UNIVERSITÉ DU QUÉBEC À MONTRÉAL

UTILISATION D'UN ALGORITHME GÉNÉTIQUE POUR LA COMPOSITION DE SERVICES WEB

MÉMOIRE

PRÉSENTÉ

COMME EXIGENCE PARTIELLE

DE LA MAÎTRISE EN INFORMATIQUE

PAR

IMED CHOUCHANI

MAI 2010

REMERCIEMENTS

Je tiens à remercier mon directeur de recherche, Monsieur Guy Tremblay, qui a supervisé cette recherche. Sa disponibilité, sa persévérance, son perfectionnisme ainsi que ses encouragements m'ont beaucoup aidé tout au long de ce travail. Je garderai spécialement un doux souvenir de son approche amicale et de son humour peu commun.

J'aimerais remercier mon ami Sami Douik pour son aide pour réaliser ce rapport et ses encouragements.

Je tiens à exprimer ma profonde gratitude à mon père, ma mère, mon épouse et mes fils Taha et Yahya qui m'ont toujours soutenu dans les moments les plus difficiles et sans qui je n'y serais jamais arrivé.

Merci à tous.

TABLE DES MATIÈRES

Table des figures

Liste des tableaux

RÉSUMÉ

L'architecture orientée services (SOA) est une évolution architecturale des systèmes d'informations qui formalise le concept d'échange et de partage inter-application. L'approche SOA utilise un annuaire de services (UDDI) qui joue un rôle de médiateur entre le fournisseur et le consommateur de services. Le fournisseur ou le producteur de services enregistre la description de son service et, par la suite, le consommateur va interroger l'annuaire afin de trouver un service approprié à ses besoins à partir des descriptions publiées.

Une mise en oeuvre possible de cette architecture consiste à utiliser le Web comme support pour la communication entre services. Une telle architecture entraîne donc que les services soient exposés sur le web, qu'on appelle *services Web*. L'avantage de cette approche est, d'une part, de créer un bassin de clientèle pour les fournisseurs et, d'autre part, de mettre en place une base de services importante à l'attention des consommateurs.

Un problème qui apparaît à la mise en oeuvre de cette architecture basée sur les services Web est que le processus de découverte devient assez complexe, et ce en raison de la multitude de services offerts sur Internet, de l'absence d'un standard de représentation des requêtes des consommateurs et en l'absence d'un moteur de recherche efficace.

Notre travail consiste à développer une application basée sur les techniques des algorithmes génétiques pour la découverte et la composition de services Web. Le consommateur exprime le service Web désiré par un fichier de description WSDL. Le système interroge un espace de recherche sous la forme de collection de services Web et donne comme résultat une liste de services possibles (population). Le critère de sélection d'un service est sa valeur de similarité avec le service cible. Les éléments de la population résultat sont des services qui existent dans l'espace de recherche (découverte d'un service approprié) ou qui ont été créés par le biais de la composition de services existants.

Notre choix d'une approche utilisant les algorithmes génétiques s'est fait parce que les opérations utilisées par un algorithme génétique — croisement, mutation, sélection — semblaient avoir une grande correspondance avec la composition de services Web. Les quelques tests que nous avons effectués pour évaluer notre approche semblent justifier notre choix de ces techniques. Toutefois, les résultats pourraient être améliorés par l'introduction de notions sémantiques et le temps de réponse pourrait être amélioré par l'utilisation de parallélisme dans nos algorithmes génétiques.

MOTS-CLÉS: Services Web, composition de services, architecture orientée services (SOA), Web Services Description Language (WSDL), Simple Object Access Protocol (SOAP), algorithmes génétiques.

INTRODUCTION

Plusieurs innovations technologiques ont eu lieu ces dernières années pour répondre aux besoins de partage des ressources et de communication des entreprises. Ceci a conduit à remettre en cause constamment l'architecture des applications et à rechercher de meilleures façons de développer les applications informatiques.

Les architectures distribuées ainsi que les architectures orientées services réduisent le problème de développement d'applications lourdes à un problème de déploiement de composants (8) ou de services Web. L'internet offre une multitude de services, ce qui complexifie la recherche d'un service qui répond à un besoin particulier, et ce en raison de l'absence d'un standard de représentation des requêtes et d'un moteur de recherche efficace.

Architecture orientée services

Au cours des derniers années, il y a eu trois vagues technologiques ayant apporté des améliorations importantes à l'architecture et au processus du développement des systèmes distribués (figure 0.1).

L'architecture orientée services (SOA) est un modèle d'application distribuée qui utilise le concept de service comme base de développement. Une mise en oeuvre possible de cette architecture est celle basée sur la technologie des services Web (Figure 0.2).

Service Web

Les services Web sont des applications accessibles via l'Internet. Le langage WSDL (*Web Services Description Languages*) (17; 20; 43) permet de décrire les informations (descriptions et interfaces) de ces services qui sont nécessaires pour les invoquer à distance par l'échange de messages.

Ces descriptions sont enregistrées dans un annuaire UDDI (*Universal Description, Discovery*

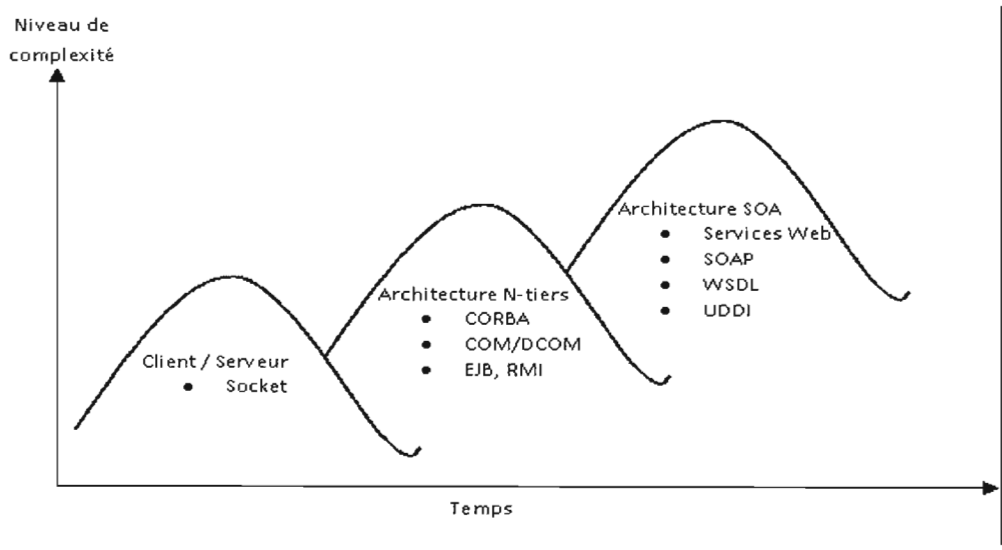

FIGURE 0.1 Évolution de l'architecture des applications (Figure tirée de (46)).

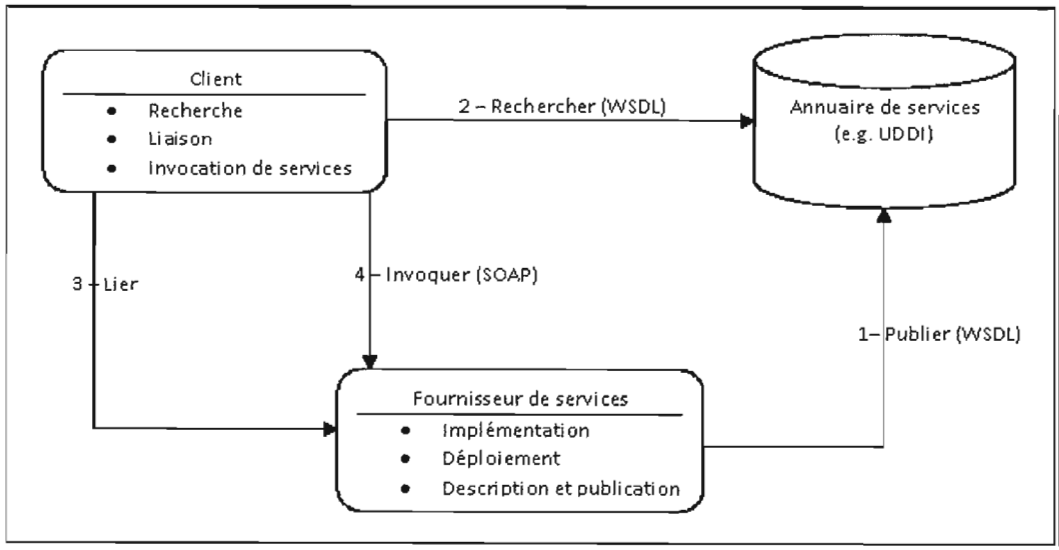

FIGURE 0.2 Architecture orientée services.

and Integration) et transmissibles, par exemple, à travers le protocole SOAP (*Simple Object Access Protocol*) utilisant le langage d'échange XML (*eXtensible Markup Language* (47)).

Le rôle d'un annuaire UDDI est tout d'abord de permettre aux fournisseurs d'enregistrer leurs services Web. Son rôle est ensuite de permettre aux clients de découvrir les services Web qu'ils souhaitent exploiter en fournissant les informations permettant de les invoquer à distance et dynamiquement. Donc, un tel annuaire UDDI favorise les communications entre clients et fournisseurs de services Web en facilitant la recherche et la découverte de ces services.

Service Web composite

Les services Web sont des applications accessibles sur Internet réalisant chacun une tâche spécifique. Pour fournir une solution à une tâche complexe, on peut regrouper des services Web pour n'en former qu'un seul ; on parle alors de composition de services Web.

Pour la création d'un service Web composite on peut utiliser les langages standards comme Java, C, etc., qui offrent des bibliothèques riches pour la manipulation des services Web. On peut aussi utiliser des standards développés à cette fin comme WS-BPEL (*Business Process Execution Language*) (6; 45; 19).

Le processus de composition de services Web comporte trois étapes :

1. Spécifier le service désiré ainsi que les différentes tâches à réaliser ;
2. Découvrir les services qui permettent de réaliser chaque tâche ;
3. Développer la composition.

Problématique

La composition automatique de services Web demeure un problème à résoudre. Plusieurs approches basées sur la description sémantique des services donnent des résultats théoriques pertinents mais l'absence d'un répertoire officiel qui contient des descriptions sémantiques de services Web rend ces solutions impossibles en pratique.

Notre contribution

Les algorithmes génétiques sont répandus dans divers domaines pour résoudre des problèmes d'optimisation et de recherche. Nous nous inspirons de ces techniques pour décrire une approche de recherche et de composition de services Web.

Notre projet consiste à développer une architecture basée sur les techniques génétiques et vise à aider le concepteur à trouver une composition qui produit le service Web désiré à partir d'un ensemble de services existant.

Architecture de notre application

La figure 0.3 présente l'architecture de notre approche. Cette architecture est basée sur un algorithme génétique qui reçoit la description du service Web désiré, recherche parmi les services Web existants pour tenter de donner comme résultat soit un service Web le plus possible similaire à celui recherché, soit une liste des services qui peuvent donner un service semblable à celui recherché par le biais d'une composition.

Description de notre approche

Notre approche consiste à utiliser un algorithme génétique pour la composition de services Web. Notre application se divise en trois phases :

1. Analyse de la requête utilisateur ;
2. Recherche sur internet des opérations identifiées dans la première phase (par un moteur de recherche comme `seakda`) ;
3. Exécution de l'algorithme génétique qui commence par analyser un petit ensemble de services (population initiale) et recherche la possibilité d'évolution (par le biais de la composition). S'il atteint un état de stabilité (pas de possibilité de composition) et si la solution trouvée n'est pas suffisamment pertinente (une similarité insuffisante avec le service désiré), il tente d'enrichir sa population par un individu externe et poursuit sa recherche et son processus d'évolution.

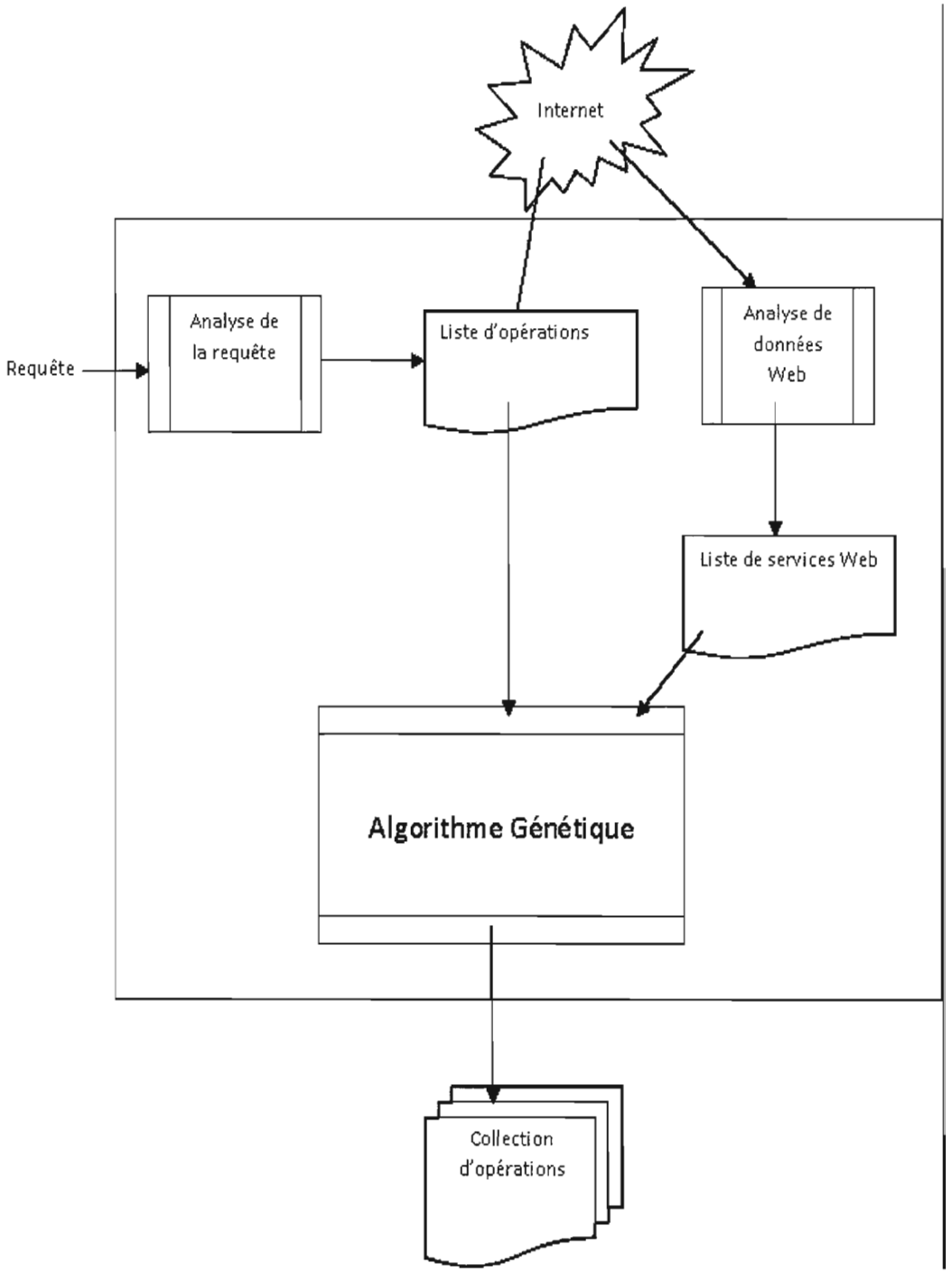

FIGURE 0.3 Architecture de notre application.

La première phase de cette application est l'analyse de la requête utilisateur ainsi que le filtrage des fichiers WSDL des services Web trouvés par une recherche simple sur internet. Cette liste de description est enregistrée dans une base de données accessible par notre algorithme génétique qui sera développé à la deuxième phase.

Organisation du mémoire

Ce mémoire comprend deux grandes parties. La première est consacrée à l'état de l'art relatif aux domaines abordés dans le mémoire. Dans le premier chapitre, nous décrivons les descriptions de services Web (syntaxique et sémantique) ainsi que les approches de composition liées à chaque description. Nous étudions ensuite quelques approches existantes et examinons leur pertinence puis nous formulons quelques observations sur leurs avantages et leurs inconvénients. Dans le troisième chapitre nous mettons en contexte les algorithmes génétiques.

La deuxième partie de notre mémoire est consacrée à la présentation de notre approche. Dans le quatrième chapitre nous présentons notre algorithme génétique. Dans le chapitre suivant, nous présentons une description de la mise en oeuvre de notre application et des tests que nous avons effectués.

CHAPITRE I

LES SERVICES WEB

Introduction

L'architecture orientée services (*SOA : Service Oriented Architecture*) est un modèle d'architecture distribuée qui définit une infrastructure d'interaction entre les services. Un service est un composant logiciel autonome (fournisseur) mis à la disposition d'autres composants logiciels (consommateurs).

Une mise en oeuvre possible de l'architecture orientée services est celle basée sur la technologie des services Web. Un service Web est un service auquel on accède par n'importe quelle technologie Web.

Un service Web est atomique ou autonome s'il ne fait appelle à aucun autre service Web dans son exécution. Sinon on dit qu'il est composite.

Le présent chapitre décrit les services Web. Dans la première section nous présentons les différentes technologies derrière la notion de services Web. La second section décrit le service Web composite: sa définition, son fonctionnement et quelques langages de définition. La dernière section est consacrée aux services Web sémantiques.

1.1 Définitions et exemples

1.1.1 Service Web

Définition

Dans le document « *Web Services Architecture* », le groupe W3C qui travaille sur le services Web a donné la définition suivante (16) :

> « *A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards* ».

Cette définition met en valeurs les principaux avantages de services Web :

- interface décrite dans un format interprétable par les machines ;
- utilisation de langages et de protocoles indépendants des platformes d'implémentation ;
- utilisation de normes actuelles du Web.

Exemple

Un service Web regroupe un ensemble d'opérations (fonctions). On appelle messages d'entrée les paramètres d'entrée de l'opération. Les résultats retournés sont appelés des messages de sortie.

La figure 1.1 montre un exemple de service Web (MathService) tiré du site Web `Seekda`[1]. Ce service Web comprend quatre opérations (Divide, Add, Multiply et Subtract). Pour chaque opération, on a un message d'entrée composé de deux réels et un message de sortie formé d'un réel pour le résultat de l'opération. L'interface d'échange de services prend en considération les divers protocoles disponibles sur le Web tels que HTTP et SOAP.

---

1. `http://webservices.seekda.com/providers/aspalliance.com/MathService?tab=usenow`

FIGURE 1.1 Le service Web `MathService`.

1.1.2 eXtensible Markup Language

Définitions

XML (*eXtensible Markup Language*) est un langage de balises, plus général que HTML, qui permet la représentation de données. Un document XML est un fichier texte composé d'un en-tête optionnel et d'un corps ayant une structure d'arbre (49). L'en-tête contient des informations sur la version de XML et le codage utilisé ; elle peut aussi référencer un XML-Schema.

XML-Schema permet de décrire d'une façon complète la structure d'un document XML (la structure de l'arbre, les types et les attributs des éléments de document).

Plusieurs outils ont été développés autour de la norme XML permettant de manipuler les données décrites dans un document XML, par exemple les analyseurs DOM et SAX (9).

Le format DOM (*Document Object Model*) permet de représenter un document XML sous forme d'un arbre d'objets dans la mémoire et fournit des interfaces permettant de le manipuler (54). Contrairement à DOM, SAX (*Simple API for XML*) ne construit pas une image dans la mémoire. SAX se base sur un modèle d'événements : il parcourt le document XML et génère un événement à chaque fois qu'il rencontre une nouvelle balise.

Exemple : La figure 1.2 présente un exemple de document XML où on remarque la présence de différentes balises comme `book`, `chapter`, etc.

1.1.3 Simple Objet Access Protocol

Définition

SOAP (*Simple Objet Access Protocol*) est un protocole d'échange de messages basé sur le langage XML (47). Un message SOAP est composé de deux parties :

Enveloppe : les informations sur le message transmis.

Corps de message : les informations à transmettre.

L'échange de messages SOAP est basé sur le modèle requête/réponse :

- Le client envoie une requête ;

```
<book>
    <chapter>
        <title>Introduction</title>
    </chapter>
    <chapter>
        <title>Récit</title>
        <subChapter>
            <title>Partie 1</title>
        </subChapter>
        <subChapter>
            <title>Partie 2</title>
        </subChapter>
    </chapter>
    <chapter>
        <title>Index</title>
    </chapter>
</book>
```

FIGURE 1.2 Exemple d'un document XML (47).

– Le serveur répond par un message réponse .

Exemple : Dans notre exemple d'un message SOAP nous prendrons une requête simple d'un client qui veut savoir le prix d'un ticket de symbole « DIS »(figure 1.3). Le serveur répond par le message SOAP de la figure 1.4 où il indique le prix (34.5) du ticket demandé.

1.2 Description syntaxique

Un service Web possède une description syntaxique qui regroupe des informations telles que noms d'opération, types de données, protocoles réseau et point d'accès (43). Une telle description est basée sur le langage de description de services Web WSDL (5).

1.2.1 Web Services Description Languages

Définition

Dans le cadre de l'architecture orientée services, le WSDL (*Web Services Description Languages*) est un langage fondé sur XML qui permet de décrire une interface publique d'accès à un

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"

<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
     <soapenv:Body>
          <m:GetLastTradePrice xmlns:m="Some-URI">
               <m:tickerSymbol>DIS</m:tickerSymbol>
          </m:GetLastTradePrice>
     </soapenv:Body>
</soapenv:Envelope>
```

FIGURE 1.3 Exemple de requête SOAP (47).

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn

<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
     <soapenv:Body>
          <m:GetLastTradePriceResponse xmlns:m="Some-URI">
               <m:price>34.5</m:price>
          </m:GetLastTradePriceResponse>
     </soapenv:Body>
</soapenv:Envelope>
```

FIGURE 1.4 Exemple de réponse SOAP (47).

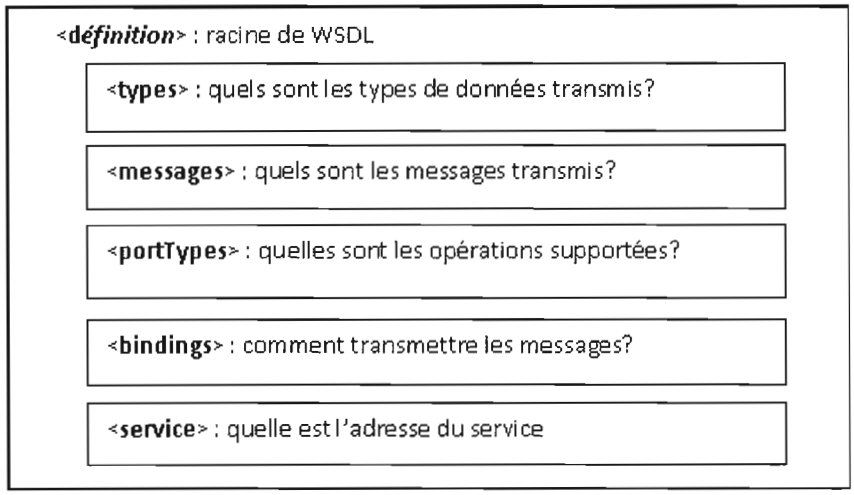

FIGURE 1.5 Structure d'un document WSDL (46).

service Web (17)

Structure d'un document WSDL

Le langage WSDL permet de décrire (Figure 1.5) :

1. Les opérations qu'un services Web peut exécuter.
2. Le type de messages XML qu'il peut traiter.
3. Les protocoles de communication qu'il supporte.
4. Le point d'accès à une instance du service Web.

Un document WSDL décrit un service Web en deux niveaux principaux (20) :

Niveau abstrait : Une partie de description de service Web qui permet de définir les types de données, les messages transmis et les ports et ce de façon indépendante des protocoles utilisés.

Niveau concret : Ce niveau permet de définir les protocoles utilisés par le service et sa localisation.

Exemple d'un fichier WSDL : La figure 1.6 montre un exemple de fichier WSDL qui décrit un service Web « Calendar » [2].

---

2. `http://www.stgregorioschurchdc.org/wsdl/Calendar.wsdl`

```
<?xml version="1.0" encoding="UTF-8" ?>
- <definitions name="Calendar"
  targetNamespace="http://www.stgregorioschurchdc.org/Calendar"
  xmlns:tns="http://www.stgregorioschurchdc.org/Calendar"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://schemas.xmlsoap.org/wsdl/">
  - <message name="EasterDate">
      <part name="year" type="xsd:short" />
    </message>
  - <message name="EasterDateResponse">
      <part name="date" type="xsd:string" />
    </message>
  - <portType name="EasterDateSoapPort">
    - <operation name="easter_date" parameterOrder="year">
        <input message="tns:EasterDate" />
        <output message="tns:EasterDateResponse" />
      </operation>
    </portType>
  - <binding name="EasterDateSoapBinding" type="tns:EasterDateSoapPort">
      <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
    - <operation name="easter_date">
        <soap:operation
          soapAction="http://www.stgregorioschurchdc.org/Calendar#easter_date" />
      - <input>
          <soap:body use="encoded"
            namespace="http://www.stgregorioschurchdc.org/Calendar"
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
        </input>
      - <output>
          <soap:body use="encoded"
            namespace="http://www.stgregorioschurchdc.org/Calendar"
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
        </output>
      </operation>
    </binding>
  - <service name="Calendar">
    - <port name="EasterDateSoapPort" binding="tns:EasterDateSoapBinding">
        <soap:address
          location="http://www.stgregorioschurchdc.org/cgi/websvccal.cgi" />
      </port>
    </service>
  </definitions>
```

FIGURE 1.6 Fichier WSDL du service Web `Calendar`.

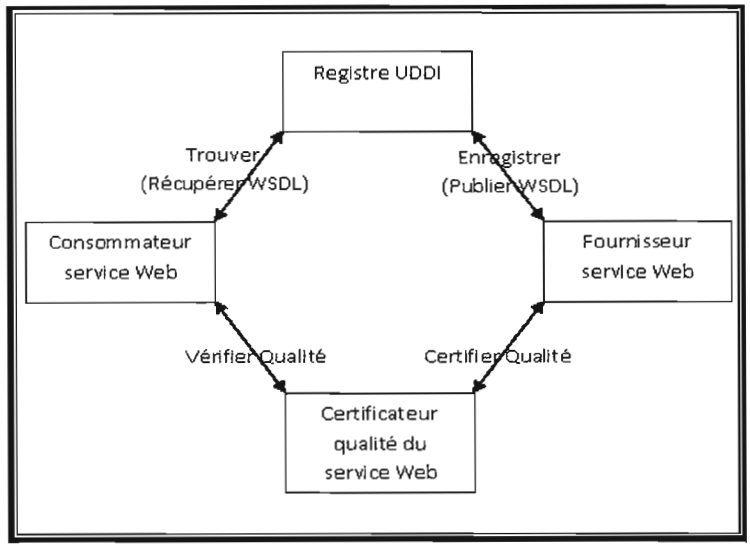

FIGURE 1.7 Modèle de registre et découverte de services Web (23).

1.2.2 Universal Description Discovery and Integration

UDDI (*Universal Description Discovery and Integration*) est un standard pour décrire un annuaire de services Web. Il permet de publier (fournisseur de services) et de découvrir (utilisateur de services) un service Web (figure 1.7).

Un annuaire UDDI est consultable de différentes manières :

- Pages blanches : Une liste des entreprise ainsi que des informations associées ;
- Pages jaunes : Une liste de services de chaque entreprise (document WSDL) ;
- Pages vertes : Des informations techniques sur les services.

1.3 Outils de développement et de manipulation

1.3.1 Outils pour la plate-forme Java

Plusieurs API et package ont été développés autour de la plate-forme Java dans le but de faciliter le déploiement et la création des services Web :

JAX-WS : *Java API for XML Web Services* (JAX-WS) est une bibliothèque de classes Java qui

permet de créer et déployer des services Web.

Apache Axis : Axis est un package Java libre qui fournit :

1. une API pour développer des services Web ;
2. des outils pour créer automatiquement les fichiers de description WSDL ;
3. des outils pour déployer et tester des services Web.

1.3.2 Outils pour la plate-forme Microsoft .NET

« .NET Framework » de Microsoft offre plusieurs outils de développement et de manipulation des services Web :

Classe System.Web.Services : Les classes `System.Web.Services` telles que `WebMethodAttribute`, `WebService`, `WebServiceAttribute` et `WebServiceBindingAttribute` permettent de créer des services Web XML à partir de clients de services Web ASP.NET et XML.

Web Services Description Language (Wsdl.exe) : Un outil qui permet de génèrer le code pour des services Web XML et les clients des services Web XML à partir de fichiers de description (WSDL), de fichiers de schéma XSD et de documents de découverte `.discomap`.

Web Services Discovery Tool (Disco.exe) : Un outil qui permet de découvrir les URL de services Web et d'enregistrer les documents liés à chaque service sur le disque local.

1.4 Services Web composites

1.4.1 Définition et fonctionnement

La composition est le fait de combiner les fonctionnalités de plusieurs services Web au sein d'un même processus métier pour répondre à une demande complexe qu'un seul service ne pourrait pas satisfaire. Un processus métier est une représentation concrète des tâches à accomplir dans une composition (53).

La réalisation d'une composition nécessite les étapes suivantes :

1. La découverte est le processus de recherche de services Web qui peuvent participer à la composition. Ce processus se fait généralement de manière manuelle par envoi de requêtes

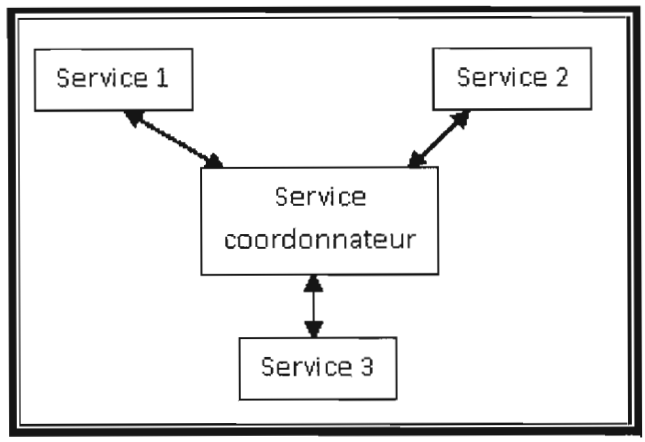

FIGURE 1.8 Exemple d'orchestation de services Web : le coordonnateur joue le rôle de « chef d'orchestration ».

aux registres de l'annuaires UDDI (33). Plusieurs travaux vise même à automatiser cette étape (23) ;

2. L'organisation des interactions entre les services Web dans une composition est définie par la technique d'orchestration (figure 1.8). L'orchestration permet aux différent services d'échanger les messages entre eux.

   Une composition est associée à une spécification pour gérer les échanges de messages et mettre en place les structures de contrôle nécessaires (27). Dans la littérature, plusieurs langages de spécification ont été proposés : WSCI, WSFL, XLANG et, plus récemment, YAWL, XPDL, BPMN et WS-BPEL (48) ;

3. L'exécution de la composition est l'étape d'invocation effective des services Web participant à une composition.

1.4.2 Classification des approches de compositions

La composition de services Web peut être classifiée en fonctions du degré d'automatisation. D'après cette classification on trouve trois catégories de compositions (10) :

Composition manuelle : L'utilisateur génère la composition à la main sans l'aide d'outils dédiés ;

Composition semi-automatique : Dans le processus de composition, l'utilisateur utilise des outils pour aider à la découverte et à la sélection des services les mieux adaptés à son besoin ;

Composition automatique : Le processus de composition est réalisé automatiquement sans l'intervention de l'utilisateur.

Selon les techniques utilisées dans le processus de composition, on peut classifier de la manière suivante les approches pour la composition :

L'approche industrielle : Cette approche est basée sur la description syntaxique de service Web et les technologies liées (WSDL, SOAP, UDDI, etc.) dans la création de processus de composition. WS-BPEL est l'outil le plus utilisé dans ce genre d'approches ;

L'approche sémantique : Une orientation sémantique pour la composition de services Web. Ce genre d'approche utilise les techniques du Web sémantique pour que le processus de composition se réalise automatiquement ;

L'approche formelle : Cette approche utilise des techniques de modélisation et de validation formelle de processus (exemple : les réseaux de Petri) (13).

1.4.3 Langages de définition

Il existe plusieurs langages de définition pour la composition de services Web. Dans cette section nous allons présenter brièvement quelques-uns d'entre eux.

WSCL (*Web Service Conversation Language*)

WSCL décrit les services Web en mettant l'accent sur les conversations de ceux-ci. WSCL manipule les fichiers WSDL pour décrire les opérations possibles ainsi que leur chorégraphie (12).

XLANG

XLANG est une extension de WSDL créée par Microsoft. Il fournit un modèle pour une orchestration des services et des contrats de collaboration entre ceux-ci.

WS-BPEL (*Business Process Execution Language for Web Services*)

BPEL est un langage qui se base sur XML (6). Il a été conçu spécifiquement comme un langage pour la définition des processus métier. Il supporte deux types différents de processus :

1. Les processus exécutables permettent de spécifier les détails du processus métier. Ils peuvent être exécutés au moyen d'un engin d'orchestration (19).
2. Les *abstracts business protocols* permettent de spécifier l'échange de messages entre partenaires du processus (45).

1.5 Services Web sémantiques

1.5.1 Définitions

Un service Web sémantique est le résultat de l'intégration d'aspects sémantiques aux définitions du service Web (35). Le service Web sémantique est la convergence de deux technologies récentes du Web : les services Web et le Web sémantique (29).

Le Web sémantique offre un ensemble de technologies pour ajouter l'aspect sémantique au Web actuel dans le but de permettre la manipulation automatique de ces ressources. Le service Web est l'une des ressources importantes du Web qui est visée par ces technologies sémantiques (21).

1.5.2 Description sémantique de services Web

Deux démarche possibles pour la description sémantique de services Web (1) :

- La première approche consiste à annoter les langages existants (WSDL, UDDI, WS-BPEL) avec de l'information sémantique (14). Le principal avantage de ce genre de solutions est la facilité pour les fournisseurs de services d'adapter leurs descriptions existantes aux annotations proposées (18; 50) ;
- La deuxième approche réécrit entièrement la description sémantique du services Web en utilisant les technologies du Web sémantique (15; 40).

1.5.3 Ontologie de services Web

La création d'une ontologie de services Web permet d'ajouter l'aspect sémantique et d'offrir un outil de recherche plus performant. Des languages ont été développés pour créer des ontologies de services Web.

Le OWL-S (*Web Ontology Language for Web services*) est un sous-ensemble du *Web Ontology Langage (OWL)* dédié à la description sémantique de services Web (44). Il est compatible avec des formats de description syntaxique tels que WSDL. Une description OWL-S se compose de trois éléments : le `service profile`, le `process model`, et le `grounding`, qui décrivent respectivement « que fait le service », « comment le service fonctionne » et « comment accéder au service »(39; 25).

1.6 Relation entre description syntaxique et sémantique de services Web

OWL-S et WSDL sont complémentaires pour l'ensemble de la spécification de services Web. Les deux langages ne couvrent pas la même espace conceptuel. Comme indiqué par la figure 1.9 [3], la spécification WSDL indique les types abstraits, tandis que les OWL-S tient compte de la définition des types abstraits en tant que classe. Cependant, WSDL ne peut pas exprimer la sémantique d'une classe de OWL-S. De même, OWL-S n'a aucun moyen, tel qu'actuellement défini, pour exprimer l'information obligatoire que WSDL saisit.

Conclusion

Les services Web peuvent être vus comme des ressources actives, dynamiques, relativement à des ressources statiques comme celles contenues dans les bases de données disponibles sur le Web. Si un service Web est vu comme une ressource, alors certains problèmes vont se poser : Comment le décrire, le composer avec d'autres, le sélectionner. La composition entre donc dans cet ensemble de tâches, elle est intrinsèquement liée au problème de la description des services Web.

La composition automatique de services Web permet de gagner du temps et de faciliter la tâche du programmeur pour découvrir ou créer un nouveau service complexe. Selon des études

3. Tiré du site : `http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/#6`

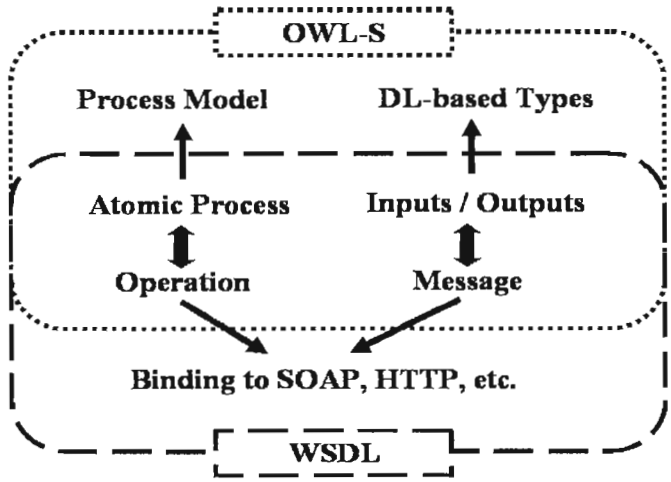

FIGURE 1.9 Relation entre OWL-S et WSDL.

faite sur des bases de connaissances restreintes (développées pour la mise en oeuvre de certains approches), la description sémantique des services Web peut résoudre le problème de la composition automatique (22). Mais l'absence d'une base de connaissances regroupant les services Web selon une ontologie ne permet pas d'utiliser à grand échelle ce genre de composition.

CHAPITRE II

LES ALGORITHMES GÉNÉTIQUES

Introduction

Les algorithmes génétiques sont des algorithmes qui s'inspirent de la science génétique développée au XIXe siècle par Darwin (36). Le but principal de ces algorithmes est de trouver une solution pour des problèmes difficiles — des problèmes où on ne connaît pas de méthodes exactes pour les résoudre en temps raisonnable (31). Ces approches permettent l'évolution d'une génération (solution) à une autre par un ensemble d'opérations (mutation et croisement) tout en cherchant à améliorer une fonction objectif. Le codage des éléments de la population est l'élément le plus important, puisqu'il permet de modéliser un problème sous forme de données informatiques manipulables (4).

Les algorithmes génétiques sont appliqués dans divers domaines pour résoudre des problèmes d'optimisation ou de recherche. Par exemple: la conception topologique de réseaux téléinformatiques à commutation de paquets (42), la recherche d'informations (58), des applications en économie (30).

Dans le présent chapitre, après avoir présenté les principes et les fonctionnalités de base d'un algorithme génétique ainsi que ses caractéristiques nous décrivons les différentes opérations génétiques telles que la mutation et le croisement. Dans la quatrième section nous abordons le principe de parallélisme dans un algorithme génétique. Nous finissons par une présentation de deux exemples d'applications de techniques génétiques qui nous aideront dans le développement de notre approche.

2.1 Principes et fonctionnalités

2.1.1 Principe de base

Les algorithmes génétiques sont des approches d'optimisation qui utilisent des techniques dérivées de la science génétique et de l'évolution naturelle : la sélection, la mutation et le croisement (28). Pour utiliser ces approches, on doit disposer des éléments suivants (7) :

1. Le codage d'un élément de population : une fonction qui permet de modéliser les données du problème réel dans des données utilisables par l'algorithme génétique (34).
2. Une fonction pour générer la population initiale : la génération de la population initiale est importante puisque cette génération représente le point de départ de l'algorithme et son choix influe sur la rapidité et l'optimalité de la solution finale.
3. Une fonction à optimiser (la fonction objectif) : une fonction qui retourne une valeur d'adaptation pour chaque individu. Cette valeur permet de déterminer la solution pertinente puisque le problème se restreint à chercher le groupe d'individus qui ont les valeurs optimums.
4. Des opérateurs qui permettent d'évoluer d'une population à une autre tout en améliorant la fonction objectif. L'opérateur de croisement recompose les gènes d'individus existant dans la population, alors que l'opérateur de mutation a pour but de garantir l'exploration de l'espace d'états.
5. Des paramètres de dimensionnement : taille de la population, nombre total de générations (critère d'arrêt), probabilités d'application des opérateurs de croisement et de mutation, etc.

2.1.2 Fonctionnement

Un algorithme génétique fonctionne de la manière suivante (figure 2.1) (30) :

Étape 1 : Initialisation on choisit μ individus qui représente la population initiale ;

Étape 2 : Évaluation on évalue chaque individu par la fonction objectif ;

Étape 3 : Sélection on définit les individus de la génération P qui vont être dupliqués dans la nouvelle population. À chaque génération il y a deux opérateurs de sélection : la sélection de reproduction, ou plus simplement sélection, qui détermine les individus qui vont se re-

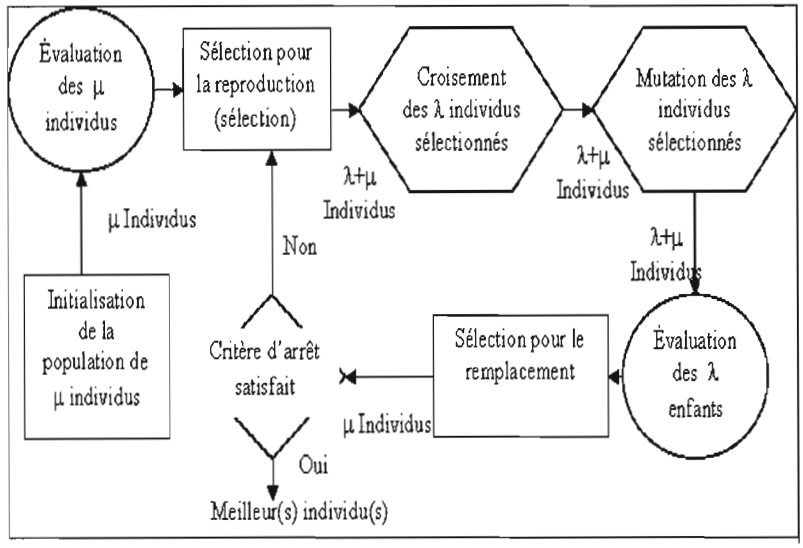

FIGURE 2.1 Structure générale d'un algorithme génétique (36).

produire durant une génération et la sélection pour le remplacement, ou plus simplement le remplacement, qui détermine quels individus devront disparaître de la population.

Étape 4 : Reproduction on utilise des opérateurs génétiques (croisement et mutation) pour produire la nouvelle génération. Les opérateurs de mutation modifient un individu pour en former un autre tandis que les opérateurs de croisement engendrent un ou plusieurs enfants à partir de combinaisons de deux parents.

La forme classique d'un algorithme génétique

Un algorithme génétique a la forme classique suivante (46; 7) :

1. Initialiser la population initiale P
2. Évaluer P
3. Tantque (Pas Convergence) faire :
   (a) P' = Sélection des Parents dans P
   (b) P' = Appliquer Opérateur de Croisement sur P'

(c) P'' = Appliquer Opérateur de Mutation sur P'

(d) P = Remplacer les anciens de P par leurs descendants de P''

(e) Évaluer P

Fin Tantque

Le critère de convergence peut être de nature diverse, par exemple :

- Un taux minimum qu'on désire atteindre d'adaptation de la population au problème ;
- Un certain temps de calcul à ne pas dépasser ;
- Une combinaison de ces deux points.

2.2 Caractéristiques des algorithmes génétiques

Les algorithmes génétiques, en tant qu'approche de résolution de problèmes, se caractérisent par certains aspects particuliers : le codage des paramètres du problème à traiter, l'espace de recherche et la fonction d'évaluation qui permet de déterminer la pertinence d'une solution trouvée et l'évolution d'une génération à une autre par la sélection des chromosomes qui participent à la reproduction et les chromosomes à disparaître (42).

2.2.1 Le codage

Le codage est une fonction qui permet de passer de la donnée réelle du problème traité à la donnée utilisée par l'algorithme génétique (figure 2.2). Le choix du codage est l'élément le plus important dans la conception de l'algorithme puisqu'il permet d'une part de représenter les données, les paramètres et les solutions et d'autre part il influe sur la mise en oeuvre des opérations génétique telles que le croisement et la mutation qui influent directement sur le bon déroulement de l'algorithme génétique et de leur convergence vers la bonne solution.

Généralement on a trois types de codage les plus utilisés :

Représentation binaire : Chaque gène dispose du même alphabet binaire 0, 1. Un gène est alors représenté par un entier, les chromosomes, qui sont des suites de gènes sont représentés par des tableaux de gènes et les individus de l'espace de recherche sont représentés par des tableaux de chromosomes.

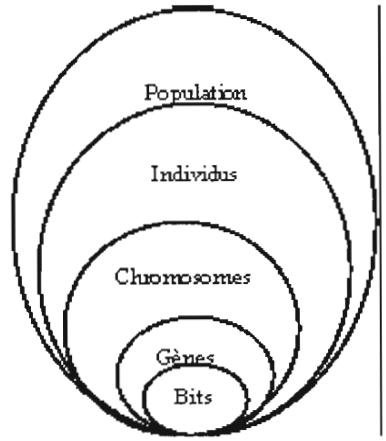

FIGURE 2.2 Les cinq niveaux d'organisation d'une population d'un algorithme génétique (7).

Représentation avec réels : Contrairement au codage binaire, un gène est représenté par une suite de bits (un bits dans le code binaire) qui est associé à un réel. Ce type de codage peut être utile notamment dans le cas où l'on recherche le maximum d'une fonction réelle.

Exemple :

10010011	11101011	00011010
gene1	gene2	gene3
x1=3,256	x2=0,658	x3=10,26

Représentation à l'aide d'arbres syntaxiques : Ce type de codage utilise une structure arborescente (une racine de laquelle peuvent être issus un ou plusieurs fils eux mêmes des arbres). Un arbre syntaxique est un arbre contenant deux types de noeuds (35) :

1. les noeuds internes ou « symboles non terminaux »
2. les feuilles ou « symboles terminaux »

Ce type de codage peut être utilisé lorsque la taille du problème ou de la solution n'est pas finie. Son inconvénient est qu'on peut trouver des arbres de solutions de taille importante difficile à analyser.

2.2.2 La fonction d'évaluation

La fonction de performance — qu'on appelle aussi fonction d'adaptation, fonction objectif ou

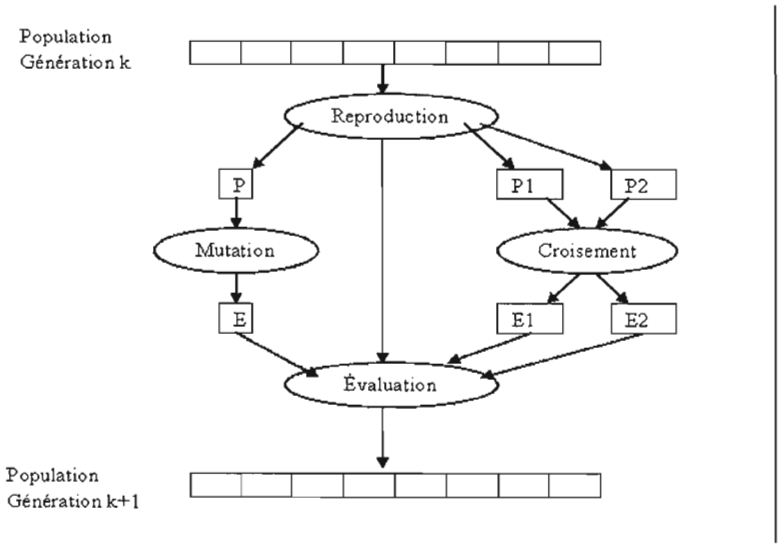

FIGURE 2.3 Principe général de l'évolution d'une population d'un algorithme génétique (4).

fonction *fitness* — associe une valeur de performance à chaque individu ce qui offre la possibilité de le comparer à d'autres individus et permet à l'algorithme génétique de déterminer qu'un individu sera sélectionné pour être reproduit ou pour déterminer s'il sera remplacé (36).

2.3 Les opérateurs génétiques

La reproduction est le processus qui permet de construire une population $k+1$ à partir d'une population k. Ce processus est constitué par l'utilisation de l'opération de sélection, de l'opération de croisement ou/et de l'opération de mutation (Figure 2.3).

2.3.1 La sélection

L'opération de sélection permet de déterminer quels individus sont plus enclins à obtenir les meilleurs résultats (36). On trouve deux types de sélection :

1. La sélection pour la reproduction : On l'appelle tout simplement l'opération de sélection, et elle permet de choisir les individus qui participent à une reproduction (croisement ou mutation). Cette opération choisit, généralement, les individus les plus forts (meilleurs scores

d'adaptation) pour produire les enfant les plus performants.

2. La sélection pour le remplacement : On l'appelle tout simplement l'opération de remplacement, et elle choisit les individus les plus faibles pour être remplacés par les nouveaux.

On trouve plusieurs techniques de sélection :

- **Sélection par rang :** Choisir toujours les individus possédant les meilleurs scores.
- **Sélection par tournoi :** Utiliser la probabilité de sélection proportionnelle à l'adaptation sur des paires d'individus, puis choisir parmi ces paires celui qui a le meilleur score d'adaptation.
- **Sélection uniforme :** Choisir aléatoirement sans faire intervenir la valeur d'adaptation.

2.3.2 Le croisement

L'opération de croisement (*crossover*) est une opération de reproduction qui permet l'échange d'information entre les chromosomes (individus). Il utilise deux parents pour former un ou deux enfants. Les deux parents sont choisis par l'opération de sélection. Le croisement permet l'innovation (les enfants sont différents de leurs parents) et repose sur l'idée que deux parents performants produiront des enfants plus performants.

Dans le cas d'une représentation binaire, le croisement de deux chromosomes peut se faire en un seul point de coupure (Figure 2.4) ou en deux points de coupure (Figure 2.5).

Le taux de croisement détermine la proportion des individus qui sont croisés parmi ceux qui remplaceront l'ancienne génération.

2.3.3 La mutation

Le rôle de cet opérateur est de modifier aléatoirement, avec une certaine probabilité, la valeur d'un composant de l'individu (57). La figure 2.6 présente un exemple de mutation de chromosome tel que le gène g_i est retiré aléatoirement et est remplacé par la gène g_i'.

La mutation est un phénomène rare mais permet d'explorer de nouvelles zones dans l'espace de recherche et aide l'algorithme génétique à possiblement aller vers une solution optimale globale, sans resté pris dans une solution optimale locale.

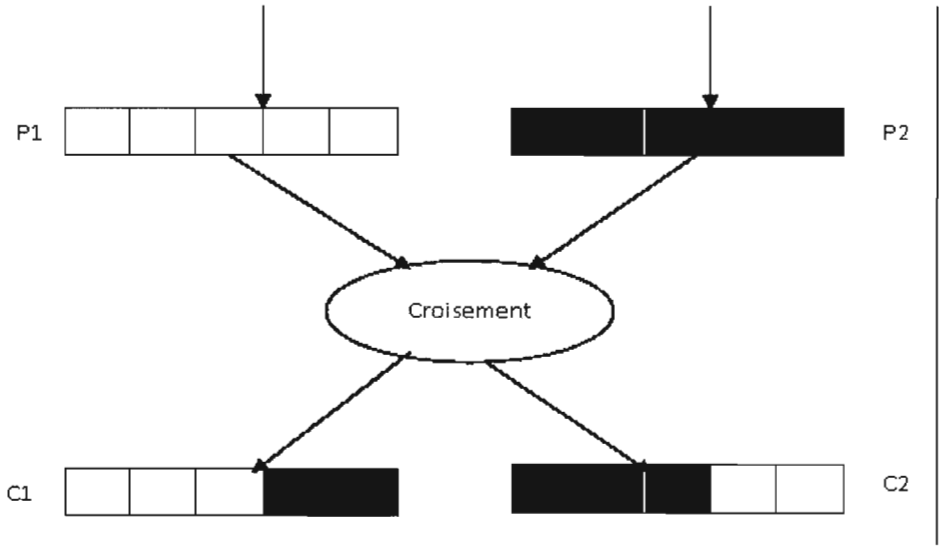

FIGURE 2.4 Croisement à un point (4).

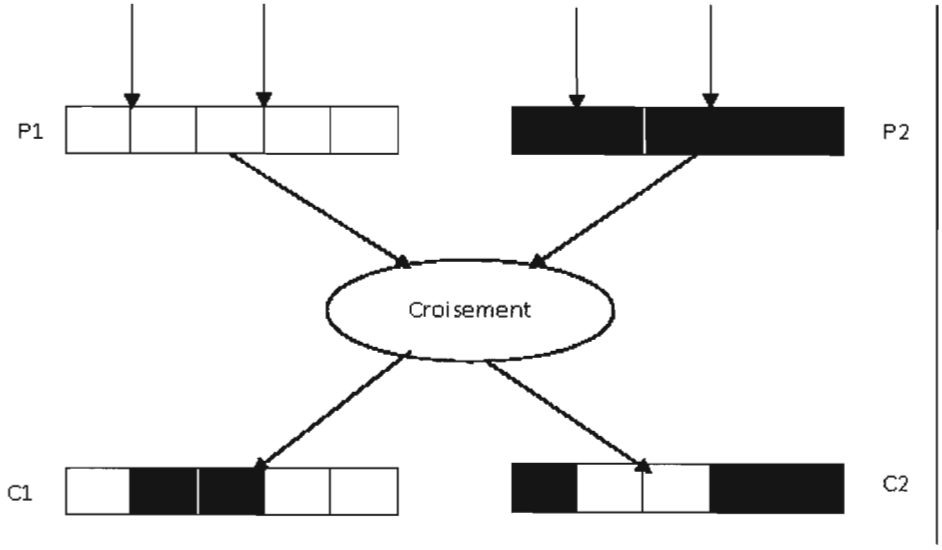

FIGURE 2.5 Croisement à deux points (4).

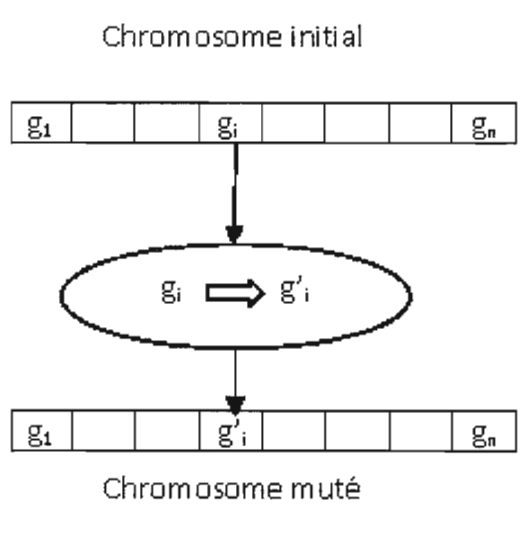

FIGURE 2.6 Illustration du principe de la mutation.

2.4 Techniques avancées

2.4.1 Recherche multi-objectif

Dans certain cas, les individus ont des critères d'évaluation multiples. Donc, d'une part, on a besoin d'utiliser une fonction d'évaluation multi-objectif et, d'autre part, on a besoin d'une technique avancée de sélection (56).

Le problème de la sélection est dû aux valeurs retournées par la fonction d'adaptation multi-critères. Puisque la fonction retourne plusieurs valeurs d'évaluation pour un seul individu, comment décider qu'un individu est meilleur qu'un autre ?

Une solution possible de ce problème est d'introduire une valeur seuil : un individu est meilleur qu'un autre si le nombre des valeurs contenues dans son vecteur d'adaptation qui sont supérieures aux valeurs correspondantes dans l'autre individu dépasse un certain seuil.

2.4.2 Diploïdie et dominance

Une cellule biologique est diploïde (par opposition à haploïde) si les chromosomes qu'elle contient sont présents par paires. Par exemple l'être humain possède 23 paires, chaque paire contient un chromosome issu du père et un chromosome issu de la mère. Ces chromosomes contiennent les mêmes gènes mais n'ont pas forcément les mêmes valeurs. Le problème est de déterminer la valeur que va exprimer le gène qui a deux valeurs différentes (une valeur dans chaque chromosome).

Dans la nature pour déterminer, à chaque gène, l'allèle qui va s'exprimer on définit la notion de dominance. Un allèle est dite dominant s'il s'exprime automatiquement, quelque soit sa valeur sur l'autre chromosome.

Pour intégrer la notion de diploïdie et dominance aux algorithmes génétiques, on code chaque individu par une paire de chromosomes construite sur trois valeurs $(-1, 0, 1)$ telle qu'on définit la relation de dominance : $-1 < 0 < 1$. Les opérations génétiques (croisement, mutation et sélection) ne se font pas entre individus mais entre les chromosomes homologues d'un seul individu (Figure 2.7). La reproduction donne naissance à un enfant qui possède un chromosome issu du premier parent et un chromosome issu du deuxième parent.

2.4.3 Parallélisme

Le but principal de la parallélisation est de réduire le temps de calcul. Il existe deux méthodes utilisées pour la parallélisation des algorithmes génétiques (31) :

1. La première consiste à diviser la population de taille μ en sous-populations de taille N ($N < \mu$) traitées chacune sur une machine ;
2. La seconde maintient la population totale sur une seul machine, et les autres machines sont utilisées pour calculer la fonction d'évaluation des individus en même temps.

Pour en savoir plus sur le sujet de parallélisme des algorithmes génétiques voir les articles (41; 31) et des exemples d'utilisation dans les références (55; 51).

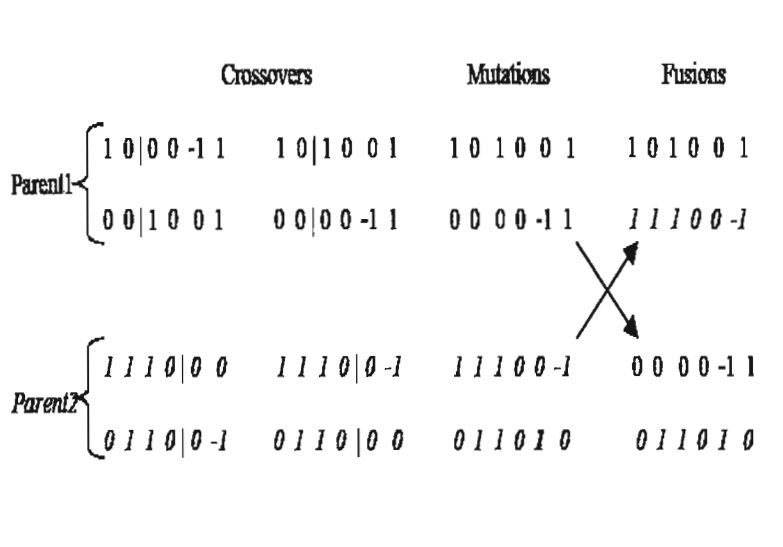

FIGURE 2.7 Exemple de reproduction avec la technique diploïdie et dominance.

2.5 Exemples d'application

2.5.1 Classification et récupération intelligentes de composants logiciels

Problématique

Des composants logiciels peuvent être conçus en tant qu'unités indépendantes et autonomes qui sont offertes par des fournisseurs et peuvent être assemblées, par l'intermédiaire de leurs interfaces publiques, pour former un logiciel complet (8).

À cet effet, les utilisateurs des composants de logiciel doivent rechercher parmi les composants disponibles sur le marché, évaluer leurs caractéristiques et acquérir les plus appropriés pour les intégrer selon les conditions et les contraintes fonctionnelles de leur projet.

Le problème fondamental pour des fournisseurs de composants logiciels est celui de classifier les composants par catégories pour simplifier la découverte et la récupération rapide.

Solution génétique

Codage : Le codage est la façon de présenter les caractéristiques d'un composant logiciel. Un composant logiciel est représenté par certaines caractéristiques qu'on peut coder sous forme binaire pour simplifier la tâche de classification et de découverte. La figure 2.8 montre un exemple de codage binaire d'un composant logiciel.

Fonction d'adaptation : Elle calcule une valeur de similarité entre les composants logiciels. Par exemple si la valeur d'adaptation entre `composant_1` et `composant_2` est égale a 40% alors les deux composants sont identiques à 40% relativement à leurs différentes caractéristiques.

On utilise un seuil pour déterminer si deux composants appartiennent à la même classe ou non.

Algorithme : Il procède comme suit :

1. Créer une population aléatoire de N chromosomes
2. Pour chaque génération de l'algorithme génétique :
   (a) Appliquer le croisement à chaque paire de classificateurs, où chaque paire est choisie aléatoirement selon une probabilité de croisement
   (b) Appliquer la mutation à un classificateur aléatoirement choisi selon une probabilité de mutation
3. Exécuter la classification des composants : pour chacun des M classificateurs :
   (a) Comparer chaque classificateur de valeurs avec les caractéristiques de chaque composant. Si un élément est assez proche (déterminé par un seuil) d'un classificateur alors attribuer le composant à la classe représentée par ce classificateur. Normalement, un grand nombre de composants sera attribué à chaque chromosome.
   (b) Choisir les K classificateurs principaux (chromosomes) en terme du nombre de composants assignés.
   (c) Si l'adaptation moyenne de la génération (la moyenne de la fonction `fitness` des composants) actuelle est plus grande que celle de la génération précédente alors créer une nouvelle population en sélectionnant les chromosomes en fonction de leur aptitude et répéter l'étape 3. Sinon, répéter l'étape 2.

2.5.2 Algorithmes génétiques en recherche d'information

Problématique

La recherche d'information tente de trouver les documents qui répondent à un besoin exprimé par un usager, et ce parmi une grande collection de documents (58). Soltan (52) présente des techniques qui permettent d'évaluer le poids d'un terme dans un document appartenant à une collection.

La recherche d'information dans une collection de données dépend de l'indexation des documents, l'analyse sémantique de la requête et le calcul de la similarité entre la requête exprimée

par l'utilisateur et les documents de la collection pour permettre par la suite de classer les documents par leurs valeurs de similarité à la requête (38). La difficulté réside dans la recherche de la sémantique associée aux documents et à la requête.

Solution génétique

Codage : Étant donné un document d_h, avec $h = 1, \ldots, n$ et un ensemble de terme T_j avec $j = 1, \ldots, m$, le descripteur associé à d_h possède la forme suivante :

$$d_h = (t_{1h}, t_{2h}, ..., t_{mh}) \tag{2.1}$$

tel que t_{jh} poids du terme j dans le document h.

Un descripteur d'un document est la concaténation des poids (gène) de tous les termes.

On a donc un individu

$$i = (d_1, d_2, ..., d_n) = (t_{11}, ...t_{m1}, t_{12}, ..., t_{m2}, ..., t_{1n}, ...t_{mn}) \tag{2.2}$$

Pour arriver à une représentation binaire de ces vecteurs, les valeurs t_{hj} sont discrétisées et remplacées par des entiers entre 0 et 10 qui seront représentés sur 4 bits.

La population de base contient un individu obtenu par le processus d'indexation et sept autres construits à partir des jugements de pertinence répartis équitablement.

Fonction de performance La fonction de performance est la similarité entre la requête et un document de la collection. Cette similarité est calculée par l'équation 2.3.

$$sim(d_h, q) = \frac{\sum(t_{hj} t_{qj})}{\sqrt{\sum(t_{hj}^2) \sum(t_{qj}^2)}} \tag{2.3}$$

tel que :

$sim(d_h, q)$: similarité entre la requête q et le document d_h

t_{hj} : poids du terme j dans le document h, calculé avec l'équation 2.4

$$t_{hj} = ntf_{hj} * nidf_j \tag{2.4}$$

$$\text{avec } ntf_{hj} = \frac{tf_{hj}}{max_g(tf_{hg})} \text{ et } nidf_j = \frac{log(n) - log(df_j)}{log(n)}$$

tel que :

n : nombre de documents dans la collection.

tf_{hj} : fréquence du terme j dans le document h.

df_j : fréquence du terme j dans la collection.

Algorithme

- Les paramètres sont le nombre d'individus qui forment la population initiale et le nombre de générations.
- La population initiale contient des individus obtenus par le processus d'indexation et d'autres construits à partir de jugement de pertinence. Le jugement de pertinence de requêtes est une approche d'apprentissage en recherche d'informations proposé par Vrajitoru dans sa thèse de doctorat.
- L'opération de croisement utilisée est nommée « opération de croisement dissocié », laquelle consiste à faire une distinction entre les points de croisement des deux parents (Figure 2.9).

Comme dans la plupart des algorithmes génétiques, le choix des valeurs des différents paramètres s'avère important pour la performance. Enfin, il semble que le nouvel opérateur de croisement apporte une amélioration significative au niveau de qualité de solutions obtenus comparativement à d'autres algorithmes génétiques.

Conclusion

Les algorithmes génétiques sont les approches métaheuristique les plus répandues pour résoudre des problèmes difficiles d'optimisations et de recherche. Leur efficacité est déterminée par les opérateurs génétiques qui sont utilisés et par la fonction d'évaluation.

Une fonction d'évaluation multi-critères permet d'augmenter le champs d'utilisation de ces approches génétiques. Alors que la possibilité d'utiliser le parallélisme dans l'exécution de ces algo-

rithmes permet un gain important en temps d'exécution.

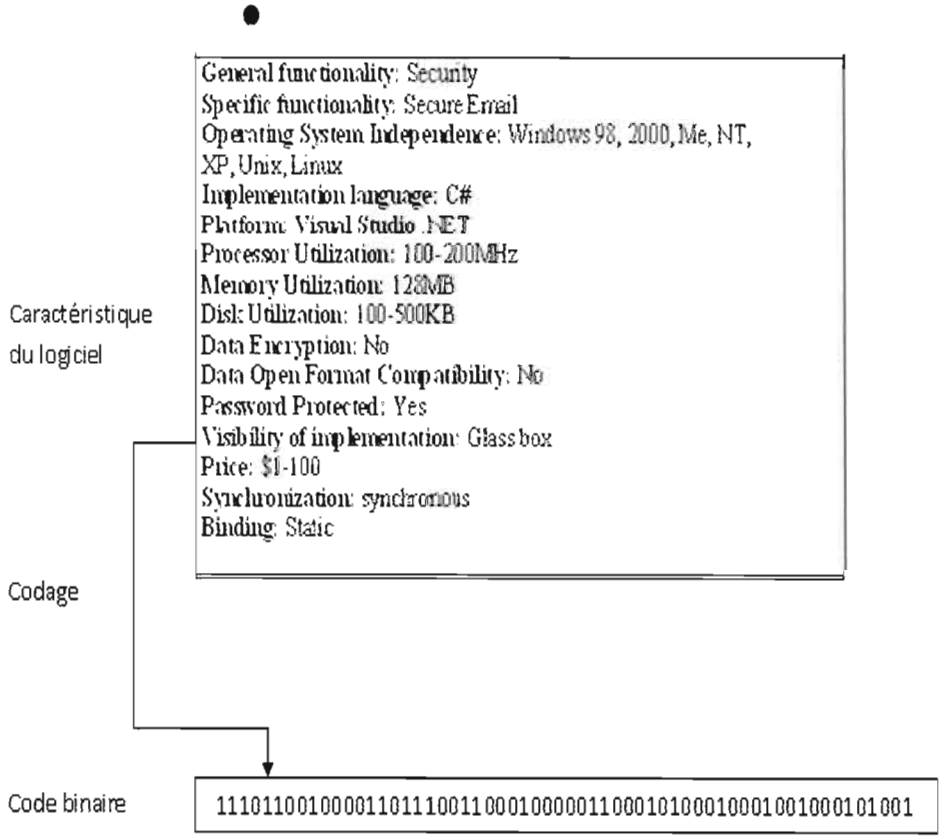

FIGURE 2.8 Exemple de codage d'un composant logiciel (tiré de (8)).

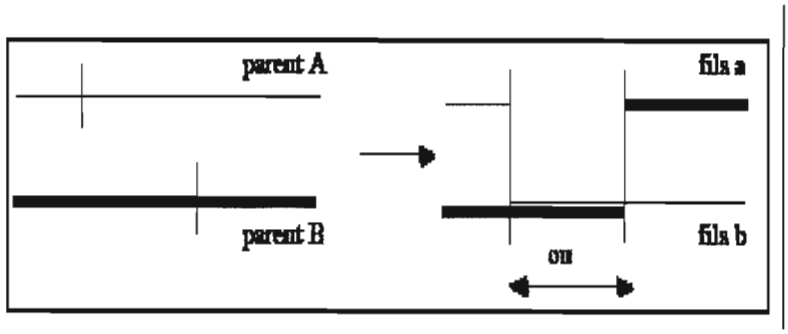

FIGURE 2.9 Croisement dissocié (tiré de (58)).

CHAPITRE III

ANALYSE DE QUELQUES APPROCHES

Introduction

Après avoir présenté les services Web et les techniques liées à leur description ainsi que leur cycle de vie et leur composition, nous avons abordé les approches génétiques et quelques exemples d'applications. Dans le présent chapitre nous allons étudier quelques approches de composition et de découverte de services Web qui nous aideront dans notre recherche.

La première approche que nous présentons est une thèse de doctorat qui cherche à développer une architecture de composition automatique et optimale de services Web (22). Dans cette approche, la découverte de services Web utilise une description sémantique. Dans la phase d'optimisation, le processus de composition fait appel à l'algorithme génétique NSG-II, utilisé dans plusieurs domaines d'application.

Dans la deuxième approche, l'auteur restreint sa recherche à la description syntaxique des services Web (3). Il utilise les technologies et les normes de services qui sont standardisées par le W3C. Cet exemple donne une architecture de composition de services Web basée sur leur signature.

Finalement, la composition de services Web par couverture de fonctions est une autre approche pour la composition des services que nous étudierons.

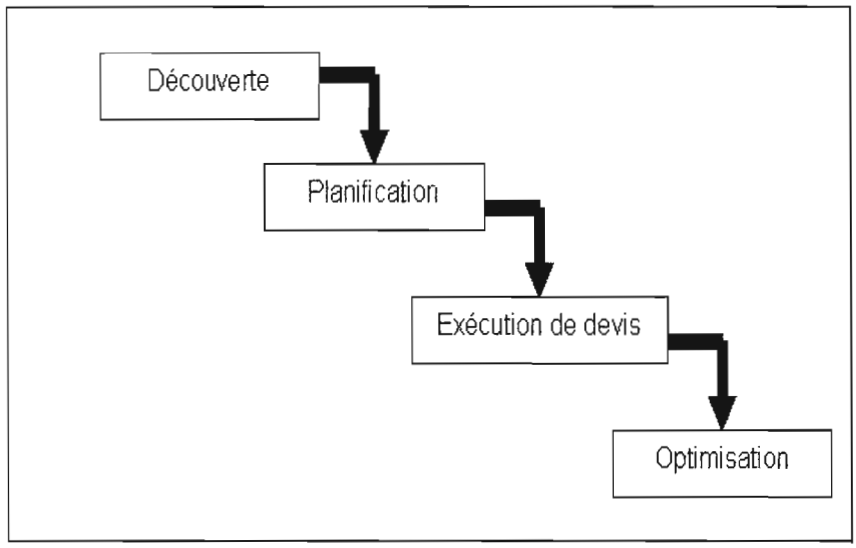

FIGURE 3.1 Les phases de SPOC (tiré de (22)).

3.1 SPOC : Semantic based Planning Optimized Compositions

3.1.1 Description de l'approche

Barreiro Claro propose dans sa thèse de doctorat (22) un canevas appelé SPOC (*Semantic based Planning Optimized Compositions*) pour composer de manière automatique et optimale des services Web. Plus précisément, SPOC est un canevas [1] dédié à la réalisation de devis en quatre phases (Figure 3.1) qui permet la composition automatique de services Web :

la découverte de services : cette phase permet d'identifier les services à partir d'une ontologie de services UDDI. Dans cette phase, SPOC utilise le web sémantique comme langage de description des services Web.

Dans cette première phase, l'auteure propose une ontologie du domaine qui organise les services Web selon une similarité sémantique. Une fois une tâche définie, le processus de découverte cherche dans cette ontologie les services Web qui le réalisent.

1. « le canevas est un synopsis général schématisant les lignes principales du scénario ». `http://fr.wikipedia.org/wiki/Canevas`

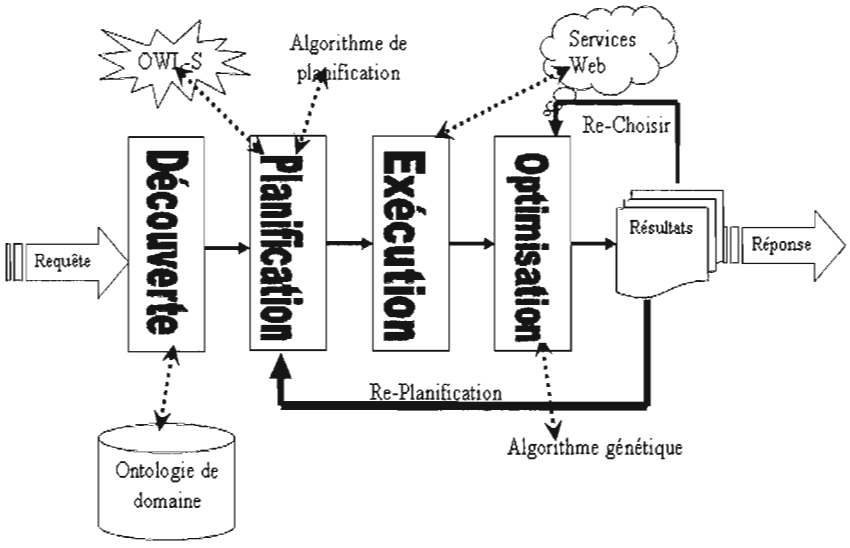

FIGURE 3.2 Le canevas SPOC (22).

la planification : cette phase concerne la composition automatique. La planification permet d'ordonner les tâches à réaliser et de choisir les services Web qui peuvent réaliser chaque tâche. Dans SPOC, Claro utilise JESS (*Java Expert System Shell*), qui est un moteur de règles de planification, pour déterminer les services Web qui participeront à la composition ainsi que leur ordonnancement.

l'exécution : cette phase envoie une requête à chaque service pour obtenir des informations comme le coût, la durée, etc.

l'optimisation : cette phase a pour but de proposer à l'utilisateur un certain nombre de compositions optimisées selon des critères de qualité prédéfinis (e.g., le coût, le prix, le chiffre d'affaire, la réputation) en utilisant l'algorithme génétique NSGA-II (*Non-Dominated Sorting Genetic Algorithm*) (37).

La figure 3.2 montre le canevas SPOC et ses interactions avec l'environnement extérieur.

Il est important de remarquer que SPOC utilise une ontologie du domaine pour la découverte de services Web et un algorithme génétique dans la phase d'optimisation.

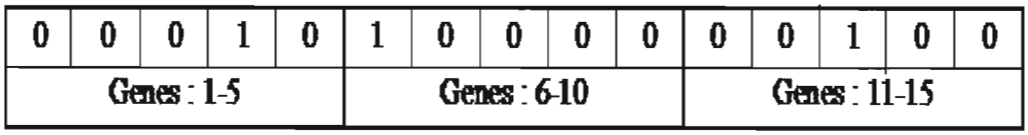

FIGURE 3.3 Exemple d'un chromosome dans SPOC (tiré de (22)).

L'application de l'algorithme génétique dans le processus de composition de services Web

L'algorithme génétique NSGA-II (37) est utilisé dans SPOC pour optimiser les compositions de services Web (dans la phase optimisation). La figure 3.3 présente un exemple de chromosome dans lequel chaque gène a une valeur binaire. Cette valeur détermine si un service appartient ou non à la composition proposée.

Dans cet exemple (figure 3.3), le chromosome représente une composition avec 15 services et 3 tâches : les services de 1 à 5 pour la tâche 1, les services de 6 à 10 pour la tâche 2 et les services de 11 à 15 pour la tâche 3. L'algorithme choisit dans ce cas le service 4 pour la tâche 1, le service 6 pour la tâche 2 et le service 13 pour la tâche 3.

NSGA-II est développé en langage C. Il a prouvé son efficacité dans la résolution de nombreux problèmes d'optimisation et de classification (24). Le canevas SPOC fait un appel direct à NSGA-II.

Dans la phase d'exécution, SPOC enregistre les paramètres [2] trouvés dans un fichier . Ces valeurs sont utilisées par l'algorithme génétique NSGA-II qui génère un ensemble de fichiers contenant les solutions.

3.1.2 Les apports de cette approche

Dans cette approche, l'auteure propose une architecture de composition automatique et optimale de services Web. Les idées les plus importantes que nous avons identifiées sont :

2. Les paramètres d'entrées et de sorties d'une composition de services Web

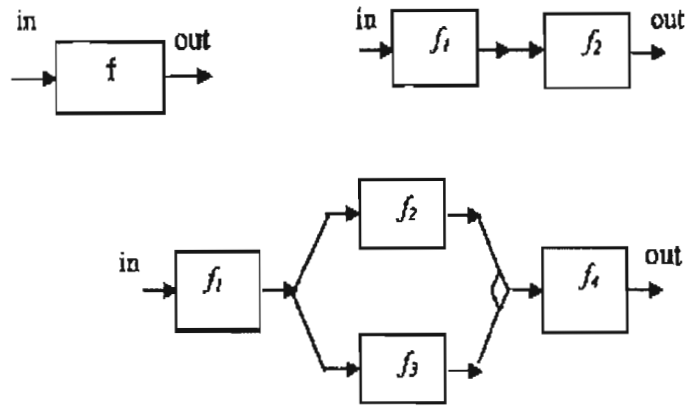

FIGURE 3.4 Exemples de composition de fonctions (2).

- L'utilisation de la description sémantique de services Web dans la phase de découverte ;
- L'utilisation d'un algorithme génétique pour optimiser la composition de services Web.

Signalons que SPOC est basé sur les services Web sémantiques. Il utilise dans la phase de découverte une ontologie de domaine qui permet la découverte automatique de services Web. Toutefois, jusqu'à ce jour, cette ontologie n'existe pas — pour ses tests, l'auteure utilise OPS[3] (*Ontology to Publish Services*).

3.2 Composition de services Web par appariement de signatures

3.2.1 Description de l'approche

Dans son mémoire de maîtrise (3), Alkamari propose une méthode de composition de services Web par couverture de fonctions. La composition de services par couverture de fonctions est une technique proposée Mili et al. (32). Dans cette technique on traite les opérations des services Web comme des fonctions telles que les types de messages d'entrées de l'opération sont les entrées de la

3. Une ontologie décrivant un domaine spécifique en utilisant des classes et des propriétés.

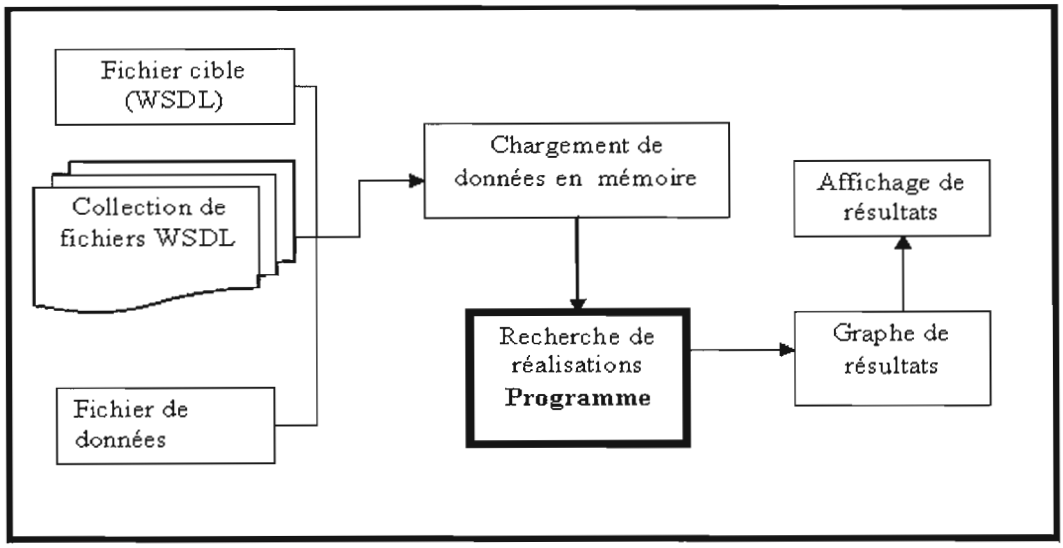

FIGURE 3.5 Architecture de l'approche (tiré de (3)).

fonction et les types de messages de sorties en sont les sorties.

La recherche d'une composition d'opérations pour obtenir une opération spécifique se limite à la recherche des compositions de fonctions correspondantes pour obtenir la fonction de l'opération désirée. La figure 3.4 montre un exemple simple de composition de fonctions.

La figure 3.5 présente l'approche de composition de services Web par appariement de signatures. Le programme principal, au centre de la figure, prend comme entrées une collection de fichiers WSDL et une description du service désiré. Il parcourt toutes les réalisations possibles pour trouver celles qui amènent vers le service Web recherché.

Une réalisation est le chemin comprenant un ou plusieurs services qui mène d'un ensemble de type A à un ensemble de type B.

Le travail est basé sur les algorithmes suivants :

Algorithme EnumerateRealizations : Cet algorithme (Figure 3.6) (2) permet de trouver la réalisation minimale d'une fonction g parmi une collection de fonctions $F_0, \ldots, F_{k-1}$.

Algorithme de génération de composition à partir de réalisations : Cet algorithme (32)

```
Algorithm EnumerateRealizations
(Input k      : StageNumber;
 Input F_{k-1}  : CollectionOfFunctions;
 Input T_{k-2}  : SetOfTypes;
 Input g      : AFunction;
 Output all the realizations with prefix
                    F_1, F_2, ..., F_{k-1}));
begin
(1)  T_{k-1} - T_{k-2} ∪ Output (F_{k-1});
(2)  C_{k-1} - {f | Input (f) is a subset
               of T_{k-1} but not of T_{k-2}};
(3)  for every subset H of C_{k-1} do
(4)     if F_{k-1} is a minimal cover of
        (Input (H) ∪ Output (g) ∩ Output
        (F_{k-1})) - T_{k-2} then
(5)        if T_{k-1} ∪ Output (H) contains
                    Output (g) then
(6)           if H is a minimal cover of
                 Output (g) ∩ Output (H) -
                    T_{k-1} then
(7)              display the realization
              end
(8)        elsif Output (H) is
              not contained in T_{k-1} then
(10)          F_k - H;
(11)          EnumerateRealizations (k+1,
                          F_k, T_{k-1}, g)
(12)       end
(13)    end
(14) end
end EnumerateRealizations
```

FIGURE 3.6 Algorithme EnumerateRealizations (tiré de (2)).

permet de générer toutes les compositions possibles de la réalisation trouvée par l'algorithme.

3.2.2 Les apports de cette approche

Les apports à souligner de cette approche sont :

- L'utilisation de la description normalisée de services Web (WSDL, UDDI, etc.) ;
- L'utilisation de théories mathématiques liées aux fonctions pour réaliser des compositions d'opérations.

On remarque aussi que :

1. l'application va chercher toutes les réalisations possibles, ce qui augmente de façon importante le temps d'exécution si le nombre de services qui constituent la base de données est important.
2. une opération est identifiée uniquement par les types des messages d'entrées et de sorties. Or cette identification est incomplète. Le nom d'opération, les noms des messages ainsi que la description d'opération qui peut exister dans le fichier WSDL peuvent jouer un rôle important dans l'identification d'opérations.

3.3 Autres approches

3.3.1 Woogle – Web Service Search Engine

Woogle est un projet de l'université de Washington visant à produire un moteur de recherche de services Web basé sur la technique de similarité entre les opérations.

Cette approche est basée sur la description syntaxique de services Web (figure 3.7) en tenant compte de sa description et de la description des différentes opérations. La recherche des services Web similaires est déterminée par la recherche des opérations similaires. Deux opérations de fonctionnalités sont similaires si leurs entrées et leurs sorties sont similaires.

L'algorithme du moteur de recherche Woogle permet la composition d'opérations. S'il trouve deux opérations telles que la sortie d'une est similaire à l'entrée de l'autre, il génère l'opération composite.

Woogle introduit les technologies sémantiques dans son processus de recherche. En tant que

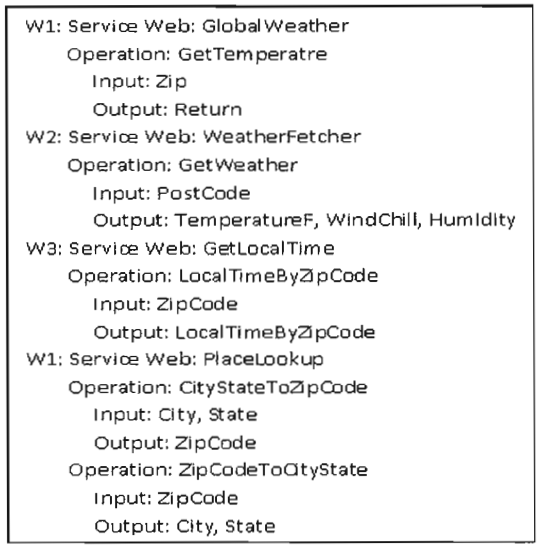

FIGURE 3.7 Exemple de présentation de services Web (26).

mécanisme de base, Woogle est une solution générique qui est conforme à la norme WSMO [4] (33)(*Web Service Modeling Ontology*).

Les apports à souligner par cette approche sont :

- la représentation de services Web sous forme simple et complète ;
- la technique de similarité entre les entrées-sorties des opérations ;
- le volet sémantique de l'approche et la conformité avec les normes de Web sémantiques.

3.3.2 WSPAB : A Tool for Automatic Classification et Selection of Web Services Using Formal Concept Analysis

Description de l'approche

L'objectif de cette approche est le développement d'une application, WSPAB (*Web Service Personal Address Book*), qui facilite la découverte des services Web les plus pertinents à une requête donnée.

4. Une approche pour la réalisation des services Web sémantiques compatible avec le standard WSDL.

WSPAB fonctionne avec les autres approches de découverte de services Web, comme `seekda`, dans le but de donner une solution pertinente et précise. L'auteur remarque que le nombre de services d'une solution retournée par les moteurs de recherche actuels est important. Par conséquent, le travail de sélection manuelle d'un service Web demeure difficile.

La figure 3.8 présente l'architecture de l'application WSPAB et définit les étapes d'exécution suivantes :

- Analyse des services Web ;
- Filtration des services Web ;
- Extraction de signatures d'opérations ;
- Identification des services pertinents ;
- Tri des signatures.

La classification de services Web se réalise à l'aide d'un outil d'analyse formelle de concepts appelé Galicia (*Galois Lattice Interactive Constructor*), qui analyse les relations binaires entre les services et retourne le résultat sous forme graphique.

Les apports à souligner par cette approche sont :

- l'interaction entre l'application et l'outil de découverte de services Web (seekda) ;
- l'utilisation de signatures d'opérations ;
- l'utilisation d'un outil d'analyse formel pour la classification des services Web pertinents.

Conclusion

Dans ce chapitre, nous avons présenté quelques approches pour la découverte et la composition de services Web. Chacune d'entre elles traite un point particulier et utilise des techniques appropriées mais aucune ne permet de combiner la recherche et la composition de services Web ainsi que profiter de sa description syntaxique et sémantique.

Dans le chapitre suivant, nous présentons notre approche de découverte et de composition de services Web qui utilise les descriptions syntaxiques (fichier WSDL).

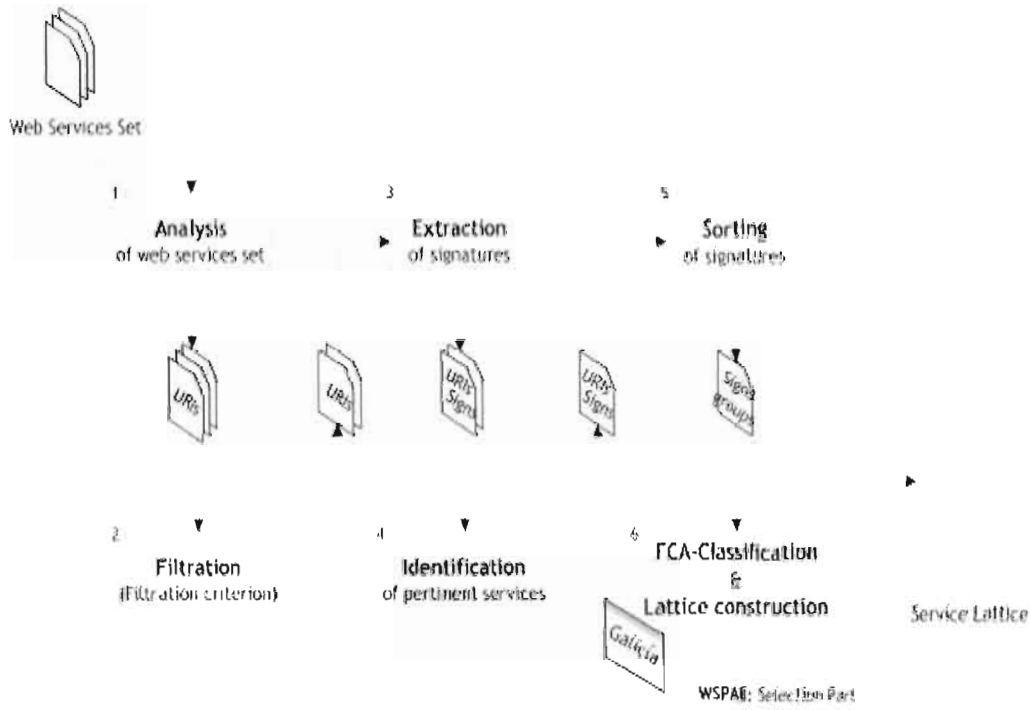

FIGURE 3.8 Architecture de l'application WSPAB (tiré de (11)).

CHAPITRE IV

UN ALGORITHME GÉNÉTIQUE POUR LA RECHERCHE ET LA COMPOSITION DE SERVICES WEB

Introduction

Un service Web est un regroupement d'opérations. Une opération est définie par un nom et un ensemble de messages d'entrées et de sorties. Pour trouver un service Web particulier, il faut donc trouver les différentes opérations qui le constituent.

Dans notre recherche, on parle d'ailleurs des opérations, d'une liste d'opérations qui forment notre espace de recherche, d'une opération cible et d'un algorithme génétique qui permet de trouver, s'il y a lieu, une opération de l'espace de recherche similaire à celle recherchée.

Il est possible de trouver une opération de l'espace de recherche proche de celle désirée. Mais, dans la plupart des cas, la composition de certaines opérations donne un résultat plus approprié. Parfois une petite modification sur l'entrée ou la sortie de quelques opérations peut améliorer le résultat et donner une solution plus pertinente.

Notre algorithme (Figure 4.1) reçoit en entrée une liste d'opérations qui représentent l'espace de recherche ainsi que l'opération à rechercher. Le résultat de l'algorithme est une liste d'opérations qui ont une bonne similarité avec l'opération recherchée. Une opération appartient à la solution retournée par l'algorithme peut être :

- une opération de l'espace de recherche;

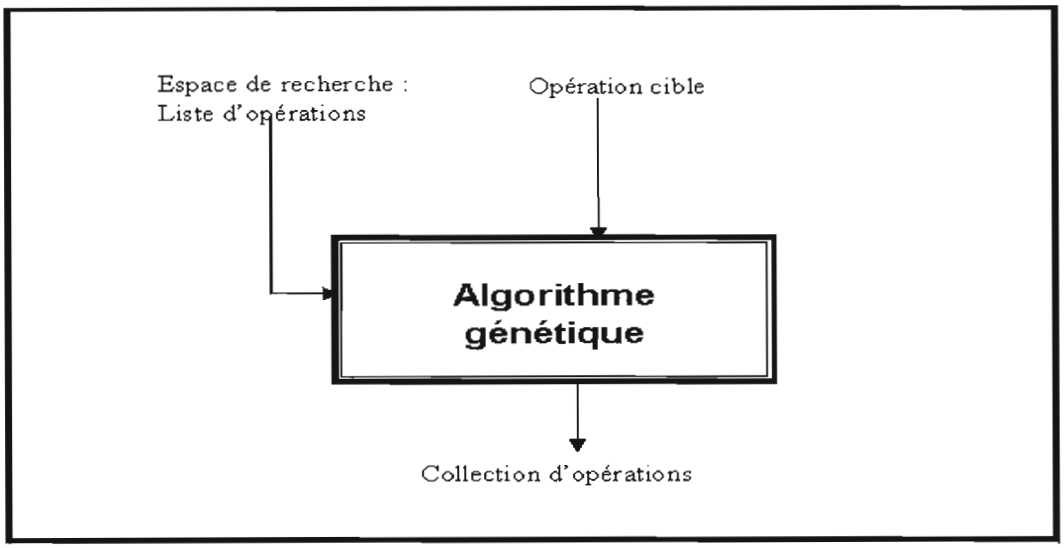

FIGURE 4.1 Architecture de notre approche génétique.

- le résultat d'une suite de composition des opérations mutées ou non mutées.

4.1 Codage

Dans notre approche on utilise un codage sous forme d'arbre syntaxique. On code génétiquement une opération d'un service Web comme suit (Figure 4.2) :

Géne : une variable d'entrée ou de sortie (un nom de variable et un type) ;

Chromosome : une liste de variables d'entrée ou de sortie ;

Individu : une opération définie par un nom (nom de l'individu), un chromosome d'entrée et un chromosome de variables de sortie ;

Génération (appelée aussi *population*) : M individus (opérations) qui représentent un ensemble de solutions potentielles à un instant donné.

Exemple :

La figure 4.3 présente un exemple d'encodage de l'opération `translate` du service Web `NLGTranslateService` de la Figure 4.4.

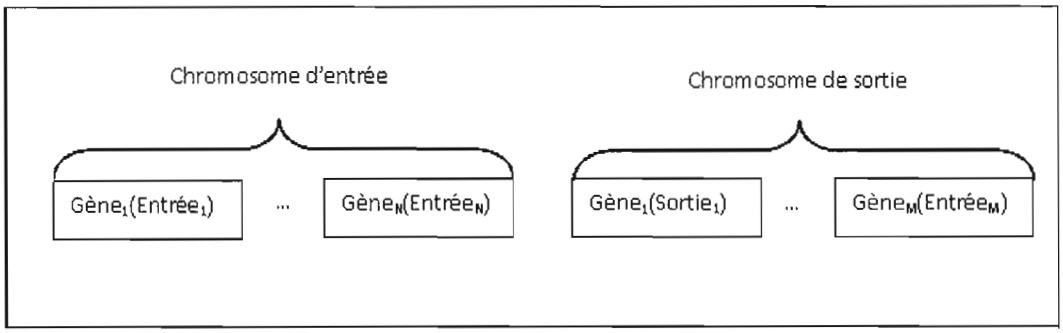

FIGURE 4.2 Encodage d'une opération d'un service Web.

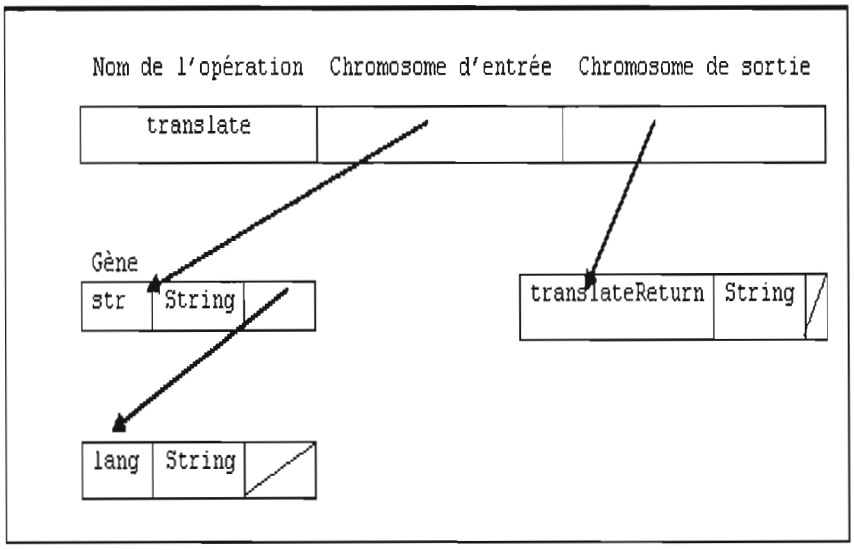

FIGURE 4.3 Encodage de l'opération `translate`.

```
Nom Opération : translate
   Entrées :
      str : String
      lang : String
   Sorties :
      tranlateReturn : String
```

FIGURE 4.4 Opération `translate` du service `NLGTranslateService`.

4.2 Algorithme

Notre algorithme génétique (Figure 4.5) permet de découvrir les opérations pertinentes et d'améliorer l'ensemble de solutions potentielles par le biais de la composition. Il procède de la façon suivante (diagramme de la figure 4.6) :

1. Choisir aléatoirement N opérations parmi les opérations des services Web qui forment l'espace de recherche ;
2. Faire évoluer d'une génération à une autre par le biais de croisement tout en cherchant à améliorer la similarité avec l'opération cible ;
3. Si l'algorithme atteint un état de stabilité (impossible de croiser des opérations de la génération en cours), explorer l'espace de recherche pour enrichir la population par un individu externe ;
4. Terminer l'algorithme si l'opération recherchée est trouvée ou si l'espace de recherche est exploré dans sa totalité. Sinon retour à l'étape 2.

4.3 Opérateurs génétiques

4.3.1 Sélection

La fonction de sélection permet de choisir les deux meilleurs opérations qui vont se composer.

Le choix du couple d'opérations pour le croisement dépend de deux paramètres : possibilité de croisement et couples déjà sélectionnés. Le premier détermine les couples d'opérations de la génération en cours qu'on peut croiser tandis que le deuxième détermine les couples déjà sélectionnés pour un croisement dans les générations antérieures.

Notre algorithme choisi le premier couple d'opérations dans la liste des couples qui peuvent se croiser et qui n'est pas sélectionné pour un croisement dan une génération antérieure.

4.3.2 Croisement

L'opérateur de croisement permet de fusionner deux opérations qui ont un ou plusieurs points d'attache. Un point d'attache entre deux opérations est un point où la sortie de la première opération est identique à l'entrée de la deuxième.

Algorithme génétique

Début

Créer la population initiale

Tant que (opération non trouvée) **ET** (il existe des opérations externes non explorées) **Faire**

Tant que (possibilité de croisement) **Faire**

Sélectionner deux opérations

Enfant = croisement des deux opérations sélectionnées

Évaluer Enfant

P = Les N meilleures opérations de P U {Enfant}

Fin Tant que

Choisir une opération externe

Évaluer l'opération

P = Les N meilleures opérations de P U {Opération choisie}

Fin Tant Que

Fin

FIGURE 4.5 Notre algorithme génétique.

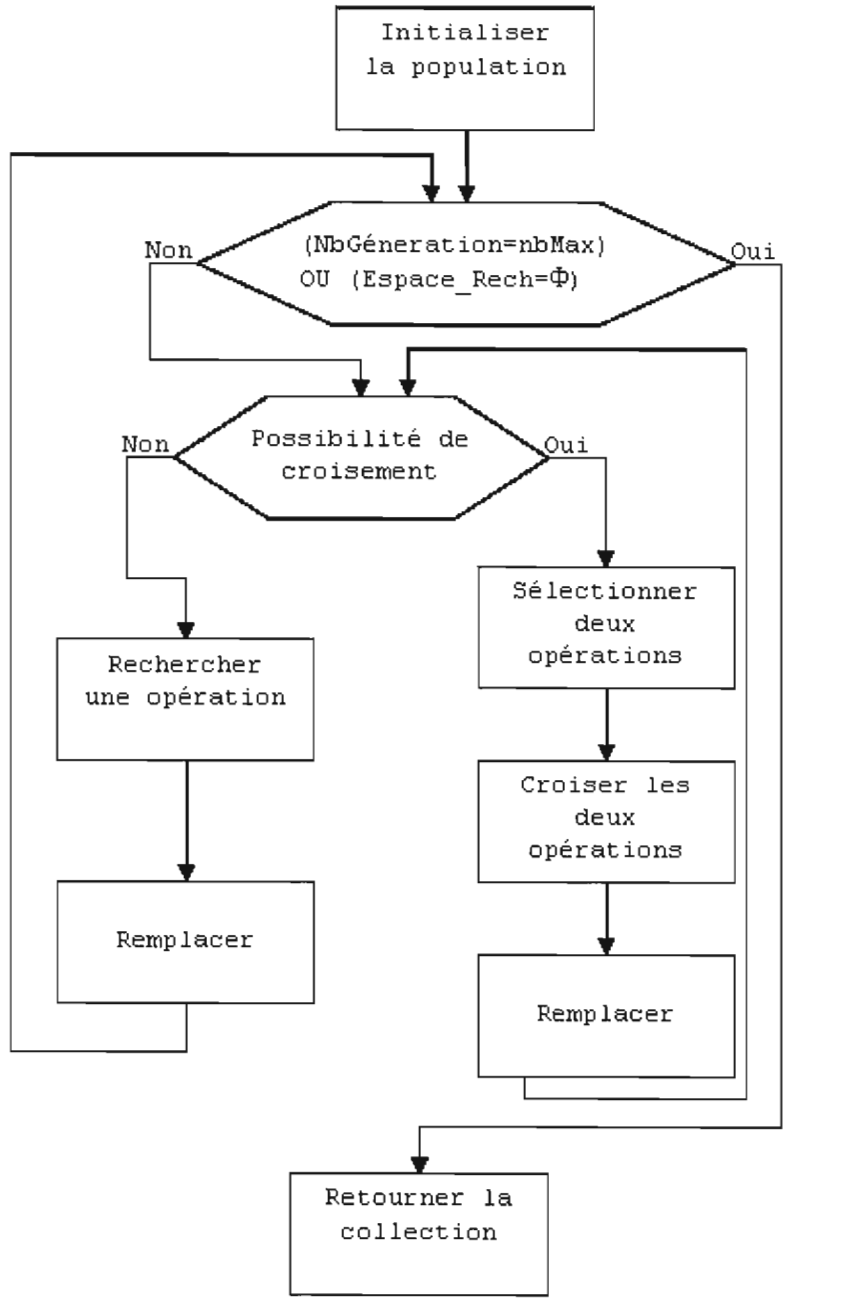

FIGURE 4.6 Diagramme de l'algorithme génétique.

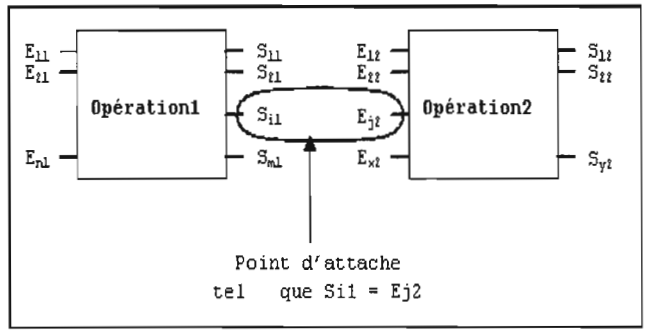

FIGURE 4.7 Exemple de deux opérations qui peuvent se croiser.

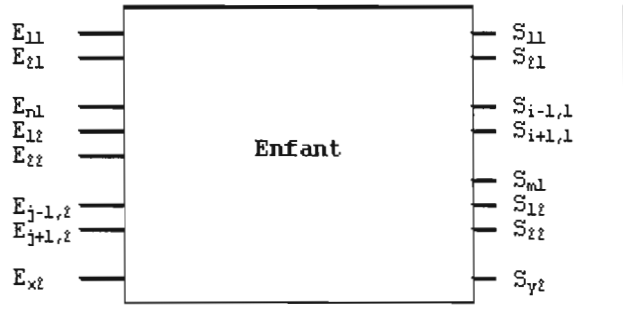

FIGURE 4.8 Résultat du croisement.

La figure 4.7 montre deux opérations (`Opération1` et `Opération2`) où la sortie S_{i1} de l'opération `Opération1` est identique à l'entrée E_{j2} de l'opération `Opération2`. La figure 4.8 montre que l'opération `Enfant` résulte du croisement des opérations `Opération1` et `Opération2`.

4.3.3 Remplacement

L'opération de remplacement détermine quel individu devra disparaître de la population à chaque génération et être remplacé par le nouvel individu. Le critère de sélection est toujours la valeur d'adaptation retournée par la fonction *fitness* – voir section 4.4. Cette opération va choisir N individus (taille de la population) les plus adaptés au problème (la meilleure valeur de similarité avec l'opération cible) parmi les $N + 1$ individus (N individus de la population + le nouveau né). Le procédure de remplacement présentée à la figure 4.9 permet de créer la nouvelle population et

```
Procedure Remplacement
Debut
    i = taille_population;
    Tant que Fitness(population[i])<fitness(enfant) et i<2 faire
        population[i]=population[i-1];
        i--;
    Fin Tant que
    si Fitness(population[i]>fitness(enfant) et i<taille_population alors
        popultaion[i+1]=enfant;
    else if Fitness(population[i]<fitness(enfant) elors
            popultaion[i]=enfant;
        Fin si
    Fin si
Fin
```

FIGURE 4.9 procedure de l'opération de remplacement.

de conserver l'ordonnancement selon la valeur d'adaption des individus (dans l'ordre croissant).

L'évaluation des individus faite par la fonction fitness a un rôle primordial dans le bon fonctionnement de cette opération pour que l'opération de remplacement prenne les bonnes décisions.

4.4 Fonctions d'évaluation

La fonction d'évaluation (appelée aussi fonction d'adaptation ou *fitness*) permet de calculer la similarité entre deux opérations. Une opération est définie par son nom, les variables d'entrée et les variables de sortie. Une variable d'entrée ou de sortie est décrite par son nom et son type.

Une fonction *fitnessIn* calcule une valeur de similarité entre les entrées de deux opérations. Une fonction *fitnessOut* calcule une valeur de similarité entre les sorties de deux opérations. L'algorithme génétique utilise la valeur d'adaptation (*fitness*) dans l'opération de remplacement, tandis que *fitnessIn* et *fitnessOut* jouent un rôle dans le croisement et la mutation.

On suppose qu'on cherche à calculer la similarité entre l'opération `R` qui a N entrées et M sorties et l'opération `E` qui a X entrées et Y sorties. Les valeurs de *fitnessIn*, *fitnessOut* et *fitness*

sont calculées respectivement par les équations 4.1, 4.2 et 4.3.

$$\textbf{fitnessIn(R, E)} = (\sum_{i=1}^{N} Max_{j=1}^{X}(simNom(Nom\_R_i, Nom\_E_j) * simType(Type\_R_i, Type\_E_j)))/N \quad (4.1)$$

$$\textbf{fitnessOut(R, E)} = (\sum_{i=1}^{M} Max(\bigcup_{j=1}^{Y} simNom(Nom\_R_i, Nom\_E_j) * simType(Type\_R_i, Type\_E_j)))/M \quad (4.2)$$

$$\textbf{fitness(R, E)} = pNom * simNom(Op\_1.nom, Op\_2.nom) + pEntrees * fitnessIn(Op\_1.entrees, Op\_2.entrees) + pSorties * fitnessOut(Op\_1.sorties, Op\_2.sorties) \quad (4.3)$$

tel que :
- $simNom(nom\_1, nom\_2)$ = (plus long préfixe Commun)/ longueur de `nom_1` ;
- $simType($`type_1, type_2`$)$ = 1 si `type_1 = type_2`, 0 sinon ;
- $pNom = 4$;
- $pEntrees = 48$;
- $pSorties = 48$.

Nous signalons que notre fonction d'adaptation, avec le calcul de similarité entre deux noms, ne tient compte que des premiers caractères similaire tandis qu'il existe d'autres techniques telles que la distance d'édition [1] qui permet de mesurer la similarité entre deux chaînes de caractères. Nous choisissons cette approche car elle nous semble plus simple.

Exemple

Soit les deux opérations de la figure 4.10. Le détail du calcul de la fonction `fitness` est :

fintnessIn = 0.5

fintnessOut = 0

d'où fitness = $4 * 3/12 + 48 * 0.5 + 48 * 0 = 25\%$

1. Ou la distance de Levenshtein = le nombre minimal de caractères qu'il faut supprimer, insérer ou remplacer pour passer d'une chaîne à l'autre.

```
Nom Opération : getCountries
   Entrées :
      token : String
      datasource : String
   Sorties :
      result : ArrayOfString
```

```
Nom Opération : getAddress
   Entrées :
      point : Point
      token : String
      addressFinderOptions : AddressFinderOptions
   Sorties :
      result : Address
```

FIGURE 4.10 Les deux opérations `getAddress` et `getCountries`.

4.5 Exemple détaillé d'une série d'itérations de l'algorithme

4.5.1 Hypothèses

- on a les cinq opérations présentées à la figure 4.11 dans notre espace de recherche ;
- à la figure 4.11, deux variables de deux opérations reliées par une ligne signifie qu'elles sont identiques (i.e., leurs noms et leurs types sont identiques) ;
- l'opération cible est présentée à la figure 4.12.

4.5.2 Déroulement de l'algorithme

On va voir comment notre algorithme peut réaliser la composition présentée à la figure 4.13 pour obtenir l'opération recherchée.

Étape N

On suppose qu'à l'étape N on a la génération N (tableau 4.1) telle que la population contient les cinq opérations définie précédemment et que les cinq opérations `Opération_6` à `Opération_10` ont une valeur d'adaptation égale à 0.

À cette étape, l'algorithme choisit le meilleur couple à croiser (`Op_1` , `Op_3`), crée l'opération résultant de croisement (`Op_1-Op_3`) et remplace l'opération la plus mauvaise (`Op_10`) par celle-ci.

On signale que l'opération de remplacement insère le nouveau né (résultat de croisement) à la bonne position pour conserver la population ordonnée selon la valeur d'adaptation des opérations.

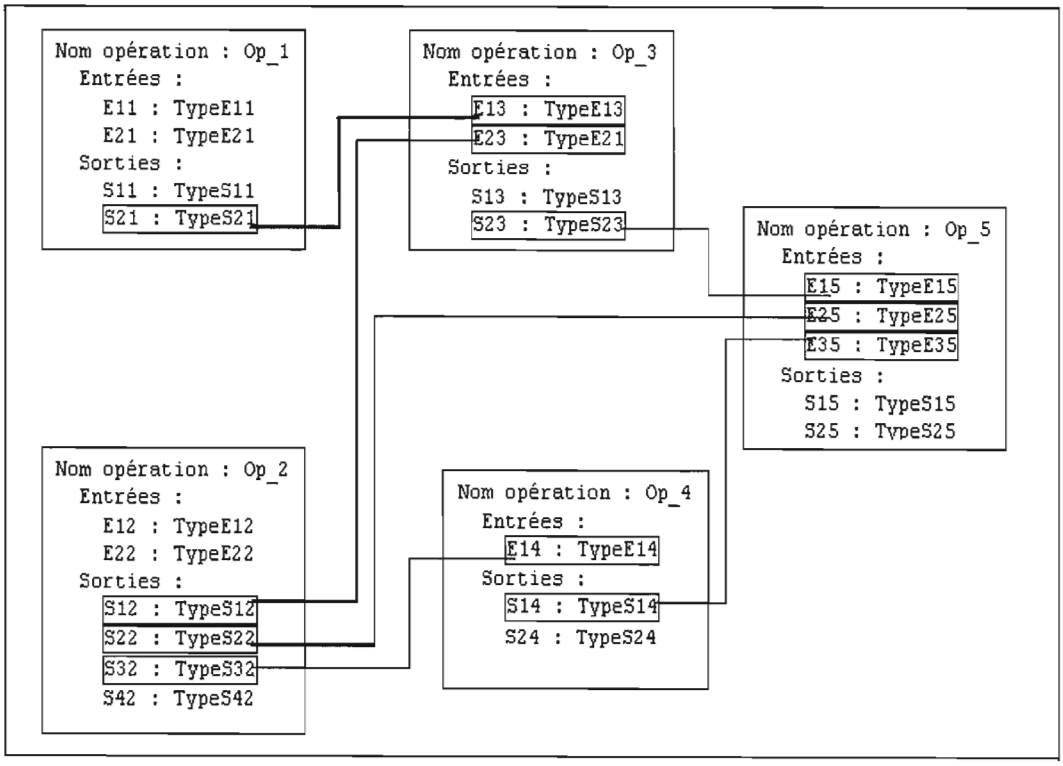

FIGURE 4.11 Les cinq opérations examinées par l'exemple.

```
Nom opération : Opération_R
  Entrées :
    E11 : TypeE11
    E21 : TypeE21
    E12 : TypeE12
    E22 : TypeE22
  Sorties :
    S11 : TypeS11
    S13 : TypeS13
    S15 : TypeS15
    S25 : TypeS25
    S24 : TypeS24
    S42 : TypeS42
```

FIGURE 4.12 Opération cible.

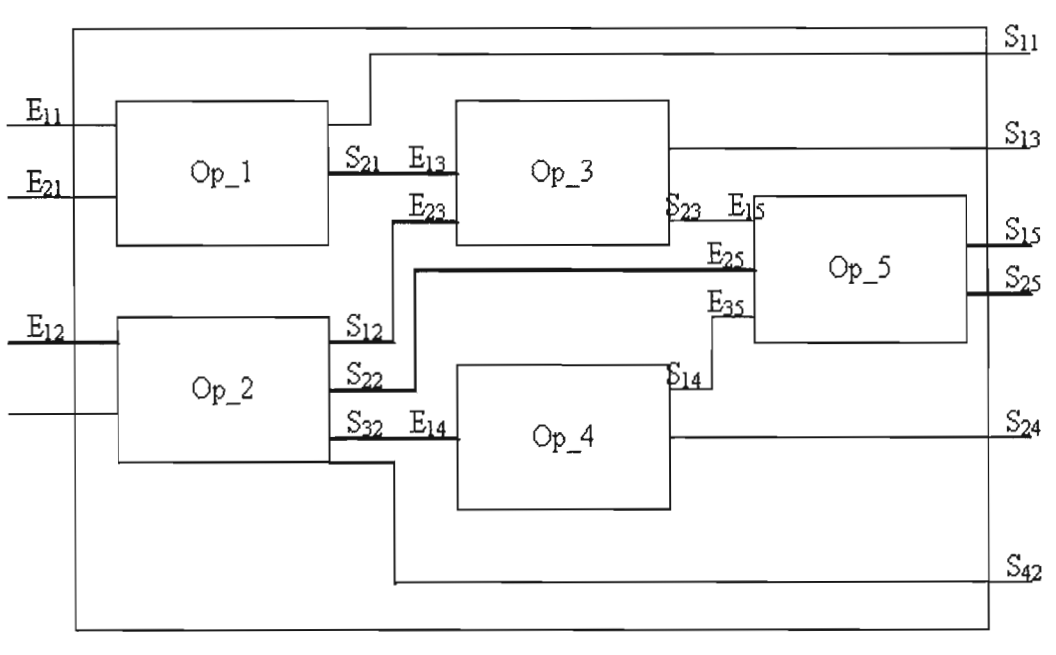

FIGURE 4.13 Exemple de composition possible pour obtenir l'opération recherchée.

TABLE 4.1 Génération N

Population : Op_1 Op_2 Op_5 Op_3 Op_4 etc.
Fitness : 28,8 28,8 19,2 9,6 9,6
Possibilités de croisement : (Op_1,Op_3), (Op_2,Op_3), (Op_2,Op_4), (Op_3,Op_5), (Op_4,Op_5).
Déjà sélectionnés : ϕ
Sélection : (Op_1,Op_3)
Croisement : (Op_1,Op_3) $\Longrightarrow$ Op_1-Op_3
tel que
Nom opération : Op_1-Op_3
  Entrées :
    E11 : TypeE11
    E21 : TypeE21
    E23 : TypeE23
  Sorties :
    S11 : TypeS11
    S13 : TypeS13
    S23 : TypeS23
fitness (Op_1-Op_3) = 38,4
Remplacement : L'opération « Op_1-Op_3 » remplace « Op_10 »

Étape N+1

À l'étape $N+1$, la génération $N+1$ (tableau 4.2) produit l'opération `Op_1-Op_3-Op_5`. L'algorithme insère cette opération dans la population et poursuit son évolution à la génération $N+2$ (tableau 4.3).

Étape N+3

La génération $N+3$ (tableau 4.4) permet la production de l'opération `Op_2-Op_1-Op_3-Op_4` avec une valeur d'adaption de 76,8.

Étape N+4

À cette étape (génération présentée au tableau 4.5), l'algorithme produit l'opération composite présentée à la figure 4.13. La valeur d'adaptation est égale à 96 puisque le nom de cette opération est différent du nom de l'opération cible, alors que les signatures des entrées sont équivalentes.

TABLE 4.2 Génération N+1

Population :	Op_1-Op_3	Op_1	Op_2	Op_5	Op_3	Op_4	etc.
Fitness :	38,4	28,8	28,8	19,2	9,6	9,6	

Possibilités de croisement : (Op_1-Op_3,Op_5), (Op_1,Op_3), (Op_2,Op_1-Op_3), (Op_2,Op_3), (Op_2,Op_4), (Op_3,Op_5), (Op_4,Op_5).

Déjà sélectionnés : (Op_1,Op_3)

Sélection : (Op_1-Op_3,Op_5)

Croisement : (Op_1-Op_3,Op_5) $\Longrightarrow$ Op_1-Op_3-Op_5

tel que

Nom opération : Op_1-Op_3-Op_5

Entrées :

E11 : TypeE11
E21 : TypeE21
E23 : TypeE23
E25 : TypeE25
E35 : TypeE35

Sorties :

S11 : TypeS11
S13 : TypeS13
S15 : TypeS15
S25 : TypeS25

fitness (Op_1-Op_3) = 57,6

Remplacement : L'opération « Op_1-Op_3-Op_5 » remplace « Op_9 »

TABLE 4.3 Génération N+2

Population : Op_1-Op_3-Op_5 Op_1-Op_3 Op_1 Op_2 Op_5 etc.
Fitness : 57,6 38,4 28,8 28,8 19,2
Possibilités de croisement : (Op_1-Op_3,Op_5), (Op_1,Op_3), (Op_2,Op_1-Op_3), (Op_2,Op_3), (Op_2,Op_4), (Op_3,Op_5), (Op_4,Op_5).
Déjà sélectionnés : (Op_1,Op_3), (Op_1-Op_3,Op_5)
Sélection : (Op_2,Op_1-Op_3)
Croisement : (Op_2,Op_1-Op_3) $\Longrightarrow$ Op_2-Op_1-Op_3
tel que
Nom opération : Op_2-Op_1-Op_3
Entrées :
E11 : TypeE11
E21 : TypeE21
E12 : TypeE12
E22 : TypeE22
Sorties :
S11 : TypeS11
S13 : TypeS13
S23 : TypeS23
S22 : TypeS22
S32 : TypeS32
S42 : TypeS42
Fitness (Op_1-Op_3-Op_5) = 67,2
Remplacement : L'opération « Op_2-Op_1-Op_3 » remplace « Op_8 ».

TABLE 4.4 Génération N+3

Population :	Op_1-Op_3-Op_5	Op_1-Op_3-Op_5	Op_1-Op_3	Op_1	etc.
Fitness :	67,2	57,6	38,4	28,8	

Possibilités de croisement : (Op_1-Op_3-Op_5,Op_4), (Op_1-Op_3-Op_5,Op_5), (Op_1-Op_3,Op_5), (Op_1,Op_3), (Op_2,Op_1-Op_3), (Op_2,Op_3), (Op_2,Op_4),(Op_3,Op_5), (Op_4,Op_5).

Déjà sélectionnés : (Op_1,Op_3), (Op_1-Op_3,Op_5),(Op_2,Op_1-Op_3)

Sélection : (Op_1-Op_3-Op_5,Op_4)

Croisement : (Op_1-Op_3-Op_5,Op_4) $\Longrightarrow$ Op_2-Op_1-Op_3-Op_4

tel que

Nom opération :Op_2-Op_1-Op_3-Op_4

Entrées :

E11 : TypeE11
E21 : TypeE21
E12 : TypeE12
E22 : TypeE22

Sorties :

S11 : TypeS11
S13 : TypeS13
S23 : TypeS23
S22 : TypeS22
S14 : TypeS14
S24 : TypeS24
S42 : TypeS42

Fitness (Op_1-Op_3-Op_5) = 76,8

Remplacement : L'opération « Op_2-Op_1-Op_3-Op_4 »remplace « Op_7 ».

TABLE 4.5 Génération N+4

Population :	Op_2-Op_1-Op_3-Op_4	Op_1-Op_3-Op_5	Op_1-Op_3-Op_5 etc.
Fitness :	76,8	67,2	57,6

Possibilités de croisement : (Op_2-Op_1-Op_3-Op_4,Op_5), (Op_1-Op_3-Op_5,Op_4), (Op_1-Op_3-Op_5,Op_5), (Op_1-Op_3,Op_5), (Op_1,Op_3), (Op_2,Op_1-Op_3), (Op_2,Op_3), (Op_2,Op_4), (Op_3,Op_5), (Op_4,Op_5).

Déjà sélectionnés : (Op_1,Op_3), (Op_1-Op_3,Op_5), (Op_2,Op_1-Op_3), (Op_1-Op_3-Op_5,Op_4)

Sélection : (Op_2-Op_1-Op_3-Op_4,Op_5)

Croisement : (Op_2-Op_1-Op_3-Op_4,Op_5) $\Longrightarrow$ Op_2-Op_1-Op_3-Op_4-Op_5

tel que

Possibilités de croisement : (Op_2-Op_1-Op_3-Op_4, Op_5), (Op_1-Op_3-Op_5, Op_4), (Op_1-Op_3-Op_5, Op_5), (Op_1-Op_3, Op_5), (Op_1, Op_3), (Op_2, Op_1-Op_3), (Op_2, Op_3), (Op_2, Op_4), (Op_3, Op_5), (Op_4, Op_5).

Déjà sélectionnés : (Op_1 , Op_3), (Op_1-Op_3 ,Op_5), (Op_2,Op_1-Op_3), (Op_1-Op_3-Op_5,Op_4)

Sélection : (Op_2-Op_1-Op_3-Op_4,Op_5)

Croisement : (Op_2-Op_1-Op_3-Op_4,Op_5) $\Longrightarrow$ Op_2-Op_1-Op_3-Op_4-Op_5

tel que Op_2-Op_1-Op_3-Op_4-Op_5 :

Nom opération : Op_2-Op_1-Op_3-Op_4-Op_5

Entrées :

E11 : TypeE11

E21 : TypeE21

E12 : TypeE12

E22 : TypeE22

Sorties :

S11 : TypeS11

S13 : TypeS13

S15 : TypeS15

S25 : TypeS25

S24 : TypeS24

S42 : TypeS42

Fitness (Op_2-Op_1-Op_3-Op_4-Op_5) = 96

Remplacement : L'opération « Op 2-Op 1-Op 3-Op 4-Op 5 » remplace « Op 6 ».

Conclusion

La fonction d'adaptation joue le rôle le plus important dans notre algorithme, puisqu'elle détermine l'allure d'évolution de la population d'une génération à une autre et influe sur la solution finale et la similarité entre les opérations obtenues et la cible.

Nous allons tenter de montrer dans la prochaine section que le calcul de l'adaptation basé sur la description syntaxique des opérations peut donner de bons résultats.

CHAPITRE V

MISE EN OEUVRE ET TEST

Introduction

Le présent chapitre est consacré à la mise en oeuvre de notre approche génétique. Notre algorithme présenté au chapitre 4 nécessite une structure de données particulière que nous allons détailler plus loin. Aussi, nous avons besoin d'un programme qui permet de transformer les informations décrites dans un fichier de description syntaxique d'un service Web dans la forme requise par notre algorithme. D'où la nécessité d'un analyseur de fichiers WSDL.

Dans la première section nous présentons la plate-forme de développement de notre application (le langage de développement et l'outil de programmation) et les arguments ayant influencé notre choix.

La mise en oeuvre de notre approche génétique est décrite en quatre sous sections. Dans la première nous définissons le flux d'entrées/sorties avec lequel interagit notre application. Dans la deuxième partie, nous allons présenter la structure de données utilisée par l'implémentation des divers algorithmes. Par la suite nous détaillerons notre utilitaire d'analyse des fichiers de description WSDL qui prépare les données pour notre algorithme génétique. Cet algorithme est présenté dans une dernière sous-section où nous présentons le diagramme de classes ainsi que la mise en oeuvre des opérations génétiques telles que la sélection, le remplacement et le croisement.

La troisième section est une étude sur la performance de notre approche. Cette dernière section contient principalement les résultats de deux types de tests:

- Test pour la découverte d'une opération
- Test pour la création d'un service web composite.

5.1 Plate-forme de développement

Notre choix de plate-forme de développement a été déterminé par les trois critères suivants :

Coût minimal : Plate-forme libre ;

Efficacité : Offre les services nécessaires pour réaliser notre approche ;

Simplicité : Tient compte de notre expérience dans le domaine de développement d'applications.

Nous avons donc choisi la plate-forme Java plus spécifiquement :

- le langage de développement orienté objet Java ;
- la plate-forme Java, édition standard version 6 ;
- l'éditeur JDeveloper version 11.1.1.

5.2 Mise en oeuvre

5.2.1 Analyseur de fichiers WSDL

L'analyseur est le programme qui permet de préparer les entrées de l'algorithme génétique, sujet de notre mémoire. Il reçoit en entrée un fichier de description de service Web (WSDL) et donne comme sortie l'ensemble des opérations sous la forme présentée dans la section 5.2.2.

Dans notre analyseur, nous utilisons la classe `javax.wsdl` de la bibliothèque WSDL4J (*Web Services Description Language for Java*) qui permet de lire un fichier WSDL et de définir les différents champs qui le constituent. Le résultat de cette manipulation est un arbre conforme au format DOM (*Document Object Model*), ce qui nous permet ensuite de parcourir cet arbre et de retirer l'information dont nous avons besoin (figure 5.1).

WSDL4J est un projet à code source ouvert (*open source*) développé par IBM en langage Java. WSDL4J fournit une interface standard Java qui peut être utilisée pour analyser des documents existants en WSDL ou pour programmer la création de nouveaux documents WSDL.

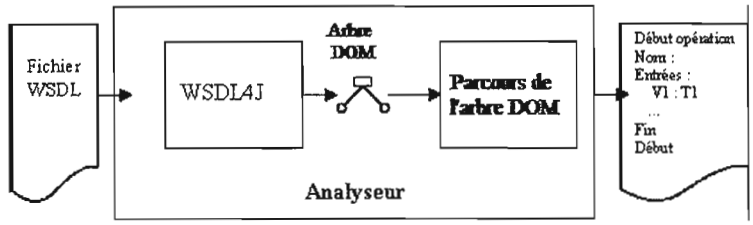

FIGURE 5.1 Fonctionnement de l'analyseur de fichiers WSDL.

```
Début de l'opération
  Nom opération : nom_Opération
        Entrées:
              nom_Entrée_1 : Type_Entrée_1
                    ...
              nom_Entrée_N : Type_Entrée_N
        Sorties:
              nom_Sortie_1 : Type_Entrée_1
                    ...
              nom_Sortie_M : Type_Sortie_M
Fin de l'opération
```

FIGURE 5.2 Représentation d'une opération.

5.2.2 Structures de données

Dans le contexte de l'algorithme génétique, on utilise les termes population, individu, chromosome et gène, tels qu'une population est un ensemble d'individus, un individu est une suite de chromosomes et un chromosome est une suite de gènes.

Notre algorithme génétique manipule des opérations sous la forme présentée à la figure 5.2. Dans cette section nous décrivons la structure de données que nous utilisons pour présenter les paramètres génétiques de notre application.

Gène : Une représentation d'une variable (d'entrée ou de sortie). Une variable a un nom et un type. La classe `Gene` présente la mise en oeuvre de ce paramètre.

Chromosome : Une liste de gènes. Dans la mise on oeuvre du chromosome on utilise la structure `List de Java` telle que :

```
List<Gene> chromosome = new ArrayList <Gene> ();
```

Individu : La structure `Individu` est la représentation d'une opération où on définit les données suivantes :

nom : Nom de l'opération ;

Chromosome d'entrées : Une liste définissant le message d'entrée de l'opération ;

Chromosome de sorties : Une liste définissant le message de sortie de l'opération ;

double val : Un double qui détermine la valeur d'adaptation de l'opération avec l'opération cible ;

Population : Une collection d'individus, appelée aussi génération.

5.2.3 Algorithme génétique

L'implémentation en Java de notre approche génétique nécessite la mise en oeuvre des classes présentées dans la figure 5.3 du diagramme de classes. Ces classes se divisent en trois catégories :

1. La classe principale qui décrit le processus de l'algorithme génétique ;
2. Les classes de mise en oeuvre des opérations génétiques ;
3. Les classes de mise en oeuvre de la structure de données.

Classe principale :

Cette classe est le code Java de notre algorithme génétique détaillé dans le chapitre 4. Il contient le constructeur et une méthode `Exécuter` qui reçoit en entrée le fichier `Opérations.txt` et envoie les sorties vers le fichier `Resultats.txt`. La figure 5.4 est un extrait du code Java de cette classe.

La trace d'exécution de cette classe est enregistrée dans le fichier `Trace.txt`. Le but de cette capture est d'informer l'utilisateur sur le déroulement du processus et de l'aider à choisir les services Web qui amènent aux résultats désirés.

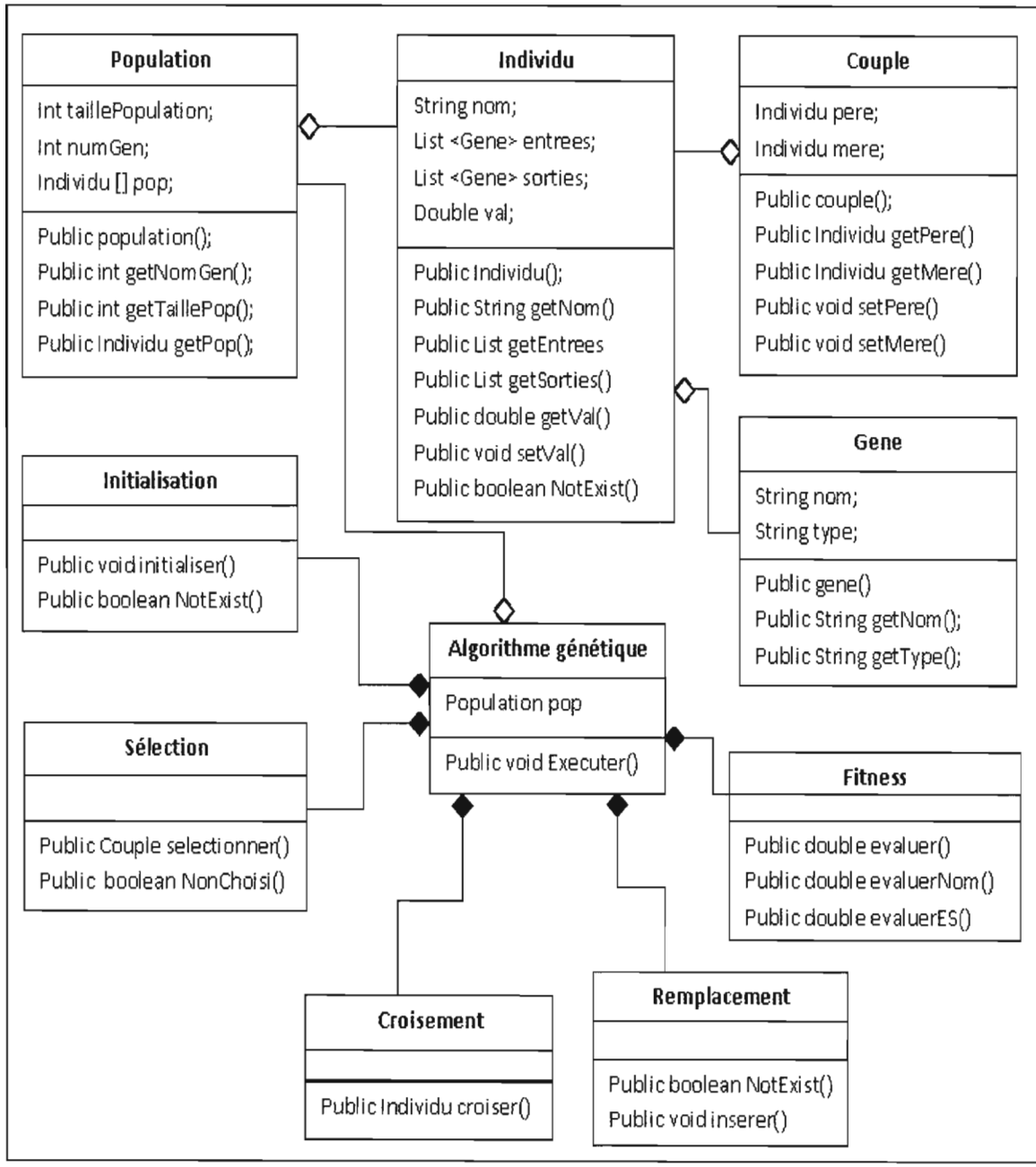

FIGURE 5.3 Diagramme de classe de notre application.

```
Init.Initialiser(Pop,FileIn,IndR,fichier);
Aff.AfficherPopulationF(fichier,Pop);
while ((Pop.GetNumGen()<nbMaxGeneration)&&(ExisteOperation)&&(OperationNonTrouve)){
    while ((C=selectionner.Selctioner(Pop,CoupleChoisi,NbCoupleChoisi))!=null){
        CoupleChoisi[NbCoupleChoisi]=C;
        NbCoupleChoisi++;
        Individu Enfant=new Individu("",null,null);
        Enfant=Croiser.Croiser(C.GetPere(),C.GetMere(),IndR,fichier);
        nbCroisement++;
        fitness.Evaluer(Enfant,IndR);
        if (Pop.GetPop()[taillePopulation-1].GetVal()<Enfant.GetVal()){
            if ((remplir.Inserer(Pop,Enfant,fichier))!= -1){
                if (Enfant.GetVal()>=96){
                    OperationNonTrouve=false;
                }
            }
        }
    }
    Individu IndA = new Individu("",null,null);
    IndA=Rech.RechIndividu(FileIn);
    nbOperation=nbOperation++;
    if (IndA!=null){
        fitness.Evaluer(IndA,IndR);
        remplir.Inserer(Pop,IndA,fichier);
    }
    else
        ExisteOperation=false;
    if (Pop.GetPop()[0].GetVal()>=96)
        OperationNonTrouve=false;
}
```

FIGURE 5.4 Extrait de code Java de l'algorithme génétique.

Classes des opérations génétiques :

Initialisation : Une classe qui permet d'initialiser la population. Elle cherche d'une manière aléatoire (utilisation de la fonction mathématique `random`) les opérations dans le fichier `Operations.txt`, crée les individus et les dépose dans la structure `population` ;

Sélection : Une classe qui permet de sélectionner un couple (`Pere` et `Mere`) pour l'opération de croisement. Cette classe utilise les informations suivantes :

Possibilité de croiser Une liste des couples d'individus qui peuvent être croisés ;

Déjà sélectionnés Une liste des couples d'individus qui ont été croisés dans des générations antérieures.

Fitness : Une classe qui permet de calculer la valeur d'adaptation d'un individu (opération) selon l'équation 4.3 (voir page 59) ;

Croisement : Une classe de mise en oeuvre de l'opération de croisement. Elle prend comme entrée un couple d'individus déterminé par l'opération de sélection (deux individus qui peuvent se croiser et qui ne sont pas choisis dans une génération antérieure) et retourne le nouveau né (résultat du croisement).

La figure 5.5 présente un extrait de code Java de la classe croisement qui montre comment obtenir les entrées de l'individu `enfant` à partir du couple `Pere` et `Mere`.

Remplacement : Une classe pour la mise en oeuvre de l'opération de remplacement, qui décide de la position pour insérer le nouvel individu après le retrait d'un ancien.

Le nouvel individu peut être le résultat de l'opération de croisement comme il peut être un individu externe à la population (exploration de l'espace de recherche).

Le remplacement d'un individu par un autre dépend de leurs valeurs d'adaptations retournées par la fonction *`Fitness`*. Si la valeur d'adaptation du nouvel individu est supérieure à la plus mauvaise de la population courante, on insère le nouveau à la bonne position (telle que la population demeure triée) et on retire le plus mauvais. Dans le cas contraire, en rejette le nouvel individu et on cherche un autre.

Classes de structure de donnée : La spécification des classes de mise en oeuvre de la structure de données ne contient que les données et les méthodes de manipulations (lecture et écriture).

Gene : une classe qui définit une variable d'entrée ou de sortie d'une opération.

```
public Individu Croiser(Individu pere, Individu mere, Individu indR,BufferedWriter fichier) throws IOException {
    List <Gene> entrees = new  ArrayList<Gene>();
    List <Gene> sorties = new  ArrayList<Gene>();
    List <Gene> entreesM = new  ArrayList<Gene>();
    List <Gene> sortiesP = new  ArrayList<Gene>();
    for (int x =0; x<Pere.getEntrees().size();x++){
        entrees.add(pere.getEntrees().get(x));
    }
    for (int y=0;y<mere.GetSorties().size();y++){
        sorties.add(mere.GetSorties().get(y));
    }
    for (int x =0; x<mere.getEntrees().size();x++){
        entreesM.add(mere.getEntrees().get(x));
    }
    for (int y=0;y<pere.GetSorties().size();y++){
        sortiesP.add(pere.GetSorties().get(y));
    }
    for (int t=0;t<sortiesP.size();t++){
        for (int k=0;k<entreesM.size();k++){
            if (entreesM.get(t).GetNom().equals(sortiesP.get(k).GetNom())&&
                entreesM.get(t).GetType().equals(sortiesP.get(k).GetType())){
                entreesM.remove(t);
                sortiesP.remove(k);
            }
        }
    }
    for (int i=0;i<entreesM.size();i++){
            entrees.add(entreesM.get(i));
    }
    for (int j = 0; j<sortiesP.size();j++){
            sorties.add(sortiesP.get(j));
    }
    Individu Enfant = new Individu(Pere.GetNom()+"-"+Mere.GetNom(),entrees,sorties);
    Fitness Fit = new Fitness();
    Fit.Evaluer(Enfant, IndR);
}
```

FIGURE 5.5 Extrait du code Java de la classe `croisement`.

Données :

`nom` : le nom de la variable que ce soit d'entrée ou de sortie de l'opération,

`type` : le type de variable,

Individu : une classe qui décrit les attributs d'une opération (figure 5.6).

Données :

`nom` : le nom de l'opération,

`chromosomeEntrees` : le message d'entrée,

`chromosomeSorties` : le message de sortie,

`val` : la valeur d'adaptation de l'opération,

Population une classe décrivant une population d'individus.

Données :

`individus` : tableau d'individus ;

`nBGeneration` : le numéro de génération en cours.

Couple : une classe qui représente un couple d'individus (père et mère) ; deux opérations qui peuvent se croiser ;

Données :

`pere` : Individu qui peut jouer le rôle du père dans un croisement,

`mere` : Individu qui peut jouer le rôle de la mère dans un croisement,

Remarques

L'aspect aléatoire s'applique à la recherche d'opérations. L'opération de recherche utilise la fonction mathématique `random` pour générer une valeur entre 0 et le nombre maximum d'opérations existant dans l'espace de recherche. Elle utilise cette valeur aléatoire pour determiner l'opération à retourner.

Dans la mise en oeuvre de notre approche on utilise des classes utilitaires qui ne sont pas décrites ici car elles ne sont pas spécifiques à notre approche et n'influent pas sur le résultat obtenu.

```
package Composition.Services.Web.Genetic.Codages;

import java.util.ArrayList;

public class Individu {
    private String nom;
    private List <Gene>  entrees = new ArrayList <Gene> ();
    private List <Gene> sorties = new ArrayList <Gene> ();
    private double val;
    public Individu(String nomS, List <Gene> entres, List <Gene> sortis) {
        nom=nomS;
        entrees=entres;
        sorties=sortis;
    }
    public String GetNom(){
        return nom;
    }
    public List <Gene>  getEntrees(){
        return entrees;
    }
    public List <Gene> GetSorties(){
        return sorties;
    }
    public double GetVal(){
        return val;
    }
    public void SetVal(double vall){
        val=vall;
    }
}
```

FIGURE 5.6 Extrait de code Java de la classe Individu.

5.3 Tests

Dans cette section nous présentons le flux de traitement ainsi que les différents tests que nous avons réalisés pour évaluer notre approche génétique. Le flux de traitement est une présentation de l'environnement de notre application c'est à dire les fichiers d'entrées et de sorties qui nous avons préparés pour réaliser les différents tests.

Nous classifions les tests en deux types :

- le premier consiste à évaluer la découverte d'une opération ; c'est à dire que l'opération cible est une opération existante de l'espace de recherche ;
- le deuxième consiste à évaluer la construction d'une opération composite, c'est à dire que l'opération cible peut être obtenue à partir des opérations existantes de notre espace de recherche par le biais de composition.

Signalons que, après plusieurs essais, nous avons choisi pour tous les tests les paramètres suivants pour notre algorithme génétique :

- nombre maximum de génération = 100 ;
- taille de population = 10 ;
- nombre d'opérations dans l'espace de recherche = 650.

5.3.1 Flux de traitement

Dans cette section, nous présentons le flux de traitement de notre application (figure 5.7). Au départ, nous avons besoin des fichiers d'entrées suivants :

Collection de fichiers WSDL : Un répertoire local qui contient l'ensemble des fichiers WSDL qui forme l'espace de recherche. Cette collection peut être construite manuellement (collecte des services Web) ou bien être le résultat d'une requête de recherche dans un annuaire UDDI (exemple : seekda) ;

Fichier WSDL cible : Description syntaxique du service Web cible.

Après l'analyse de cette collection, les sorties sont envoyées vers les fichiers suivants :

Collection d'opérations (Fichier Opérations.txt) : Un fichier qui contient toutes les opérations trouvées par l'analyseur dans la collection de fichiers WSDL ;

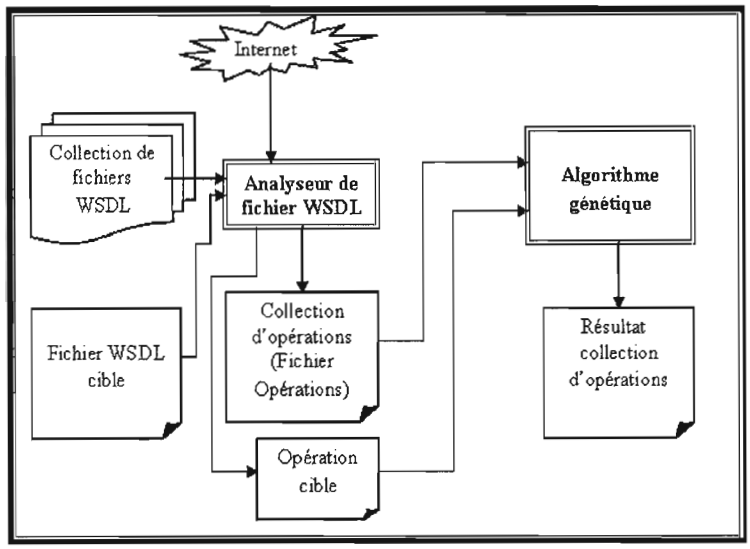

FIGURE 5.7 Flux de traitement de l'application.

Opération cible : Un fichier qui contient l'opération cible (voir section 5.2.2) ;

Les entrées de notre algorithme génétique sont prêtes à être utilisées. L'algorithme cherche les solutions pertinentes et les enregistre dans les fichiers suivants :

Résultat : Une collection d'opérations qui forment la génération retenue par l'algorithme génétique comme une solution de problème ;

Trace : Un fichier texte où on enregistre la trace de l'exécution de l'algorithme génétique contenant entre autres les informations suivantes : la population initiale, les croisements et les remplacements effectués tout au long de l'exécution de l'algorithme.

5.3.2 Évaluation pour la découverte d'opérations

Au cours de ces tests, on cherche à évaluer notre algorithme pour la découverte d'une opération, c'est à dire on demande à notre application de trouver une opération existante dans notre espace de recherche.

Nous réalisons ces tests pour savoir si notre approche est capable de trouver une opération parmi la collection définie, pour comparer notre algorithme à une approche séquentielle linéaire et

```
DEBUT d'OPERATION :
Nom Operation :  findAddress
    Entrees :
        token : string
        address : Address
        addressFinderOptions : AddressFinderOptions
    Sorties :
        Result : LocationInfo
FIN d'OPERATION :
```

FIGURE 5.8 L'opérations `getAdress`.

pour préciser les limites de notre application génétique.

Tests pour la découverte d'opérations

Nous faisons plusieurs essais pour évaluer notre approche dans le cas de recherche d'une opération existant dans notre espace de recherche. Ces essais permettent de vérifier que notre algorithme génétique peut donner une solution exacte si elle existe, et de donner une idée sur le déroulement de l'exécution.

Nous faisons plusieurs essais pour la recherche de l'opération `getAdresse` (figure 5.8). D'un essai à autre l'ordre des opérations dans le fichier `Operation.txt` est changé. À cet effet, nous avons développé un programme qui permet de mélanger d'une manière aléatoire les opérations.

Résultats des tests pour la découverte d'opérations

Le tableau 5.1 resume le contenu du fichier `resultats.txt` suite à l'exécution de notre algorithme pour la recherche de l'opération `getAddress`.

Le tableau 5.2 résume le déroulement de l'exécution de notre algorithme pour la recherche de l'opération définie précédemment. À partir de ceci on peut remarquer, au contraire de l'algorithme séquentiel (de fouille linéaire), que le temps d'exécution de notre algorithme génétique pour la

Opération	Message d'entrée	Message de sortie	Fitness
getAddress	point : Point token : string addressFinderOptions :AddressFinderOptions	Result : Address	100 %
pointQuery	queryOptions :QueryOptions token : string point : Point	Result : ResultSet	32 %
findNearest	proximityOptions : ProximityOptions token : string point : Point	Result : ResultSet	32 %
getCountries	datasource : string token : string	Result : ArrayOfstring	17.2 %
getMyMarkerNames	token : string iconDataSource : string	Result :ArrayOfstring	17.2 %
getThematicMap	thematicData : ArrayOfKeyValue mapExtent : Envelope mapImageOptions : MapImageOptions token : string thematicOptions : ThematicOptions	Result : MapImageInfo	17.2 %
envelopeQuery	envelope : Envelope queryOptions : QueryOptions token : string	Result : ResultSet	16.0 %
findRoute	routeFinderOptions : RouteFinderOptions routeStops : ArrayOfRouteStop token : string	Result : RouteInfo	16.0 %
findNearest	proximityOptions : ProximityOptions line : Geometry token : string	Result : ResultSet	16.0 %
findNearest	proximityOptions : ProximityOptions line : Geometry token : string	Result : ResultSet	16.0 %

TABLE 5.1 Résultat pour la découverte de l'opération `getAddress`.

	Algorithme génétique		Recherche linéaire	
Essai	Nb d'opérations examinées [1]	Temps d'exécution (en secondes)	Nb d'opérations examinées	Temps d'exécution (en secondes)
Essai 1	12	4	1	0
Essai 2	145	36	341	87
Essai 3	226	64	594	112
Essai 4	40	15	255	66
Essai 5	311	82	418	102
Moyenne	146.8	40.2	321.8	73.4

TABLE 5.2 Résultat pour la découverte de l'opération `getAdress`.

découverte d'une opération est indépendant de la position de celle-ci. Par contre il dépend du nombre d'opérations vérifiées.

5.3.3 Évaluation pour la composition d'opérations

Pour évaluer notre algorithme dans la composition d'opérations, nous allons faire deux tests :

1. Le premier test consiste à lancer notre application pour trouver une composition simple d'opérations (composition de deux opérations) ;
2. Le deuxième test consiste à évaluer notre algorithme dans une composition complexe (exemple traité a la section 4.5).

Évaluation pour une composition simple

Tests d'une composition simple :

On suppose que notre opération cible est `meteo` (figure 5.9). Une opération qui reçoit en entrée un `codezip` et affiche en sortie les informations météo de la région définie par ce code.

Pour confirmer que notre algorithme donne toujours la bonne solution et pour savoir les différentes façons dont l'algorithme procède nous faisons plusieurs essais de ce test.

Résultats du test d'une composition simple

1. Nombre d'opérations vérifiées avant de trouver l'opération cible.

```
DEBUT d'OPERATION :
Nom Operation :  meteo
        Entrees :
            zipCode : string
        Sorties :
            GetWeatherResult : string
FIN d'OPERATION
```

FIGURE 5.9 L'opération `meteo`.

Essai	Nombre d'opérations	Nombre de générations	Nombre de croisements	Temps d'exécution (en secondes)
Premier essai	609	13	101	122
Deuxième essai	389	13	50	76
Troisième essai	275	14	101	69

TABLE 5.3 Les trois essais de recherche de l'opération `meteo`.

La meilleure solution retenue par notre approche est l'opération composite `zipCodeToAddress - getWeather` (figure 5.10). Cette opération est une composition de deux opérations `zipCodeToAddress` et `getWeather` (figure 5.11).

Les trois essais que nous avons réalisés donnent toujours le bon résultat défini précédemment. Ils sont différents dans leurs déroulements en termes des opérations vérifiées ainsi que des croisements effectués et du temps d'exécution demandé (tableau 5.3).

Évaluation pour une composition complexe

Test d'une composition complexe

Notre dernier test consiste à évaluer l'approche dans le cas où l'opération cible peut être créée par une composition un peu plus complexe de plusieurs opérations. Parmi les opérations de notre espace de recherche on ne trouve pas un exemple pertinent qui nous permet d'atteindre notre objectif. Nous choisissons donc de réaliser l'exemple présenté à la section 4.5.

```
La solution retenue par l'algorithme est :
La taille de population est : 10
Le numéro de génération est : 16

L'individu numéro : 1
Le nom de l'opération est : (zipCodeToAddress-getWeather)
Le Fitness de l'opération est : 96.0
  Entrees:
      zipCode:  String
  Sorties:
      state:  String
      setWeatherResult :  String

L'individu numéro : 2
Le nom de l'opération est : getWeather
Le Fitness de l'opération est : 48.0
  Entrees:
      cityName:  String
      countryName:  String
  Sorties:
      getWeatherResult :  String

L'individu numéro : 3
Le nom de l'opération est : zipCodeToAddress
Le Fitness de l'opération est : 48.0
  Entrees:
      zipCode:  String
  Sorties:
      State: String
      cityName :  String
      countryName :  String

L'individu numéro: 4
Le nom de l'opération est: getThreeDigitZipCodesWithin
Le Fitness de l'opération est : 18.0
  Entrees:
      distance:  Float
      zip:  String
  Sorties:
      getThreeDigitZipCodesWithinResult :  ArrayOfZipCodeDistances

L'individu numéro : 5
Le nom de l'opération est : getDistanceBetweenZipCodes
Le Fitness de l'opération est : 18.0
  Entrees:
      zip1: String
      zip2: String
  Sorties:
      getDistanceBetweenZipCodesResult :  Float
```

FIGURE 5.10 Les cinq premières opérations du résultats de recherche de l'opération `meteo`.

```
Nom Operation :  GetWeather
        Entrees :
            CountryName : string
            CityName : string
        Sorties :
            GetWeatherResult : string
FIN d'OPERATION :

DEBUT d'OPERATION :
Nom Operation :  ZipCodeToAddress
        Entrees :
            zipCode : string
        Sorties :
            CountryName : string
            CityName : string
            State : string
FIN d'OPERATION :
```

FIGURE 5.11 Les deux opérations `zipCodeToAddress` et `getWeather`.

On ajoute à notre espace de recherche les opérations présentées à la figure 4.11 (page 61) et nous exécutons l'application pour la recherche de l'opération cible définie à la figure 4.12 (page 61).

Résultats du test d'une composition complexe

La figure 5.12 montre quatre opérations du fichier `resultats.txt`. Nous faisons plusieurs essais pour arriver à ce résultat, et ce :

- parce qu'il existe une infinité de compositions possibles ;
- parce que l'algorithme explore l'espace de recherche de manière aléatoire.

Conclusion

À travers les quelques tests que nous avons réalisés, nous avons obtenu des résultats qui semblent intéressants. Ceci semble donc confirmer que notre application est réalisable et peut résoudre certains problèmes de découverte et de composition des services Web. Mais nous remarquons certaines limites :

- dans le cas de recherche d'opération, les termes utilisés par l'opération cible doivent être identiques aux termes utilisés par l'opération existante ;

```
La solution retenue par l'algorithme est :
La taille de population est : 10
Le numéro de génération est : 10
L'individu numéro : 1
Le nom de l'opération est : ((Operation2-((Operation1-Operation3)-Operation5))-Operation4)
Le Fitness de l'opération est : 96.0
  Entrées:
      E22: TypeE22
      E12: TypeE12
      E21: TypeE21
      E11: TypeE11
  Sorties:
      S13: TypeS13
      S11: TypeS11
      S15: TypeS15
      S25: TypeS25
      S42: TypeS42
      S24: TypeS24
L'individu numéro : 2
Le nom de l'opération est : (Operation2-((Operation1-Operation3)-Operation5))
Le Fitness de l'opération est : 88.0
  Entrées:
      E11: TypeE11
      E21: TypeE21
      SE3514: TypeSE3514
      E12: TypeE12
      E22: TypeE22
  Sorties:
      SE3214: TypeSE3214
      S42: TypeS42
      S25: TypeS25
      S15: TypeS15
      S11: TypeS11
      S13: TypeS13
L'individu numéro : 3
Le nom de l'opération est : ((Operation2-(Operation1-Operation3))-((Operation1-Operation3)-
Operation5))
Le Fitness de l'opération est : 88.0
  Entrées :
      SE1223 : TypeSE1223
      E11 : TypeE11
      E21 : TypeE21
      SE3514 : TypeSE3514
      E22 : TypeE22
      E12 : TypeE12
  Sorties :
      S11 : TypeS11
      S13 : TypeS13
      SE2315 : TypeSE2315
      S42 : TypeS42
      SE3214 : TypeSE3214
      S25 : TypeS25
      S15 : TypeS15
L'individu numéro : 4
Le nom de l'opération est : (((Operation2-(Operation1-Operation3))-((Operation1-
Operation3)-Operation5))-Operation5)
Le Fitness de l'opération est : 88.0
  Entrées :
      SE2225 : TypeSE2225
      SE3514 : TypeSE3514
      E21 : TypeE21
      E11 : TypeE11
      E12 : TypeE12
      E22 : TypeE22
      SE3514 : TypeSE3514
      SE1223 : TypeSE1223
  Sorties :
      S13 : TypeS13
      S11 : TypeS11
      S15 : TypeS15
      S25 : TypeS25
      SE3214 : TypeSE3214
```

FIGURE 5.12 Extrait du fichier `resultats.txt` pour la recherche de l'opération `Operaton_cible`.

- dans le cas de composition, l'ordre d'apparition des opérations dans les générations est très important, si cet ordre est changé l'algorithme génétique peut retourner une solution différente de la meilleure solution possible.

Nous signalons qu'il faudrait aussi faire des tests et des expérimentations plus approfondis, mais que cela est difficile étant donné la difficulté de trouver un bon corpus (une bonne base de services web).

CONCLUSION

L'architecture orientée services, dont une implémentation possible est celle basée sur les services Web, amène au monde du développement de logiciels un nouveau concept. La production d'une application n'est plus seulement une question de développer du code mais plutôt un problème de déploiement des services existants.

Dans ce contexte, les organismes qui sont intéressés par cette nouvelle architecture tels que W3C, IBM, etc, développent des techniques et spécifient des normes et des protocoles (WSDL, SOAP, UDDI) qui permettent la mise en place de cette architecture. Le but principal de cette évolution architecturale et de ces technologies est d'implémenter un système de communication entre les applications au niveau d'un réseau étendu tel que le Web.

Après la spécification de besoins, le programmeur tente de trouver les services Web qui réalisent les différentes tâches définies. Généralement, on rencontre deux problèmes pour faire fonctionner d'une façon convenable cette architecture :

1. Les moteurs de recherches actuels de services Web ne donnent pas des solutions précises ;
2. Il est souvent nécessaire d'affiner les tâches pour trouver des services qui peuvent donner la solution désirée par le biais de la composition.

Dans ce travail, nous avons proposé une approche de découverte et de composition de services Web. Nous nous basons sur la description syntaxique de services (WSDL). Nous avons développé un algorithme génétique qui permet de trouver un service Web cible (s'il existe dans notre espace de recherche) ou proposer une composition, s'il y a lieu.

La fonction d'évaluation (*fitness*) permet de calculer une valeur de similarité entre le service Web cible et le service trouvé. Cette valeur a une influence primordiale sur la précision des résultats retournés. Dans notre application, on ne tient compte que de l'aspect syntaxique de la description du services Web ; on analyse les fichiers WSDL et on cherche la similarité entre les noms et les

types de variables par une simple comparaison de caractères (en ne tenant pas compte de l'aspect sémantique, par exemple, possible synonymie des mots).

Les tests que nous avons réalisés donnent des bons résultats en termes de solutions retenues et de temps d'exécution relativement aux approches séquentielles. Mais, on peut toujours faire des améliorations pour une utilisation plus générale. Dans le cas de découverte d'une opération, par exemple, l'utilisateur doit utiliser les mêmes termes que l'opération existante pour spécifier la cible pour que notre application la retrouve avec une similarité élevée.

Nous proposons comme travaux futurs d'intégrer l'aspect sémantique dans la fonction d'évaluation et d'améliorer la performance de l'algorithme génétique par le biais de parallélisme.

L'aspect sémantique permet de tenir compte d'une similarité réelle entre les entités (opération, nom, types). Deux noms d'opération, par exemple, sont similaires non seulement lorsque les deux chaînes de caractères qui se présentent sont égales mais lorsque les deux termes signifient la même chose (synonyme). Dans ce contexte, on pourrait travailler avec une fonction d'évaluation qui prendrait en compte la similarité sémantique entre les différents champs de l'opération.

Le parallélisme, que ce soit le parallélisme de données ou de tâches, permettrait aussi d'améliorer le temps d'exécution de notre algorithme parallèle, ce qui pourrait être nécessaire dans le cas où on aurait un espace de recherche important à explorer.

Bibliographie

(1) Rama Akkiraju, Joel Farrell, John Miller, Meenakshi Nagarajan, Marc-Thomas Schmidt, Amit Sheth, and Kunal Verma. Web service semantics - WSDL-S. Technical report, Large Scale Distributed Information Systems, 2005.

(2) Aniss Alkamari, Hafedh Mili, and Abdel Obaid. Signature-based composition of web services. *International MCETECH Conference on e-Technologies*, pages 104–115, 2008.

(3) Aniss Alkamri. Composition de services web par appariement de signatures. Maîtrise en informatique, Université du Québec à Montréal, 2008.

(4) Jean-Marc Alliot and Nicolas Durand. Algorithmes génétiques. Technical report, Laboratoire d'Optimisation Globale de ENAC, `http://www.recherche.enac.fr/opti/GA/FAG/ag.pdf`, Mars 2005.

(5) Guestavo Alonso, Fabio Casati, Harumi Kuno, and Vijay Machiraju. *Web Services : Concepts, Architectures and Applications.* Data-Centric Systems and Applications. Springer, 2004.

(6) Alexandre Alves, Assaf Arkin, Sid Askary, Charlton Barreto, Ben Bloch, Francisco Curbera, Mark Ford, Yaron Goland, Alejandro Guízar, Neelakantan Kartha, Canyang Kevin Liu, Rania Khalaf, Dieter König, Mike Marin, Vinkesh Mehta, Satish Thatte, Danny van der Rijn, Prasad Yendluri, and Alex Yiu. Web services business process execution language version 2.0. Technical report, OASIS, 2007.

(7) Souquet Amédée. Algorithmes génétiques. Master's thesis, Université de Nice, 2004.

(8) George A. Papadopoulos Andreas S. Andreou, Dimitrios G. Vogiatzis. Intelligent classification and retrieval of software components. *IEEE Computer Society Washington, DC, USA*, pages 37–40, Septembre 2006.

(9) Vidur Apparao, Steve Byrne, Mike Champion, Scott Isaacs, Ian Jacobs, Arnaud Le Hors, Gavin Nicol, Jonathan Robie, Robert Sutor, Chris Wilson, and Lauren Wood. Document object model (DOM) level 1 specification. Technical report, W3C, 1998.

(10) Nerea Arenaza. Composition semi-automatique de services web. Projet de mastèr, École polytechnique fédérale de Lausanne, 2006.

(11) Zeina Azmeh, Marianne Huchard, Chouki Tibermacine, Christelle Urtado, and Sylvain Vauttier. WSPAB : A tool for automatic classification and selection of web services using formal concept analysis. *Web Services, European Conference on*, 0 :31–40, 2008.

(12) Arindam Banerji, Claudio Bartolini, Dorothea Beringer, Venkatesh Chopella, Kannan Govindarajan, Alan Karp, Harumi Kuno, Mike Lemon, Gregory Pogossiants, Shamik Sharma, and Scott Williams. Web services conversation language (WSCL) 1.0. Technical report, W3C, 2002.

(13) Amel Benna, Nacer Boudjlida, and Hassina Talantikite. SAWSDL, mediation and XQuery fror web services discovery. In *NOTERE '08 : Proceedings of the 8th international conference on new technologies in distributed systems*, pages 1–10, New York, NY, USA, 2008. ACM.

(14) Djamel Amar Bensaber and Mimoun Malki. Development of semantic web services : model driven approach. In *NOTERE '08 : Proceedings of the 8th international conference on new technologies in distributed systems*, pages 1–11, 2008.

(15) Bianchini, Valeria De Antonellis, Barbara Pernici, and Pierluigi Plebani. Ontology-based methodology for e-service discovery. *Inf. Syst.*, 31(4) :361–380, 2006.

(16) David Booth, Hugo Haas, Francis McCabe, Eric Newcomer, Iona Michael, Chris Ferris, and David Orchard. Web services architecture. Technical report, W3C, `http://www.w3.org/TR/ws-arch/`, février 2004.

(17) David Booth and Canyang Kevin Liu. Web services description language (WSDL) version 2.0 part 0 : Primer. Technical report, W3C, `http://www.w3.org/TR/2007/REC-wsdl20-primer-20070626/`, june 2007.

(18) Antonio Brogi, Sara Corfini, and Razvan Popescu. Semantics-based composition-oriented discovery of web services. *ACM Trans. Internet Technol.*, 8(4) :1–39, 2008.

(19) Thomas Erl ; John Evdemon ; Diane Jordan ; Khanderao Kand ; Dieter König ; Simon Moser ; Ralph Stout ; Ron Ten-Hove ; Ivana Trickovic ; Danny van der Rijn et Alex Yiu Charlton Barreto ;, Vaughn Bullard ;. Web services business process execution language version 2.0 primer. Technical report, OASIS, 2007.

(20) Roberto Chinnici, Jean-Jacques Moreau, Sanjiva Weerawarana, and Arthur Ryman. Web services description language (WSDL) version 2.0 part 1 : Core language. Technical report, W3C, `http://www.w3.org/TR/2007/REC-wsdl20-20070626/`, june 2007.

(21) Pierre Châtel. Une architecture pour la découverte et l'orchestration de services web sémantiques. Technical report, Thales Communications France, 2007.

(22) Daniela Barreiro Claro. *SPOC - Un canevas pour la composition automatique de services web dédiés à la réalisation de devis.* Thèse de doctorat en informatique, Université d'Angers, 2006.

(23) Ricardo de la Rosa-Rosero. Découverte et sélection de services web pour une application Mélusine. Master mathématiques informatique, Institut d'Informatique et de Mathématiques Appliquées de Grenoble, 2004.

(24) S. Dehuri, S. Patnaik, A. Ghosh, and R. Mall. Application of elitist multi-objective genetic algorithm for classification rule generation. *Appl. Soft Comput.*, 8(1) :477–487, 2008.

(25) Stefan Dietze, Alessio Gugliotta, and John Domingue. Towards context-aware semantic web service discovery through conceptual situation spaces. In *CSSSIA '08 : Proceedings of the 2008 international workshop on Context enabled source and service selection, integration and adaptation*, pages 1–8, New York, NY, USA, 2008. ACM.

(26) Xin Dong, Alon Halevy, Jayant Madhavan, Ema Nemes, and Jun Zhang. Similarity search for web services. In *VLDB '04 : Proceedings of the Thirtieth international conference on Very large data bases*, pages 372–383. VLDB Endowment, 2004.

(27) Helga Duarte, Marie-Christine Fauvet, Marlon Dumas, and Boualem Benatallah. Vers un modèle de composition de services web avec propriétés transactionnelles. *Ingénierie des Systèmes d'Information*, 10(3) :9–28, 2005.

(28) Nicolas Durand. *Algorithmes Génétiques et autres méthodes d'optimisation appliqués à la gestion de trafic aérien.* PhD thesis, Thèse d'habilitation, 2004.

(29) Patrick Kellert et Farouk Toumani. Les web services sémantiques. *Information - Interaction - Intelligence (I3 Journal)*, pages 135–160, Janvier 2004.

(30) Thomas Vallee et Murat Yildizoglu. Présentation des algorithmes génétiques et de leurs applications en économie. *http://beagle.u-bordeaux4.fr/yildi/files/agpresf.pdf*, septembre 2001.

(31) F. Fallet-Kahn. Algorithmes génétiques. Technical report, Télécom ParisTech, `http://www.infres.enst.fr/~charon/MOD/rapports/AG.pdf`, Janvier 2005.

(32) Anne-Elisabeth Caillot Radouane Ben Tamrout Hafedh Mili, Guy Tremblay and Abdel Obaid. Web service composition as a function cover problem. In H. Mili and F. Khendek, editors, *MCeTech Montreal Conference on eTechnologies*, pages 61–71, Montréal, Canada, Jan. 2005.

(33) Stephan Hagemann, Carolin Letz, and Gottfried Vossen. Web service discovery : Reality check 2.0. *Next Generation Web Services Practices, International Conference on*, pages 113–118, 2007.

(34) Denis Huet. Application des algorithmes génétiques aux problèmes SAT et CSP. Technical report, `http://www.recherche.enac.fr/opti/papers/reports/techhuet94.pdf`, Juillet 1994.

(35) Philippe Laublet et Chantal Reynaud Jean Charlet. *Le Web sémantique.* Cépaduès-Èditions, Avril 2005.

(36) Eric Taillard Patrick Siarry Johann Dréo, Alain Petrowski. *Métaheuristiques pour l'optimisation difficile.* Eyrolles, Juillet 2003.

(37) Amrit Pratap et T Meyarivan Kalyanmoy Deb, Samir Agrawal. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization : NSGA-II. Technical report, Indian Institute of Technology Kanpur, 2000.

(38) Bangorn Klabbankoh and Ouen Pin-ngern. Applied genetic algorithms to information retrieval. *International Journal of the Computer, the Internet and Management (IJCIM)*, 7(3) :30–35, Septembre 1999.

(39) Michael Klein, Birgitta Konig-Ries, and Michael Mussig. What is needed for semantic service descriptions ? *International Journal on Web and Grid Services (IJWGS)*, pages 328–364, 2005.

(40) Jacek Kopecký and Elena Simperl. Semantic web service offer discovery for e-commerce. In *ICEC '08 : Proceedings of the 10th international conference on electronic commerce*, pages 1–6, New York, NY, USA, 2008. ACM.

(41) Yann Lefablec. Optimisation par algorithmes génétiques parallèles et multi-objectifs. Master's thesis, Rapport DEA IFP., 1995.

(42) Gisèle Legault. Un algorithme génétique pour la conception topologique de réseaux téléinformatiques à commutation de paquets. Mémoire de maîtrise en mathématique, Université du Québec à Montréal, Décembre 2004.

(43) Amelia A. Lewis, Jean-Jacques Moreau, Hugo Haas, David Orchard, Sanjiva Weerawarana, and Roberto Chinnici. Web services description language (WSDL) version 2.0 part 2 : Adjuncts. Technical report, W3C, `http://www.w3.org/TR/2007/REC-wsdl20-adjuncts-20070626/`, june 2007.

(44) David Martin, Mark Burstein, Jerry Hobbs, Drew McDermott, Sheila McIlraith, Srini Narayanan, Massimo Paolucci, Bijan Parsia, Terry Payne, Evren Sirin, Naveen Srinivasan, and Katia Sycara. OWL-S : Semantic markup for web services. Technical report, W3C, 2004.

(45) Benny Mathew et Poornachandra Sarang Matjaz B. Juric. *Business Process Execution Language for Web Services.* PACKT, second edition, Janvier 2006.

(46) Hacène Mechedou. Un environnement de composition de services web. Mémoire de maîtrise en informatique, Université du Québec à Montréal, Août 2007.

(47) Nilo Mitra and Ericsson Yves Lafon. SOAP version 1.2 part 0 : Primer (second edition). Technical report, W3C, `http://www.w3.org/TR/2007/REC-soap12-part0-20070427/`, Avril 2007.

(48) Michäel Mrissa. *Médiation Sémantique Orientée Contexte pour la Composition de Services Web.* PhD thesis, Université Claude Bernard Lyon I, November 2007.

(49) R. Nassrallah and N. Bouraqadi. Les services web : un condensé. Technical Report 2003-5-5, Ecole des Mines de Douai, May 2003. (In French).

(50) Joachim Peer. Semantic service markup with SESMA. In *Proceedings of Workshop Web Service Semantics : Towards Dynamic Business Integration, Chiba, Japan*, pages 100–116, 2005.

(51) Fabien Picarougne. *Recherche d'information sur Internet par algorithmes évolutionnaires.* Thèse de doctorat, Université François Rabelais Tours, Novembre 2004.

(52) Gerard Salton. *Automatic text processing : the transformation, analysis, and retrieval of information by computer.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.

(53) Biplav Srivastava and Jana Koehler. Web service composition — current solutions and open problems. In *ICAPS 2003 Workshop on Planning for Web Services*, pages 28–35, 2003.

(54) Johnny Stenback, Philippe Le Hégaret, and Arnaud Le Hors. Document Object Model (DOM) level 2 HTML specification. Technical report, W3C, 2003.

(55) Mohamed Anouar Taleb. Parallélisation d'un algorithme génétique pour le problème d'ordonnancement sur machine unique avec temps de réglages dépendants de la séquence. Maîtrise en informatique, Université du Québec à Chicoutimi, 2008.

(56) Lynda Tamine and Mohand Boughanem. Un algorithme génétique spécifique à une reformulation multi-requêtes dans un système de recherche d'information. *Sciences et Traitement de l'Information*, 1(1) :49–76, 2001.

(57) Sophie Voisin. Application des algorithmes génétiques à l'estimation de mouvement par modélisation markovienne. Technical report, Centre National de la Recherche Scientifique (CNRS), `http://imaging.utk.edu/people/svoisin/rapportDEA.pdf`, Juin 2004.

(58) D. Vrajitoru. *Algorithmes génétiques en recherche d'informations*, chapter Apprentissage : des principes naturels aux méthodes artificielles, pages 271–278. `http://www.cs.iusb.edu/~danav/papers/AidriDV.pdf`, 1998.