le diagramme de classes. La propriété method_name indique le nom de l'opération à appeler parmi celles mises à disposition par le service. De la même manière, cette opération doit figurer dans les méthodes de la classe « Webservice » correspondante. L'opération appelée dans la figure est envoyerSecouristes(). Celle-ci prend une variable notée lieu en paramètre et sa valeur de retour est stockée dans la variable délai.

Les données représentent une partie importante de la composition de services. En effet, chaque service prend généralement des paramètres en entrée et retourne le plus souvent une valeur. Lors de la mise en œuvre de la composition, il n'est pas rare de devoir assigner la sortie d'un service à l'entrée d'un autre. Il est même parfois nécessaire de procéder à une transformation des données entre les appels afin de se conformer aux exigences d'interfaçage des services. Les données prennent donc une place importante au sein de la modélisation avec UML-S et nous introduisons pour les représenter le concept de variable qui est familier à tout développeur. Chaque donnée est modélisée dans le diagramme d'activité UML-S sous la forme d'une variable et identifiée par son nom. Dans le cas où il est nécessaire de procéder à une transformation des données, ils ont proposé de modéliser ce comportement à l'aide d'une nouvelle fonctionnalité ajoutée à la sémantique d'UML 2.0. En effet, il est désormais possible dans UML de spécifier un comportement dit de transformation sur un arc où il y a passage de données. Ce type de comportement est généralement représenté par les outils de modélisation comme un commentaire, c'est à dire un rectangle avec le coin haut-droit replié, marqué du stéréotype « transformation ». Ce choix de représentation est illustré dans la figure 4.2

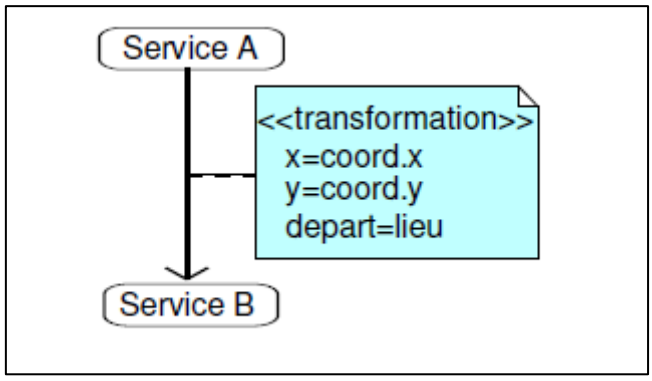

Fig 4.2 : Transformation des données avec UML-S.

IV.3. Implémentation

L'approche que nous proposons, vise à transformer les diagrammes d'activité vers le langage d'orchestration BPEL. Pour réaliser ça, nous devons d'abords créer un métamodèle des diagrammes d'activités. Ce dernier va nous permettre de générer l'outil de modélisation et grâce à cet outil nous pouvons créer un diagramme d'activité quelconque. Pour transformer ces diagrammes en code BPEL nous avons créé une grammaire qui se compose de 35 règles. Chaque règle définit un code BPEL pour chaque partie du diagramme d'activité.

IV.3.1. Métamodélisation du diagramme d'activité

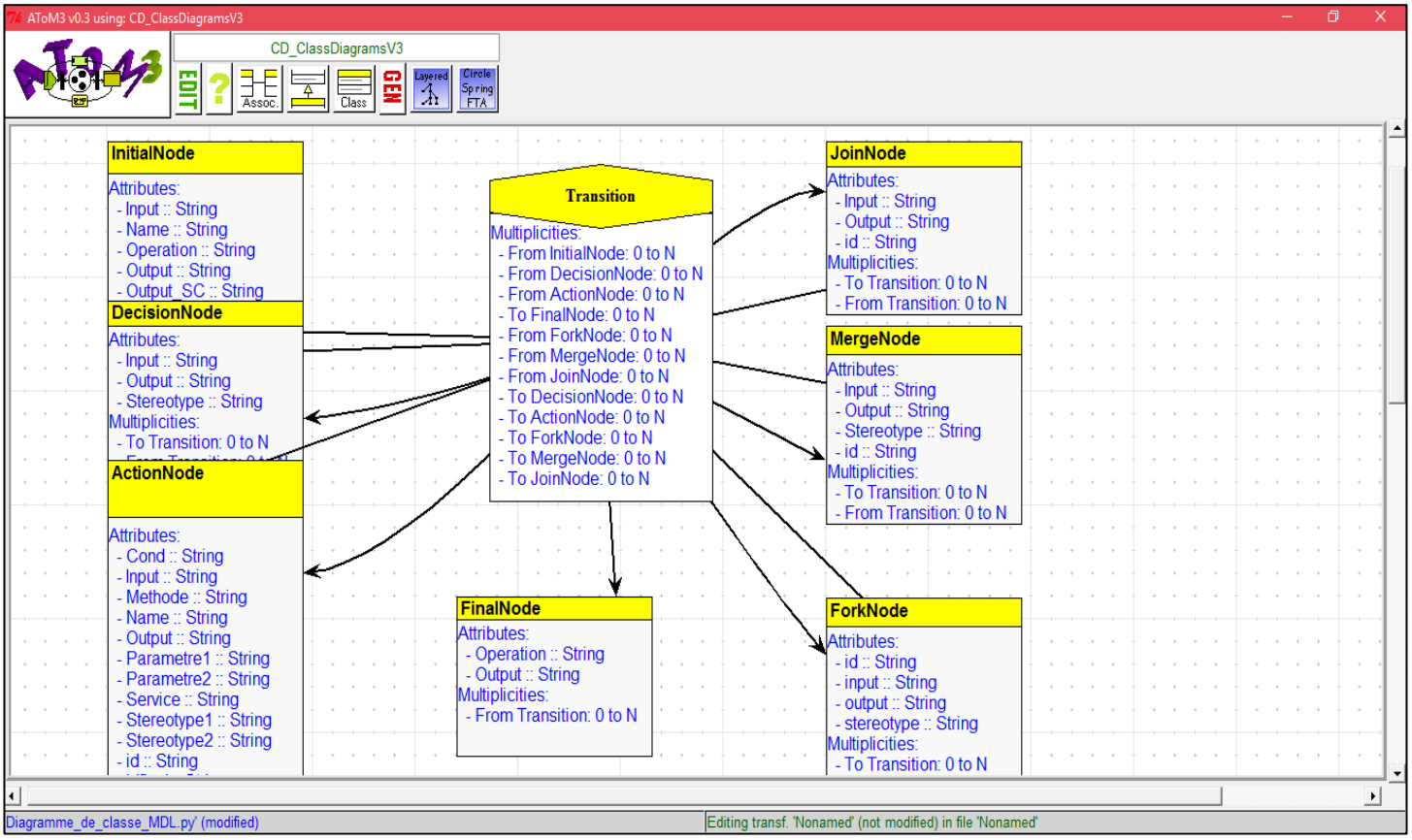

Fig 4.3: Métamodèle du diagramme d'activité.

- Chaque classe du métamodèle représente un des nœuds du diagramme d'activité et contient toutes les informations liées à ces derniers comme attributs et ces derniers vont être utilisés dans le fichier BPEL :
    1. Classe InitialNode : représente le début d'un diagramme d'activité représenté par un nœud noir.
    2. Classe ActionNode : Etape de l'activité représenté par un rectangle aux bords arrondis.
    3. Classe FinalNode : Noeud finale d'activité indique une terminaison avec succès représenté par un rond noir entouré par un cercle plus large.
    4. Classe DecisionNode : Nœud de décision permettant de spécifie les différentes alternatives possibles. Elle a un seul arc entrant et deux arcs sortants qui sont gardés par des condition et est représenté par un losange.
    5. Classe JoinNode : Nœud de jonction de branches cette classe rassemble plusieurs flots entrants en un seul flot sortant. Représenté par une barre verticale.

6. Classe MergeNode : Nœud de synchronisation il a un seul arc entrant et plusieurs arcs sortant gardé par une condition. Un des arcs sortants va être déclenché selon la condition vérifié représenté par un losange.
7. Classe ForkNode : elle représente un nœud de synchronisation qui possède un seul arc entrant et plusieurs arcs sortants qui doivent être déclenchés simultanément représenté par une barre verticale.

- Pour chaque constituant du métamodèle, on va éditer ses apparences visuelles en prenant en compte les stéréotypes ajoutés au profil UML proposé par CDUMEZ (UML-S). Le diagramme d'activité est utilisé pour décrire le comportement interne d'une opération fournie par le service composé. L'appel de l'opération est modélisé par le nœud initial et son retour par le nœud final. Si l'opération prend des paramètres on lui assigne le stéréotype <<input>> et on indique les noms des variables. Si l'opération retourne une valeur on assigne au nœud final le stéréotype output et on indique le nom des variables. Une action symbolise un appel à un service et on lui assigne le stéréotype <<WSCALL>> :

1. Classe InitialNode : stéréotypé avec <<input>> :

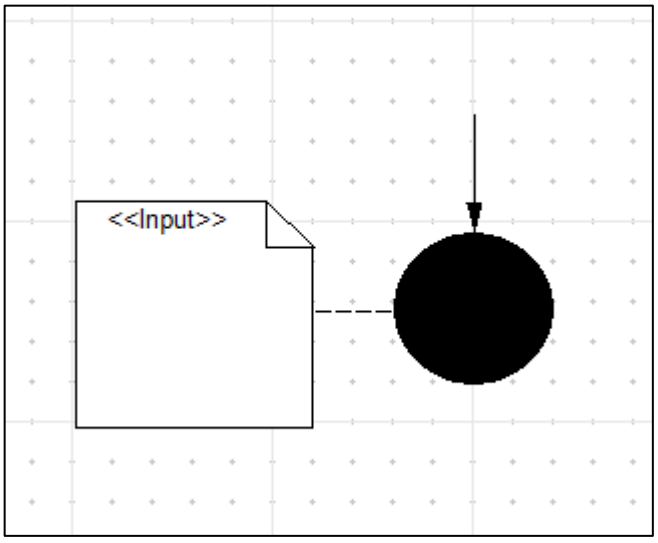

Fig 4.4: Apparence visuelle du nœud initiale.

2. Classe ActionNode : stéréotypé avec <<WSCALL>> et chaque action contient une transformation de variable stéréotypé avec <<Transformation>>.

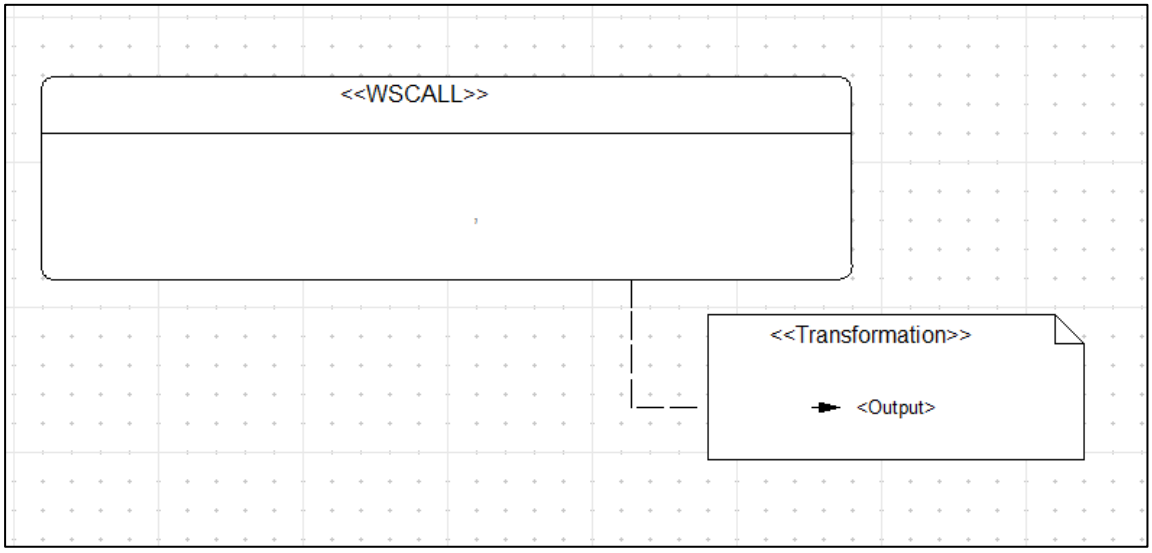

Fig 4.5: Apparence visuelle du nœud action.

3. Classe FinalNode : stéréotypé par <<output>>.

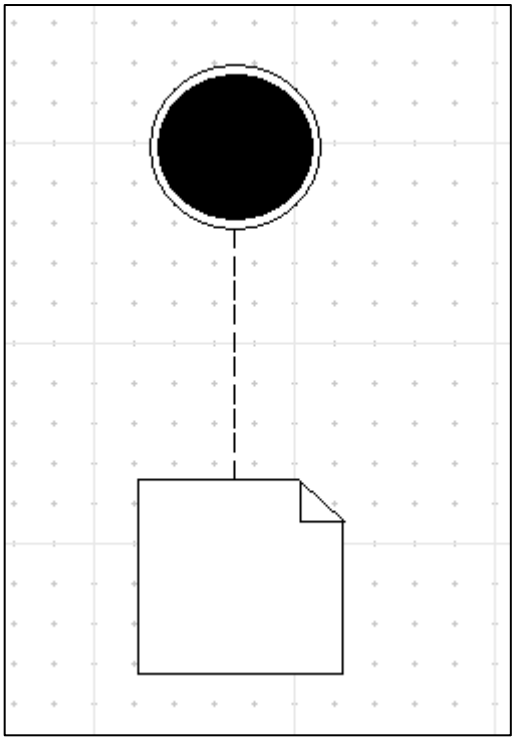

Fig 4.6: Apparence visuelle du nœud final.

4. Classe DecisionNode : stéréotypé par <<XOR>> qui représente un choix exclusif.

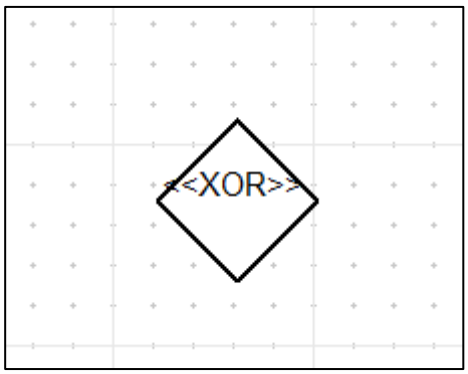

Fig 4.7: Apparence visuelle du nœud de décision.

5. Classe JoinNode :

Fig 4.8: Apparence visuelle du nœud de jointure

6. Classe MergeNode : stéréotypé par <<OR>>.

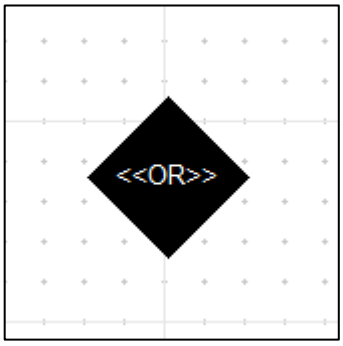

Fig 4.9: Apparence visuelle du nœud de synchronisation merge.

7. Classe ForkNode :

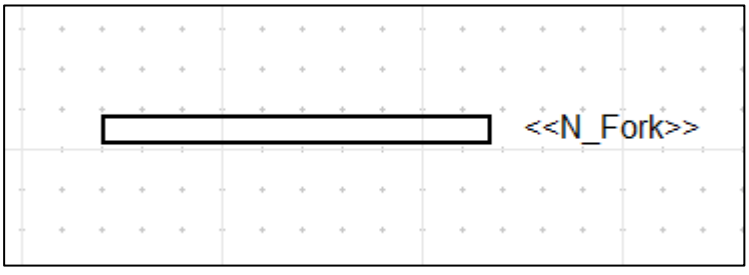

Fig 4.10: Apparence visuelle du nœud de synchronisation fork.

- Puis, on va générer automatiquement l'outil de modélisation des diagrammes d'activité qui apparait dans la figure ci-dessous (Figure 4.11).

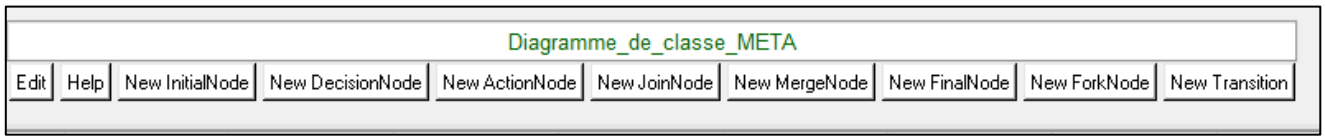

Fig 4.11: L'outil de modélisation des diagrammes d'activité.

- Une fois l'outil de modélisation générée, on va spécifier les règles de la grammaire de transformation puis exécuter la grammaire qui génère le code BPEL

IV.3.2. Règles de la grammaire de transformation

Nous avons proposé une grammaire nommée *Gram_bpel*. Elle a une action initiale permettant de créer un fichier texte pour écrire le code BPEL, et une variable globale « Visited » pour éviter l'exécution d'une règle sur le même nœud plusieurs fois. L'exécution de cette grammaire sur un diagramme d'activité, permet d'obtenir automatiquement le code BPEL équivalent à ce diagramme.

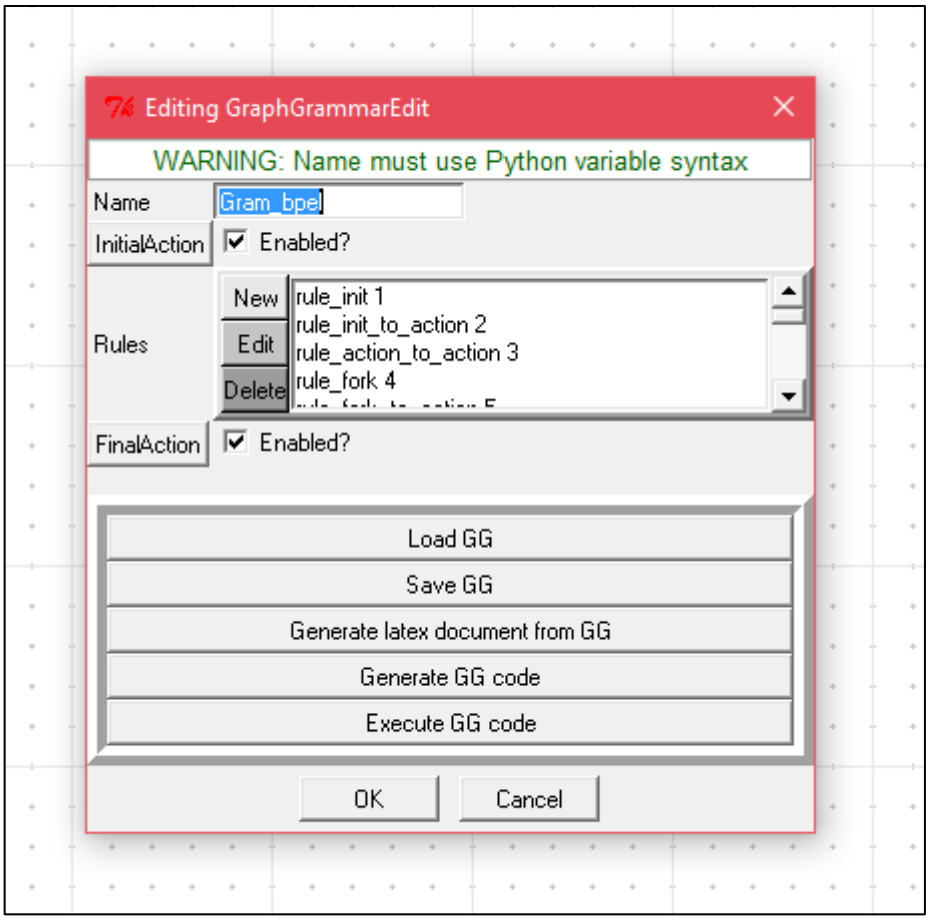

Fig 4.12: Gram_bpel.

Chaque activité UML-S a un lien avec les activités BPEL comme le montre la figure suivante :

Stéréotype UML-S	Activité BPEL
<<input>>	<receive>
<<output>>	<reply>
<<WSCALL>>	<invoke>
<<Transformation>>	<assign>

Fig 4.13: Relation entre les activités UML-S et BPEL.

BPEL étant un langage d'exécution de processus métier, celui-ci permet de représenter de manière directe et précise un certain nombre de structures de contrôle. Le lien entre ces structures de contrôle et les constructions BPEL correspondantes est présenté dans la figure 4.14.

Structure de contrôle	Equivalent BPEL
Début d'activité	<receive createInstance="yes" operation=….. Variable=……../>
Séquence d'activité	<sequence>
Branchement multiple (fork)	<flow> <sequence> <activity1> …. </flow>
Choix exclusif (décision)	<switch> <case condition= …..> <activity1> …… <case condition= …..> <activity2> …….. </case> </switch>
Choix altéré (merge)	<if condition=…… > <activity1> <elseif condition=….> <activity 2> …….. </elseif> </if>
Fin d'activité	<reply> … </reply>

Fig 4.14: Principales structures de contrôle prises en charge par BPEL.

- **Description des règles :**

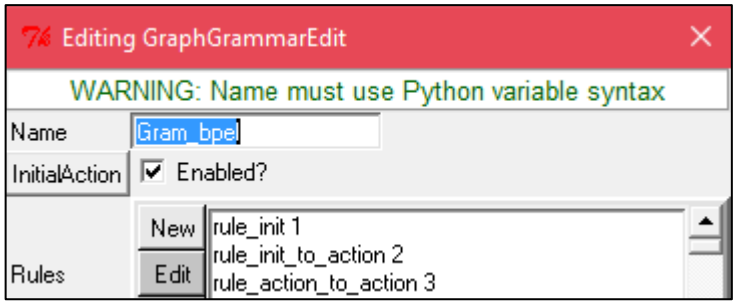

Fig 4.15: Rules 1 to 3.

La première règle permet de créer un fichier BPEL ainsi que localiser un nœud initial et donner son équivalent en code BPEL. La seconde règle permet de localiser un nœud action venant après un nœud initial et donner son équivalent en code BPEL. La troisième règle localise un nœud action venant après une autre action et donne son code BPEL.

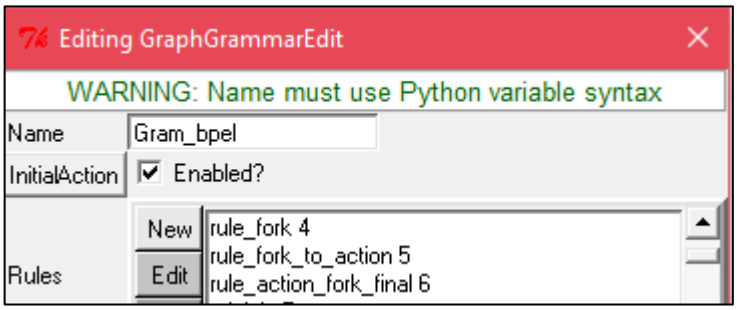

Fig 4.16: Rules 4 to 6.

Ces trois règles gèrent code équivalent au nœud fork, le code BPEL généré à partir de ces dernières est comme suite (la figure ci-dessous est juste un jeu de données ne contient pas un exemple réel ce dernier va être présenté dans l'étude de cas) :

```
<flow>
<sequence>


<invoke partnerLink="action5"
operation="action5"
inputVariable="action5"
outputVariable="action5"/>
<assign>
<copy>
<from variable="action5"/>
<to variable="action5"/>
</copy>
</assign>

<invoke partnerLink="action6"
operation="action6"
inputVariable="action6"
outputVariable="action6"/>
<assign>
<copy>
<from variable="action6"/>
<to variable="action6"/>
</copy>
</assign>
</sequence>
</flow>
```

Fig 4.17: Code BPEL du ForkNode.

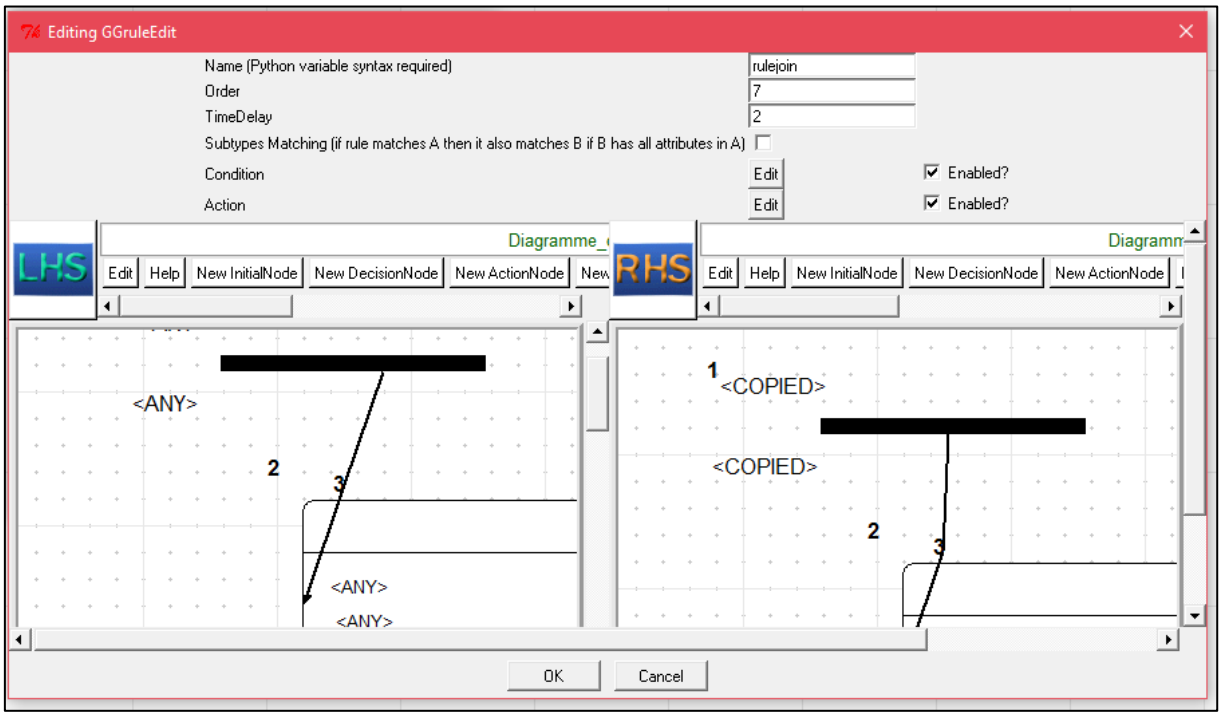

Fig 4.18: Règle de synchronisation (JoinNode).

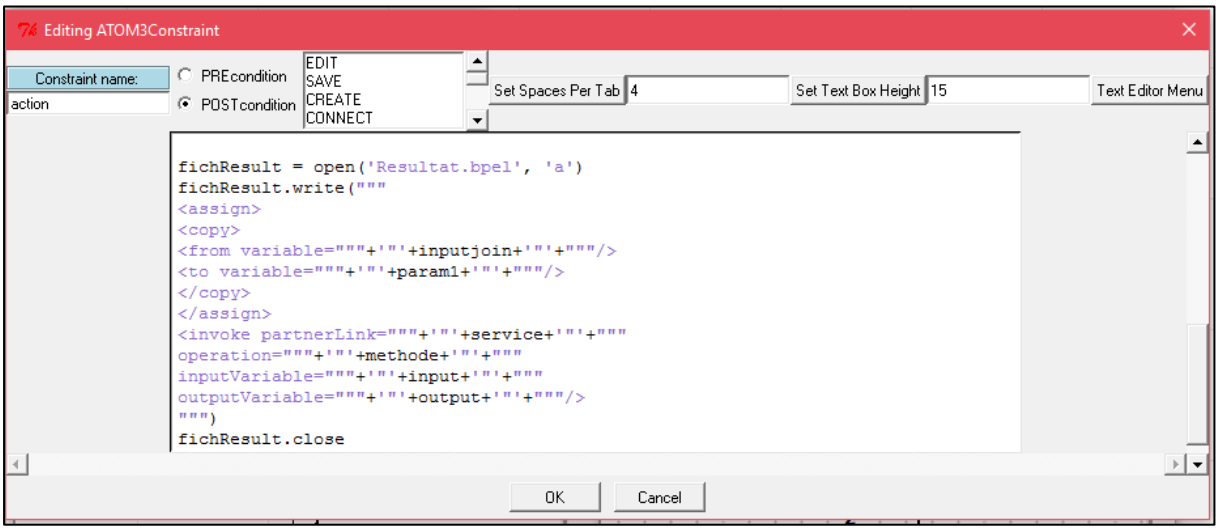

Fig 4.19: Règle de synchronisation (JoinNode).

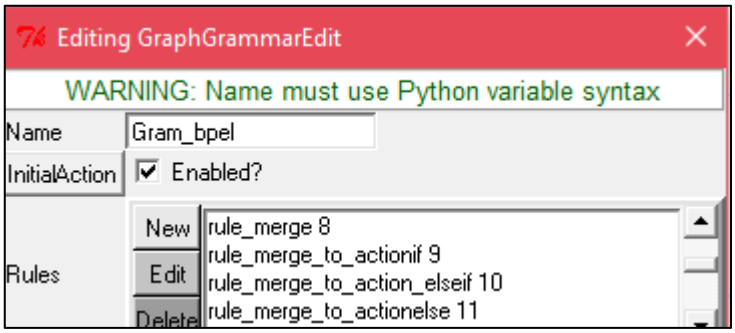

Fig 4.20: Rules 8 to 11.

Ce bloc gère le MergeNode qui décrit le choix altéré, le code BPEL généré à partir de ces dernières est comme suite (la figure ci-dessous est juste un jeu de données ne contient pas un exemple réel ce dernier va être présenté dans l'étude de cas) :

```
<if condition="cond1">
<invoke partnerLink="action8"
operation="action8"
inputVariable="action8"
outputVariable="action8"/>
<assign>
<copy>
<from variable="action8"/>
<to variable="action8"/>
</copy>
</assign>

<elseif condition="COND2">
<invoke partnerLink="action9"
operation="action9"
inputVariable="action9"
outputVariable="action9"/>
<assign>
<copy>
<from variable="action9"/>
<to variable="action9"/>
</copy>
</assign>
</elseif>
</if>
```

Fig 4.21: Code BPEL équivalent au MergeNode.

IV.4. Etude de cas [30]

Nous allons traiter dans cette étude de cas le développement d'un service composé utilisant des services existants et réalisant des interactions entre ces services définies par un scénario prédéterminé. Nous allons présenter dans cette section les services qui vont être composer ainsi que le scénario de composition considéré. Nous allons composer quatre services existants nommés Hôpital, SAMU, Police et BaseAccidents. Le service Hôpital fournit ici une méthode nommée hopitalPlus-Proche() qui retourne les coordonnées GPS de l'hôpital le plus proche de la position passée en paramètre. Le service SAMU fournit une méthode appelée envoyerSAMU() qui va envoyer une équipe de soins d'urgence à la position passée en paramètre (adr_acc). C'est aussi ce service qui s'occupe d'acheminer le ou les blessés à l'hôpital situé à la position adr_hop passé en paramètre. La méthode du service SAMU retourne une valeur entière indiquant la durée prévue du trajet vers le lieu de l'accident. Le service Police fournit une méthode envoyerPatrouille() qui permet d'envoyer une patrouille de Police à la latitude et la longitude passées en paramètre. Enfin, le service BaseAccidents permet d'interroger ou d'ajouter des éléments à une base de données répertoriant les accidents

rapportés. Le service en question fournit ainsi deux méthodes : ajouterAccident() qui permet d'ajouter un rapport d'accident à la base de données et accidentExiste() qui permet de vérifier si un accident a déjà été rapporté. Les accidents sont ici identifiés par leur position GPS. En utilisant ces quatre services, nous allons élaborer un scénario de réponse d'urgence qui réalisera une séquence d'actions à chaque rapport d'accident. Nous désirons ici créer un service composé de réponse d'urgence que nous allons appeler 119, en référence au numéro d'appel d'urgence en Union Européenne. Ce service permettra aux utilisateurs de rapporter un accident à une position donnée. Le service prendra alors les mesures nécessaires et retournera à l'utilisateur le délai prévu d'intervention en secondes. Le scénario interne qui dirigera les interactions entre les services est le suivant. Le service va d'abord interroger la base d'accidents afin de vérifier si l'accident rapporté est déjà connu. Dans le cas où celui-ci a déjà été rapporté, une valeur négative est simplement retournée à l'utilisateur. Dans le cas contraire, nous allons d'abord rechercher l'hôpital le plus proche du lieu de l'accident et nous enverrons ensuite à la fois le SAMU et la police. Nous ajouterons ensuite ce rapport à la base d'accidents avant de retourner à l'utilisateur le délai.

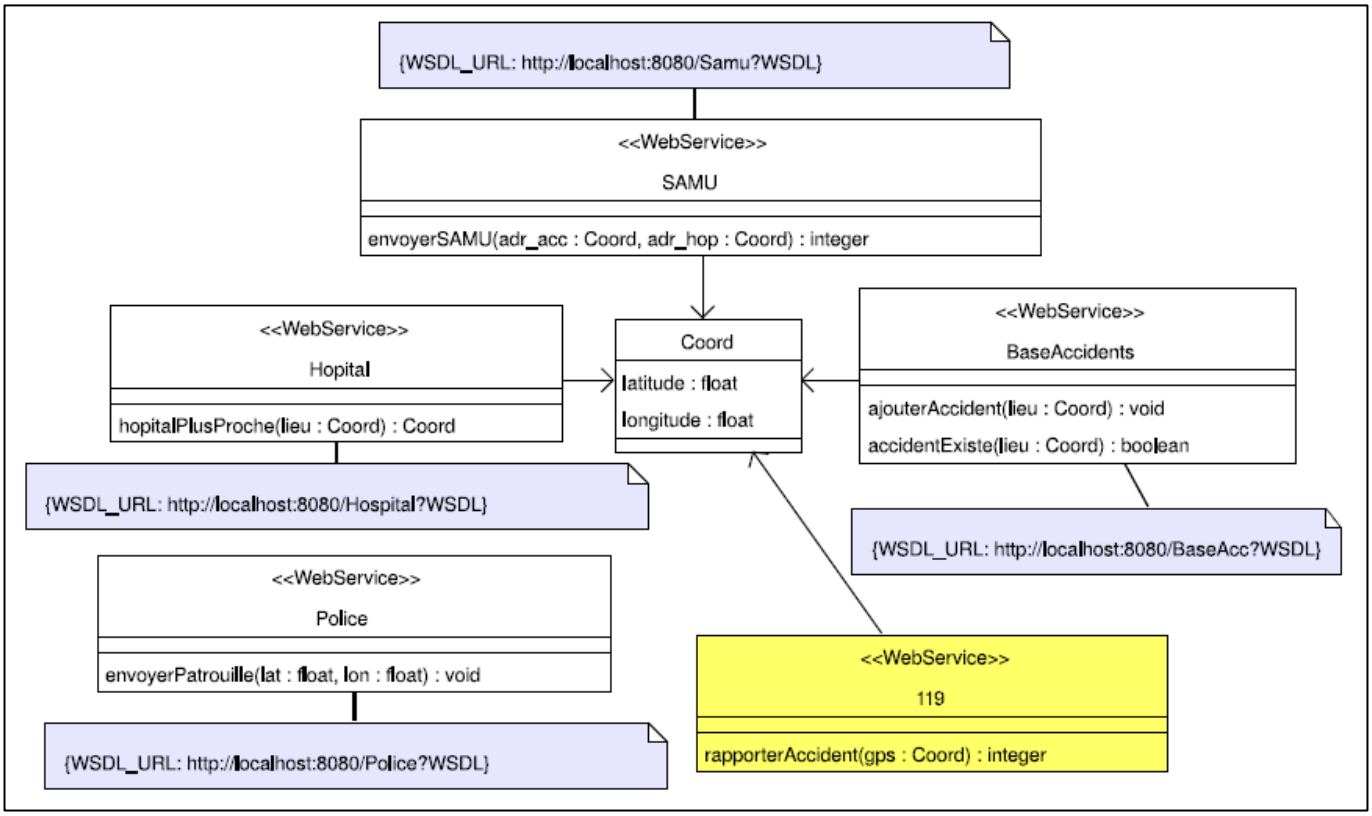

Fig 4.22 : Diagramme de classes UML-S.

Lorsque le diagramme de classes est achevé, nous obtenons une représentation statique ou structurelle des services Web impliqués dans la composition. Il convient désormais de fournir une représentation dynamique ou comportementale de la composition. Plus précisément, nous allons spécifier, sous la forme d'un diagramme d'activité UML-S, le comportement interne de la méthode rapporterAccident(). Le diagramme d'activité correspondant au scénario prédéfini et qui est réalisé manuellement à l'aide de l'outil de modélisation générée à partir du métamodèle. Sur le diagramme d'activité de la figure 4.22, les nœuds initiaux et finaux correspondent respectivement à l'appel et au retour de l'opération rapporterAccident() du service composé 119. Puisque l'opération prend la position de l'accident en paramètre, nous avons assigné le stéréotype « Input » au noeud initial et nous avons déclaré en valeur étiquetée la variable lieu_acc à laquelle la valeur passée en paramètre sera assignée. De la même manière, puisque l'opération rapporterAccident() retourne une valeur, nous avons assigné le stéréotype « Output » au nœud final et nous avons indiqué en valeur étiquetée le nom de la variable dont la valeur sera retournée, ici délai.

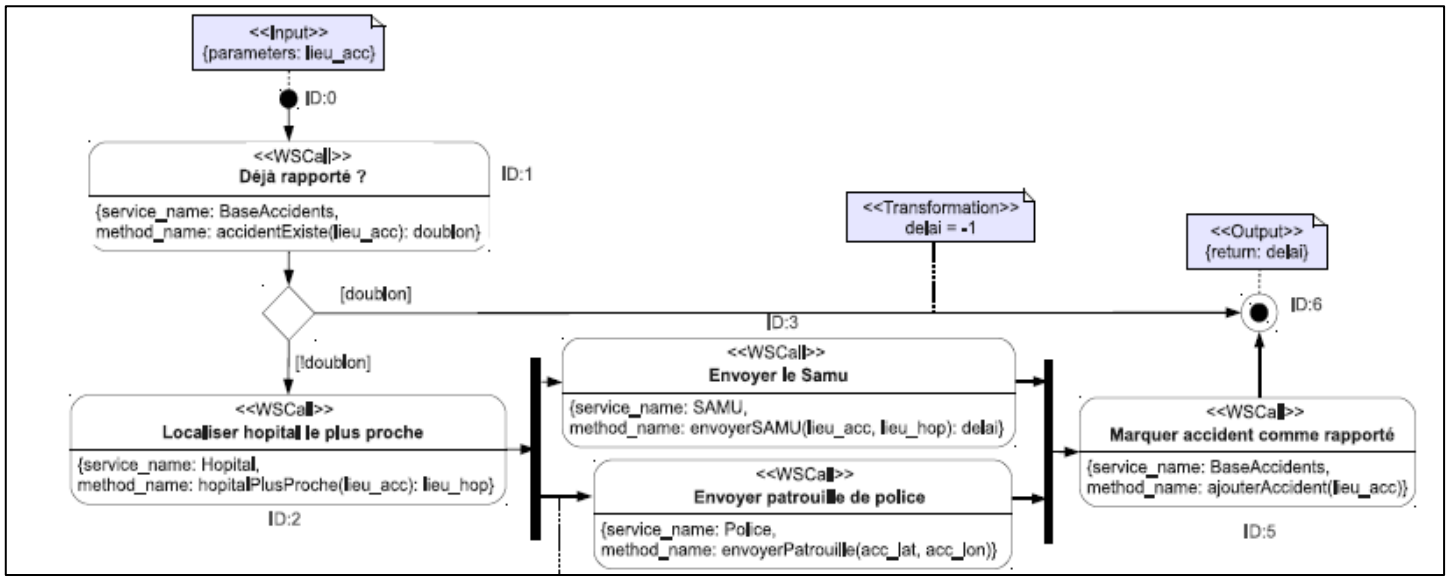

Fig 4.23 : Diagramme d'activité UML-S.

Les autres nœuds de l'activité sont des actions qui représentent des appels ou invocations à des services Web existants. Ces actions sont représentées graphiquement sous la forme de rectangles aux bords arrondis. Ces invocations sont également caractérisées par le stéréotype « WSCall » et ses valeurs étiquetées service_name et method_name qui contiennent respectivement le nom du service Web appelé et le nom de la méthode invoquée parmi celles publiées par le service. Chaque action « WSCall » fait donc obligatoirement référence à une classe « WebService » du diagramme de classes et plus spécifiquement à une opération déclarée dans cette classe. Le workflow est composé des actions suivantes :

- Tout d'abord, le service BaseAccidents est invoqué afin de savoir si l'accident situé à lieu a déjà été rapporté. Le résultat de cet appel est une valeur booléenne que nous assignons à la variable doublon.
- Nous utilisons alors cette variable comme condition de garde au niveau des branches sortantes d'un nœud décisionnel (losange clair). Dans le cas où doublon est évalué comme étant vrai, cela signifie que l'accident a déjà été rapporté et que le service n'a donc aucun traitement à effectuer. Nous assignons a la variable délai une valeur négative, avant de la retourner au niveau du nœud final. Cette création de variable est effectuée à l'aide d'un comportement de transformation, représenté par un rectangle au coin supérieur-droit replié et identifiable grâce au stéréotype « transformation ».
- Dans le cas où doublon est évalué comme étant faux, il s'agit d'un nouveau rapport d'accident et le système doit alors le traiter. Ce traitement consiste d'abord à invoquer le service Hopital afin de déterminer la position de l'hôpital le plus proche de l'accident.
- Nous assignons la position de l'hôpital sélectionné à la variable lieu_hop. Les services SAMU et Police sont alors appelés de manière concurrentielle. Le parallélisme est représenté graphiquement par des barres verticales noires et épaisses.
- La méthode envoyerSAMU() du service SAMU prend en paramètre la position de l'accident (lieu_acc) et la position de l'hôpital (lieu_hop), puis retourne le temps de réponse prévu du SAMU que nous assignons à la variable delai.
- Nous effectuons ensuite une synchronisation entre les appels aux services SAMU et Police afin d'attendre leur terminaison avant de marquer l'accident comme rapporté et traité. Ce marquage est réalisé via un appel à la méthode ajouterAccident() du service BaseAccidents. Enfin, la valeur stockée dans la variable délai est retournée à l'utilisateur afin qu'il connaisse le délai d'intervention des secours.

- **Génération du code**

  Le début est caractérisé par la réception de la requête client et l'invocation du service BaseAccidents afin de vérifier si l'accident a déjà été rapporté. La réception de la requête client est ici représentée par l'instruction <receive> qui fait référence au nom de l'opération invoquée (rapporterAccident). La position de l'accident est ici passée par le client en paramètre (rapporterAccidentIn) et celle-ci est ensuite affectée à la variable lieu_acc, comme précisé sur le diagramme d'activité. La position est également affectée à l'entrée de l'opération accidentExiste, c'est à dire accidentExisteIn, avant son invocation à l'aide de l'instruction <invoke>. Enfin, nous affectons la valeur retournée par l'opération à la variable doublon. L'affectation d'une variable ou d'un de ces

membres à une autre variable est réalisé en BPEL à l'aide de l'instruction <assign>. Le service BaseAccidents nous a donc retourné ici une valeur booléenne indiquant si cet accident a déjà été rapporté ou non.

```
<?xml version="1.0" encoding="UTF-8"?>
    <process name="911"xmi:version="2.0"xmlns="bpel20">
    <partnerLinks>...</partnerLinks>
    <variables>...</variables>
    <sequence>
    <receive createInstance="yes" operation="RapporterAccident" partnerLink="Client" variable"rapporteraccidentIn"/>
    <assign>
    <copy>
    <from variable="rapporteraccidentIn"/>
    <to variable="lieu_acc"/>
    </copy>
    </assign>

    <invoke partnerLink="BaseAccident"
    operation="AccidentExist"
    inputVariable="AccidentExistIn"
    outputVariable="doublon"/>
    <assign>
    <copy>
    <from variable="AccidentExistIn"/>
    <to variable="doublon"/>
    </copy>
    </assign>
```

Fig 4.24 : Réception de la requête et invocation du service BaseAccidents.

Le processus doit ensuite réaliser un choix exclusif basé sur la valeur de la variable doublon. Dans le cas où l'accident a déjà été rapporté et donc où la variable doublon contient la valeur true, le processus affecte simplement une valeur négative à la variable délai.

Dans le cas où l'accident n'a pas déjà été rapporté, la variable doublon possède la valeur false. Ce code contient tout d'abord l'invocation de l'opération hopitalPlusProche du service Hopital afin de localiser l'hôpital le plus proche du lieu de l'accident. Comme indiqué sur le diagramme

```
/***taking a decision***/
<switch>

<case condition="accident non rapporte">
<invoke partnerLink="Hopital"
operation="HopitalPlusProche"
inputVariable="lieu_acc"
outputVariable="lieu_hop"/>
<assign>
<copy>
<from variable="lieu_acc"/>
<to variable="lieu_hop"/>
</copy>
</assign>
```

Fig 4.25 : Appel du service HOPITAL.

d'activité, la variable lieu_acc est passée en paramètre à l'opération et sa valeur de retour est affectée à la variable lieu_hop.

L'étape suivante consiste à invoquer de manière concurrentielle les services SAMU et Police. L'exécution concurrentielle d'activités est réalisée en BPEL à l'aide de l'instruction <flow>. Les invocations des opérations envoyerSAMU et envoyerPolice à l'aide d'instructions <invoke> ont donc été placées dans un <flow>. L'opération envoyerSAMU retourne ici le délai de réponse de l'équipe de secours qui est affecté à la variable délai, fidèlement au diagramme d'activité.

```
<flow>
<sequence>


<invoke partnerLink="SAMU"
operation="envoyer SAMU"
inputVariable="envoyer SAMUIn"
outputVariable="delai"/>
<assign>
<copy>
<from variable="envoyer SAMUIn"/>
<to variable="delai"/>
</copy>
</assign>

<invoke partnerLink="Police"
operation="envoyer patrouille"
inputVariable="envoyer patrouilleIn"
outputVariable="envoyer patrouilleOut"/>
<assign>
<copy>
<from variable="envoyer patrouilleIn"/>
<to variable="envoyer patrouilleOut"/>
</copy>
</assign>
</sequence>
</flow>

</case>
</switch>
```

Fig 4.26 : Appel des services SAMU et Police.

Un <flow> permet de réaliser à la fois les structures de branchement multiple et de synchronisation. En effet, les actions figurant après le <flow> ne sont exécutées qu'après synchronisation des branches parallèles d'exécution. Elle fait appel à l'opération ajouterAccident du service BaseAccidents juste après le comportement de synchronisation.

```
<assign>
<copy>
<from variable="lieu_acc"/>
<to variable="ajouter accidentIn"/>
</copy>
</assign>
<invoke partnerLink="Base Accidents"
operation="ajouter accident"
inputVariable="ajouter accidentIn"
outputVariable=""/>
```

Fig 4.27 : Ajout de l'accident.

La dernière partie du code BPEL fait référence à la réponse au client. Il s'agit ici de la réponse à l'invocation par le client de l'opération rapporterAccident du service 119. Comme indiqué sur le diagramme d'activité, la valeur de la variable délai est ici retourné au client afin que celui-ci connaisse le délai prévu dans l'arrivée des secours sur le lieu de l'accident. La réponse au client est réalisée en BPEL à l'aide de l'instruction <reply>.

```
<reply operation="rapporter accident" partnerLink="Client"
 variable="rapporter accidentOut"/>
</sequence>
</process>
```

Fig 4.28 : Réponse au client.

IV.5. Conclusion

Dans ce chapitre nous avons proposé une approche automatique pour transformer les diagrammes d'activités d'UML vers BPEL. La méthode proposée se base sur la transformation de modèles vers code, et utilise l'outil de modélisation et de métamodélisation multi-formalismes ATOM^3. Afin de réaliser cette méthode, nous avons proposé un méta-modèle des diagrammes d'activités et une grammaire. Le méta-modèle permet de générer un outil visuel de modélisation des diagrammes d'activités, et la grammaire transforme ces derniers en code BPEL équivalent. Enfin, nous avons donné un exemple pour illustrer cette transformation.

CONCLUSION

CONCLUSION

Le travail présenté dans ce mémoire s'inscrit dans le domaine de l'ingénierie dirigée par les modèles. Il se base essentiellement sur l'utilisation combinée de méta modélisation et de transformation de modèle.

Le résultat de notre travail est une approche automatique pour transformer les diagrammes d'activités vers le langage d'orchestration BPEL. L'approche proposée est basée sur la transformation de modèle vers code, et est réalisée à l'aide de l'outil ATOM³.

Le travail est réalisé dans deux étapes :

1. La première étape consiste à proposer un méta-modèle des diagrammes d'activité, afin de générer un outil visuel permettant la modélisation de ces diagrammes.
2. La deuxième étape propose une grammaire de graphe permettant de générer automatiquement le code BPEL sous forme textuelle, à partir des diagrammes d'activités graphiques.

Comme perspectives, nous comptons dans le but de valider notre approche, nous devrons vérifier les diagrammes obtenus en BPEL. Et afin d'arriver à développer une approche totalement automatique, incluant tous les diagrammes UML, nous proposerons de continuer la transformation des autres diagrammes UML (diagramme de séquence, diagramme de cas d'utilisation, diagramme d'état/transition, etc. …) vers le langage BPEL en utilisant toujours la transformation de modèles vers code et l'outil ATOM³.

BIBLIOGRAPHIE

[1] : W3C World Wide Web Consortium; “Extensible Markup Language” (XML) http://www.w3.org/XML/ (20-02-2017)

[2] : W3C World Web Consortium “Soap version1.2 Partie1: Structure pour l’échange de message, Recommandation W3C 24 june 2003” https://www.w3.org/2002/07/soap-translation/soap12-part1.html (20-02-2017)

[3] : W3C World Wide Web Consortium; “Web Services Description Language (WSDL) 0http://www.w3.org/TR/2005/WD-wsdl20-adjuncts-20050803/ (20-2-2017)

[4] : Site du consortium OASIS: http://www.oasis-open.org/home/index.php (20-02-2017)

[5] : W3C World Wide Web Consortium; “Web Services Architecture”; W3C Working Group Note 11; February 2004; http://www.w3.org/TR/ws-arch/ (25-02-2017)

[6] : “WSDL Usage Experience with XML Schema” http://www.w3.org/2005/05/25-schema/WSDL.html (25-02-2017)

[7] : Exemple du fichier de description d’un service web (WSDL) http://staff.um.edu.mt/cabe2/supervising/undergraduate/owlseditFYP/TemperatureService.wsdl (25-02-2017)

[8] : W3C World Wide Web Consortium; “XML Schema” http://www.w3.org/XML/Schema (25-02-2017)

[9] : UDDI Executive Overview: Enabling Service-Oriented Architecture, disponible à http://www.uddi.org/whitepapers.html (25-02-2017)

[10] : Découvrir et décrire un service Web en utilisant UDDI, 1ère partie https://msdn.microsoft.com/fr-fr/library/aa480517.aspx (26-02-2017)

[11] : Les services web https://openclassrooms.com/courses/les-services-web (26-02-2017)

[12] : Peltz, Chris. Web Services Orchestration: a review of emerging technologies, tools, and Standards. Hewlett Packard, Co. Janvier 2003. (30-02-2017)

[13] : Sanlaville, Sonia. Environnement de procédé extensible pour l’orchestration Application aux services web. Thèse de doctorat, Université Joseph Fourier, Décembre 2005. (30-02-2017)

[14] : MOHAMMED FAICAL ABOUZAID Analyse formelle d’orchestration de services web. Thèse de doctorat : Ecole polytechnique de Montréal. (30-02-2017)

[15] : Bernard Coulette Etat de l’art sur le développement logiciel bas sur la transformation de modèles https://www.researchgate.net/profile/Bernard_Coulette/publication/220575143_Etat_de_l'art_sur_le_developpement_logiciel_base_sur_les_transformations_de_modeles/links/0deec534fb73835450000000.pdf (30-02-2017)

[16] : Meta Object Facility MOF https://fr.wikipedia.org/wiki/Meta-Object_Facility (01-03-2017)

[17] : Unified Modelling Language UML http://www.uml.org/ (01-03-2017)

[18] :Object Constraint Language OCL http://www.omg.org/spec/OCL/ (01-03-2017)

[19] : Xavier Blanc. MDA en action. EYROLLES, March 2005. (01-03-2017)

[20] : Généralités sur l'approche MDA http://laine.developpez.com/tutoriels/alm/mda/generalites-approche-mda/ (02-03-2017)

[21]: K. Czarnecki and S. Helsen, "Classification of Model Transformation Approaches", OOPSLA'03 Workshop on Generative Techniques in the Context of Model Driven Architecture, 2003. (01-03-2017)

[22] : Atom^3 http://atom3.cs.mcgill.ca/ . (01-03-2017)

[23] : J. de Lara and H. Vangheluwe, "AToM3 : A tool for multi-formalism and metamodelling," in FASE (R.-D. Kutsche and H. Weber, eds.), vol. 2306 of Lecture Notes in Computer Science, pp. 174–188, Springer, 2002. (05-03-2017)

[24] : J. de Lara, H. Vangheluwe, and M. Alfonseca, "Meta-modelling and graph grammars for multi-paradigm modelling in AToM3," Software and System Modeling, vol. 3, no. 3, pp. 194–209, 2004. (05-03-2017)

[25] : G. Booch, J. Rumbaugh and I. Jacobson, *"The Unified Modeling Language User Guide*", Publisher: Addison Wesley, First Edition October 20, 1998. (05-03-2017)

[26] : L. Perochon, *"UML : langage graphique de modélisation"*, Juin 2009, http://www.projet-plume.org/ressource/uml(INRA) (05-03-2017)

[27] : *"OMG Unified Modeling Language TM (OMG UML), Superstructure", Février 2009,* URL Standard du document : http://www.omg.org/spec/UML/2.2/Superstructure (08-03-2017)

[28] : L. Audibert, *"UML 2", edition 2007-2008,* adresse du document : http://www-lipn.univ- paris13.fr/audibert/pages/enseignement/cours.htm (08-03-2017)

[29] : Uml profile diagrams : http://www.uml-diagrams.org/profile-diagrams.html (01-04-2017)

[30] : Christophe Dumez Approche dirigée par les modèles pour la spécification, la vérification formelle et la mise en œuvre de services web composés. Thèse de doctorat. L'université de technologie de Belfort-Montbéliard. (01-02-2017)