Convertibilité entre types d'entrée et de sortie pour la composition de services en bio-informatique

Mouhamadou Ba, Sébastien Ferré, Mireille Ducassé

Convertibilité entre types d'entrée et de sortie pour la composition de services en bio-informatique

Mouhamadou Ba[1] Sébastien Ferré[2] Mireille Ducassé[1]

[1] IRISA/INSA Rennes
[2] IRISA/Université Rennes 1
{mouhamadou.ba, sebastien.ferre, mireille.ducasse}@irisa.fr

Résumé

L'hétérogénéité des données et des formats de données en bio-informatique entraine souvent une incompatibilité des entrées et sorties des services, rendant difficile leur composition. Ce travail propose de réduire ces incompatibilités par la détection automatique de convertibilité d'une donnée de sortie vers une donnée d'entrée. Cette détection repose sur un langage de type, proche de XML Schéma, permettant d'abstraire les données tout en exploitant finement leur structure composite. Notre expérience sur des services et des types bio-informatiques, avec une implémentation de notre approche, montre que les convertibilités détectées sont nombreuses et pertinentes d'un point de vue biologique.

Mots Clef

workflow, services, composition, langage de type, convertibilité, bio-informatique

Abstract

The heterogeneity of data and data formats in bioinformatics often entails a mismatch between inputs and outputs of different services, making it difficult to compose services. This article proposes to reduce those mismatches by the automatic detection of the convertibility from output data to input data. This detection relies on a language of types, close to XML Schema, allowing to abstract data while precisely accounting for its composite structure. Our experiment on bioinformatic services and datatypes, performed with an implementation of our approach, shows that the detected convertibilities are numerous and relevant from a biological point of view.

Keywords

workflow, services, composition, language of types, convertibility, bioinformatics

1 Introduction

L'hétérogénéité des données et des formats de données en bio-informatique entraine souvent une incompatibilité des entrées et sorties des services, rendant difficile la composition de services [1]. Les formats pour représenter les données d'entrée et de sortie peuvent être textuels ou basés sur les technologies XML. Les formats textuels, souvent spécifiques à un ou quelques services, sont adaptés aux données qu'ils représentent et ils ont l'avantage d'être faciles à interpréter par un être humain [2]. Cependant, ils ne facilitent pas les traitements automatiques. Leur nature hétérogène exige analyse syntaxique et conversions spécifiques pour le transfert des données entre services. Plusieurs plateformes (ex., Emboss [3], Galaxy [4], Mobyle [5]), destinées à l'analyse de données, utilisent des formats de données textuels et doivent donc définir manuellement des services spéciaux pour la conversion entre formats. Ces derniers deviennent rapidement difficiles à maintenir lorsque le nombre de services, et donc le nombre de formats, croit. En effet, le nombre de conversions potentielles est quadratique avec le nombre de formats. Les formats basés sur les technologies XML décrivent les types de données de manière indépendante des outils, ce qui améliore l'interopérabilité entre les services [6, 7]. BioXSD [8], par exemple, propose une représentation standard pour l'échange de données entre services en bio-informatique. Il permet aussi d'ajouter des méta-informations provenant d'ontologies, ce qui augmente la précision des représentations. Les formats XML seuls ne suffisent toutefois pas à résoudre le problème d'appariement des entrées et sorties. D'une part, la plupart des formats restent textuels, il est donc important de pouvoir apparier des services utilisant des formats XML et textuels. D'autre part, quand bien même tous les formats seraient en XML et standardisés, il resterait le problème qu'un service peut ne consommer qu'une partie de ce que produit un autre service.

Les travaux relatifs à l'incompatibilité des données sont nombreux. Par exemple, Li et al. [9] classifient les incompatibilités dans le cadre de la composition de services en distinguant les incompatibilités fonctionnelles et non-fonctionnelles au niveau sémantique et syntaxique. Elizondo et al. [10], avec une catégorisation similaire, automatisent la détection des incompatibilités des données pour la composition. Ils proposent, en plus, d'identifier les convertisseurs existants capables de résoudre les incompatibilités détectées. Plus largement, Stroulia et al. [11] propose de calculer la similarité entre deux services web. Ils

se basent sur la structure des types de données, les opérations ainsi que sur la description textuelle des services. Ces travaux montrent les bénéfices de la prise en compte de la structure des données pour détecter et résoudre des incompatibilités. Cependant, ils se limitent à la notion de similarité entre types de donnée ou services, ce qui permet la découverte de convertisseurs et services existants, mais pas la génération de nouveaux convertisseurs. Certaines approches [12, 13] proposent d'exploiter des ontologies telles que EDAM [14] pour faciliter l'appariement de services en définissant des hiérarchies de formats et types de données. Une première limite de ces solutions est que les relations de convertibilité doivent être explicitées manuellement dans l'ontologie. Une deuxième limite est que la structure des données est peu ou pas prise en compte. L'approche de Dibernado et al. [12] offre davantage en permettant d'établir la convertibilité entre une donnée composite et une de ses composantes. Cependant, elle n'exploite pas toute la structure composite des données.

La contribution de cet article est de proposer une méthode d'appariement automatique des types d'entrée et sortie de services prenant en compte la structure composite des données. Nous définissons des règles de convertibilité des données en exploitant la (dé)composition ainsi que la spécialisation et la généralisation de types. Ces règles s'appuient sur une abstraction de la structure composite des données, qui est explicite pour du XML et implicite pour du textuel. Cette abstraction est exprimée dans un langage de types proche de XML Schéma. Nous rapportons une expérience sur des services et des types bio-informatiques, avec une implémentation de notre approche. Les résultats, analysés avec une équipe de la plateforme bio-informatique GenOuest [1], montrent que les convertibilités détectées sont nombreuses et pertinentes d'un point de vue biologique.

Dans la suite, la section 2 introduit la représentation abstraite des types. La section 3 définit les règles de convertabilité. La section 4 montre comment instancier notre méthode à la bio-informatique. La section 5 présente l'expérience sur les données bio-informatiques. La section 6 compare notre approche avec des travaux connexes.

2 Représentation des types

Dans cette partie nous présentons le langage de types utilisé pour décrire les données. Ce langage est défini à partir d'un ensemble ouvert de types primitifs et d'un jeu de constructeurs de types. Du point de vue sémantique, un type dénote un ensemble d'arbres XML. Notre langage de types reprend les principales constructions de XML Schéma, mais pour simplifier la présentation nous utilisons les expressions régulières inspirées des travaux de Hosoya et al [15]. Dans la suite, les types sont notés par des majuscules (ex., T, T_1), et la fonction $XML(T)$ définit la sémantique du type T par un ensemble de documents XML.

Les types primitifs : $T = p$, où p est un type primitif. Les types primitifs servent d'ingrédients de base pour fabriquer d'autres types. En termes de structure, les types primitifs sont atomiques ; ils ne sont pas décomposables. Les instances XML d'un type primitif sont des CDATA (textes). Par exemple, *int* est un type primitif représentant les entiers.

Le constructeur tag : $T = l[T_1]$ où l est un label. Cette expression de type dénote les documents XML dont la balise racine est l et dont le contenu est de type T_1 : $XML(l[T_1]) = \{$<*l*> x </*l*>$\mid x \in XML(T_1)\}$. Les labels apportent une information sémantique sur les données et peuvent être dérivés des concepts d'une ontologie. En XML Schéma, le label est exprimé par une balise ou par l'attribut *name* d'une balise *element*.

Le constructeur tuple : $T = T_1T_2$. Cette expression de type dénote les documents XML qui sont la concaténation d'une instance de T_1 et d'une instance de T_2 : $XML(T_1T_2) = \{x_1x_2 \mid x_1 \in XML(T_1), x_2 \in XML(T_2)\}$. Ce constructeur permet de former des données composites et des séquences. En XML Schéma, ce constructeur correspond aux balises *xs :complexType* et *xs :sequence*.

Le type vide : $T = \varepsilon$. Le type vide dénote la séquence vide (ou document vide) : $XML(\varepsilon) = \{\varepsilon\}$.

Le constructeur union : $T = T_1|T_2$. Cette expression de type dénote l'union des instances de T_1 et des instances de T_2 : $XML(T_1|T_2) = XML(T_1) \cup XML(T_2)$. Le constructeur *union* est adéquat lorsqu'on désire, par exemple, considérer plusieurs types différents et effectuer certains traitements sans distinction. En XML Schéma, ce constructeur correspond à la balise <*xs :choice*>.

Le constructeur liste : $T = T_1+$. Cette expression de type dénote les séquences non-vides d'instances de même type T_1 (listes homogènes) : $XML(T_1+) = \{x_1 \dots x_n \mid x_1, \dots, x_n \in T_1\}$. En XML Schéma, ce constructeur correspond aux attributs *minOccurs* et *maxOccurs* associés à la balise *sequence*.

Le constructeur optionnel : $T = T_1?$. Cette expression de type est équivalente à $T_1|\varepsilon$.

3 Règles de convertibilité

Nous définissons la convertibilité $T_1 \rightarrow T_2$ comme l'existence d'une fonction permettant de passer de T_1 à T_2. On dit dans ce cas que T_1 est convertible en T_2. Nous proposons des règles de convertibilité (voir figure 1) pour chacun des constructeurs de types introduits à la section précédente. Pour décider de la convertibilité entre deux types nous supposons connues les relations de convertibilité sur les labels, $l_1 \rightarrow_l l_2$, et sur les types primitifs, $p_1 \rightarrow_p p_2$. Celles-ci dépendent du domaine d'application et correspondent à des fonctions de conversion bien définies (ex., des flottants vers les entiers). Nous croisons systématiquement tous les constructeurs pour traiter tous les cas. Les

1. http ://www.genouest.org/

$$p_1 \to p_2 \qquad si\ (p_1 \to_p p_2) \qquad (1)$$
$$A \to \varepsilon \qquad (2)$$
$$l_1[A] \to l_2[B] \qquad si\ (l_1 \to_l l_2)\ et\ (A \to B) \qquad (3)$$
$$l[A] \to B \qquad si\ (A \to B) \qquad (4)$$
$$A \to (B_1 B_2) \qquad si\ (A \to B_1)\ et\ (A \to B_2) \qquad (5)$$
$$(A_1 A_2) \to B \qquad si\ (A_1 \to B)\ ou\ (A_2 \to B) \qquad (6)$$
$$A \to (B_1|B_2) \qquad si\ (A \to B_1)\ ou\ (A \to B_2) \qquad (7)$$
$$(A_1|A_2) \to B \qquad si\ (A_1 \to B)\ et\ (A_2 \to B) \qquad (8)$$

FIGURE 1 – Règles de convertibilité

règles concernant les constructeurs *liste* et *optionnel* se déduisent des autres, elles ne sont donc pas détaillées. La règle (1) permet de vérifier la convertibilité entre types primitifs. La règle (2) dit que tout type est convertible en le type vide. Cette règle permet la conversion de tout document XML en un document vide. Deux tags sont convertibles à condition que leurs labels soient convertibles et que les types qu'ils taguent soient eux aussi convertibles (règle (3)). Si cette règle n'est pas satisfaite, nous proposons d'ignorer le label à gauche. Cette situation revient à supprimer la balise racine dans le document à convertir (voir règle (4)). Ces quatre premières règles permettent de gérer le cas des tags et du type vide. Les quatre règles suivantes permettent la convertibilité de tout type par rapport aux types tuple (règles (5) et (6)) et union (règle (7) et (8)). On notera l'inversion des connecteurs logiques selon que le constructeur tuple/union est à gauche ou à droite.

Ces règles correspondent à des fonctions de conversion élémentaires qui peuvent être enchaînées arbitrairement. Il suffit qu'il existe un enchaînement de telles conversions d'un type A vers un type B pour établir la convertibilité de A vers B. Par exemple, pour vérifier la convertibilité $ABC \to C$ où A, B et C sont des expressions de type quelconques, il faut appliquer deux fois la règle (6) pour oublier le A et le B. Pour vérifier la convertibilité $ABC \to CA$ il faut d'abord appliquer la règle (5) pour décomposer la partie droite et se ramener aux vérifications de $ABC \to C$ et $ABC \to A$ de même nature que la vérification précédente. Dans ce cas, commencer par la règle (6) mène à un échec.

Nous avons implémenté notre système de règles dans un programme qui décide de la convertibilité entre deux expressions de type quelconques. Ce programme est directement dérivé des règles ci-dessus et combine *pattern matching* sur les expressions pour identifier les constructeurs et appels récursifs sur les sous-expressions de type. L'examen des règles montre que les appels récursifs impliquent des couples d'expressions toujours plus petits, ce qui assure la terminaison du programme dans tous les cas. Il reste à étendre notre implémentation à la génération de convertisseurs qui passera par leur spécification basée sur les règles proposées et le choix d'un langage pour représenter des convertisseurs exécutables (ex., XQuery).

4 Instanciation à la bio-informatique

Selon les besoins et la nature des données dans les plateformes bio-informatiques, plusieurs formats sont proposés. En génomique, divers formats textuels (ex., FastQ, BED) [2] et XML (ex., BioXSD [3], phyloXML [4]) existent. Les formats textuels sont les plus couramment utilisés. Outre les formats de données, des ontologies, telle que l'ontologie EDAM [5], sont proposées pour décrire, organiser et classer les ressources notamment les types de données et formats en bio-informatique. Notre travail exploite ces ressources pour définir les types d'entrée et sortie de services. Nous abstrayons les données en mettant l'accent sur leur structure composite et les informations qu'elles contiennent.

```
> Accession = accession[string]
> SimpleSequence = simpleSequence[string]
> NucleotideSequence = ns[SimpleSequence]
> AminoAcidSequence = as[SimpleSequence]
> Biosequence =
    NucleotideSequence | AminoAcidSequence
> ComplexBiosequence =
    complexBiosequence[
        sequence[Biosequence]
        species[string] source[string]
        name[string]
        version[string] note[string]?]
> ComplexProteinSequence =
    complexProteinSequence[
        sequence[AminoAcidSequence]
        species[string] source[string]
        name[string]
        version[string] note[string]?]
> ListOfComplexBiosequences =
    ComplexBiosequence+
```

FIGURE 2 – Exemples de types en bio-informatique

La figure 2 montre des exemples simples de types définis manuellement à partir de formats et types de données existants. $Accession$ et $SimpleSequence$ sont des types simples définis avec le constructeur *tag* et un primitif. Ils représentent, respectivement, des numéros d'accès et des séquences brutes. De la même manière, $NucleotideSequence$ (représentant les séquences nucléotides) et $AminoAcidSequence$ (représentant les sequence acides aminées) sont définis en utilisant $SimpleSequence$, ils spécialisent les séquences. Leur union forme $Biosequence$, une sequence biologique générale. $ComplexBiosequence$ et $ComplexProteinSequence$

2. http ://genome.ucsc.edu/FAQ/FAQformat.html/
3. http ://bioxsd.org/
4. http ://www.phyloxml.org/
5. http ://edamontology.org

sont des types composites contenant plusieurs types à l'aide du constructeur *tuple*. Ces types pour des séquences biologiques définissent des informations nécessaires (ex., $sequence[Biosequence]$) et optionnelles (ex., $note[string]$?), $ComplexProteinSequence$ étant plus spécifique que $ComplexBiosequence$. $ListOfComplexBiosequences$ définit une liste de $ComplexBiosequence$ à l'aide du constructeur *liste*. Tous les types que nous utilisons sont définis de la même manière que les types précédents. Les labels sont inspirés de l'ontologie EDAM.

Nous couvrons les types et formats de données textuels couramment utilisés dans les plateformes telles que EMBOSS [6]. Nous offrons une structure permettant de définir les données par composants en spécifiant la nature des informations de chaque composant et ses éventuelles relations avec d'autres composants. Par rapport aux formats XML tels que BioXSD, notre abstraction se place à un niveau plus élevé. Nous considérons seulement les aspects pertinents pour notre appariement entre entrées et sorties de services. Nous ignorons, par exemple, certains attributs et les restrictions sur les types primitifs.

L'abstraction de types est directe pour les formats XML grâce aux schémas. Pour les formats de données textuels, les spécifications souvent informelles doivent être étudiées pour dériver la représentation structurelle. L'expérience avec les types en génomique montre que des expressions de types reviennent fréquemment, elles peuvent être réutilisées après avoir été définies une fois. Le coût de description moyen d'un service diminue donc quand le nombre de services déjà décrits augmente. Les formats XML existants couvrent 2/3 des types de données courants utilisés comme entrées et sorties de services [8]. Des formats XML spécialisés, tels que phyloXML [7] pour les données phylogéniques et PDBML [16] pour les systèmes biologiques, existent pour certains sous-domaines bio-informatiques. Par ailleurs, des alternatives XML sont fournies pour certains formats textuels (ex., GFF [17]) et certaines plateformes définissent leurs propres formats XML (ex., Uniprot XML [18]). Le développement de telles solutions facilitera l'abstraction de types. Nous pouvons espérer, dans ce cas, des solutions automatiques et semi-automatique d'abstraction.

Les types définis représentent les entrées et sorties de services en génomique. Les tables 1 et 2 montrent des exemples de services et de types. Dans notre approche, ajouter un nouveau service opérationnel se fait en deux étapes. La première consiste à reconnaitre ou définir les types abstraits utilisés par le service. La deuxième étape consiste à implémenter, s'ils n'existent pas, les convertisseurs entre notre format XML et chaque format concret. Contrairement à d'autres approches, il n'est pas nécessaire de définir un convertisseur pour toutes les paires de formats existants. Cette étape est l'une des perspectives de ce travail.

6. http ://emboss.sourceforge.net/docs/Themes

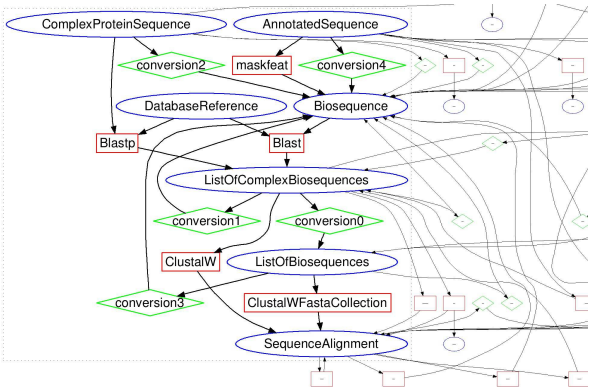

FIGURE 3 – Extrait du graphe de liaisons entre services

5 Expérience

Nous avons utilisé notre algorithme pour construire un graphe d'appariement de services bio-informatiques. Nous avons sélectionné 30 services provenant des plateformes EMBOSS [3], EBI (European Bioinformatics Institute) [19], BioMoby [20] et des services définis pour supporter le format BioXSD. Les services pris en compte couvrent la plupart des signatures de services. D'autres variantes et des catégories de services similaires sont proposées dans les plateformes mais les types des entrées et sorties ne changent pas en général, les ajouter apporterait peu à notre expérience.

A partir de notre échantillon de services nous avons généré automatiquement un graphe qui représente l'ensemble des liaisons qu'il est possible d'établir entre les services en utilisant notre programme d'appariement. La figure 3 montre un extrait du graphe obtenu. Le graphe complet est disponible en ligne, ainsi que le tableau complet des services [7]. Trois sortes de composants figurent dans le graphe, les services, les convertisseurs de types et les types qui définissent les entrées/sorties. Les services sont représentés par des rectangles, les convertisseurs par des losanges et les types d'entrée/sortie par des ellipses. Les services sont associés à leurs entrées et sorties par des flèches respectivement entrantes et sortantes. De la même manière chaque convertisseur est associé à un type de départ et un type d'arrivée, il matérialise la convertibilité d'un type en un autre par une conversion détectée automatiquement. Notre programme retrouve les liaisons directes entre des services capables de produire un type et ceux capables de le consommer. Dans notre graphe, ces liaisons sont créées lorsque les services utilisent la même représentation (le même type) pour définir les mêmes données. C'est le cas, par exemple, de la liaison entre le service *Blast* et le service *ClustalW*.

Des liaisons indirectes sont aussi détectées. Elles correspondent aux cas où il faut adapter une sortie d'un service pour qu'il puisse être accepté en entrée d'un autre service. Dans le graphe les liaisons indirectes sont matérialisées par les *conversions*. Avec *conversion4*, une séquence

7. http ://www.irisa.fr/LIS/Members/moba/graph/view

Service	Source	Entrées	Sorties	Tâche effectuée
Blast	BioXSD	BioXSD biosequence database URI	BioXSD biosequence	Recherche
Blastp	EMBL-EBI	Fasta sequence (typée) database URI	BlastResult	Recherche
ClustalWFastaCollection	BioMoby	Fasta files	MSF	Alignement
ClustalW	BioXSD	BioXSD biosequence (>=2)	BioXSD alignment	Alignement
maskfeat	EMBOSS	EMBL sequence	Fasta sequence	Manipulation et affichage

TABLE 1 – Extrait de la liste de tâches

Type d'origine	Type dans notre représentation	Représente des
Fasta sequence	ComplexProteinSequence | ComplexBiosequence	séquences typées ou non
BlastResult, Fasta files	listOfComplexBiosequences | listOfBiosequences	listes de séquences
EMBL sequence	AnnotatedSequence	sequences annotées
BioXSD biosequence	Biosequence | ComplexBiosequence	séquences simples ou complexes
BioXSD Alignment, MSF	SequenceAlignment	alignements de séquences
Database URI	DatabaseReference	référence à une base de données

TABLE 2 – Exemples de formats et types

simple est déduite d'une séquence annotée . Il décompose ainsi un type composite et sélectionne certaines de ses composantes pour alimenter des services. Avec *conversion0*, une liste de séquences simples est composée à partir d'une liste de séquences complexes. *Conversion2* permet de montrer la possibilité de spécialiser/généraliser un type. Notre programme détecte qu'une séquence de protéines est aussi une séquence biologique. Il considère également les éléments d'une liste individuellement. Avec *conversion3*, chaque élément d'une liste de séquences simples peut être sélectionné. Dans de nombreuses situations une combinaison de conversions est nécessaire pour passer d'un type à un autre. Par exemple, avec *conversion2*, la décomposition et la généralisation sont combinées pour passer d'une séquence complexe de protéines à une séquence biologique simple. *Conversion1* est un autre exemple de combinaison permettant de considérer chaque élément d'une liste d'un type composite.

Le graphe complet contient 264 liaisons entre services sur les 900 possibles avec 30 services (30x30). Notre programme trouve donc de nombreuses liaisons mais reste suffisamment spécifique pour être utile. Parmi ces liaisons, 88 sont directes, c'est-à-dire ne nécessitent pas de conversion. Notre relation de convertibilité permet donc de multiplier par 3 le nombre de liaisons entre services. Les 30 services utilisent 26 types différents, dont 10 sont en fait des types ad-hoc et non-décomposables (ex., images, rapports). Notre programme a identifié 27 conversions possibles entre les 16 types composites, sur les 256 possibles avec 16 types (16x16).

Nous avons présenté le graphe de l'expérience à des développeurs et utilisateurs de la plateforme bio-informatique GenOuest. Ils ont souligné que "il est remarquable que le rôle central de l'alignement de sequences soit si visible sur le graphe". Ils ont affirmé que "le graphe produit a un intérêt pédagogique". En effet, dans un document type [8], un graphe produit à la main à partir d'une libraire de services bio-informatiques contient des services et des liaisons similaires à ceux de notre graphe. Cependant, étant plus complète dans la modélisation des données d'entrée et sortie, notre approche offre plus de flexibilité. Par conséquent, elle fournit des connexions et une différentiation plus explicites entre les catégories de services. Elle révèle aussi des conversions possibles entre les données d'entrée et sortie, ce qui crée de nouvelles connexions. Par ailleurs, notre graphe est destiné au traitement automatique, il montre la preuve des connexions entre services et peut facilement prendre en compte les changements au niveau des types de données et des services. Avec un ensemble grandissant d'environ 1500 outils actuellement disponibles, il est peu probable que l'on puisse produire le graphe à la main.
Notre objectif principal est de guider le biologiste à composer des workflows. Durant la construction d'un workflow, les entrées et sorties jouent un rôle central dans la sélection des services en posant des contraintes sur les services applicables, cependant ils ne sont pas les seuls critères de sélection pour le biologiste. Il est aussi nécessaire de représenter les services par leurs propriétés fonctionnelles et non-fonctionnelles (ex., tâche bio-informatique, qualité des résultats, provenance, efficacité, popularité). Les utilisateurs de GenOuest mentionnent cependant, entre autres possibilités offertes, que le graphe peut servir de support utile à un utilisateur avec une connaissance du domaine pour sélectionner des services destinés à un workflow. Ainsi, ils valident le fait que le graphe généré par notre approche détecte des informations pertinentes et fournit des connaissances, généralement produites à la main, de manière systématique. Ils soulignent des perspectives intéressantes. L'étape couteuse de notre approche est la

8. Présentation de services du package Wisconsin- Olivier Collin - CNRS Roscoff - Formation Génopole Ouest - november 2002

production des types abstraits actuellement effectuée à la main. Nos abstractions des données pourraient être enrichies par l'utilisation d'ontologies, en particulier EDAM, ce qui permettrait de faciliter l'étape d'abstraction et d'intégrer des facettes métier. Il est également question de modéliser d'autres données en génomique (ex., arbres phylogéniques) et dans d'autres domaines (ex., métabolisme).

6 Travaux connexes

Le développement de solutions automatiques pour la composition de services est une réponse aux méthodes manuelles, souvent longues et difficiles à mettre en place, utilisées actuellement pour la sélection et la l'appariement de services durant la composition. Gérer l'appariement de services requière l'identification des catégories d'incompatibilité entre services. Le travail de Li et al. [9] fournit une classification multi-dimensionnelles des incompatibilités. Il identifie les incompatibilités sémantiques et syntaxiques sur les propriétés fonctionnelles et non-fonctionnelles, l'incompatibilité des propriétés fonctionnelles incluant l'incompatibilité au niveau des signatures de services. Cette classification systématique décompose le problème d'incompatibilité et aide à trouver des solutions adaptées a chaque situation. Basés sur des classifications similaires, Velasco-Elizondo et al. [10] résolvent des incompatibilités de données détectées et proposent d'identifier des convertisseurs existants adaptés aux attentes des utilisateurs. Ils utilisent les formats et la structure mais ne permettent pas de considérer, par exemple, une partie des données pour déterminer un convertisseur. Stroulia et al. [11] proposent d'établir la similarité entre spécifications WSDL (Web Service Description Language) à travers la structure des types de données, des opérations et des descriptions textuelles. Cette approche exploite la structure composite des données mais est destinée à la découverte de services, elle n'est pas destinée à la conversion et à l'appariement entre entrées et sorties de services. Les travaux précédents prennent en compte la nature syntaxique (structure) des données. Cependant, ils se limitent au calcul de similarité entre types ou services en faisant l'agrégation de similarités à travers la structure jusqu'au niveau élémentaire. Ils ne donnent pas, par exemple, une explication sur le passage d'un type à un autre. Notre objectif est de générer une spécification formelle du passage d'un type à un autre, ce qui permettra de générer automatiquement des convertisseurs de données entre services.

De nombreux travaux de recherche utilisent la description sémantique des ressources et proposent des méthodes plus ou moins automatiques de gestion des entrées/sorties lors de la composition de services. Nous allons dans le même sens que le travail proposé par Dibernado et al. [12] pour la gestion des types d'entrée et sortie des services. Elle prend en compte les types composites en se basant sur les relations de généralisation et de composition des ontologies. Cette approche d'appariement n'exploite cependant pas toute la structure composite des types, elle n'utilise que la relation *hasA* des ontologies. Elle n'explicite pas la prise en compte du tag, de l'union et de l'optionnel. Elle permet l'extraction d'un élément pour le type à gauche mais ne permet pas de décomposer le type à droite. Par contre, étant basée sur les ressources ontologiques, elle peut tirer profit de nouvelles relations. Une de nos perspectives est d'utiliser les ontologies pour compléter la description des types au delà de notre description structurelle.

Lebreton et al. [13] proposent une approche de vérification de la compatibilité sémantique des paramètres de services. Ils se basent sur la description sémantique des ressources pour l'appariement entre paramètres de services et confirment les avantages d'utiliser ces technologies sémantiques déjà démontrés dans des travaux antérieurs [21, 22]. Ce travail gère les cas d'appariement par généralisation et par spécialisation mais ne sait pas (dé)composer des types. Galaxy [4] utilise une bibliothèque de convertisseurs pour gérer des liaisons entre des services. L'appariement entre paramètres de services se base sur les formats des entrées et sorties. Les services prennent en compte plusieurs formats. Lorsqu'un format n'est pas prévu, un convertisseur de format défini à la main est utilisé. Dans cette approche il y a une forte dépendance entre services et formats, ce qui alourdit l'évolution des outils. De plus, les formats utilisés sont textuels, ce qui, comme discuté précédemment, ne favorise pas l'automatisation de la conversion entre formats. L'idée de séparer les types ou formats de données des services qui les utilisent est l'objet de beaucoup de travaux, voir, par exemple, Kalas et al. [8]. Dans le but de faciliter l'interopérabilité des outils, des formats communs basés sur XML sont proposés pour représenter les données bio-informatiques. Pour l'instant, peu d'implémentations de services utilisent ces technologies. La généralisation de leur utilisation renforcerait notre approche, car elle faciliterait la spécification des types abstraits.

7 Conclusion

Dans cet article nous avons proposé d'apparier les types d'entrée et sortie des tâches en exploitant la structure composite des types et en considérant la convertibilité des données. A travers une représentation de types de données, nous avons proposé des règles de convertibilité entre types. Nous avons ensuite montré les cas d'appariement qu'il est possible d'établir automatiquement avec des types et services bio-informatiques en utilisant notre représentation et les règles de convertibilité. Ces appariements ont été validés par une équipe de bio-informaticiens.

Remerciements. Nous remercions Olivier Collin, Olivier Dameron, François Moreews et Olivier Sallou pour leur expertise dans le domaine des services et workflows bio-informatiques ainsi que pour d'enrichissantes discussions.

Références

[1] T. Oinn, M. Greenwood, M. Addis, J. Ferris, K. Glover, C. Goble, D. Hull, D. Marvin, P. Li, and P. Lord,

"Taverna : Lessons in creating a workflow environment for the life sciences," *Concurrency and Computation : Practice and Experience*, vol. 18, no. 10, pp. 1067–1100, 2006.

[2] S. Gundersen, M. Kalas, O. Abul, A. Frigessi, E. Hovig, and G. K. Sandve, "Identifying elemental genomic track types and representing them uniformly," *BMC Bioinformatics*, vol. 12, p. 494, 2011.

[3] P. Rice, I. Longden, and A. Bleasby, "Emboss : the european molecular biology open software suite," *Trends in genetics*, vol. 16, no. 6, pp. 276–277, 2000.

[4] J. Goecks, A. Nekrutenko, J. Taylor, and T. G. Team, "Galaxy : a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences," *Genome Biology*, vol. 11, no. 8, p. R86, 2010.

[5] H. Ménager, V. Gopalan, B. Néron, S. Larroudé, J. Maupetit, A. Saladin, P. Tufféry, Y. Huyen, and B. Caudron, "Bioinformatics applications discovery and composition with the mobyle suite and mobylenet," in *RED*, pp. 11–22, 2010.

[6] P. N. Seibel, J. Krüger, S. Hartmeier, K. Schwarzer, K. Löwenthal, H. Mersch, T. Dandekar, and R. Giegerich, "Xml schemas for common bioinformatic data types and their application in workflow systems," *BMC Bioinformatics*, vol. 7, p. 490, 2006.

[7] M. V. Han and C. M. Zmasek, "phyloxml : Xml for evolutionary biology and comparative genomics," *BMC Bioinformatics*, vol. 10, p. 356, 2009.

[8] M. Kalas, P. Puntervoll, A. Joseph, E. Bartaseviciute, A. Töpfer, P. Venkataraman, S. Pettifer, J. C. Bryne, J. C. Ison, C. Blanchet, K. Rapacki, and I. Jonassen, "Bioxsd : the common data-exchange format for everyday bioinformatics web services," *Bioinformatics*, vol. 26, no. 18, 2010.

[9] X. Li, Y. Fan, and F. Jiang, "A classification of service composition mismatches to support service mediation," in *GCC*, pp. 315–321, 2007.

[10] P. V. Elizondo, V. Dwivedi, D. Garlan, B. R. Schmerl, and J. M. Fernandes, "Resolving data mismatches in end-user compositions," in *IS-EUD*, pp. 120–136, 2013.

[11] E. Stroulia and Y. Wang, "Structural and semantic matching for assessing web-service similarity," *Int. J. Cooperative Inf. Syst.*, vol. 14, no. 4, pp. 407–438, 2005.

[12] M. DiBernardo, R. Pottinger, and M. Wilkinson, "Semi-automatic web service composition for the life sciences using the biomoby semantic web framework," *Journal of Biomedical Informatics*, vol. 41, no. 5, pp. 837–847, 2008.

[13] N. Lebreton, C. Blanchet, D. B. Claro, J. Chabalier, A. Burgun, and O. Dameron, "Verification of parameters semantic compatibility for semi-automatic web service composition : a generic case study.," in *Int. Conf. on Information Integration and Web Based Applications and Services* (D. Taniar, E. Pardede, H.-Q. Nguyen, J. W. Rahayu, and I. Khalil, eds.), pp. 845–848, ACM, 2010.

[14] J. C. Ison, M. Kalas, I. Jonassen, D. M. Bolser, M. Uludag, H. McWilliam, J. Malone, R. Lopez, S. Pettifer, and P. M. Rice, "Edam : an ontology of bioinformatics operations, types of data and identifiers, topics and formats," *Bioinformatics*, vol. 29, no. 10, pp. 1325–1332, 2013.

[15] H. Hosoya, J. Vouillon, and B. C. Pierce, "Regular expression types for xml," in *ICFP*, pp. 11–22, 2000.

[16] J. D. Westbrook, N. Ito, H. Nakamura, K. Henrick, and H. M. Berman, "Pdbml : the representation of archival macromolecular structure data in xml," *Bioinformatics*, vol. 21, no. 7, pp. 988–992, 2005.

[17] R. D. Dowell, R. M. Jokerst, A. Day, S. R. Eddy, and L. Stein, "The distributed annotation system," *BMC Bioinformatics*, vol. 2, p. 7, 2001.

[18] "The universal protein resource (uniprot) in 2010," *Nucleic Acids Research*, vol. 38, no. Database-Issue, pp. 142–148, 2010.

[19] H. McWilliam, F. Valentin, M. Goujon, W. Li, M. Narayanasamy, J. Martin, T. Miyar, and R. Lopez, "Web services at the european bioinformatics institute-2009," *Nucleic Acids Research*, vol. 37, no. Web-Server-Issue, pp. 6–10, 2009.

[20] M. D. Wilkinson and M. Links, "Biomoby : An open source biological web services proposal.," *Briefings in Bioinformatics*, vol. 3, no. 4, pp. 331–341, 2002.

[21] E. Sirin, J. Hendler, and B. Parsia, "Semi-automatic composition of web services using semantic descriptions," in *Web services : modeling, architecture and infrastructure workshop in ICEIS*, vol. 2003, Citeseer, 2003.

[22] J. Ríos, T. J. M. Karlsson, and O. Trelles, "Magallanes : a web services discovery and automatic workflow composition tool," *BMC Bioinformatics*, vol. 10, p. 334, 2009.