ORIGINAL ARTICLE

NOUVELLE APPROCHE FONCTIONNELLE DE TRANSFORMATION AUX SPECIFICATIONS ABSTRAITES DES SERVICES WEB BASEES SUR LES AFD A PARTIR DES PROGRAMMES BPEL

NEW FUNCTIONAL APPROACH TO TRANSFORMING ABSTRACT SPECIFICATIONS OF WEB SERVICES BASED ON AFD FROM BPEL PROGRAMS

| Khawla Bouafia *[1] | Ali Khebizi [2] | and | Bálint Molnár [3] |

[1.] Eötvös Loránd University | Département des systèmes d'information | Faculté d'informatiques | Budapest | La Hongrie|
[2.] Université 8 Mai 45 | Département d'Informatique | Guelma | Algérie |
[3.] Eötvös Loránd University | Département des systèmes d'information | Faculté d'informatiques | Budapest | La Hongrie |

| Received Marchl 02, 2020 | | Accepted April 17, 2020 | | Published April 22, 2020 | | ID Article | Khawla-Ref.1-ajira060420|

RESUME

Aujourd'hui, les services web sont devenus très utilisés notamment pour les entreprises afin de rendre accessibles leurs processus métiers et leurs données via le Web. Dans de tels environnements ; la composition des services web est un sujet qui suscite l'intérêt des chercheurs et des utilisateurs. Elle offre la possibilité de traitement de procédures complexes, même avec des services existants simples, tout en coopérant entre eux. Toutefois, cette tâche reste très complexe et nécessite pour son accomplissement des techniques, des langages et des outils pour sa vérification. Le sujet que nous abordons dans cet article entre dans ce contexte et vise à proposer une approche qui consolide l'aspect vérification des programmes écrits dans le langage de composition BPEL. En même temps l'approche permet d'extraire des spécifications formelles reflétant le comportement contenu dans les programmes BPEL. Dans notre travail nous proposons une approche de transformation de modèle pour convertir un code BPEL vers un automate d'états finis déterministe (AFD). Dans cette optique, nous abordons le problème en utilisant l'approche de transformation verticale basée sur le pattern Matching. Enfin, nous présentons un outil logiciel qui implémente l'approche proposée.

***Mots-clés** : Service Web, Composition de services Web, Pattern, BPEL, Automates d'états finis déterministe (AFD), Transformation de modèles.*

ABSTRACT

Nowadays, web services have become very used especially by organizations to make available their business and data via the Web. The composition of web services is a topic of interest among researchers and final users. It offers the possibility of handling complex business even with simple existing services while cooperating with each other. However, this task is very complex and requires for its fulfillment techniques, languages, and verification tools. In this paper we deal with such context and we aim to suggest an approach that manages BPEL programs verification and which allows extracting formal specifications contained in BPEL programs. Additionally, the resulting abstract model is very useful for maintenance and verification tasks. In our work we propose a model transformation approach to convert a BPEL code to a finite state machine. With this in mind, we address the problem using the vertical transformation approach based on pattern matching. Finally, we present a software tool that implements our proposed approach.

1. INTRODUCTION

La naissance de l'Internet a bouleversé la façon dont les entreprises gèrent leurs activités et la manière avec laquelle elles communiquent avec l'environnement. En effet, plusieurs technologies ont été introduites ces dernières années pour renforcer la communication et le partage des ressources à travers le web. Ces avancés ont mené les experts du domaine à inventer et à découvrir de nouvelles méthodes pour résoudre certains nombres des problèmes informatiques, tels que l'échange des données et l'invocation des applications à distances.

Parmi ces techniques qui sont apparus au début du 21ème siècle, et qui ont été adoptées par les entreprises, c'est la technologie des services Web [1, 2]. Les services Web sont des briques logicielles de bases et grâce à leurs nombreux avantages, ils sont utilisables à distance via n'importe quel type de plate-forme. Ils peuvent servir au développement d'applications distribuées et sont accessibles depuis n'importe quel type de clients.

Par ailleurs, les services Web appartiennent à des applications capables de collaborer entre elles de manière transparente pour l'utilisateur, car leur mise en œuvre repose sur une architecture décentralisée. Cette architecture orientée services (SOA) est un style architectural fondé sur la description des services et de leurs interactions. En ce sens, la composition de services web est un processus qui permet d'assembler plusieurs services afin de créer une nouvelle fonctionnalité à forte valeur ajoutée. La technique de composition des services Web a besoin de langages et modèles pour gérer les processus métiers complexes. BPM est une approche globale qui permet de gérer le cycle de vie de ces processus métiers.
Néanmoins, les différentes situations, la gestion des processus métiers pour des raisons diverses exigent de faire des transformations de et vers des modèles (réingenering, maintenance, collaboration) et les langages des processus métiers. Plusieurs recherches ont largement traité la problématique de la transformation de modèles [3, 4 ,5, 6], mais l'exploration de cette littérature fait ressortir qu'aucune étude ou approche n'a été proposée pour permettre la

***Corresponding author Author & Copyright Author © 2020: | Khawla Bouafia |.** All Rights Reserved. All articles published in *American Journal of Innovative Research and Applied Sciences* are the property of Atlantic Center Research Sciences, and is protected by copyright laws CC-BY. See: http://creativecommons.org/licenses/by-nc/4.0/.

transformation de programmes écrit en BPEL en des descriptions formelles, particulièrement, en automate d'états finis déterministe (AFD) [7].

Pour remédier à cette insuffisance, nous focalisons dans cet article à l'étude de cette problématique. Ce sujet est motivé par les raisons suivantes: (i) Simplifier la collaboration entre partenaires dans le cadre des échanges entre eux par l'exploitation d'outils graphiques. (ii) Permettre la vérification du programme BPEL, après la transformation, en utilisant les outils formels de vérification (model-checking). (iii) Passage d'un programme de bas niveau vers un modèle formel qui sera plus adéquat à des taches de maintenance et d'évolution.

L'approche que nous proposons s'articule autour des règles de transformation qui permettent de passer du programme écrit en langage BPEL vers des structures explicitant les éléments descriptifs de l'automate à construire. Pour atteindre cet objectif, nous avons identifié un ensemble de situations (portions de code BPEL) qui sont récurrents. Ces situations sont modélisées sous forme de patterns. L'approche proposée a été implémenté, dans un environnement qui concrétise la mise en œuvre des patterns. En plus, l'outil logiciel développé réalise des fonctions très utiles aux gestionnaires de protocoles, telles que : générateurs de code BPEL, vérification et visualisation de certaines propriétés de l'automate.

L'article compose apes l'introduction d'un état de l'art, la troisième partie contient problématique et motivations de notre travail après les diffèrent notions et concepts de base utilisent, la cinquième section constitue notre contribution présente les fondements de l'approche fonctionnelle proposée, Implémentation et expérimentation sont présentés dans la dernière section, à la fin on présente une conclusion et les perspectives à ce travail.

2. État de l'art

La discipline informatique manipule largement le concept de modèle. Avec le temps plusieurs types de modèles de conception ont connu les jours, pour faire face à différents problèmes de conception. Cependant, cette tendance a engendré une variété de modèles avec différents niveaux d'expressivité et d'abstraction.

Cette situation a induit un besoin pour des techniques de transformations inters modèles. Dans le contexte des services, les modèles existants permettent de représenter les différentes vues (protocoles des services, composition, exécution...) associées aux étapes du cycle de vie du service. Ils évident que les communautés, les concepteurs et les utilisateurs n'utilisent pas les mêmes modèles ni langages, d'où la nécessité de trouver des mécanismes de passage d'un modèle / langage à un autre.

Différents travaux de recherche ont abordé d'une manière générale le problème de la transformation. Ces approches varient de la transformation entre les différents types de modèles ou de modèle vers langage ou bien encore de langage vers modèle. Dans ce qui suit nous citons ces travaux et les outils industriels proposés dans le domaine des transformations. Les travaux les plus proches de ce travail et qui sont généralement liés aux transformations de modèles sont récapitulés dans la table A noter qu'il existe plusieurs autres outils qui offrent des transformations inters modèles. Mais, aucune tentative de transformation de code BPEL vers des automates n'a été faite.

Tableau 1: Le tableau présente la synthèse de classification des transformations existantes.

References	Modèle source	Modèle cible	Commentaries et insuffisances
[3,4]	BPMN	BPEL	Les aspects graphiques comme les partitions et les sous-partitions des modèles BPMN sont perdus.
[8]	UML	BPEL	La transformation ne génèrent pas les artefacts BPEL : fault Handlers; event Handlers; compensation Handler; jeter; rethow; compenser ; ensemble de corrélation.
[9]	S-GNet	BPEL	Manque certains détails de déclaration d'éléments de processus dans le code BPEL obtenu.
[10]	AUML	BPEL	N'offre aucune structure formelle pour vérifier, valider et évaluer les performances du système avant les phases d'implémentation.
[10]	BPEL	Pi-calcul	Le pi-calcul de l'algèbre des processus qui nécessite la vérification de protocole à l'aide de la logique et des outils pour corriger et valider la modèle.
[3]	BPEL	RDP	Le problème du blocage et la vivacité dans les RDP.
[5]	RDP-TE	AHL	Le problème de ce travail concerne les mesures de performances et des vérifications de propriétés de vivacité ainsi que l'analyse de l'atteignabilité.
[6]	UML	GSPN	Les outils de transformation utilisés ne sont pas visuels comme AGG ou ATOM3, et la modèle résultant après la transformation n'est pas optimal et n'accepte pas l'analyse et la vérification avec des outils d'analyse comme TINA.
[11]	BPEL	multi-agent JADE	Besoin des moteurs d'orchestration des services Web à deux niveaux : dynamicité et la décentralisation.
[12]	UML state machine	YAWL	Pas optimal et aucune an alyse et vérification après la transformation.
[13]	AFD	Hypergraph	Manque de vérifications des propriétés de l'AFD.

Après l'analyse de l'état de l'art et les travaux connexes, les travaux existants, aussi bien dans le domaine de la recherche académique que dans le monde industriel, la notion de transformation et ses approches la prochaine section explique notre problème, les motivations de ce travail

3. PROBLEMATIQUE RT MOTIVATION

En ce sens, nous avons opté pour les automates d'états finis déterministes comme "modèle cible de représentation" en raison de plusieurs avantages. En parallèle, le "langage source" à traiter est le langage de composition BPEL. Le choix des services Web implémentés entant que programme BPEL est plus motivé. La question qui se pose dans ce contexte est : comment opérer la transformation du langage source vers la modèle cible ?
Dans la figure suivante, nous illustrons cette problématique :

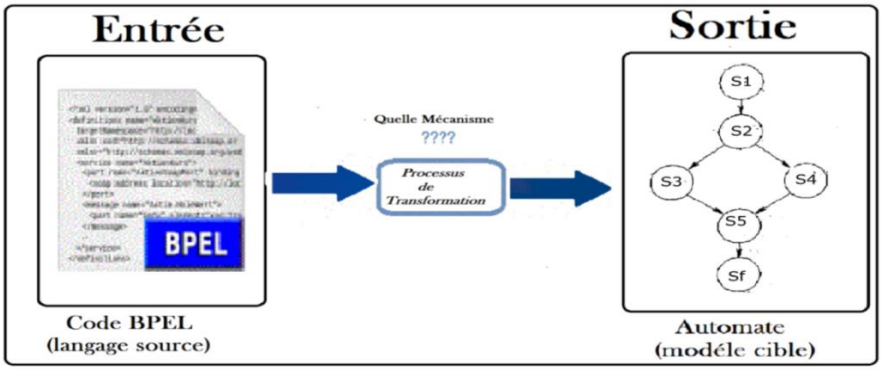

Figure 1 : Problématique.

Fondamentalement, la translation de modèle est une transformation d'un modèle ou langage vers un autre modèle ou langage équivalant, mais dans une représentation différente. Une des raisons pour l'utilisation de la translation de modèle est de faciliter l'interopérabilité et la communication. Il est souvent souhaitable d'utiliser et d'échanger des modèles entre les différents outils de modélisation et même entre les différents langages de transformation.

Comme illustré dans la figure 1, notre problématique consiste, justement, à partir d'un programme BPEL vers un modèle visuel de son comportement (plutôt que le simple code). La modèle cible permettrait de faciliter l'ajout de nouvelles fonctionnalités, de nouvelles contraintes ou règles et il servira d'outils de collaboration qui est facilement vérifiable. En d'autres termes, dans notre cas le problème traité est celui de prendre un code source écrit dans le langage de composition des services Web est de "convertir" sa partie abstract, qui exprime le comportement du service, en une représentation plus abstraite et qui est exprimée par un automate d'états finis déterministe.
Le sujet abordé dans cet article est très intéressant dans le domaine des services web et il est d'une importance capitale pour les raisons suivantes :

1. Passage d'un modèle textuel vers un modèle graphique : c'est-à-dire vers une représentation de haut niveau comme les automates d'AFD qui utilisent un formalisme solide, simple et populaire.
2. L'existence d'outils logiciels (software) de vérification (model-checking) pour les automates d'AFD : il est important par une action de transformation d'opérer des vérifications de certaines propriétés.
3. Répondre aux besoins d'évolution et de maintenance : assurer la maintenance et l'évolution des services Web qui sont des opérations très importantes dans le cycle de vie du service. On peut aussi facilement détecter les erreurs pour garder un contenu bien organisé sans bugs et qui est mis à jour.
4. Outils de communication et de collaboration : Simplifier la collaboration entre partenaires dans le cadre des échanges entre eux rapidement via des modèles graphiques compréhensibles pour les divers profils d'utilisateurs.
5. Processus d'ingénierie inverse (BPR) : dans plusieurs situations, repartir d'un code qui est déjà implémenté et opérationnel, en vue d'arriver à un modèle abstrait est une opération très pertinente pour les concepteurs. En ce sens, il est important de reconvertir la modèle comportemental d'un programme en l'exprimant via des automates d'états finis déterministes, qui seront plus adéquats à d'éventuelles activités de vérification, d'actualisation...
6. Recherche sélective de structures : les approches existantes de transformation traitent l'analyse du code d'une manière monolithique, ou le programme en entrée est considéré en tant que tout indissociable et un modèle cible correspondant est produit en sortie.

Or, dans plusieurs situations, les gestionnaires de protocoles peuvent être intéressés par des éléments particuliers pour les raisons de collaborations ou d'amélioration des règles de gestion. À titre d'exemple, un utilisateur veut savoir si son processus métier contient des boucles spécifiques à un état (c'est le cas de sélection d'articles dans le processus). Dans ce cas, il veut rendre la boucle limitée afin d'accorder une remise au client. Le système que nous voulons concevoir permettra de prendre en charge cette préoccupation. Ainsi, l'utilisateur pourra choisir une structure déterminée et vérifier son existence dans le code source.

D'une manière générale, le problème de la transformation consiste à fixer un modèle ou langage source et de se focaliser sur un autre modèle cible en spécifiant les mécanismes subséquents permettant d'atteindre cet objectif. Depuis les dernières années, l'utilisation de services Web a connu une popularité grandissante. Ces services sont très utilisés notamment par les entreprises pour rendre accessible leurs processus métiers ou leurs données via le Web. le problème du choix du modèle ou langage le plus communément utilisé et le plus adapté reste à spécifier. Une fois ce modèle cible fixé comme objectif, alors la source sur laquelle nous allons se concentrer pour atteindre la cible constitue notre premier souci.

Après avoir posé le problème et donner les motivations, dans la prochaine section nous allons donner la définition de transformation et les différentes classifications des types de transformations existantes.

4. NOTION ET CONCEPTS DE BASES

4.1 Notion de transformation de modèles et les patterns de construction

Dans [14], on trouve les définitions suivantes :

Définition : une transformation est la génération automatique d'un modèle cible à partir d'un modèle source, suivant une spécification de la transformation.

Après avoir illustré la notion de transformation, on s'intéresse à présent à la notion de pattern qui est aussi très utile dans notre contexte. On va la présenter ci-après :
En génie logiciel, un patron de conception (design pattern en anglais) est un concept destiné à résoudre des problèmes récurrents. Il décrit des solutions standards pour répondre à des problèmes d'architecture et de conception des logiciels. Ce n'est pas un algorithme, mais une structure générique qui permet de résoudre le problème identifié. Il est indépendant du langage de programmation, il est standardisé, suffisamment pour que tous puissent s'y référer. C'est l'expérience qui montre que la solution est valide. Ce qui nous ramène à la définition suivante :

Définition : les designs patterns ou motifs de conception sont des recueils de bonnes pratiques de conception pour un certain nombre de problèmes récurrents.

Selon cette définition les patterns de conception permettent d'assurer que les mêmes conditions ou phénomènes conduisent aux mêmes résultats ou effets.
Dans notre contexte, nous allons utiliser ce concept pour exprimer différentes situations qui conduisent à la déduction des mêmes éléments du protocole à construire. Ces situations à traiter sont explicitées par les diverses primitives contenues dans le code du programme BPEL.

4.2 Classification des approches de transformation de modèles

Dans [15], les auteurs ont publié une classification basée sur plusieurs critères. On distingue deux types d'approches
Pour la notion de transformation de modèle :

1. **Les transformations du type Modèles-Modèles :** ce type de transformation permet la génération de plusieurs modèles intermédiaires avant d'atteindre la modèle de code, afin d'étudier les différentes vues du système, son optimisation, la vérification de ses propriétés et sa validation. Nous distinguons cinq techniques de transformation modèles vers modèles :
    - Approche par manipulation directe
    - Approche relationnelle
    - Approche basée sur les transformations de graphes
    - Approche basée sur la structure
    - Approche hybride
2. **Les transformations du type Modèles-Code** Il existe deux types de transformations. Modèle vers Code : la première est basée sur le principe du visiteur où l'on se sert du modèle et ses éléments qui rapprochent sa sémantique du langage de programmation pour obtenir le code. La deuxième est basée sur le principe des patrons2 avec laquelle on accède aux informations du modèle source en utilisant les fragments de méta-code dans le code ou modèle cible [6].
3. **Classification multicritères** les auteurs présentent un état de l'art contenant une classification des différentes transformations de modèles selon les critères suivants :
    - Similarité des espaces techniques,
    - Transformation endogène / transformation exogène
    - Transformation horizontale / transformation verticale
    - Transformation syntaxique /transformation sémantique

On se basant sur les critères présidents, on constate que le problème abordé dans cet article consiste en une transformation du type "Modèle-code" basé sur les patterns, elle est exogène, verticale et syntaxique.

4.3 Les processus métiers et les AFDs

Un processus métier est un composant élémentaire du processus global dans la chaine logistique d'une organisation. Un exemple typique de processus métier est le traitement d'un ordre d'achat. Les activités constituant ce processus commencent par la prise en compte de la commande et se terminent par la livraison au client. Entre ces deux extrémités, des activités intermédiaires se déroulent selon un chemin connu à l'avance afin de garantir le fonctionnement correct et cohérent de ce processus [16].

Dans cette section, nous allons nous focaliser sur le concept de processus métier. Selon [17] les processus métiers sont définis comme suit :

Définition. Un processus métier est un ensemble d'activités, entreprises dans un objectif déterminé [18, 19, 20]. La responsabilité d'exécution de tout ou partie des activités par un acteur correspond à un rôle. Le déroulement du processus utilise des ressources et peut être conditionnées par des événements, d'origine interne ou externe. L'agencement des activités correspond à la structure du processus.

Toute entreprise doit clairement identifier et définir les procédures qui entrent en ligne de compte dans ses relations commerciales, sociales ou administratives. Lorsqu'une entreprise atteint cet état, elle a une base solide pour évaluer son cadre de travail et adopter des changements bénéfiques. Toute tâche qui n'appartient pas ou ne supporte pas un processus métier est considérée comme redondante ou inutile et peut être éliminée.

Modélisation du processus métier un modèle de processus métier consiste en un ensemble de modèles d'activités et de contraintes d'exécution entre eux. Cette modélisation représente la phase où l'importation des définitions des processus métiers dans des outils techniques est possible, afin d'ajouter le lien avec les applications du SI. Les capacités de modélisation des outils devraient permettre de déployer directement les processus modélisés sans passer par une phase d'implémentation.

Il existe beaucoup de modèles pour modéliser les processus métiers comme : AFD, les réseaux de pétri, UML…, mais nous sommes plus intéressées des automates.
Un automate est un modèle très connu dans le domaine de la spécification formelle de systèmes. Un automate est composé d'un ensemble d'états ou nœuds; un ensemble d'actions et un ensemble de transitions étiquetées entre les états. Les actions sont modélisées par des étiquètes et une transition étiquetée modélise ainsi l'exécution de l'action lors du franchissement de celle-ci [21]. Formellement un automate est décrit par : son alphabet de symboles, son état initial, ses états finaux, les états intermédiaires dans lesquels il peut se trouver, ainsi que les transitions, c'est-à-dire comment on passe d'un état à un autre.

De nombreux types d'automates existent: les automates à état finis, entrée/Sortie, temporisés..., L'utilisation des AFD pour la modélisation des processus métiers est motivée par les raisons suivantes :

- Ils sont faciles à comprendre par les utilisateurs.
- Appropriés pour décrire le comportement réactif et les systèmes dynamiques.
- Utilisé pour le contrôle d'exécution de services.
- Les automates sont des outils populaires et très répandus.
- L'existence d'outils logiciels pour leur vérification.
- Dotée d'une assise théorique solide permettant la vérification (model-checking).

C'est pour ces raisons que ce modèle de représentation sera utilisé pour représenter les processus métiers dans la suite de notre travail.

La prochaine section sera consacrée à la composition des services Web et nous mettrons l'accent sur BPEL, en tant que langage standard dans le cadre de la composition des services Web

4.4 La composition des services Web et le langage BPEL

La composition des services Web est un sujet qui a suscité l'attention des chercheurs qui tentent de fournir des modèles rigoureux, des langages et des plateformes pour la proposition des résultats efficaces pour cette issue. Plusieurs définitions ont été proposées pour le concept de composition, nous citons dans ce qui suit les plus communes et les plus acceptées : Dans [22] on trouve la définition suivante :
Définition : La composition des services Web est la capacité d'offrir des services à valeur ajoutée en combinant des services existants probablement offerts par différentes organisations. Les techniques de composition des services Web peuvent être décrites sous deux angles : la chorégraphie ou l'orchestration.

La définition d'une application Web repose sur la description du flot de données et du flot de contrôle. Pour cela de nombreux langages ont vu le jour pour tenter de solutionner le problème de la composition des services Web. Ces langages décrivent les interactions entre différents fournisseurs de services Web et leurs clients.
Le tableau suivant résume les différents langages de composition de services Web existants :

Tableau 2: le tableau montre l'aperçu des langages de composition des services Web.

Nom de langage	Désignation	Organisme
XLANG	eXtensible LANGuag	Microsoft
WSFL	Web Services Flow Language	IBM
EbXML	Electronic Business using eXtensible Markup Language	OASIS et UN/CEFACT
WS-CDL	Web Services Choreography Description Language	W3C
BPML	Business Process Modeling Language	BPMI
WSCI	Web Service Choreography Interface	BEA
BPEL	Business Process Execution Language	IBM, Microsoft et BEA
BPMN	Business Process Modeling Notation	BPMI et OMG

BPEL est le standard de description des processus métier. Et il est le mieux accepté par les développeurs des services. Le langage d'exécution des processus métier, est un standard nécessaire pour assurer les liens entre plusieurs processus métier. Il est donc conçu pour intégrer un grand nombre d'applications, publiées sous forme de services. Selon [23] BPEL est défini comme suit :

Définition : BPEL est un langage composition qui représente une convergence des deux langues de flux de travail, WSFL et XLANG. WSFL a été conçu par IBM et est basé sur le concept de graphes orientés. XLANG, combine les deux approches et fournis un vocabulaire riche pour la description des processus métiers.

Structurellement, BPEL correspond à une grammaire XML qui décrit des processus métiers.
La construction des boucles, la déclaration de variables, la copie et l'affectation des valeurs, etc.., est aussi possible. De plus, il est possible de combiner toutes ces constructions et de définir des processus métier complexes d'une manière algorithmique en manipulant les structures itératives et conditionnelles.

- Boucles While, Swich
- Boucles If-Then-Else, Repeat Until, For Each

Le langage BPEL [3] a plusieurs caractéristiques, on cite les plus importantes :

- Exprimer les processus métiers par un langage standard.
- Décrire la logique des processus métiers à travers la composition de services Web.
- Permettre d'annuler le traitement fait par une activité en cas d'échec.
- Les processus BPEL ont une durée de vie bien déterminée et leur cycle de vie peut être contrôlé afin de maintenir l'exécution des activités pour une longue période.

Au sein de l'entreprise, BPEL est utilisé pour normaliser l'intégration des applications d'entreprise, ainsi que d'étendre l'intégration aux systèmes hétérogènes. Entre entreprises, BPEL permet une intégration plus facile et plus efface avec les partenaires socioéconomiques.

5. FONDAMENT DE LAPPRROCHE FONCTIONNELLE PROPOSE

On va s'intéresser à la présentation d'une approche pour la transformation des programmes BPEL vers des spécifications abstraites des services Web. L'approche proposée est basée sur une transformation caractérisée par les propriétés suivantes :

1. Le passage du code source vers un modèle abstrait.
2. La transformation est exogène.
3. Elle est syntaxique dans le sens où elle n'ajoute pas de sémantique.
4. Elle est basée sur des patterns.
5. Elle est verticale. La transformation à proposer sera composée d'un ensemble de règles de conversion, qui permettent d'extraire les structures du modèle cible.

Nous rappelons que la modèle cible est une spécification formelle exprimée par des AFD, nous introduisons les deux concepts de base qui constituent le fondement de notre approche : la transformation de modèles et les patterns de construction,
Nous percevons la problématique de la transformation des programmes BPEL en spécifications formelles, comme étant le problème d'analyse du code source BPEL dans la perspective de la détection de certains motifs récurrents qui conduisent aux mêmes structures de l'automate à construire.

Tel qu'illustré dans la table, pour traiter ce problème nous proposons une approche fonctionnelle, dans le sens où elle est basée sur l'analyse des fonctions réalisées par le code source BPEL, essentiellement sa partie abstract, et nous abordons le problème en trois étapes.

- Premièrement, nous allons définir la règle générale de transformation et nous donnerons sa spécification formelle (niveau 1 de la table).

- Deuxièmement, nous allons identifier un ensemble de patterns de construction où chacun permettra d'aboutir à un élément particulier de l'automate cible (niveau 2 de la table).
- Une fois, les patterns identifiés ils seront exploités pour découvrir les structures cibles (niveau 3 de la table);

Tableau 3 : Le principe de l'approche proposée.

Niveau	Préoccupation	Objectifs
Niveau 1	Comment passer d'un langage vers un modèle	Spécification générale de la transformation
Niveau 2	Quelles sont les situations qui se répètent?	Formalisation et identification des patterns
Niveau 3	Quelles sont les conditions a satisfaite pour que le pattern se produit?	Instantiation des patterns

5.1 Règle générique de transformation

Une règle générique de transformation est une règle de passage (ou de conversion) qui permet de prendre une portion d'un code source identifié par des primitives spécifiques du langage BPEL et de chercher la structure de l'automate correspondante à ces primitives. A cet effet, la règle à utiliser aura comme attributs deux éléments :

- **La portée de la règle** : elle est introduite en entrée et permet de sélectionner (rechercher) uniquement les primitives concernées et identifiées comme attribut Source.
- **La correspondance** : c'est le résultat de l'application de la transformation exprimée comme élément de l'AFD Cible.

Plus formellement, soit la transformation T qui prend les éléments Source et qui les convertit en des éléments Cible de l'automate selon des conditions à satisfaire, alors on représentera la transformation par :

$$Source \xrightarrow{T} Cible \qquad (1)$$

Tels que :

- Source : portion de code d'un programme BPEL éventuellement avec des primitives spécifiques.
- Cible : la structure de l'automate à états finis déterministe correspondante à la portion du code Source.
- T : la transformation a réalisé, si les conditions sur les primitives de Source sont vérifiées.

5.2 Formalisation et identification des patterns de construction

La règle de transformation précédente est très générale et ne permet pas donc de détecter les éléments particuliers d'un AFD, tels que les branchements ou les séquences. Pour remédier à cette difficulté nous allons particulariser cette règle de transformation afin de l'adapter à différentes situations rencontrées dans le code source à analyser.
De ce fait, la notion de pattern expliquée précédemment est très utile dans cette perceptive. Nous commençons par donner la spécification formelle des patterns, puis nous allons présenter l'ensemble des patterns identifiés dans notre étude.

Spécification formelle des patterns

La transformation des programmes BPEL consiste à déterminer les primitives et les éléments correspondants dans l'AFD selon des patterns quelconques. Un pattern est représenté formellement comme suit :

$$Primitive \xrightarrow{PC} Element \qquad (2)$$

Tels que :

Primitive : est une primitive d'un programme BPEL

- Element : est un élément de l'automate à états finis déterministe à construire.
- PC: est un prédicat d'ordre 2 (deux paramètres) exprimant la sémantique du pattern. PC correspond à pattern de construction

La sémantique véhiculée par cette modélisation est la suivante : une structure nommée Élément est déduite d'un programme BPEL si : le prédicat : *PC (Primitive, Élément)* est évalué à Vrai. Ce prédicat est Vrai si les contraintes exigées par la description d'élément sont vérifiées dans la partie primitive. Autrement dit, si les primitives d'un programme BPEL expriment la structure de l'élément à construire.

Identification de l'ensemble des patterns : L'analyse approfondie des différents codes BPEL a permis de déceler un ensemble de patterns aboutissants toujours aux mêmes structures de l'automate.
Les patterns identifiés sont récapitulés dans la table suivante : pour chaque pattern identifié, on donnera : son principe général, sa représentation graphique, sa spécification formelle et sa sémantique.

Tableau 4 : Aperçu général des patterns identifiés.

Nom du Pattern	Label
Pattern de sequence simple	$PC1_s$
Pattern de sequence générale	$PC1_g$
Pattern de Parallélisme simple	$PC2_s$
Pattern de Parallélisme générale	$PC2_g$
Pattern de choix exclusif	PC3
Pattern de fusion	PC4
Pattern de cycle	PC5

A signaler que l'étape trois de l'approche proposée (instantiation des patterns) sera introduite au fur et à mesure de la présentation de chaque pattern.

1. *Pattern de Séquence d'activités (séquence)* : dans cette séquence, l'activité B est disponible une fois que l'activité A ait terminé son exécution. De la même manière, l'activité c'est disponible une fois que l'activité B soit terminée. La séquence d'activités dans BPEL contient une primitive s'appelant *sequence* pour désigner la succession des activités. Cette primitive permet de définir qu'un ensemble d'activités de BPEL sera exécutée séquentiellement. La spécification formelle du pattern de séquence est décrite comme suit :

$$PrimitiveSeque \xrightarrow{PC1s} (A1, A2) \qquad (\mathbf{3})$$

Tels que :

- Primitive Séquence : Portion du code dans un programme BPEL qui exprime la séquence d'activités
- A1, A2 : Succession correspondante des activités dans l'AFD
- PC1s : Le prédicat qui guide la transformation

La sémantique correspondante : une séquence d'activités entre A1 et A2 est détectée (A1 suit A2) dans un programme BPEL si : Le prédicat *PC1s(PrimitiveSéquence,(A1,A2))* est évalué à Vrai. Ce prédicat est Vrai si et seulement si la commande : sequence est détectée dans la portion de code en entrée.

Remarque : bien que très simpliste, ce pattern pourra être utilisé dans la construction des séquences de plusieurs activités. Il suffit de renommer les variables. A titre d'exemple on peut trouver la séquence A1, A2, A3, A4..., dans ce cas A2 devient A1 et A3 devient A2 après A3 devient A1 et A4 devient A2... Le pattern général est le suivant :

$$PrimitiveSeque \xrightarrow{PC1g} (\mathrm{A1}, \mathrm{A2}, \mathrm{A3}\ldots) \qquad (\mathbf{4})$$

2. *Pattern de Parallélisme (parallel split)* : Le parallélisme représente un point dans le processus où un lien de contrôle simple est divisé en plusieurs liens de contrôle s'exécutant en parallèle). Le principe général est : au moment de terminer l'exécution de la première activité A, le lien de contrôle passe aux activités B et C. BPEL contient un type d'activité nommé *Flow.* Ce type d'activités permet de spécifier les activités qui seront exécutées de façon simultanée, dans chaque chemin d'activité il est possible de rencontrer n'importe quel type de structures (séquence, choix exclusif, un autre parallélisme (récursivité)...).

Le pattern de parallélisme (parallel split) fait la transformation des activités incluses dans la structure flow d'un programme BPEL en un état correspondant à partir duquel sortent deux ou plusieurs activités à destination des états cibles de l'AFD à construire.
La spécification formelle est décrite par l'équation suivante :

$$PrimitiveFlow \xrightarrow{PC2s} \big(\mathrm{A1}, (\mathrm{A2}, \mathrm{A3})\big) \qquad (\mathbf{5})$$

Où :

- PrimitiveFlow : est la primitive Flow dans un programme BPEL
- A1, A2, A3 : sont des activités qui vérifient la règle suivante : l'état atteint par A1 est un état Si / ∈ F à partir duquel sortent les deux activités parallèles A2 et A3. Évidemment ces deux dernières auront leurs propres états cibles.
- PC2s : le prédicat qui permet de détecter l'activité flow.

La sémantique correspondante de cette primitive est : l'état si cible de l'activité A1 à partir duquel sortent les activités en parallèle A2 et A3 est déduit d'un programme BPEL si : Le prédicat *$PC2_s$ (PrimitiveFlow,(A1,(A2,A3)))* est évalué à Vrai. Ce prédicat est Vrai si et seulement si la commande *flow* est rencontrée dans le code BPEL associé aux activités.

Remarque : Bien que la spécification contenue dans l'équation (5) ne prend en charge que deux activités, elle peut être généralisée pour exprimer le parallélisme de n activités. Le pattern général est le suivant :

$$PrimitiveFlow \xrightarrow{PC2g} (A1, A2, A3\ldots) \qquad (\mathbf{5})$$

3. Pattern de Choix Exclusif (exclusive choice) : c'est un point de contrôle de processus métier, il permet de faire un choix du chemin d'exécution qui sera décidé au moment de l'exécution. Cette décision est basée sur une information ou une donnée de gestion. Contrairement au parallélisme, une seule activité dans le flux de contrôle est invoquée. Le principe général correspond est : une fois l'activité A terminée, il existe une sélection pour le flux de contrôle pour décider de la continuité du processus : soit l'activité A, soit l'activité B. BPEL utilise le type d'activité switch pour permettre de réaliser le choix exclusif. Chaque chemin inclus dans le choix exclusif est contenu dans un élément case. Au moment de l'exécution du switch, chaque case contient la séquence d'activités qui sera exécutée si la condition liée aux cases est vraie. Il est à signaler que dans chaque chemin il est possible de rencontrer n'importe quel type de structure (séquence, parallélisme, un autre choix (récursivité)). Le switch peut utiliser également un autre élément : le *otherwise*. Cet élément est optionnel, il représente un chemin qui sera choisi par défaut si aucune case ne contient une condition vraie. Du fait que cet élément est optionnel, s'il n'existe pas explicitement dans la définition de processus, le moteur génèrera un lien *otherwise* avec un type d'activité empty. Le pattern de choix exclusif fait la transformation des activités associées aux switch du programme BPEL vers une activité divisée en plusieurs activités dans l'AFD. Sa spécification formelle est donnée par l'équation suivante :

$$PrimitiveSwitch \xrightarrow{PC3} (\mathrm{A1}, (\mathrm{A2}, \mathrm{A3})) \qquad (\mathbf{6})$$

Tels que :

- PrimitiveSwitch : est le primitive switch dans un programme BPEL ;
- A1,(A2,A3) : sont des activités qui vérifient la règle suivante : l'état atteint par A1 est un état si $\notin$ F à partir duquel sortent les deux activités A2 et A3, évidemment ces deux dernières auront leurs propres états cible (S1,S2) ;
- PC3 : le prédicat qui permet de détecter l'activité switch. La sémantique de ce pattern est : L'état si cible de l'activité A1 à partir duquel sortent les activités A2 et A3 est déduit d'un programme BPEL si : le prédicat *PC3 (PrimitiveSwitch,(A1,(A2,A3*) est évalué à Vrai. Ce prédicat est Vrai si et seulement si les primitives sont les commandes suivantes : *switch* et pick.

4. Pattern de Fusion simple (simple merge) : La fusion simple est un point dans le processus dans lequel une ou plusieurs branches du flux de contrôle se joignent sans nécessité de synchronisation. Le principe général de la fusion simple est représenté dans lors de la terminaison de l'activité A, ou l'activité B, ou des deux à la fois, le processus continue avec l'activité C. Cette construction est représentée dans un programme BPEL par la fin d'une activité de type switch ou flow. Pour compléter son exécution, ce type d'activité attend la terminaison de la branche activée parmi les branches qui y sont connectées. La spécification formelle du pattern de fusion est décrite comme suite :

$$PrimitiveFusion \xrightarrow{PC4} (\mathrm{A1}, (\mathrm{A2}, \mathrm{A3})) \qquad (\mathbf{7})$$

Tels que :

- PrimitiveFusion : portion du code dans un programme BPEL qui exprime la fusion des activités ;
- (A1, A2), A3) : sont des activités qui vérifient la règle suivante : les états atteints par A1 et A2 (si $\notin$ F) sont des états fusionnés dans un seul état ;
- PC4 : est un prédicat qui permet de détecter la fusion.

La sémantique correspondante est : Les états si atteints par les activités A1 et A2 conduisent vers un seul état dans l'AFD si : Le prédicat PC4 (PrimitiveFusion, (A1, A2), A3) est évalué à Vrai. Ce prédicat est Vrai si et seulement s'il existe une fusion d'activités dans un programme BPEL après branchement des activités tel que Flow et switch.

5. Pattern des Cycles (Les boucles) : Une boucle est une structure qui permet de répéter les mêmes instructions plusieurs fois. Tout comme pour les conditions, il y a plusieurs façons de réaliser des boucles. Au bout du compte, cela revient à faire la même chose : répéter les mêmes instructions un certain nombre de fois. La représentation générale de la boucle est représentée par : On lit les instructions de haut en bas (comme d'habitude) puis, une fois arrivé à la fin de la boucle, on repart à la première instruction, puis on recommence à lire les instructions et répéter de la même manière jusqu'à ce que la condition de boucle ne soit pas vérifiée. Le pattern de boucle fait la transformation de l'activité while dans un programme BPEL vers une boucle dans l'AFD. Le pattern de boucle est représenté formellement comme suit :

$$PrimitiveBoucle \xrightarrow{PC5} (\mathrm{A}) \qquad \mathbf{(7)}$$

Tela que :

- PrimitiveBoucle : portion du code dans un programme BPEL qui exprime la boucle while ;
- A : l'activité concernée par la boucle ;
- PC5 : le prédicat qui, une fois satisfait permet de reconnaitre la boucle.

La sémantique correspondante de ce pattern des boucles dans l'AFD est déduite dans un programme BPEL si: Le prédicat PC5 (Primitive Boucle, A) est évalué à Vrai. Ce prédicat est Vrai si et seulement si la commande while est détectée.

Si la boucle contient une seule activité alors la boucle devient une boucle simple (selfloop) c'est-à-dire itère sur le même état.
Remarque : Il est à signaler que de l'AFD de par sa nature, ne prend pas en charge la synchronisation des activités. Donc, les primitives de synchronisation d'activités de BPEL ne sont pas transformées en structures correspondantes dans l'AFD.

Après avoir identifié le problème de la transformation des programmes BPEL en spécifications abstraites des services Web avec la proposition d'une approche basée sur les patterns de construction pour construire un AFD qui représente le comportement externe du service Web. Les patterns identifiés génèrent à des structures descriptives pour spécifier l'AFD correspondant. En effet, sept patterns relatifs aux structures de contrôle (boucle, séquence, choix...) ont été identifiés et formalisés pour chacun l'expression logique permettant l'évaluation à vrai du prédicat a été exprimée. La prochaine section sera consacrée à l'implémentation de l'approche proposée.

5.3 Présentation de l'environnement de travail

Dans le développement de notre application, les différentes fonctionnalités à réaliser ont besoin de différents outils. Ces outils seront présentés dans cette section ainsi que les raisons qui ont motivé leurs choix.

Eclipse et Java : Comme environnement de programmation nous avons choisi le couple Eclipse/Java. Eclipse est un environnement de développement intégré (IDE) gratuit. Il permet de développer des programmes en plusieurs langages : Java, PHP.... Il utilise le principe intéressant des plugins, ce qui permet d'ajouter les fonctionnalités selon les besoins (l'extensibilité). C'est un IDE à complétion contextuelle, c'est-à-dire qu'il aide à corriger les erreurs syntaxiques du code.

Le langage Java, né en 1995, est un langage orienté objets, il permet d'écrire de façon simple et claire des programmes portables sur la majorité des plateformes. De plus, il bénéficie d'une très grande bibliothèque de classes avec lesquelles l'utilisateur pourra composer des interfaces graphiques, créer des applications multithreads, animer une page HTML par des applets ou encore communiquer en réseau. De plus Java assure une totale indépendance des applications vis-à-vis de l'environnement d'exécution : c'est-à-dire que toute machine supportant Java est en mesure d'exécuter un programme sans aucune adaptation (ni recompilation, ni paramétrage de variables d'environnement). Il est à signaler que malgré la grandeur de la bibliothèque native de Java, certaines fonctionnalités, tel que la manipulation des documents XML, l'accès à une base de données et bien d'autres, nécessitent des APIs qui ne sont pas intégrés dans Java par default, on les appelle les APIs externes. Donc il faut les intégrer et cela se fait d'une manière très simple à travers l'IDE Eclipse. Dans notre travail, les APIs externes utilisées sont :

- **JDOM :** est une API permettant la manipulation (création, modification, suppression...) de document XML via une structure arborescente. Il présente quelque similitude avec DOM (Document Objet Model). Cependant, il s'en distingue parce que JDOM est spécifiquement conçu pour JAVA et que du point de vue du développeur JAVA, il s'avère beaucoup plus pratique à utiliser (à ce propos, il convient de noter que contrairement à ce qui est parfois écrit, le J de JDOM suit la nomenclature NAA- not en abréviation de Sun : JDOM veut ainsi dire JDOM et rien d'autre). La raison qui nous a poussé à l'utiliser est sa simplicité et la possibilité de sa manipulation avec java.
- **Mysql-connector** : L'API qui permet l'accès aux bases de données sous java se nomme JDBC. Son fonctionnement requiert un divers qui peut être de différents types (il existe cinq types de drivers). Comme nous utilisons mySQL en guise de SGBD alors le type 04, pour plus de performances en termes de vitesse et d'utilisation des ressources, est adéquat à notre application mysql-connector est une API qui fournit ce driver le type 04 conforme à MySQL. Chaque SGBD à son propre driver. C'est une fichier exécutable jar (Java ARchive) qui englobe toutes les classes nécessaires à l'accès à la base de données.

Le choix XML : XML est devenu omniprésent dans le monde de l'informatique. De nombreux standards sont apparus et permettant à des applications différentes de stocker mais surtout de partager des documents. Ce succès de XML est en grande partie dû à ses qualités qui sont :

- Séparation stricte entre contenu et présentation ;
- Simplicité, universalité et extensibilité ;
- Format texte avec gestion des caractères spéciaux ;
- Structuration forte ;
- Modèle de document (DTD et schémas XML) ;
- Format libre : facilité d'exporter un document XML sous divers autres formats pdf, html... ;
- Existence d'outils de manipulation : JDOM, DOM... .

Dans notre application, nous l'utilisons pour la représentation des automates qui modélisait les BP. Après avoir présenté l'environnement de réalisation de notre application, nous passons à la présentation de son architecture et ses fonctionnalités [19].

6. IMPLEMENTATION ET EXPREMENTATION

Afin de vérifier la faisabilité de l'approche proposée, dans cette section où présentera son implémentation. Le prototype logiciel développé est succinctement décrit et quelques aspects pratiques sont montrés

6.1 L'architecture du système

Cette section présente l'architecture générale de notre application ainsi que les interfaces principales avec les grandes fonctionnalités. Nous avons aussi testé et opéré quelques expérimentations détaillant les différents modules qui constituent son architecture

Architecture du système L'application BPEL-AFD (transformation des programmes BPEL vers des automates d'états finis déterministe (AFD)), que nous avons implémenté contiennent sept modules qui manipulent les informations contenues dans deux bases de données. La figure suivante, présente une vue globale de l'architecture de notre système ainsi que l'interaction entre les divers composants

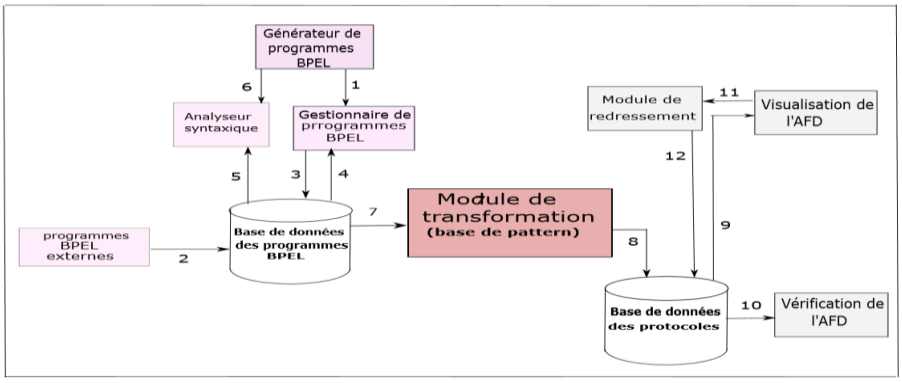

Figure 2 : le figure montre l'architecture du système BPEL-AFD.

L'architecture de notre système contient deux bases de données, à savoir :

- **Une base de données des programmes BPEL** : pour enregistrer les programmes écrit par le générateur de programmes BPEL, ou les programmes après leur modification, ainsi que les programmes importés ;
- **Une base de données XML (BDD XML)** : en fait c'est un ensemble de données xml représentant les protocoles. Ceci facilite la manipulation des éléments (états et transitions) de ces derniers. Un exemple de description d'un AFD sous forme XML est donné dans la figure suivante :

```
-<Automate>
 -<Etats>
    <Etat Type="Initial" Position="(249,68)">Etat01</Etat>
    <Etat Type="Simple" Position="(172,182)">Etat02</Etat>
    <Etat Type="Simple" Position="(329,258)">Etat03</Etat>
    <Etat Type="Final" Position="(212,353)">Etat04</Etat>
  </Etats>
 -<Transitions>
    <Transition Etat_Source="Etat01" Etat_Destination="Etat02" Position="(211,125)">Message0</Transition>
    <Transition Etat_Source="Etat02" Etat_Destination="Etat03" Position="(250,220)">Message1</Transition>
    <Transition Etat_Source="Etat03" Etat_Destination="Etat04" Position="(271,305)">Message2</Transition>
  </Transitions>
</Automate>
```

Figure 3: le figure montre la description d'un AFD sous forme XML.

La structure conceptuelle des deux bases de données est illustrée dans la Figure suivante :

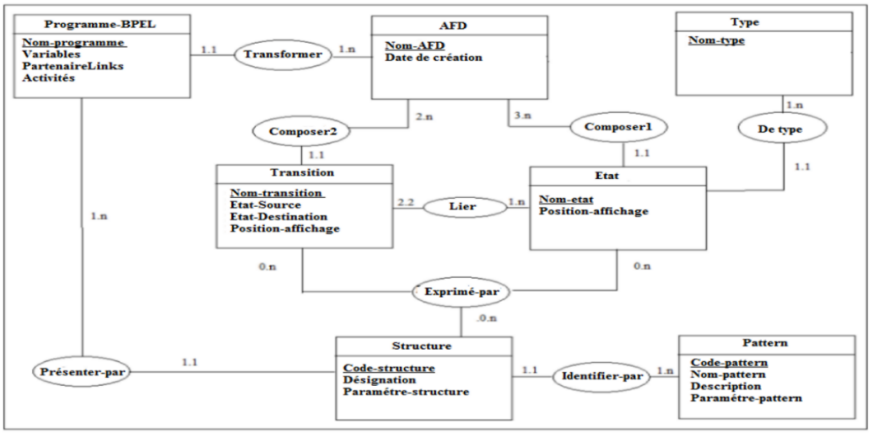

Figure 4: La figure montre le MCD du domaine.

Le MLD correspondant au MCD est le suivant :

Programme-BPEL (Nom-programme, Variables, PartenaireLinks, Activités, Nom-AFD*).
AFD (Nom-AFD, Date de création).
Etat (Nom-etat, Position-a-chage, Nom-AFD*, Nom-type*).
Type (Nom-type).
Transition(Nom-transition,Etat-Source,Etat-Destination,Position-a-chage,Nom-AFD*, Nom-Etat*).
Structure(Code-structure,Désignation,Paramétre-structure,Nom-programme*,Code-pattern*).
Pattern(Code-pattern, Nom-pattern, Description, Paramétre-pattern).
Exprimé-par(Nom-etat, Nom-transition, Code-structure).

Les modules de la figure 2, permettent de réaliser différentes fonctions d'entrée/sortie, de vérification et évidemment, la fonction principale de transformation du code en AFD

1. Le module générateur des programmes BPEL donne la main pour créer un programme complet qu'on peut modifier, supprimé à l'aide du module gestionnaire de programmes BPEL.
2. Les programmes BPEL externes (importés) seront enregistrés dans la base de données des programmes BPEL.
3. Les programmes BPEL générés seront enregistrés dans la base de données des programmes BPEL.
4. Les anciens programmes déjà existant dans la base des programmes BPEL sont passés par le gestionnaire.
5. Le module analyseur syntaxique utilise les programmes BPEL déjà existants pour détecter leurs anomalies.
6. Les programmes générés peuvent passer par le module analyseur syntaxique.
7. Le module principal de cette architecture qui transforme les données enregistrées dans la base des programmes BPEL.
8. Les programmes BPEL transformés en protocoles (AFD) seront enregistrés dans la deuxième base de données des protocoles.
9. Les protocoles résultants peuvent être visualisés.
10. Vérification des protocoles des AFDs est possible pour détecter les erreurs si elles existent.
11. Si l'AFD n'est pas claire, le module de redressement donne la main de modifier manuellement, puis l'enregistrer dans la base des protocoles.

Dans la suite de la section nous allons exposer les fonctionnalités de notre système.

6.2 Fonctionnalités du système :

L'application BPEL-AFD que nous avons implémentée contient plusieurs fonctionnalités permettant de réaliser les tâches suivantes :

1. **Génération des programmes BPEL** : c'est une interface graphique qui permet de créer des programmes BPEL d'une manière interactive.

2. **Gestionnaire de programmes BPEL** : Le gestionnaire des programmes BPEL est un module nécessaire dans notre application. Il permet de faire la manipulation des programme BPEL, cette manipulation est représentée par un ensemble de fonctions qui sont :

    - Visualisation des programmes BPEL : est une fonction dont le but est d'afficher tous les codes des programmes ou bien un seul programme existant dans la base de données des programmes BPEL. Pour chaque visualisation on donne les noms des programmes existants, la date de dernière modification et leurs tailles.
    - Mise à jour des programmes : c'est une fonction qui consiste à faire la modification interne dans le programme qui peut être : ajouter, supprimer et renommer le nom ou l'opération d'une activité de programme.

3. **Vérification** : c'est une fonction qui permet de faire une analyse syntaxique des programmes BPEL et d'afficher les anomalies existantes.

4. **La transformation en automate** : Ce module est le noyau de notre travail. Il permet de produire un automate graphique correspond au code BPEL sélectionné (code complet) ainsi que l'automate correspondant à une portion de code représentant des activités quelconques (flow, switch, séquence...). Cette transformation est représentée par l'ensemble des fonctions suivantes :

    - Affichage graphique : cette fonction affiche l'automate correspondante à un programme BPEL sélectionné.
    - Création graphique : c'est une fonction qui permet de faire la création du protocole métier.

5. **Mise à jour protocole :** c'est une fonction pour la modification interne dans le protocole métier qui peut être : ajouter, renommer et supprimer un état ou une activité.

6. **Vérification :** c'est une fonction qui permet de faire une analyse syntaxique et formelle de l'automate (état non joignable, état n'accédant pas à des états finaux, absence d'un état initial ou final...).

Une autre fonction est la transformation des éléments descriptifs du protocole (états, transitions) et de les stocker dans la base de données XML. Ce gestionnaire de protocoles communique avec une base de données de protocoles, pour leur chargement et leur enregistrement.

7. DISCUSSION

7.1 Scénario d'utilisation de BPEL-AFD :

Dans cette section, nous allons présenter un scénario réel d'exploitation de notre système. Les déférentes étapes sont détaillées d'une manière, ci-après :

1. **La création ou la sélection du code BPEL :** Avec l'application BPEL-AFD nous pouvons créer un nouveau programme à partir du Menu (Gestionnaire des programmes BPEL→ Générateur des programmes). La figure, représente l'interface correspondante à la génération interactive des programme BPEL :

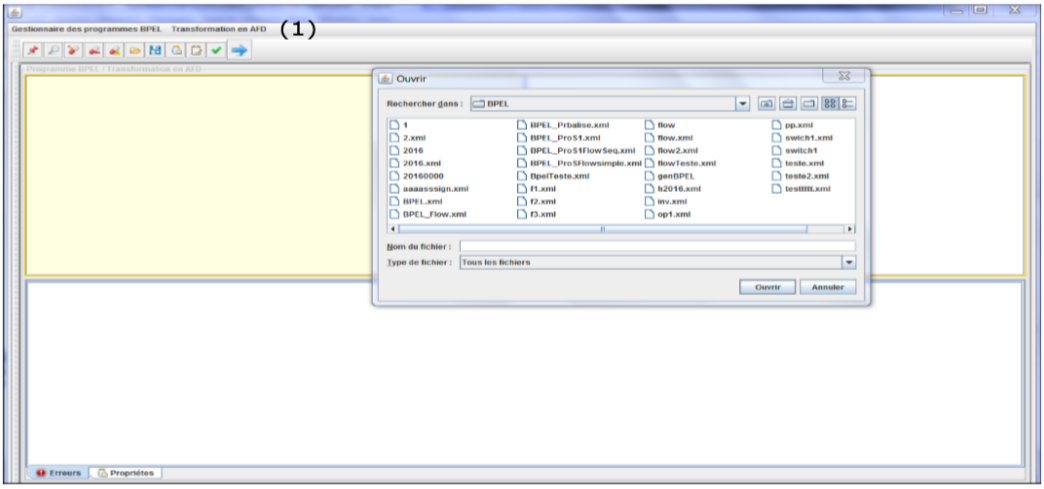

Figure 5 : Interface pour la création (généré) d'un programme BPEL.

2. **La visualisation des programmes BPEL** : Après la création et l'enregistrement d'un programme, on a une fonction à partir du Menu (Gestionnaire des programmes BPEL → Visualiser) qui nous permet de visualiser les programmes. Cette fonction affiche un tableau (A) contenant le nom de programme (1), date de création (2) et leur taille (3).

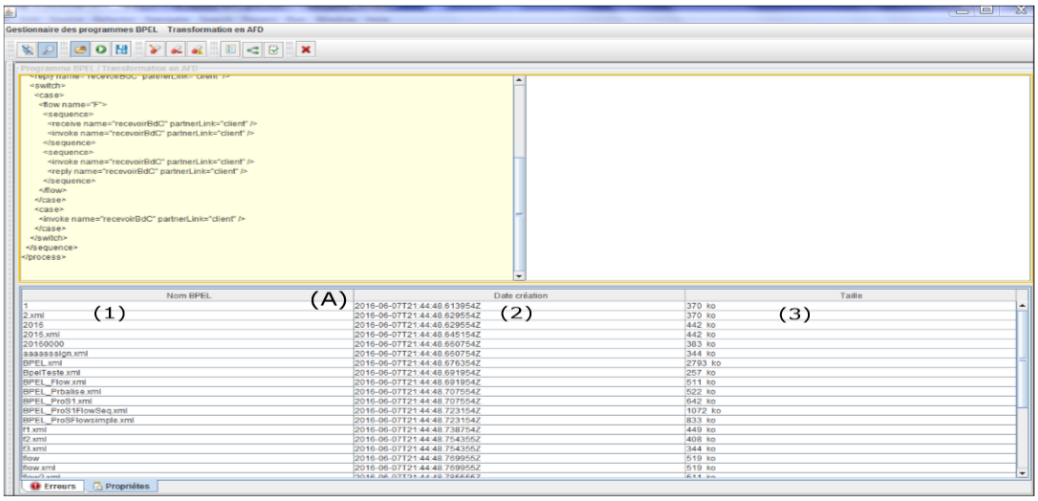

Figure 6 : Interface pour la visualisation des programmes BPEL

3. **Mise à jour des programmes :** Après l'enregistrement d'un programme, on peut le modifier (supprimer, ajouter des activités...) à partir du Menu (Gestionnaire des programmes BPEL→ Opération sur les programmes) et on pourra choisir l'opération de suppression ou l'ajout des éléments BPEL (boucles, partenaires, variables ...).

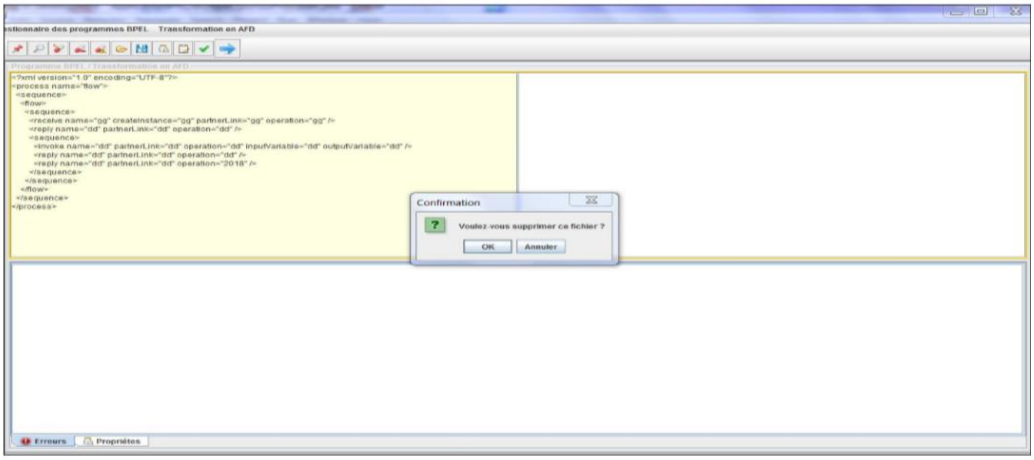

Figure 7 : Interface pour la mise à jour du programme BPEL

4. **Transformation des programmes BPEL en AFD :** Trois types d'analyses sont offertes par notre système BPEL-AFD : Analyse fine, analyse sélective et analyse globale. #
La figure présente l'interface pour l'application d'une transformation de programme BPEL en AFD. A partir du Menu (Transformation en AFD → Analyse fine), cette fonction fait l'analyse de programme et extrait les activités existantes. Après que les activités et les patterns soient affichés, l'utilisateur sélectionne l'élément désiré et clic sur le bouton >>> (A) qui déclenche la visualisation de la portion de code correspondante. Une fois cette portion enregistrée, elle pourra faire l'objet d'un chargement ultérieur en vue de sa transformation.

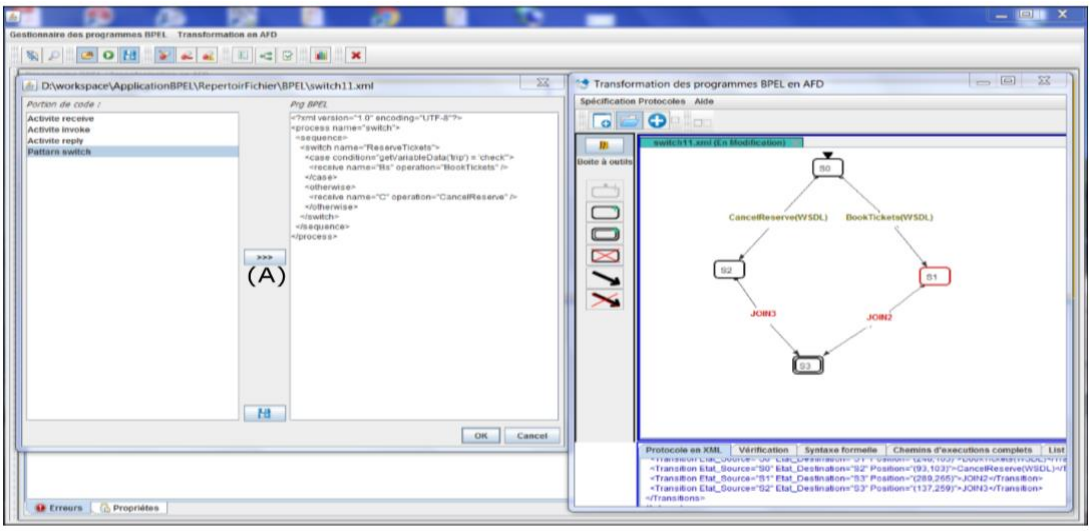

Figure 8 : Interface pour la transformation basée sur l'analyse fine.

Dans la figure dessus on montre une interface pour l'application d'une transformation de programme BPEL en AFD à partir de Menu (Transformation en AFD → Analyse sélective), on fait l'analyse de programme et on teste l'existence des patterns (A) dans le programme en cours d'analyse et on affiche la portion de code si le pattern existe sinon le message (cet élément n'existe pas dans ce programme) est affiché.

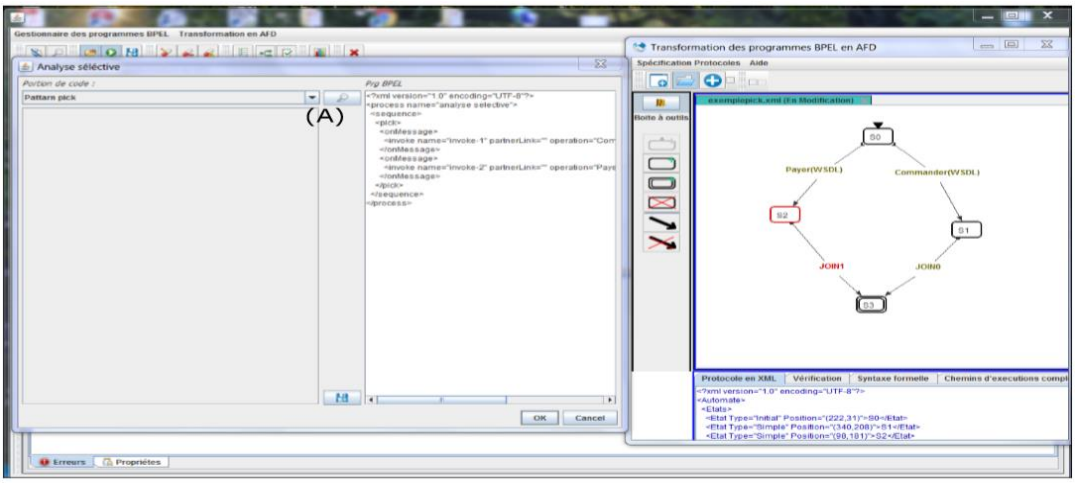

Figure 9 : interface pour la transformation basée sur l'analyse sélective.

Une autre variante est de faire une analyse globale. A partir de Menu (Transformation en AFD → Analyse globale) ou fait l'analyse du programme BPEL d'une manière monolithique. C'est-à-dire en tant que tout et l'AFD résultant sont produits en sortie.

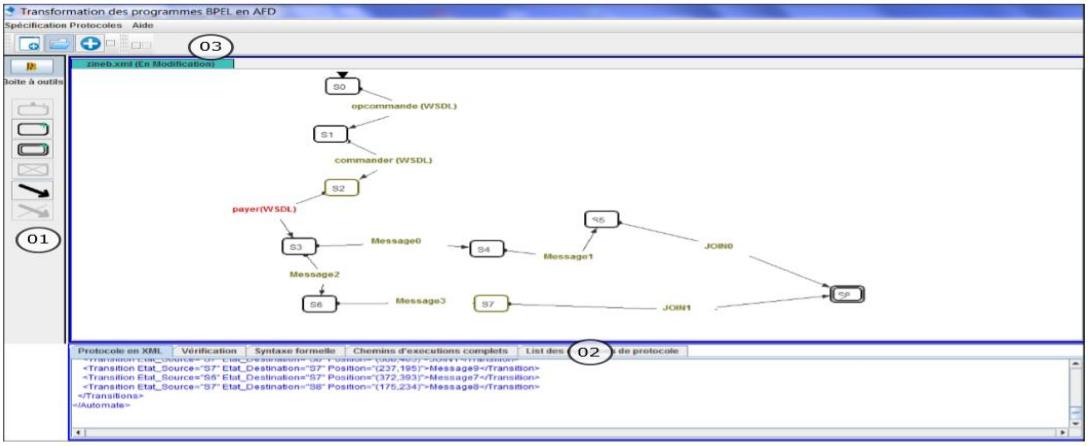

Figure 10 : Interface pour la transformation basée sur l'analyse globale.

A partir de la boîte à outils (01) on peut créer, modifier, supprimer (état ou transition) ou faire l'évolution d'un protocole métier (édition graphique du protocole). En même temps il existe un analyseur qui fait une vérification syntaxique de l'automate, une syntaxe formelle, une présentation XML et les différents chemins présents dans notre protocole. Ses différentes informations sont présentées dans le panneau d'en bas (02). A la fin en peut l'enregistrer ou ouvrir un nouveau protocole (automate) à partir du Menu (03), le programme est enregistré dans la base de données sous format XML (base des protocoles).

5. Redressement : Vu que l'affichage graphique de l'AFD est encombré et qu'il existe des zones surchargées et chevauchées, ce qui rend son exploitation complexe, alors un outil de redressement de l'affichage est proposé. L'utilisateur pourra améliorer l'affichage en cliquant / glissant sur les états et les transitions de façon à rendre l'automate plus claire et après il l'enregistre dans la base des protocoles. La figure suivante, expose un cas en cours de redressement :

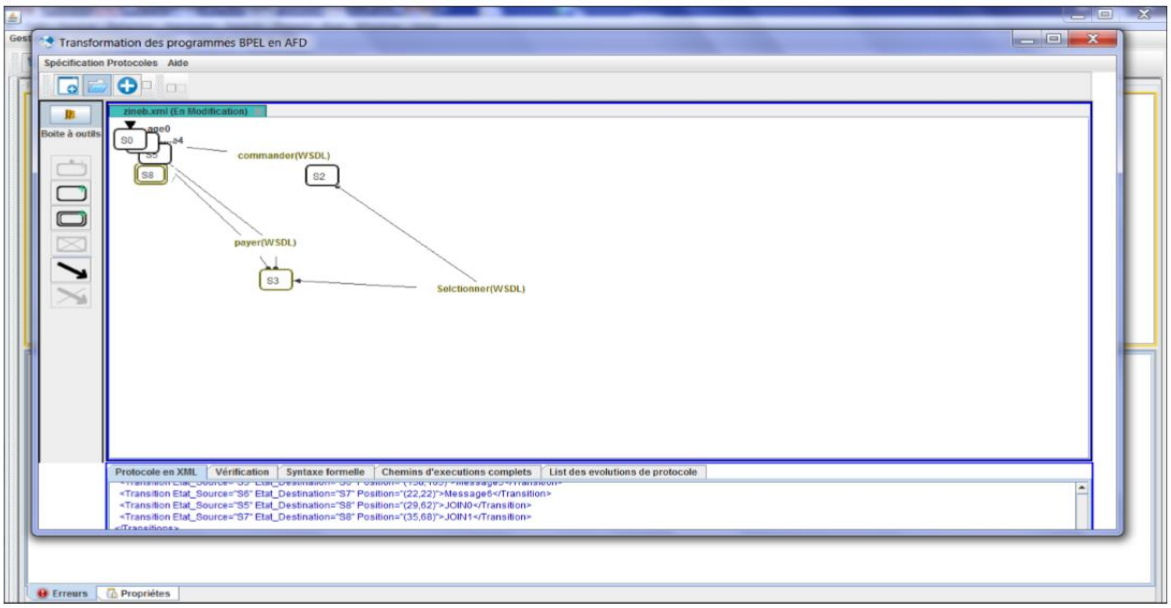

Figure 11 : Interface de redressement de l'affichage de l'AFD

7.2 Expérimentation

Pour l'expérimentation de notre prototype nous avons choisi une base de données des programmes contenant plus de 40 programmes BPEL et ou mesurant la variation du temps d'exécution.

Figure 12 : Interface de redressement de l'affichage de l'AFD

Comme le montre la figure, ce temps évolue d'une manière croissante cela est justifié par la taille et la complexité des programmes à tester.

Dans cette section, nous sommes parvenus à réaliser une application qui répond parfaitement aux objectifs fixés au début. En effet, nous avons implémenté un outil logiciel qui prend en compte les aspects conceptuels proposés dans la section des patterns. La finalité du prototype réalisé est d'extraire l'AFD correspondant au programme BPEL. Plusieurs fonctions sont aussi offertes à l'utilisateur. Cependant, comme toute application dans sa première version, beaucoup restent à faire (optimisation automatique de l'affichage de l'automate, interface, contrôle...etc) an de l'améliorer.

8. CONCLUSION

Dans cet article, nous avons traité un sujet afférent au domaine des services Web. Nous nous sommes intéressés à la problématique de la reconstruction des protocoles des services Web à partir des programmes écrits dans le langage de composition des services Web qui est BPEL.

Après avoir décrit le problème, nous avons adopté pour une approche fonctionnelle basée sur un ensemble de patterns qui permettent de convertir des portions de programmes BPEL en des sous- structures de l'automate cible à construire. L'approche proposée a été implémentée dans un outil logiciel qui à partir d'un programme BPEL en entrée produit l'automate d'états finis déterministe correspond. Le résultat en sortie est présenté sous différentes formes (formelle, XML, graphique) et des actions de vérification sont faites pour vérifier sa correction.
A la fin, comme perspectives à ce travail nous projetons de :

1. Améliorer l'outil logiciel et raffiner l'affichage graphique de l'automate d'états finis déterministe.
2. Exploiter un formalisme plus solide qui prendra en change le problème de l'attente (exprimé par le wait en BPEL), les automates temporisés, par exemple.
3. Étudier la possibilité de considérer, non seulement le flux d'activité, mais aussi les données véhiculées par ces activités lors des transformations. En ce sens les modèles de processus orientés donnés (Date driven processus) s'avèrent très adéquats pour relever ce défi.
4. Proposer un modèle pour la gestion des exceptions et des erreurs

Reconnaissance The project was supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.3-VEKOP-16-2017-00002) and by no. ED_18-1-2019-0030 (Application-specific highly reliable IT solutions) program of the National Research, Development and Innovation Fund of Hungary, financed under the Thematic Excellence Programme funding scheme

5. REFERENCES

1. MALESHKOVA, Maria, PHILIPP, Patrick, SURE-VETTER, York, et al. Smart Web Services (SmartWS)--The Future of Services on the Web. arXiv preprint arXiv:1902.00910, 2019.
2. JENSEN, Jesper. A systematic literature review of the use of semantic web technologies in formal education. British Journal of Educational Technology, 2019, vol. 50, no 2, p. 505-517.
3. Mohammed Oussama KHERBOUCHE. Contributions à la gestion de l'évolution des processus métier. PhD thesis, Décembre 2013.

4. Kholladi Mohamed Naoufel. Une Approche de transformation de la notation BPMN vers BPEL basée sur la transformation de graphe. PhD thesis, Décembre 2009.
5. Yamen Etouati, Moez Yeddes, Nejib Hadj Alouane, and Hassane Alla. Du réseau de petri temporel étendu vers les automates hybrides linéaires pour l'analyse des systèmes. In IEEE Conférence Internationale Francophone d'Automatique, CIFA 2009, pages 0104, 2009.
6. Mouna Bouarioua. Une approche basée transformation de graphes pour la génération de modèles de réseaux de Petri analysables à partir de diagrammes UML. PhD thesis, Avril 2013.
7. Molnár Bálint, Bouafia Khawla, (2017), "Adaptive case management and dynamic BP modeling a proposal for document-centric and formal approach." In: 12th International Symposium on Applied Informatics and Related Areas - AIS 2017
8. Ibm knowledge center. Uml to bpel transformations. http://www.ibm.com/ support/knowledgecenter/SS4JCV_7.5.5com.ibm.xtools.transform.uml2. bpel.doc/topics/cubpeltransf.html.
9. Faycal Bechtarzi. Une approche de composition des services web basé Transformation deGraphe. PhD thesis, Juillet 2014.
10. Tebib Assia. Concepts et Outils pour l'Intégration et l'Interopérabilité des Services Application dans le cadre du E-Government. PhD thesis, l'université de Constantine 2, 2014.
11. Sayf Eddine. transformation d'un processus BPEL à un programmeJADE. PhD thesis, fevrier 2016.
12. KHERBOUCHE, Meriem, BOUAFIA, Khawla, et MOLNÁR, Bálint. Transformation of UML State Machine To YAWL. In: 2019 Ninth International Conference on Intelligent Computing and Information Systems (ICICIS). IEEE, 2019. p. 215-220.
13. KHAWLA, Bouafia et MOLNÁR, Bálint. An FSM Approach for Hypergraph Extraction Based on Business Process Modeling. In: International Conference on Computer Science and its Applications. *Springer, Cham*. 2018. p. 158-168.
14. Jos Warmer Anneke G. Kleppe and Wim Bast. MDA Explained : The Model Driven Architecture : Practice and Promise. PhD thesis, USA, 2003.
15. Z. Hu R. Lammel A. Schurr Czarnecki, J. N. Foster and J. F. Ter williger. Bidirectional transformations: A cross-discipline perspective. pages 260-283, 2009.
16. Michael Havey. Essential business process modeling. " O'Reilly Media, Inc.", 2005.
17. Mahmoud Brahimi and Laid Bouzidi. Publiée une fois par année, la revue électronique suisse de science de l'information (ressi) a pour but principal le développement scientifique de cette discipline en suisse. *Ressi*, 2008.
18. KHAWLA, Bouafia et MOLNÁR, Bálint. Dynamic business process: comparative models and workflow patterns. In : THE 11TH CONFERENCE OF PHD STUDENTS IN COMPUTER SCIENCE. 2018. p. 45.
19. Bouafia Khawla, Molnár Bálint, (2019), A Survey on Dynamic Business Processes and Dynamic Business Processes modelling, in: 21st International Conference on Enterprise Information Systems (ICEIS). (Heraklion, Crete, Greece): 556-563.
20. BOUAFIA, Khawla et MOLNÁR, Bálint. Analysis approach for enterprise information systems architecture based on hypergraph to aligned business process requirements. *Procedia Computer Science*, 2019;164: 19-24.
21. Djbril et Moussa Mr Khebizi Ali. Thème: Analyse d'impactde l'évolution des protocoles de service web. Master's thesis, 2013.
22. F. Casati and M.-C Shan. Event-Based Interaction Management for Composite Eservices in eFlow. PhD thesis, Juillet 2002.
23. Matjaz B. Juric. A hands-on introduction to bpel. http://www.oracle.com/ technetwork/articles/matjaz-bpel1-090575.html, april 2003.

Cite this article: Khawla Bouafia, Ali Khebizi, and Bálint Molnár. UNE APPROCHE FONCTIONNELLE DE TRANSFORMATION AUX SPECIFICATIONS ABSTRAITES DES SERVICES WEB BASEES SUR LES AFD A PARTIR DES PROGRAMMES BPEL. *American Journal of Innovative Research and Applied Sciences. 2020; 10(4): 148*-164.

This is an Open Access article distributed in accordance with the Creative Commons Attribution Non Commercial (CC BY-NC 4.0) license, which permits others to distribute, remix, adapt, build upon this work non-commercially, and license their derivative works on different terms, provided the original work is properly cited and the use is non-commercial. See: http://creativecommons.org/licenses/by-nc/4.0/