HAL
open science

Composition guidée de services : application aux workflows d'analyse de données en bio-informatique

Mouhamadou Ba

To cite this version:

Mouhamadou Ba. Composition guidée de services : application aux workflows d'analyse de données en bio-informatique. Bio-informatique [q-bio.QM]. INSA de Rennes, 2015. Français. NNT : 2015ISAR0024 . tel-01301681

HAL Id: tel-01301681

https://theses.hal.science/tel-01301681

Submitted on 12 Apr 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Thèse

THESE INSA Rennes
sous le sceau de l'Université européenne de Bretagne
pour obtenir le titre de
DOCTEUR DE L'INSA DE RENNES
Spécialité : Informatique

présentée par
Mouhamadou BA
ECOLE DOCTORALE : *MATISSE*
LABORATOIRE : *IRISA*

Composition guidée de services – Application aux workflows d'analyse de données en bio-informatique

Thèse soutenue le 04.12.2015
devant le jury composé de :

Dominique Lavenier
Directeur de recherche CNRS, IRISA / Président du Jury
Mohand-Said Hacid
Professeur, Université Claude Bernard Lyon 1 / Rapporteur
Sarah Cohen Boulakia
Maître de conférences HDR, Université Paris Sud / Rapporteur
Moussa Lo
Professeur, Université Gaston Berger / Examinateur
Sébastien Ferré
Maître de conférences HDR, Université Rennes 1 / Co-directeur de thèse
Mireille Ducassé
Professeur, INSA Rennes / Directrice de thèse

Composition guidée de services – Application aux workflows d'analyse de données en bio-informatique

Mouhamadou Ba

En partenariat avec

Remerciements

Tout d'abord, je tiens à remercier Mireille Ducassé et Sébastien Ferré qui ont bien voulu encadrer cette thèse. Grâce à eux, la thèse s'est déroulée dans de très bonnes conditions. Je remercie l'ensemble des membres de l'équipe LIS de l'IRISA/INSA Rennes où s'est déroulée la thèse. Je remercie également l'équipe de la plateforme bio-informatique GenOuest ainsi que l'équipe Bibliome.

Je remercie Sarah Cohen Boulakia et Mohand-Said Hacid qui ont accepté de rapporter cette thèse ainsi que les examinateurs Moussa Lo et Dominique Lavenier.

Je remercie les amis et les collègues ainsi que les membres de DEFARSI, une mention spéciale à Soda M. Cissé, Bamba Ndiaye, Pape Fam, Abdou K. Fall, Ahmet Wade et Mamadou Diop ainsi qu'à Malaw.

Je remercie enfin mes parents, Madame Ba Adama ainsi que toute la famille Ba et la famille Séne.

Table des matières

Remerciements 1

1 Introduction générale 7
1.1 Workflows en bio-informatique 9
1.2 Thèse : composition guidée de workflows 10
1.3 Organisation du document 12

2 État de l'art 13
2.1 Services 14
2.1.1 Services web SOAP et REST 14
2.1.2 Services web sémantiques 16
2.1.3 Ontologies de domaine 18
2.1.4 Formats de données 20
2.1.5 Registres de services 22
2.2 Composition de workflows 26
2.2.1 Médiation de services 28
2.2.2 Techniques de composition de workflows 33
2.3 Conclusion 40

3 Outils et Méthodes 43
3.1 Typage 43
3.1.1 Système de types 44
3.1.2 XML et XML-Schema 46
3.2 Les systèmes d'information logiques (LIS) 53
3.2.1 Le framework ACN 53
3.2.2 Les instances LIS 54

3.3 Conclusion 56

4 Convertibility of Service Inputs and Outputs 57
4.1 Introduction 60
4.2 Abstract types 60
4.3 Convertibility Rules 62
4.3.1 Rule System 63
4.3.2 System Properties 68
4.4 Implementation 72
4.5 Instantiation and Use Case 74
4.5.1 Instantiation to Bioinformatics 74
4.5.2 Use Case : Resolving Data Mismatches in a Bioinformatics Workflow 76
4.6 Experiments 80
4.6.1 Convertibility between Bioinformatics Services 80
4.6.2 Survey with Domain Experts 84
4.7 Related Work 87
4.8 Conclusion 89

5 Guided Composition of Workflows 91
5.1 Introduction 93
5.2 Principles of our Approach 93
5.3 Formal Models 96
5.3.1 Knowledge Base 96
5.3.2 Workflow 96
5.3.3 Workflow Properties 99
5.3.4 Focus and Suggestions 100
5.3.5 Transformations 101
5.4 Properties of our Approach 103
5.5 Implementation 107
5.6 Use Case 111
5.7 Related Work 113
5.8 Conclusion 115

6 Conclusion générale 117
6.1 Conclusion 117
6.2 Perspectives 118
6.2.1 Non-déterminisme 118
6.2.2 Abstraction de types 119
6.2.3 Expressivité des workflows 120
6.2.4 Intégration d'ontologies 121

6.2.5 Workflow mining 122
6.2.6 Suggestion 122

Bibliographie 124

Table des figures 145

Index 147

Chapitre 1

Introduction générale

Ces dernières années, le développement des Technologies de l'Information et de la Communication (TICs) a impliqué l'existence de moyens efficaces d'acquisition et de traitement de l'information. Grâce à cela, le concept de *e-science* qui permet la science à travers les TICs s'est développé. La *e-science* repose sur des infrastructures importantes comprenant des machines, des réseaux et des applications pour apporter des améliorations considérables dans la production et l'exploitation des données dans le cadre d'activités scientifiques. Les améliorations se font sentir sur des aspects tels que la conduite d'expériences, le partage de ressources et la ré-utilisation de résultats [LBRM+13]. Par exemple, l'acquisition de données génomiques par des technologies efficaces de séquençage de l'ADN permet des études plus rapides sur les espèces, en utilisant des systèmes informatiques dédiés, à travers des réseaux de chercheurs.

Les systèmes informatiques existants dans le cadre des activités scientifiques en *e-science*, pour être plus efficaces dans l'intégration et l'interopérabilité, utilisent des solutions standardisées rendant accessibles des données et des outils de calcul. Parmi les solutions standardisées, il y a les services web qui sont des applications informatiques autonomes accessibles à travers Internet. Les services sont au coeur du paradigme de programmation orientée service (SOC) qui est une évolution des techniques de programmation classiques [Pap03]. Les services sont proposés pour découpler les fonctions réalisées des technologies employées et permettre à des systèmes différents de communiquer et de partager des informations à distance.

De nombreux organismes, localisés à travers le monde, ont mis à disposition des outils d'accès et de traitement de données sous forme de services. Ces services encapsulent diverses fonctions, par exemple l'extraction d'information à partir de bases de données, la comparaison de séquences biologiques ou la fouille de textes. L'existence de ces services a ouvert la voie à des solutions d'analyse de données utilisant des chaines de traitements, les *workflows* d'analyse de données [TDGS14].

Un workflow [1] est défini comme un flux de travail. Il modélise et représente un ensemble de tâches à accomplir par des entités pouvant être des personnes et/ou des machines dans le but de résoudre un problème global. Dans notre contexte, un workflow est une composition de tâches, chaque tâche désignant un service spécifique à exécuter. Les workflows permettent de représenter des expériences scientifiques *in silico*. Ils permettent d'exploiter des modèles et des données, en liaison avec des expériences *in situ*, à travers des enchainements de traitements informatiques. Cela peut être par exemple pour étudier des tremblements de terre en géophysique [MDZ+07], pour étudier des ondes gravitationnelles en astronomie [BBD+07] ou pour analyser de l'ADN en biologie [GWS+02].

Cependant, pour utiliser les workflows, il faut les créer. Et, la création de workflows peut être très complexe. En effet, créer un workflow c'est écrire un programme en utilisant les services comme des fonctions. Cela implique des difficultés liées à l'expression des besoins en termes de liaisons et d'enchainements cohérents entre services. Cela implique également des difficultés liées à la nature souvent hétérogène, distribuée des ressources [GDE+07, VIK+10]. Le nombre de services et de données ne cessent d'augmenter. Ils sont proposés sous diverses formes et concernent divers champs et domaines d'utilisation. Ainsi, bien que les services facilitent l'intégration et l'interopérabilité, des efforts supplémentaires sont nécessaires pour bien exploiter les workflows de services. Surtout, dans un contexte où les workflows sont créés et gérés par des scientifiques qui ne sont pas forcément des experts des technologies de workflow mais des utilisateurs finaux qui souhaitent faire leur science en créant facilement des workflows utilisables et modifiables en fonction de leurs hypothèses et modèles [BG07].

L'objectif de cette thèse est de proposer une solution pour faciliter la création de workflows de services, en prenant en compte l'environnement actuel des données et le contexte des domaines scientifiques. Dans la suite de ce document, nous traiterons de la création de workflows (composition de workflows), particulièrement en bio-informatique.

1. http ://www.wfmc.org/

1.1 Workflows en bio-informatique

La *e-science* est en plein essor en biologie. La communauté des biologistes a adopté les TICs dès les années 90 à cause des avantages qu'elles apportent dans l'acquisition et l'utilisation de données pour résoudre des problèmes scientifiques [Ste02, Chi02]. Ainsi, le domaine bio-informatique est aujourd'hui caractérisé par une forte augmentation des ressources informatiques et l'apparition de nouveaux environnements virtuels de recherche [LBRM+13]. Il existe plus de 1300 bases de données en biologie. Elles sont distribuées à travers le monde, par exemple DDBJ (DNA Data Bank of Japan) [2], EMBL-EBI (Protein Data Bank in Europe) [3], UniProt (Universal Protein Resource) [4] [GF12]. Les bases de données stockent diverses informations, par exemple des espèces, des organismes, des organes, des tissus, des cellules, des séquences biologiques, des ontologies. Elles stockent également des services et des expériences. Une base maintenue par EBI (European Bioinformatics Institute) contient plus de 2000 expériences [Rom08]. La plateforme myExperiment [5] contient plus de 3500 workflows. La plateforme BioCatalogue [6] contient environ 1180 services répartis entre plus de 200 fournisseurs et la platforme EMBOSS [7] propose plus de 300 outils destinés à l'analyse de séquences. Galaxy [8] partage plus de 3000 outils. De nombreuses infrastructures telles que des serveurs et des grilles sont utilisées pour le stockage et le traitement des données, par exemple à travers la plateforme GenOuest [9].

L'existence de telles ressources offre de réelles opportunités pour les expériences scientifiques. Des connaissances et des modèles mathématiques ou empiriques sont exploitables sous forme de données et programmes informatiques. Des expériences qui n'étaient possibles que sur des terrains lointains et hostiles sont devenues réalisables sur un terrain virtuel accessible. Il est possible de conduire des expériences sur ordinateur en utilisant des workflows. Cependant, comme introduit précédemment, pour exploiter pleinement les workflows, il faut relever les défis concernant leur création [AFG+03].

Concrètement, un workflow est une composition de services où plusieurs services sont liés par le fait que l'un des deux effectue une tâche à partir des données

2. http ://www.ddbj.nig.ac.jp/
3. http ://www.ebi.ac.uk/
4. http ://www.uniprot.org
5. http ://www.myexperiment.org, le 29/04/2015
6. https ://www.biocatalogue.org, le 29/04/2015
7. emboss.sourceforge.net, le 29/04/2015
8. https ://toolshed.g2.bx.psu.edu, le 29/04/2015
9. http ://www.genouest.org

résultats d'autres services. À cause des problèmes liés, d'une part, à l'incompatibilité des services et, d'autre part, au manque d'assistance aux utilisateurs, retrouver et composer des services adéquats n'est pas encore assez simple pour beaucoup d'utilisateurs.

Une catégorie d'incompatibilité entre services est causée par l'utilisation de formats de données différents au niveau des entrées et sorties des services. Elle se manifeste par l'impossibilité pour deux services d'échanger directement des données : le premier service produit des données qui ne sont pas directement consommables comme entrées du deuxième service. Il nécessite d'adapter les données produites par le premier service pour satisfaire le deuxième. Pour cela, la plupart des plateformes en bio-informatique utilisent des services spéciaux conçus pour aligner et convertir des données [WvdVW+09]. Cette solution est pénible pour un utilisateur final ; il perd beaucoup de temps, et il est exposé à des erreurs. Des solutions automatisées permettent de faciliter la gestion des incompatibilités entre services mais elles restent limitées à des correspondances entre services et à des transformations de données simples ; elles ne permettent pas des transformations sur des données complexes [WMK+08]. À cause, entre autres, des incompatibilités, du nombre et de l'hétérogénéité des services [NL05], les stratégies de composition ont du mal à facilement permettre de retrouver et composer des services spécifiques. Il est difficile de retrouver des services qui soient compatibles. Il faut en général connaitre en détails les services et leurs formats d'entrée/sortie avant de pouvoir les utiliser et les interconnecter [GSH+08]. Beaucoup de tâches sont encore très manuelles, par exemple, concernant la gestion des connections entre services. Il y a un manque de guidage dans la composition de workflows cohérents.

1.2 Thèse : composition guidée de workflows

Dans cette thèse, nous présentons nos contributions pour gérer les incompatibilités des services, et faciliter la sélection et la composition de services sous forme de workflows. Notre objectif est de guider les utilisateurs dans la composition de workflows cohérents à travers la suggestion de services et la gestion des incompatibilités entre entrées et sorties des services. Nous présentons deux contributions majeures. La première contribution concerne la détection et la résolution des incompatibilités de services. Nous proposons une méthode pour gérer la convertibilité d'une sortie d'un service vers une entrée d'un autre service. La deuxième contribution concerne des mécanismes permettant d'aider les utilisateurs à sélectionner et à interconnecter des services compatibles. Nous proposons une approche de composition guidée de services exploitant la suggestion de services et de liaisons sûres.

Contribution 1. Notre approche de convertibilité permet d'assurer systématiquement la correspondance et l'adaptation des données entre services. Elle est basée sur un ensemble de règles de convertibilité. Les règles s'appuient sur une abstraction de types qui permet de définir les types des entrées et sorties des services, en considérant la nature et la composition des données. Les règles permettent de vérifier la compatibilité d'un type de sortie d'un service par rapport à un type d'entrée d'un autre service, et de transformer les informations de la sortie pour alimenter l'entrée. Elles sont définies à travers une notion de convertibilité : un type A est convertible en un type B s'il est possible, en appliquant les règles, d'obtenir une fonction transformant toute donnée de A en une donnée de B. Les règles prennent en compte la composition, la décomposition ainsi que la généralisation et la spécialisation de types. Elles permettent non seulement une correspondance entre entrées et sorties, mais également la génération des convertisseurs de données à utiliser entre services dans les workflows.

Contribution 2. Notre approche de composition de services permet de guider un utilisateur en utilisant la convertibilité des types d'entrée et sortie des services. Le guidage est basé sur le modèle des Systèmes d'Information Logiques (LIS) [10] qui permettent des requêtes et une navigation guidées et sûres sur des données représentées avec une logique uniforme. L'approche formalise un modèle de workflow, des propriétés désirées des workflows, des mécanismes de suggestion et les actions possibles de l'utilisateur pour la sélection et la composition de worklows. Elle permet à un utilisateur de spécifier des points d'un workflow qu'il désire compléter, de bénéficier de suggestions, de sélectionner et d'établir des liaisons entre des services compatibles. Notre stratégie de composition permet de définir, étape par étape, des workflows complets et bien formés.

Nos approches ont été implémentées et évaluées avec des experts du domaine. Elles ont été appliquées à des services et types de données en génomique couramment utilisés, qu'on trouve dans les plateformes EMBOSS [11], EMBL-EBI [12] et BioCatalogue [13]. Le travail a été effectué avec la collaboration d'utilisateurs et de développeurs de la plateforme bio-informatique GenOuest [14].

10. http ://www.irisa.fr/LIS
11. http ://www.bioinformatics.nl/emboss-explorer
12. https ://www.ebi.ac.uk/services
13. https ://www.biocatalogue.org
14. http ://www.genouest.org

1.3 Organisation du document

La suite de ce manuscrit est composée de 5 grandes parties :

Chapitre 2 (Travaux existants) : Cette partie présente des travaux existants dans le cadre de la composition de services. Nous présentons d'abord des concepts et des technologies relatifs aux services, particulièrement en bio-informatique. Nous présentons ensuite la composition de workflows, avec différentes techniques pour la gestion des incompatibilités entre services et des techniques pour la composition de services.

Chapitre 3 (Outils et méthodes) : Cette partie revient sur la théorie des types qui sert de fondement à notre approche. Nous présentons des solutions XML dans la représentation des types. Nous présentons ensuite les LIS, leurs concepts de base et leurs utilisations. Les éléments de cette partie permettent de poser les outils et les méthodes sur lesquels notre travail repose.

Chapitre 4 (Convertibilité entre entrées et sorties de services) : Cette partie détaille notre approche de la convertibilité. Nous présentons le système de règles, ses propriétés et son implémentation. Nous détaillons un cas d'utilisation et exposons l'utilisation de notre approche en bio-informatique et les expériences effectuées. Ce chapitre est dérivé de l'article [BFD15b].

Chapitre 5 (Composition guidée de workflows) : Cette partie expose notre approche de composition de services. Elle détaille les éléments de notre approche, notamment les modèles, des propriétés de workflow, les mécanismes de suggestion, les actions de l'utilisateur basés sur les LIS ainsi que des preuves formelles de notre approche. Nous présentons l'implémentation et un cas d'utilisation de notre approche. Ce chapitre est dérivé de l'article [BFD15a].

Chapitre 6 (Conclusion générale) : Cette partie termine le manuscrit avec une conclusion générale et les perspectives de notre travail. Plusieurs pistes pour compléter et améliorer notre travail sont discutées dans cette partie.

Chapitre 2

État de l'art

Le développement de composants logiciels autonomes accessibles et distribués à travers le web (e.g., les services web) a ouvert des perspectives intéressantes pour l'intégration et l'interopérabilité des systèmes. Les services web offrent des moyens, rendant les activités scientifiques plus adaptées et plus efficaces qu'elles ne l'ont été, par une infrastructure d'accès et d'intégration de ressources hétérogènes et distribuées. La nouvelle donne laisse entrevoir un *monde informatique* où les composants logiciels sont disponibles à distance, capables de discuter entre eux, de se composer et d'accomplir des tâches très complexes de manière autonome. Dans ce chapitre, nous allons exposer certains des éléments qui vont participer à constituer ce monde. Nous allons présenter des solutions relatives aux services et à leur composition sous forme de workflows. Ces solutions seront exposées suivant deux parties, en se focalisant particulièrement au domaine bio-informatique. La première partie présente des solutions concernant les services : services web, ontologies, données et plateformes de services (Section 2.1). Elle caractérise l'environnement des services et des ressources en bio-informatique. La deuxième partie revient sur des notions de workflows avant de s'intéresser à des travaux existants sur la composition de workflows. Les travaux existants concerneront, d'une part, des solutions de gestion des incompatibilités entre services (Section 2.2.1) et, d'autre part, des techniques semi-automatiques de composition de workflows (Section 2.2.2).

2.1 Services

Les services[1] peuvent être vus comme des objets logiciels autonomes, utilisables et combinables en utilisant des protocoles standards [Pre04], pour assurer des fonctions spécifiques telles que réserver un billet d'avion, envoyer un tweet ou lister des produits en ligne. Ils ont été introduits pour faciliter l'interopérabilité et l'intégration des systèmes [ACKM03, CKM+03]. Les services permettent de découpler les fonctions des logiciels des langages (ou plateformes) utilisés pour les définir. Ils permettent la ré-utilisation des composants logiciels. Ils offrent des moyens d'automatiser les interactions entre des systèmes différents et distants. Les services peuvent être décentralisés et distribués à travers Internet. Ils peuvent être accessibles à travers différentes plateformes [NBCT06].

Dans les architectures orientées service (e.g., SOA) [Pap03], un service est composé d'une implémentation et d'une interface. L'implémentation fournit les opérations internes du service dans un langage de programmation tel que Java, Python ou Perl. L'interface, la description du service, définit l'identité du service ; elle permet la publication et l'accès aux fonctionnalités du service. La description d'un service peut concerner plusieurs aspects : la structure des données et des messages échangés, les protocoles utilisés, les fonctionnalités offertes, la localisation du service, la nature des données et des opérations, le fonctionnement du service, la composition du service, les fournisseurs ou encore les aspects QoS du service [PVDH07].

Dans la suite de cette partie, nous nous intéressons aux interfaces des services avec les solutions purement XML et les services dans le web sémantique. Nous présentons également différentes solutions pour annoter les services, des formats de données d'entrée et sortie ainsi que l'environnement des services (Section 2.1.5), en bio-informatique, à travers des registres de services.

2.1.1 Services web SOAP et REST

Les standards de fait, longtemps utilisés, sont les services web basés sur SOAP[2]. Il existe les services REST[3], également très utilisés, par exemple dans la plateforme Amazon[4]. Les deux solutions sont basées sur XML, et elles sont centrées données et transport. Leurs descriptions de services se situent au niveau

1. Nous utilisons le terme *service* pour designer les services web et tous les autres outils services (qui ne respectent pas pleinement les standards des services web).
2. http ://www.w3.org/TR/soap/
3. http ://www.ics.uci.edu/ fielding/pubs/dissertation/top.htm
4. http ://aws.amazon.com

syntaxique, et elles concernent des structures de messages échangés et de points d'accès à des fonctionnalités de services. Les solutions SOAP et REST sont à la base de la plupart des services utilisés, notamment dans cette thèse.

Les services SOAP sont basés sur SOAP (Simple Object Access Protocol), WSDL (Web Service Description Language) et UDDI (Universal Description Discovery and Integration). SOAP, WSDL et UDDI sont des standards ouverts basés sur les protocoles Internet. SOAP, une recommandation du W3C définie comme une référence, est un protocole d'échange de messages sous forme de données XML. Le contenu d'un message à échanger entre services est dissimulé, accompagné des règles de traitement à appliquer (Processing Model) et du protocole de transport à utiliser pour acheminer le message (Protocol Binding). WSDL est un langage XML de description de services et UDDI est un modèle de publication de services (nous revenons sur UDDI à la section 2.1.5). Un document WSDL comprend deux niveaux constitués d'éléments sous forme de structures XML. Le premier niveau définit les types de données, opérations et interfaces d'un service. Le deuxième niveau spécifie les ports d'accès et les liaisons avec les interfaces du service. Les types représentent les entrées et sorties des opérations. Les opérations sont définies à travers des messages selon des politiques de réception et de transmission. Une interface définit un ensemble d'opérations qui sont fournies par un service. Ces opérations sont accessibles de manière concrète à travers les ports et les liaisons [Alo02].

Les services REST (REpresentational State Transfer) offrent une autre vision des services. REST est introduit comme un style d'architecture (un concept, une idée) client-serveur où les services sont vus comme des ressources identifiées par des URIs. Les services REST exploitent directement le concept du web. Ils s'appuient sur la notion d'URI et sur les méthodes HTTP (POST, GET, PUT et DELETE) permettant de manipuler des ressources de manière simple. Les services REST peuvent être classés selon trois niveaux. Le premier niveau introduit l'utilisation d'URI, le deuxième niveau intègre les méthodes HTTP et le troisième niveau permet de considérer la contrainte HATEOAS (Hypermedia As The Engine Of Application State) qui permet des liens hypermédias pour l'interaction avec les services. Il n'y a pas de langage retenu pour la description des services REST. Beaucoup de services sont décrits à l'aide de textes libres ou de pages HTML. Des langages, tels que WADL (Web Application Description Language) [Had06] et RSDL (RESTful Service Description Language) [Pas12], basés sur XML, sont proposés mais restent encore peu utilisés.

Du point de vue pratique, les services SOAP offrent plus de possibilités (par exemple concernant la sécurité) mais les services REST sont plus simples à utili-

ser. Les services REST utilisent simplement HTTP là où les services SOAP utilisent plusieurs couches et des protocoles différents. Les services SOAP exposent leurs fonctionnalités sous forme d'agents exécutables à distance, et les services REST exposent leurs fonctionnalités sous forme de ressources identifiées et accessibles par HTTP. En bio-informatique, les deux solutions sont utilisées. Les services REST sont plus utilisés dans le cadre de l'accès et extraction de données, et les services SOAP semblent dominer dans les aspects analyses [KNT10].

2.1.2 Services web sémantiques

Les services web sémantiques [FFST11] sont proposés dans le but de réduire l'intervention humaine et de permettre l'automatisation et le raisonnement sur des tâches telles que la découverte, la composition et l'exécution des services. Ils utilisent comme fondation les services décrits à la section précédente et prennent en compte des aspects tels que le sens des fonctionnalités, le fonctionnement des services et le sens des données [HLL+09]. Ils intègrent des méta-informations concernant les domaines d'utilisation des services. Ils sont une convergence entre le web sémantique et les services web. Les technologies du web sémantique, telles que le langage OWL (Ontology Web Language) et les ontologies de domaine, sont exploitées pour modéliser et annoter les ressources concernant les services [MSZ01, FFST11]. Une ontologie permet de définir un modèle et un vocabulaire partagé. Elle passe par des concepts et des relations entre concepts. Nous verrons des exemples d'ontologies de domaine à la section suivante (Section 2.1.3). Nous allons présenter des modèles pour définir des services web sémantiques :

SAWSDL (Semantic Annotation for WSDL) [KVBF07] est développé par le W3C pour ajouter de la sémantique à la description des services en utilisant la couche WSDL comme base. SAWSDL ne fournit pas directement la sémantique mais prévoit les moyens d'attacher des annotations au document WSDL. Il permet d'annoter les éléments WSDL tels que les types de données et les opérations ainsi que d'ajouter des pré-conditions, post-conditions et une catégorisation des services. Une solution semblable à SAWSDL, SA-REST, est proposée par Sheth et al. [SGL07] pour ajouter des annotations à des services REST.

OWL-S (Ontology Web Language for Service) [MBH+04] est une ontologie pour décrire des services. Elle cherche à fournir les informations sur ce que réalise un service, comment l'utiliser et comment y accéder. Elle est composée des éléments *Service Profil*, *Service Model* et *Service Grounding*. L'élément *Service Profile* définit ce que le service fait. Il permet de décrire un service en termes de ses caractéristiques fonctionnelles (par exemple entrées, sorties, pré-conditions, post-conditions), non-fonctionnelles (par exemple QoS, catégorie, informations statis-

tiques) et des informations fournisseur (par exemple nom, description, propriétaire). *Service Model* décrit comment un service fonctionne en termes de *process*. Un *process* décrit l'approche utilisée pour les interactions avec un service. Il peut être atomique ou composite. Pour décrire des *process* composites, les constructeurs de structures tels que *sequence* et *split* sont utilisés. Les structures forment par exemple des séquences de tâches ou des tâches parallèles. *Service Grounding* fournit les détails d'accès à un service. Il spécifie la correspondance entre la représentation abstraite d'un service et la description des éléments concrets qui permettent d'interagir avec le service. La représentation abstraite correspond aux éléments *Service Profile* et *Service Model*. La base concrète est généralement WSDL mais peut être toute autre interface de services.

WSMO (Web Service Modeling Ontology) [RKL+05], semblable à OWL-S, est également une ontologie pour décrire des services. Elle est basée sur un ensemble de principes dont l'utilisation de la notion d'URI pour identifier les ressources, l'utilisation de concepts pour décrire toutes les ressources et la prise en compte de la médiation des données (gestion de l'interopérabilité des ressources). WSMO comporte les éléments *Ontology*, *Goals*, *Web Service* et *Mediators*. *Ontology* fournit la terminologie utilisée pour décrire les éléments. *Web Service* fournit le modèle conceptuel pour décrire les aspects fonctionnels et les interfaces des services de manière explicite et unifiée. Les *Goals* représentent les souhaits d'un utilisateur. Ils permettent l'accès aux tâches concrètes potentiellement disponibles à travers les services. Les *Mediators* permettent de gérer l'interopérabilité entre les éléments WSMO, par exemple entre deux ontologies ou entre deux services. WSMO diffère de OWL-S, entre autres, par le fait qu'elle découple la vision utilisateur d'un service de la vision fournisseur. OWL-S ne considère pas explicitement l'interopérabilité des ressources. Les deux ontologies diffèrent également sur d'autres points : différence dans l'utilisation des propriétés non-fonctionnelles, dans le langage de spécification et les niveaux d'abstraction utilisés, dans la description des interactions des services [LRPF04].

Les solutions présentées sont utilisées dans beaucoup de plateformes. Par exemple, les outils Bio-Sense [SBH06], INFRAWEBS [AM07] et AIMO [TKW+10] sont basés sur les solutions de WSMO pour utiliser des services. OWLS-XPlan [KGS05], OSIRIS Net [MS10] et SHOP2 [SPW+04] sont des solutions utilisant OWL-S, et WS-BioZard [WMK+08] utilise SAWSDL. Des extensions et des variantes sont également utilisées. Par exemple, le projet METEOR-S utilise MWSAF (METEOR-S Web Service Annotation Framework) qui est basée sur SAWSDL et le projet IRS-III utilise une extension de l'ontologie WSMO.

2.1.3 Ontologies de domaine

Comme déjà évoqué, les ontologies permettent de modéliser et de capturer les connaissances d'un domaine. Ces connaissances sont représentées sous forme de concepts et de relations entre concepts. En bio-informatique, il existe de nombreuses ontologies sur plusieurs aspects concernant par exemple des organismes, des cellules, des médicaments et des maladies [5]. Elles décrivent, classent et organisent les ressources. Elles sont destinées à faciliter l'intégration et l'interopérabilité. Dans le contexte de la composition de services, les ontologies de domaine annotent et organisent les ressources concernant les services. Elles servent à faciliter des tâches telles que la recherche, la comparaison, la sélection et l'intégration des services et des données. Nous allons présenter les ontologies de *BioMOBY*, de myGrid et *EDAM*.

BioMoby [WL02] est un projet dont le but est de mettre en place une architecture pour la découverte et la distribution de données biologiques à travers des services. Il propose une ontologie qui contient un ensemble d'instances appelées *objets*. Chaque *objet* est identifié et représente une ressource, par exemple une séquence d'acides aminés ou un format de fichier. Les objets appartiennent à des classes racines formées à partir de *MOBY Triples* composés chacun d'une ressource, de l'espace de nom de la ressource et d'un identifiant de la ressource dans l'espace de nom. Les classes sont simplifiées et entretiennent des relations d'héritage (e.g., ISA) et de contenance (e.g., HAS). L'ontologie de BioMoby fournit également des classes de services qui décrivent et organisent des catégories de tâches qui sont offertes par les services.

L'ontologie de myGrid [WAH+07] existe pour supporter la découverte et la composition de services. Elle est proposée dans le cadre du projet myGrid [GWS03]. MyGrid [6] offre une suite d'outils destinés à des scientifiques pour des e-expériences. Les outils de myGrid sont utilisés en biologie mais également dans d'autres domaines tels que sciences sociales, musique, astronomie, multimédia et chimie. L'ontologie de MyGrid permet d'annoter des outils, des interfaces, des opérations, des types et des formats d'entrée et sortie. Elle décrit également des algorithmes, des bases de données, des enregistrements de bases de données et des références. Elle est séparée en deux composants distincts, une ontologie des tâches et une ontologie de domaine. L'ontologie des tâches décrit les propriétés physiques des services (par les entrées et sorties) ainsi que la localisation des services. Dans cette ontologie, les classes de base sont nommées *Operations*. Les classes opérations définissent des tâches, des méthodes et des ressources. Elles

5. http ://bioportal.bioontology.org
6. www.mygrid.org.uk

sont liées à des classes *Parameters* et *Services*. Les classes paramètres décrivent les paramètres d'entrée et de sortie d'une opération. Les classes services regroupent et forment des unités d'opérations. Elles sont liées à des classes *Organisations* qui décrivent les organismes propriétaires. Chaque classe de l'ontologie a un nom et une description. L'ontologie de domaine décrit les types d'algorithmes, les ressources ainsi que les types de données utilisés. Elle complète les annotations de l'ontologie des tâches. Elle regroupe, sous une classe *Informatics*, les concepts clés des données, structures de données, base de données et méta-informations. Une classe, *Bioinformatics*, construite au dessus de la classe *Informatics*, contient des sources de données et algorithmes d'analyse de données spécifiques au domaine bio-informatique. Des concepts de haut niveau décrivent les types de données utilisés comme entrées et sorties des services. Deux autres classe *Tasks* et *Formats* donnent une hiérarchie concernant, respectivement, les tâches génériques et les formats de fichier.

EDAM [IKJ+13] est une ontologie des opérations, domaines d'application, types de données et formats de données en bio-informatique. Elle a été proposée dans le cadre du projet EMBRACE [PIK+10] pour créer des annotations à utiliser dans des catalogues, des services, des infrastructures collaboratives, des collections d'outils ou des workbenchs. Elle utilise l'ontologie de myGrid comme base de travail. L'ontologie EDAM contient environ 2200 concepts et est composée de 4 sous-ontologies : *Operation*, *Topic*, *Data* et *Format*. La sous-ontologie *Operation* donne les concepts permettant de décrire les fonctions des outils. Il s'agit de fonctions diverses telles que l'alignement, la recherche, l'analyse et l'édition. *Topic* fournit différents concepts (catégories ou sujets) concernant les ressources. Ces concepts concernent des domaines d'études, des données, des traitements, des analyses et des technologies. *Data* représente des informations sous forme de concepts pouvant être associés à des entrées ou à des sorties d'outils. Ces concepts permettent d'annoter des types de données pouvant être des types primitifs, des paramètres simples ainsi que des types dérivés ou des types complexes bio-informatiques. Les concepts racines sont *Core data*, *Identifier*, *Parameter* et *Report*. *Format* représente les concepts pour annoter des formats de données communément utilisés en bio-informatique. Ces formats permettent d'encoder des types de données spécifiques sous forme de fichiers ou en mémoire. Ils concernent des formats textuels, binaires et XML ainsi que divers autres formats. A travers ses sous-ontologies, l'architecture EDAM comprend des concepts identifiés (ayant un nom, une définition, des propriétés intrinsèques) et une hiérarchie (qui lie les concepts par héritage et par des relations telles que *has input*, *has format*).

De nombreuses autres initiatives vont dans le sens de la modélisation et de la description des ressources en bio-informatique. La plateforme BioPortal[7] fournit un ensemble d'ontologies sur des sujets variés tels que l'anatomie, les phénotypes, les expériences, les images et la santé. En 2008, elle contenait 134 ontologies (680 000 classes). BioPortal est accessible à travers une interface permettant aux humains de rechercher, naviguer et filtrer les ontologies. Elle propose également une palette de services, RESTful, permettant d'accéder aux ressources. Les services sont utilisés pour développer des applications utilisant des ontologies [RSBM+10].

2.1.4 Formats de données

Pour représenter les données consommées ou produites par les services, plusieurs solutions concernant les formats de données sont adoptées. En bio-informatique, on note la présence de formats textuels et de formats encodés tels que les images, les fichiers binaires et les zips [GKA+11, PL09, LHW+09, RMB+10]. Il existe également des types de données basés sur XML, sur des classes d'ontologies ainsi que sur des langages de programmation [WL02, KPJ+10, LGG11].

Les formats textuels[8] permettent de contenir des données telles que des séquences biologiques, des résultats d'alignement de séquences et des annotations de séquences. *FASTA*[9], très répandu, est un de ces formats pour les séquences biologiques. Les formats textuels sont simplement lisibles pour un utilisateur final, et ils sont facilement éditables et adaptables selon les besoins. En plus, ils sont peu verbeux. Cependant, les formats proposés n'offrent pas une représentation de données complexes, et leurs schémas pour décrire les données sont en général implicites. La plupart des représentations textuelles sont sous une forme tabulaire [Li11], une forme utilisée pour sa simplicité visuelle et sa simplicité à l'utilisation. Les formats textuels sont cependant peu adaptés à des traitements automatiques ; il faut des convertisseurs, souvent manuels, pour le transfert de données.

Les formats textuels sont souvent accompagnés de formats binaires qui sont utilisés pour leur aspect compact, adapté à un transfert rapide de données massives [KZB+10, HBN10]. Par exemple, les formats textuels et binaires (BED et BAM) sont proposés pour le NGS (Next Generation Sequencing). Des données sous forme d'images, zip et tar sont également présentes.

7. bioportal.bioontology.org
8. voir www.bioperl.org/wiki/Category :Formats
9. http ://blast.ncbi.nlm.nih.gov/blastcgihelp.shtml

Les formats basés sur XML représentent les mêmes données que les formats textuels [SKH+06, MJRS+09, PL09, WIN+05]. Certains sont généralisés, par exemple BioXSD [KPJ+10] qui fournit des formats pour des données de base en bio-informatique. D'autres sont spécialisés, par exemple *phyloXML* pour les données phylogéniques [HZ09]. XML a l'avantage d'être expressif et de permettre une structuration des données. Il offre des schémas formels pour garantir la cohérence des données. Il permet un traitement automatique. Il est pris en charge pratiquement par tous les langages de programmation avec des technologies telles que DOM [10], SAX [11] et CDuce [12]. Les services web, à la base, utilisent XML pour représenter les données. Malgré les points forts, les représentations XML ont le désavantage d'être verbeuses. Les informations de structure représentent en général une bonne partie d'un fichier XML, ce qui le rend difficilement lisible par un humain. XML pose également des problèmes dans les applications de haute performance à cause de la surcharge qu'il crée lorsqu'il encode des données [NL05]. Une représentation compacte de XML nommée EXI (Efficient XML Interchange) [13] est recommandée par le W3C. Elle peut permettre d'améliorer l'utilisation de XML dans les applications de haute performance.

Tout comme XML, JSON (JavaScript Object Notation) est un format permettant des données structurées et un traitement automatique. Il est souple et est dérivé de la notation des objets du langage JavaScript. Il permet une lecture et écriture par une machine et par un humain familier des langages orientés objet. Bien qu'utilisant une notation JavaScript, JSON est indépendant du langage ; beaucoup de langages de programmation ont intégré JSON. Il est très utilisé pour représenter les données dans les services REST. Par rapport à XML, JSON est moins expressif. Les conversions entre JSON et XML se font, cependant, en général de manière automatique.

Par ailleurs, dans des plateformes telles que BioMoby [WL02], les données sont représentées à l'aide des objets de l'ontologie de BioMoby. Ces objets encapsulent des formats existants (e.g., texte, image) sous forme de ressources. Une solution similaire est utilisée dans la plateforme SADI avec des classes OWL-DL [WVM09]. Dans la même logique, certaines représentations XML permettent d'enrichir les formats de données par des concepts d'ontologies. Par exemple, BioXSD permet d'ajouter des annotations à ses schémas de données. Cela permet de lier des types syntaxiques (e.g., XML) à des types abstraits (e.g., concepts) décrits dans des ontologies. En outre, la notion de type telle que définie dans les langages de

10. www.w3schools.com/dom/
11. http ://sax.sourceforge.net/
12. www.cduce.org/
13. www.w3.org/TR/2014/REC-exi-20140211

programmation est également utilisée pour gérer des données structurées dans certains systèmes [LGG11].

En bio-informatique, il existe un grand nombre de formats différents, et ils sont en majorité textuels. Les formats se multiplient pour répondre à un besoin de représenter des informations différentes ou additionnelles en fonction des données, des expériences ou des méthodes. Par exemple, le format BED15 est une extension du format BED pour représenter des informations supplémentaires concernant les annotations de séquences biologiques. Les formats GTF (Gene Transfer Format) et GVF (Gene Variation Format) sont basés sur le format GFF (Gene Format Feature) pour représenter des variantes d'ADN. Selon les situations, certains formats se révèlent plus adaptés que d'autres : le format BED pour le traitement, le format BAM pour le transfert ou le format BEDGraph pour la visualisation [GKA+11]. Le choix et l'utilisation des formats est propre à chaque plateforme. En plus, il y a en général une absence de schémas formels pour les formats utilisés. Par exemple, chacune des plateformes UCSC (University of California Santa Cruz) [14] et EMBOSS [15] proposent plusieurs formats textuels pour charger ou télécharger des données, et les schémas sont décrits à l'aide de textes libres. Les formats XML tels que BioXSD [16] disposant de schémas formels sont encore peu utilisés.

2.1.5 Registres de services

Un registre de services permet, d'une part, la publication de services par des fournisseurs, et d'autre part, l'accès aux services par des consommateurs. UDDI (Universal Description, Descovery and Integration Protocole) est le framework proposé dans le cadre industriel pour publier et accéder à des services web. Il fournit un modèle de données qui sert à stocker des informations dans un registre et une API permettant de gérer le registre. Les informations stockées sont constituées d'aspects fonctionnels et techniques des services ainsi que d'aspects sur les fournisseurs des services. UDDI est issu d'une initiative d'un ensemble de compagnies réunies à travers un consortium, à but non lucratif, connu sous le nom de OASIS (Advancing Open Standards for the Information Society), qui conduit le développement, la convergence et l'adoption de standards ouverts [17].

La solution UDDI est quelques fois jugée rigide et pauvre en termes de méta-informations pour le contexte bio-informatique [GSH+08, WL02]. Cependant, son concept est à la base de plusieurs des registres existants [GWS03, RKO+03]. En

14. https ://genome.ucsc.edu/FAQ/FAQformat.html
15. http ://emboss.sf.net/docs/themes/AlignFormats.html
16. http ://bioxsd.org/
17. www.oasis-open.org

pratique, les services sont accessibles à travers différentes plateformes plus ou moins formelles, privées ou publiques, centralisées ou distribuées, en local ou à distance [GSH+08]. Certaines d'entre elles prennent en compte la description sémantique des ressources [WVM09]. Des initiatives, telles que BioMoby [18] et EMBOSS [19], ont permis de collecter et de rendre accessibles des services spécifiques à la biologie [PIK+10, CHS+03]. Des plateformes comme Galaxy [20] et Mobyle [21] permettent de centraliser des registres de services et de les partager entre des membres de communautés en ligne. Le réseau GenOuest [22] propose plusieurs registres à ses utilisateurs. Les services sont dans des instances de Galaxy et Mobyle ainsi qu'en local. L'instance de Galaxy est fermée et l'instance de Mobyle est ouverte. BioCatalogue [23] est un registre ouvert, disponible en ligne, permettant de soumettre et d'accéder à des services [BTN+10]. Il est une extension de UDDI qui supporte des vues personnalisées et des méta-informations facilitant la recherche de services par des machines et par des humains [GWS03]. Des registres généraux existent également. Seekda [24] est un moteur de recherche pour tout type de services et des services sont disponibles dans d'autres registres en ligne [LH07]. Des services sont également proposés dans d'autres domaines, par exemple la plateforme Gate [Cun02] dispose d'outils pour le traitement des langues.

Comme pour les registres, les services sont également différents d'une plateforme à une autre et au sein même d'une plateforme. Ils peuvent être strictement basés sur XML, utiliser des technologies du web sémantique ou être liés à des interfaces graphiques [GSH+08]. Ils peuvent se présenter sous forme de simples scripts accessibles par des appels en lignes de commandes, des programmes avec des *wrappers* sous forme de formulaires, des services avec des descriptions sémantiques, des variantes de services web REST et SOAP ou des applications. Les services peuvent également être disponibles en local ou en ligne [WHF+13, HWS+06, MGN+10]. En bio-informatique, peu de services respectent pleinement la spécification des services web standards. Selon Lord et al. [LAWG05], sur plus de 1000 services disponibles dans le projet myGrid, seul 5% peuvent être considérés comme des services web utilisant SOAP/WSDL. Lord et al. montrent que 30% sont des services REST, et que 30% sont des services BioMoby qui sont simplifiés et annotés avec une ontologie. Les services Soaplab occupent 25%, et 10% sont des workflows. SoapLab permet d'encapsuler des outils en lignes de commandes sous forme

18. www.biomoby.org
19. www.emboss.open-bio.org
20. https ://galaxyproject.org/
21. http ://mobyle.pasteur.fr
22. www.genouest.org
23. www.biocatalogue.org
24. www.seekda.com

de services. Le registre BioCatalogue gère également plusieurs types de services dont des services SOAP, REST et Soaplab. BioCatalogue contient en 2015 plus de 1000 services et est alimentée par environ 250 fournisseurs. Une étude menée par Wassink et al. [WvdVW+09], en 2008, sur les workflows de la plateforme Taverna, montre que 57% des services sont locaux. Les services web représentent 22%, et ils sont composés de services SOAP, de services Soaplab [SRO03], de services BioMoby [WL02], de services SeqHound [MBD+02] et de services BioMart [DMK+05]. L'étude montre également que plus de 30% des services sont destinés à la conversion de données. Ils sont beaucoup plus importants que l'ensemble des services de domaine.

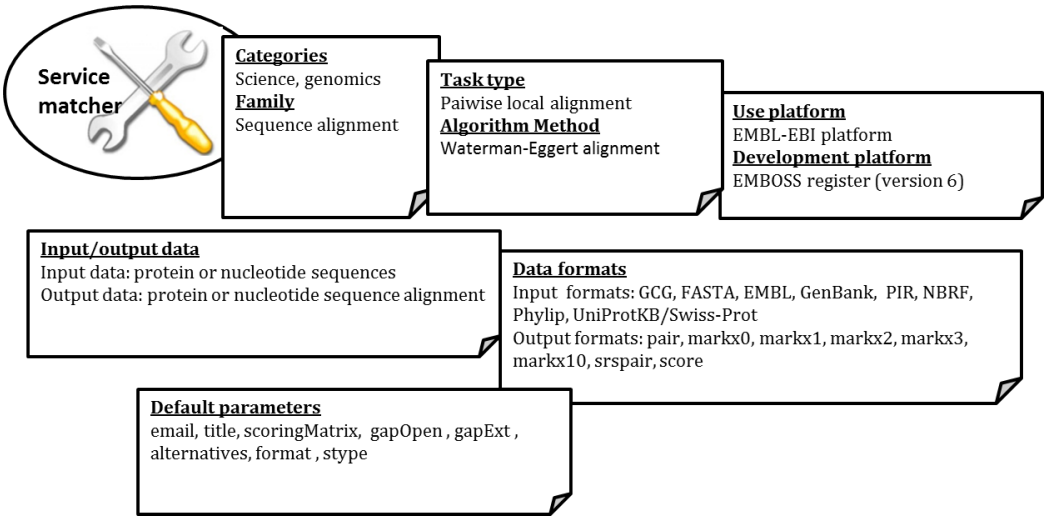

FIGURE 2.1 – Un service nommé *matcher* et des descriptions dans EMBL-EBI.

Les tâches des services de domaine sont de plusieurs sortes. Elles peuvent être génériques, par exemple, la fouille de textes et le traitement d'images. Elles peuvent également être spécifiques à un domaine, par exemple l'analyse de séquences biologiques ou de structures moléculaires. Les tâches peuvent permettre l'accès à des informations contenues dans des sources de données, l'extraction et l'analyse de (gros volumes de) données ou encore pour l'affichage et la présentation de résultats. Une bonne partie des tâches concerne la recherche de similarité, la recherche de motifs et l'extraction de séquences [SGBB01].

Un service typique est présenté à la figure 2.1. Il s'agit d'un service exploité sous forme SOAP et sous forme REST au niveau de la plateforme EMBL-EBI [25].

25. www.ebi.ac.uk/tools/psa/emboss_matcher

Il est basé sur des API Perl, Java et C#. Le service est nommé *matcher*. Les concepts qui le catégorisent sont *science* et *génétique*. Il appartient à la famille des services pour l'alignement local de séquences. Sa tâche consiste à aligner des pairs de séquences. Le service utilise la méthode d'alignement local *Waterman-Eggert* [WE87]. Il est implémenté par EMBOSS [26], dans sa version 6. Il consomme deux séquences de protéines ou de nucléotides. Il produit un alignement des deux séquences. Il accepte, en entrée, les formats textuels *GCG*, *FASTA* , *EMBL*, *GenBank*, *PIR*, *NBRF*, *Phylip* ou *UniProtKB/Swiss-Prot*. Il propose, en sortie, les formats textuels *pair* , *markx0*, *markx1*, *markx2*, *markx3*, *markx10*, *srspair* ou *score* [27]. Le service utilise également des paramètres (avec des valeurs par défaut) comprenant l'email de l'utilisateur, le titre de la tâche, une matrice de score utilisée pour comparer les séquences, la pénalité (ou score) de création de gap, la pénalité (ou score) d'extension de gap, le nombre d'alignements alternatifs pouvant être produits, le type des séquences traitées et les formats utilisés. Des descriptions sont également fournies sur la manière d'appeler le service et d'autres informations peuvent également être disponibles sous forme de commentaires et de données statistiques [28].

Les services qu'on retrouve dans les plateformes sont semblables à *Matcher*. Ils disposent de plusieurs descriptions à travers des documents WSDL ou des pages web. Dans les plateformes, il est assez rare que les services respectent un même modèle et les mêmes facettes de description. Beaucoup de facettes ne sont pas prises en compte, sont dispersées ou ne sont pas assez formellement définies. Beaucoup de services sont encore décrits à l'aide de textes libres. Pour exploiter les fichiers WSDL, il faut programmer l'accès par un expert humain. Certains services sont conçus avec des interfaces sous forme de formulaires parce que destinés à être directement consommés par des humains. Les données d'entrées et sorties sont en général représentées avec des formats textuels pouvant être très différents. Les formats peuvent changer selon qu'il s'agit d'un service pour le traitement, pour le transfert ou pour la visualisation. Beaucoup d'entre eux sont définis et maintenus par des utilisateurs en local. Beaucoup sont propres à une plateforme ou sont adaptés d'une plateforme à une autre. Les formats sont en général considérés de manière globale (non décomposable) et tiennent peu compte formellement de la structure et de la nature des données représentées. Des ontologies sont utilisées dans certaines plateformes pour annoter les ressources mais leur potentiel reste sous exploité. Les tâches offertes par des services comme *matcher* sont en général proposées sous plusieurs versions, par plusieurs outils différents et à travers différentes plateformes. Elles peuvent également être annotées ou

26. http ://emboss.open-bio.org
27. http ://emboss.sf.net/docs/themes/AlignFormats.html
28. www.programmableweb.com/api/emboss-matcher/développeurs

classées différemment d'une plateforme à une autre [GKA+11, Ste02, GSN+01].

2.2 Composition de workflows

Un workflow peut être vu comme un service global composé de plusieurs autres services. La composition de workflows consiste à lier les autres services dans le but de constituer le service global. Il s'agit d'organiser des interactions entre des services différents afin qu'ils accomplissent une tâche spécifique. Les briques de base de la composition sont des services atomiques. Un service atomique ou service élémentaire est un point d'accès vers une application qui ne repose pas sur un autre service pour accomplir sa tâche. Il est différencié d'un service composite qui est constitué d'autres services composites ou élémentaires [SQV+14, MWT+10, WW08, DS05, MM04]. Les workflows sont en général gérés par des systèmes appelés *Workflow Management Systems* (WfMS) [VDAVH04]. Dans la suite, nous présentons quelques notions de la composition de workflows et nous passons, ensuite, en revue des travaux sur la composition de workflows.

Les deux formes de composition sont l'orchestration et la chorégraphie [Pel03]. L'orchestration représente un processus central qui coordonne les interactions entre différents services. Elle définit de manière fixe un ensemble d'étapes et des interactions décrivant un processus exécutable. Elle est utilisée pour décrire un service composite caractérisant un produit, par exemple lorsqu'il y a un besoin de gérer tout un processus avec éventuellement une définition graphique. La chorégraphie représente une collaboration entre plusieurs services où chaque service joue son rôle de manière autonome. Contrairement à l'orchestration, la chorégraphie définit un ensemble de processus concurrents ; elle est associée aux interactions qui se produisent entre les différents services. La chorégraphie est utilisée, par exemple, dans le cas de processus très dynamiques, pour maintenir une indépendance entre partenaires, ou pour permettre à des partenaires de personnaliser leur processus.

Des langages appelés langages de workflows sont utilisés pour représenter concrètement les workflows. Parmi les langages, il y a BPEL4WS (Business Process Execution Language for Web Services) pour orchestrer des services web. Il est proposé par IBM et Microsoft. La composition avec BPEL donne un résultat exécutable nommé *process*. Un processus BPEL est un fichier XML contenant plusieurs éléments dont des activités primitives et des structures. Les activités primitives sont composées d'éléments permettant, par exemple, d'appeler des opérations, de manipuler des variables ou de générer des réponses. Les structures ou contrôles combinent les activités primitives en une activité globale. Elles

exploitent des structures telles que la formation de séquences, de choix ou de séquences parallèles. BPEL est un langage utilisé pour l'orchestration mais il permet également de définir une chorégraphie [SK03].
WS-CDL est un langage destiné à la chorégraphie. Il permet de composer des interactions pair à pair entre des services. Il permet de se mettre d'accord sur l'ordre et les contraintes pour l'échange de messages entre des participants. Le modèle de WS-CDL est composé d'entités définissant des interactions basées sur des rôles exposés par des participants et contraints par des relations. Un rôle énumère le comportement potentiel d'un participant. Un participant regroupe toutes les parties d'une collaboration pouvant être implémentées par une entité. Une relation définit les obligations mutuelles pour qu'une collaboration fonctionne. Les informations sont échangées entre les participants sous forme d'une chorégraphie. Des éléments spécifiques (e.g., variables et alias) permettent de contenir les informations dans une collaboration. D'autres langages similaires à BPEL4WS et WS-CDL existent : XPDL (XML Process Definition Language), BPML (Business Process Modeling Language) [29].

En bio-informatique, le langage Scufl est proposé [LBW+04]. Il s'agit d'un langage simplifié par rapport aux solutions précédentes. Il a été adapté aux besoins en bio-informatique, dans le cadre du projet myGrid. Il permet un niveau d'abstraction élevé par rapport à BPEL. Il suit une logique flux de données et minimise les aspects structures de contrôle. Il permet de considérer des descriptions sémantiques et d'intégrer les notions de provenance et de suivi des données [DCBE+07]. Le langage Scufl est constitué des entités *processeur*, *liaison* et *contrainte*. Un processeur est vu comme une fonction avec un ensemble d'entrées et un ensemble de sorties. Les entrées et sorties sont associées à des ports identifiés. Une liaison représente la consommation d'une donnée d'un port de sortie par un port d'entrée. Elle est définie par un nom, un processeur source, un processeur destination, un port d'entrée et un port de sortie. Une contrainte permet d'ajouter des conditions additionnelles pour l'exécution des processeurs. Elle peut consister, par exemple, en des actions de blocage ou de déblocage d'un processeur durant l'exécution.

La composition peut être statique ou dynamique selon les langages et les plateformes [ACMS08]. Pour une composition statique, le choix et l'interconnexion des composants se fait à la création. Les services sont connus ; ils sont sélectionnés, composés et déployés. La composition statique est utilisée dans les situations où les participants sont fixes, et où les fonctionnalités et les besoins établis ne changent pas beaucoup. Elle n'est pas flexible à des modifications ou adaptations durant l'exécution. Les outils Biztalk [Vas01] et WebLogic [GEW01] sont des

29. http ://www.ebpml.org/

solutions pour la composition statique en entreprise. Contrairement à la composition statique, la composition dynamique permet de déterminer et de remplacer des composants à l'exécution. Elle nécessite, cependant, des mécanismes automatiques pour sélectionner et interconnecter des services. Bien que la composition statique soit adaptée aux environnements dynamiques en facilitant les mises à jour et en permettant d'adapter les applications en fonction du contexte, sa mise en place est complexe. Elle impose des techniques pour garantir les aspects tels que la sureté, le temps d'exécution et les transactions. On retrouve dans cette catégorie les plateformes Sword[PF02] et eFlow [CIJ+00].

Le cycle de vie d'un workflow comprend plusieurs phases [SQV+14]. Parmi elles, une phase de définition du workflow (composition de services) qui doit garantir une bonne expression des besoins et une cohérence de la composition. La définition du workflow se matérialise par une modélisation des interactions des services participants, des flux de données, des structures de composition ainsi que des contraintes temporelles, transactions et exceptions. Une phase concerne la découverte de services [HLL+09]. Elle permet de trouver les composants réels correspondant aux tâches désirées. D'autres phases traitent des aspects tels l'exécution, la configuration, le diagnostic, le suivi d'exécution et la gestion des workflows [VdAvDH+03].

Dans les prochaines parties, nous nous intéressons à la définition de workflows impliquant sélection et composition de services. Il sera question de travaux sur la médiation de services pour la composition, à la section 2.2.1, et de techniques de composition de services, à la section 2.2.2. Les workflows qui nous concernent sont orientés flux de données, proche du modèle de Scufl ; la composition est du type orchestration.

2.2.1 Médiation de services

La compatibilité des services est l'un des problèmes critiques de la composition. En effet, lors de la composition de services, il est assez courant que les services ne soient pas directement compatibles. Cette incompatibilité est due à l'hétérogénéité des ressources dont nous avons parlé à la section 2.1.5 : multiplication des solutions et des plateformes, existence de formats et de services hétérogènes, formats textuels libres, absence de descriptions formelles. Face à l'incompatibilité, des solutions de médiation de services sont proposées. La médiation fait l'objet d'une attention particulière dans le contexte de l'ingénierie logicielle [DSW06, YS97, ZW97]. Elle permet de résoudre des problèmes d'interopérabilité entre les composants logiciels.

Des travaux [LFJ07, KNB+09] ayant porté sur l'identification et la classification des incompatibilités dans la composition de services font état de plusieurs sortes d'incompatibilité. Parmi elles, il y a l'incompatibilité au niveau syntaxique et l'incompatibilité au niveau sémantique. Pour chacun de ces deux niveaux, il y a l'incompatibilité non-fonctionnelle et l'incompatibilité fonctionnelle. L'incompatibilité non-fonctionnelle concerne en particulier les aspects QoS et fiabilité [LFJ07]. L'incompatibilité fonctionnelle concerne en particulier les protocoles et les interfaces des services. Les protocoles utilisés par les services peuvent être incompatibles à cause de l'ordre des messages, de messages supplémentaires, de pertes de messages ou du non respect d'un nombre de messages. Les incompatibilités qui arrivent au niveau des interfaces concernent des services utilisant des paramètres d'entrée et sortie différents [KNB+09]. Dans la suite, nous nous concentrons sur les incompatibilités au niveau des paramètres des services, plus précisément, au niveau des données en entrée et sortie des services.

Utilisation de "shims"

Le mot "shims" désigne à l'origine une cale utilisée dans le bâtiment pour aligner des bouts de métal. Les shims sont ici des services spéciaux utilisés pour résoudre des incompatibilités entre services pour la composition [HSL+04, LLF+09, MLK14, AK07, KLC13]. Ce sont des convertisseurs, des médiateurs de données, qui font le pont entre les entrées et sorties des services. Nous utilisons les mots *shim* et *convertisseur* indifféremment.

Plusieurs solutions différentes existent pour gérer les incompatibilités des données entre services. Certaines utilisent des convertisseurs existants ou génèrent de convertisseurs, ce que nous allons voir dans les paragraphes suivants. D'autres solutions permettent une correspondance entre services, ce que nous allons voir par la suite en 2.2.1.

Utilisation de convertisseurs manuels. Des registres de convertisseurs sont maintenus dans les plateformes pour les besoins de la conversion de données. Les convertisseurs sont développés à la main par des experts pour répondre à l'incompatibilité générée par les formats de données souvent textuels. Les convertisseurs sont placés entre deux services ou directement injectés dans le code des services. Cette solution a l'avantage d'offrir une utilisation en lignes de commandes (appréciée de certains utilisateurs), une gestion transparente des erreurs ainsi qu'une exécution simplifiée dans un environnent restreint. Cependant, elle atteint rapidement ses limites, notamment dans un environnement ouvert. Il devient pénible de gérer les convertisseurs lorsque les formats augmentent et changent, ce qui est souvent le cas en bio-informatique [RLS+06].

Pour faciliter l'utilisation des convertisseurs manuels, des travaux se basent sur la description et la classification des ressources [HSL+04]. Par exemple, le projet myGrid [WAH+07] fournit une classification partielle de types de *shims* en fonction des paramètres d'entrée et sortie et des opérations de transformation. Les types de *shims* concernent des transformations sur la syntaxe, sur la structure et sur la sémantique des données. Les transformations sur la syntaxe concernent le passage d'une représentation concrète X à une représentation concrète alternative Y (ex., passage d'un format à un autre), les transformations sur la structure concernent les modifications sur la représentation interne d'une donnée (ex., réorganisation, extraction) et les transformations sur la sémantique concernent les changements sur la valeur d'une donnée (ex., transformation d'une séquence de nucléotides en une séquence de protéines). Grâce à la classification, il est possible de stocker et de retrouver des *shims*. On peut utiliser des solutions telles que Feta [LAWG05] permettent de faciliter la recherche à la main. Feta est une architecture permettant la découverte de services web sémantiques. Elle fournit un modèle de données pour décrire les services. Le modèle est une version simplifiée de OWL-S qui utilise une sous-partie de *Service Profile*. Feta permet la découverte et le filtrage des services, y compris des *shims*. Elle est intégrée à des plateformes de composition de services, par exemple Taverna. Cependant, pour utiliser des solutions telles que Feta, il faut avoir créé et annoté les *shims*. Il faut également que l'utilisateur, manuellement, décrive ses besoins.

Il existe des solutions permettant une recherche automatique. Elles détectent les incompatibilités entre services et recherchent automatiquement les *shims* adéquats. Elles se basent sur la description des types de données, formats et services. L'approche de Valasco-Elizondo et al. [EDG+13] est l'une d'elles. Cette approche utilise la description des formats pour retrouver des convertisseurs permettant d'aligner deux services. Les descriptions sont fournies dans un langage nommés *ADL*. Ce langage permet de définir les formats légaux, des informations de structure sur les formats ainsi que les relations entre services et *shims*. Valasco-Elizondo et al. considèrent également l'aspect QoS dans la sélection des *shims* et des services en général.

Bien que les approches de découverte de *shims* facilitent la médiation de services par la recherche et la sélection de *shims*, elles reposent sur l'existence préalable des *shims*. L'implémentation des *shims* ainsi que les descriptions doivent être fournies à l'avance pour que ces approches s'appliquent.

Utilisation de conversions automatiques. Les approches proposant de gérer automatiquement les convertisseurs des données entre les services sont une alternative aux approches présentées dans le paragraphe précédent. Parmi elles,

il y a des approches optant pour une médiation de données totalement automatique et transparente à l'utilisateur. Elles implique un système qui se charge de faire les correspondances et d'appliquer automatiquement les transformations adéquates [LLF+09]. Le travail de Kashlev et al. [KLC13] va dans ce sens. Il réduit le problème des *shims* à un "runtime coercion problem" où les *shims* sont insérés automatiquement entre les services. Leur approche utilise des types de données XML pour représenter les entrées et sorties des services. Les types peuvent être primitifs, par exemple *integer*, *float* et *string*, ou des types structurés tuples. L'approche définit des règles de sous typage entre deux tuples et entre les types primitifs. Elle définit également des règles de réflexivité et de transitivité de la relation de sous typage. Elle proposent, en plus, un ensemble de règles définissant un workflow bien formé. A partir de ces éléments, le système de Kashlev et al. assure la correspondance des données ; il insère des *shims* ou coercitions invisibles rendant les workflows bien formés exécutables. Cependant, le système de Kashlev et al. ne permet pas une médiation de données complexes. Les types restent assez simples pour éviter les cas de conversions multiples qui sont difficiles à gérer de manière automatique. L'un des problèmes de la médiation implicite est lié à la spécification des besoins. Il est difficile de fournir une spécification claire et complète pour un système totalement automatique. Il est également difficile de garantir des conversions implicites et complexes sans ambiguïté dans certaines situations, par exemple lorsqu'une donnée contient plusieurs éléments d'un même type.

Il existe des approches proposant de générer des composants *shims*. Par rapport aux approches précédentes, ces approches fournissent des convertisseurs visibles et accessibles. Bien que les *shims* soient visibles, leur production reste en général totalement automatique. Un avantage de ces approches est que les conversions à effectuer peuvent faire intervenir un utilisateur. Ces approches peuvent ainsi bénéficier des connaissances de l'utilisateur final, par exemple pour gérer une ambiguïté. Plusieurs travaux adoptent cette approche. Par exemple, Browers et al [BL04] se sont basés sur des ontologies pour générer des convertisseurs. Ils considèrent des types sémantiques et des types structurels. Les types sémantiques sont définis par des concepts d'ontologies. Les types structurels associent les types sémantiques par des chemins contextuels. Dans leur approche, deux types sémantiques sont compatibles lorsqu'ils entretiennent une relation d'héritage à travers une ontologie. Deux types structurels sont compatibles lorsque leurs chemins contextuels utilisent le même contexte, et que les types sémantiques associés sont compatibles. Lorsque deux types sont compatibles, les conversions de données sont générées à l'aide des chemins contextuels sous forme de requêtes utilisant l'approche XPath. Mohan et al. [MLK14] proposent la génération de *shims* dans le contexte du Big Data. Leur approche manipule des types XML et

elle définit un algorithme produisant automatiquement des *shims* à l'aide de deux adaptateurs *pCombiner* et *pSplitter*. L'adaptateur *pCombiner* prend en entrée un ensemble de types quelconques et produit un type composite. L'adaptateur *pSplitter* prend en entrée un type et produit en sortie les composantes du type. Leur système considère des types primitifs, des représentations sous forme de tables, des collections clés et valeurs et des fichiers. Cependant, comme mentionné dans leur article, leur approche n'existe qu'avec les services utilisant des types simples, elle ne prend pas en compte les types complexes (e.g., arbre XML).

Correspondance de services

Certaines approches proposent la correspondance entre services [PKPS02]. Ce sont des approches qui ne s'intéressent pas directement à la conversion de données, elles se contentent de vérifier la correspondance entre des services. Elles sont utilisées pour la composition et pour la sélection de services [ECH10]. On distingue la correspondance au niveau sémantique et la correspondance au niveau structurel. La correspondance au niveau sémantique se base en général sur des annotations (avec des concepts) sémantiques. Elle intègre une correspondance directe et une correspondance indirecte. La correspondance directe ou exacte cherche à déterminer une égalité parfaite entre concepts. La correspondance indirecte fait intervenir des relations ontologiques telles que la relation d'héritage pour comparer deux concepts. La correspondance au niveau sémantique peut utiliser des techniques de calcul de similarité sur le lexique des mots. La correspondance au niveau structurel compare deux structures ou schémas de données. Le résultat des correspondances est booléen ou fourni sous forme de degrés de similarité [SW05]. Lebreton et al. [LBC+10] évaluent la compatibilité d'une sortie d'un service par rapport à l'entrée d'un autre service par la correspondance sémantique directe et indirecte (égalité, spécialisation ou généralisation de concepts). Stroulia et al. [SW05] proposent un algorithme calculant la similarité entre deux documents WSDL. L'algorithme donne un score de similarité. Il est utilisé pour la recherche de services. Les médiateurs définis avec le formalisme WSMO sont utilisés pour gérer les incompatibilité des services. Ces médiateurs sont de deux sortes et s'attaquent aux problèmes d'alignement d'ontologies, de protocoles ou de buts. Le médiateur qui s'attaque principalement à l'alignement des données des services est nommée *WW-mediators*. Il permet de définir des relations (impliquant la médiation) entre des services. La médiation est exprimée sous forme d'expressions WSML (Web Service Modeling Language) qui fournit la syntaxe et la sémantique formelles de WSMO. La médiation concerne le niveau sémantique et syntaxique mais ne génère pas de convertisseurs [PSS04].

```
rule rule_example:
      input: 'source/{dataset}.csv'
      output: 'target/{dataset}.pdf'
      shell: 'somecommand {input} {output}'
```

Figure 2.2 – Exemple d'une règle dans Snakemake

2.2.2 Techniques de composition de workflows

A cause de l'hétérogénéité des ressources, il est difficile de composer des workflows. Il est difficile de retrouver les services adéquats à composer. Lorsque l'on arrive à trouver les services adéquats, il est aussi difficile de garantir la compatibilité entre leurs entrées et sorties [OGA+06b]. Lors de la composition l'utilisateur est obligé de gérer des tâches manuelles complexes : accéder à des registres, rechercher des services, sélectionner des services, interconnecter des services, insérer des *shims*. Pour faciliter les tâches lors de la composition, plusieurs solutions sont proposées. Dans cette partie, nous présentons des solutions de composition de workflows. Nous verrons des outils de bas niveau et des systèmes et approches de composition automatisés.

Systèmes utilisant des composants textuels

Il existe des systèmes proposant d'utiliser des langages textuels. Ces langages sont simplifiés et concis pour définir des workflows. Ils suivent l'esprit de langages connus, tels que Python ou les scripts shell. L'environnement de travail des systèmes utilisant des composants textuels est relativement simplifié et dépend de peu de technologies pour fonctionner.

Par exemple, Snakemake [KR12] permet de créer des workflows en utilisant un langage simplifié proche de Python. Un workflow est constitué de règles où chaque règle définit comment obtenir certaines sorties à partir de certaines entrées. Une règle contient un nom, des fichiers d'entrée, des fichiers de sortie et une commande (commande shell ou code Python). Les dépendances, dans le workflow, sont établies implicitement entre les entrées et sorties définies par des règles qui se suivent. Un chemin formé par des dépendances définit une exécution en série et les chemins disjoints peuvent être exécutés indépendamment. La figure 2.2 montre comment une règle est définie à travers une entrée, une sortie et une commande à lancer.

Snakemake peut prendre en charge tout service dont les formats ou fichiers d'entrée et sortie sont bien définis. Il permet de visualiser un workflow construit sous forme d'un graphe à l'aide de l'outil de visualisation GraphViz [EGK+02].

```
tache = {
exec "la_commande_a_executer"
}

Bpipe.run {
 tache1 + tache2
}
```

Figure 2.3 – Exemple de définition d'une tâche et d'une structure de construction dans Bpipe

Tout comme Snakemake, Bpipe [SPO12] propose un langage permettant de décrire des workflows. La création de workflows respecte deux étapes. Une première étape définit une tâche spécifique à exécuter à l'aide d'une commande. Une deuxième étape de structuration combine les tâches définies à l'aide d'opérateurs. Les opérateurs indiquent des constructions telles qu'une exécution séquentielle, en parallèle ou sous forme de batchs sur les tâches définies. Un exemple de tâches et d'une construction sous forme d'une séquence (avec l'opérateur +) de deux tâches est présenté à la figure 2.3.

Un inconvénient des systèmes que nous venons de décrire est que la construction reste encore très manuelle. Leurs langages sont destinés à des utilisateurs initiés. Les systèmes n'aident pas assez l'utilisateur. Par exemple, la médiation de services est manuelle : il faut connaitre en détails les services et les données pour composer un workflow. De plus, ils ne gèrent pas assez bien certains aspects : modèles de workflows ré-utilisables, historiques des données, navigation à travers les ressources ou le suivi d'exécution.

Plateformes de gestion de workflows

Des systèmes de gestion de workflows sont proposés comme une alternative aux solutions manuelles. Ils proposent un environnement opérationnel permettant non seulement de composer des workflows mais également de gérer leur cycle de vie. Ils permettent la construction, l'exécution, le partage et la réutilisation de workflows. Ils permettent également différentes opérations utiles telles que l'annotation, la description, la recherche, la navigation, le filtrage, l'historique, le suivi d'exécution et la gestion de logs. Ces systèmes sont aussi en général accompagnés d'environnements graphiques adaptés à des utilisateurs experts de domaine [WMK+08, SBH06].

Par exemple, Galaxy [GNTT10, GRH+05] permet l'analyse par l'intégration d'outils en génomique à travers une interface web. Il assure l'accessibilité, la reproduction et le partage des données, des outils et des expériences d'analyse de données. Galaxy offre une large palette de services édités sous divers langages de programmation tels que Python, Perl, R, Shell. Les services dans Galaxy regroupent des fonctions d'accès à des jeux de données, d'analyse et de visualisation. Ils sont spécifiés à travers des fichiers de configuration. Les fichiers encapsulent les détails d'invocation et les paramètres d'entrée et sortie et les exposent sous forme de formulaires. Dans Galaxy, une analyse (un workflow) se présente sous forme d'une liste qui définit l'application, étape par étape, de traitements à des données chargées ou produites par des traitements intermédiaires. Elle est définie en accédant à des jeux de données provenant de plateformes telles que UCSC et en appliquant des traitements offerts par des services disponibles dans le registre de Galaxy. Les résultats des traitements peuvent être visualisés sous différentes formes à l'aide des services de visualisation. Un workflow formé peut être représenté, à l'aide de composants graphiques, sous forme d'un modèle de workflows. Le modèle permet d'appliquer un workflow à des données différentes. Il est éditable pour, par exemple, ajouter des composants ou considérer de nouvelles étapes. Galaxy offre également une vérification des liaisons entre services à travers les formats d'entrées et sortie des services. L'interactivité de Galaxy se fait à travers des pages web. Les pages permettant la création d'expériences, l'inspection des détails concernant les données et les étapes d'un workflow ainsi que la modification, la reproduction et le partage des ressources. Pour assurer une reproductibilité des analyses, Galaxy gère des méta-informations sur les jeux de données, les outils et les paramètres utilisés. Il maintient des historiques et permet des annotations et des descriptions des différentes étapes d'une analyse. Galaxy propose également un modèle de partage, des répertoires publics et des solutions pour le partage et la vulgarisation des expériences. Il permet ainsi le partage des contenus (par exemple des jeux de données, des historiques ou des workflows) à l'aide de liens et de pages web. Galaxy offre également les moyens de rechercher, de filtrer et de trier les contenus des répertoires en utilisant les informations telles que les noms, auteurs et annotations. Des solutions sont également proposées pour des suggestions durant la construction de workflows dans Galaxy [DCZ+12]. L'application Galaxy est téléchargeable et adaptable. Par exemple, la plateforme bio-informatique GenOuest [30] intègre une instance de Galaxy [31] où des outils sont ajoutés pour traiter des données de protéomique, phylogénie, génétique des populations et génétique quantitative [LBRM+13].

30. www.genouest.org
31. http ://galaxy.genouest.org

Mobyle [NMM+09] est une plateforme qui offre des fonctions semblables à celles de Galaxy. C'est une portail web pour l'analyse de données en bio-informatique. Elle fournit un accès à des outils en lignes de commandes via une interface web. Mobyle se définit comme un système centré utilisateur facilitant la gestion des ressources et comme une architecture distribuée permettant d'utiliser des services en local et à distance. Mobyle intègre des mécanismes pour la description, la publication et l'exécution des outils. Elle intègre également une suite MobyleNet [MGN+10] dont le rôle est de créer un réseau de plateformes. MobyleNet est ainsi une fédération de portails web qui utilisent la solution Mobyle. Chaque portail maintient des outils qui peuvent être accessibles aux autres membres de la communauté. Dans Mobyle, la description des services est fournie sous forme XML. Ces descriptions donnent les détails d'accès et d'utilisation des tâches des services et des interfaces, sous forme de formulaires, d'interaction avec les services. Les services sont constitués de programmes, de workflows et de *viewers*. Les programmes sont appelés à l'aide de commandes shell, exécutables coté serveur et fournissent des résultats stockés sous forme de fichiers. Les workflows sont des interconnexions de plusieurs tâches de différents services et les *viewers* sont des outils de visualisation. La description et la classification des ressources dans Mobyle permettent de supporter la recherche et le filtrage des services. Mobyle fournit également des mécanismes de suggestions d'interconnexion de services et de recharge de données de l'historique. Ces mécanismes sont basés sur l'évaluation de la compatibilité entre des types abstraits (*data types* et *biotype*) et entre des formats. La compatibilité est établie lorsqu'une conversion automatique est possible au niveau des formats.

Taverna [HWS+06] est un outil du projet myGrid destiné à la composition et à l'exécution de workflows en sciences de la vie. Il offre un environnement permettant d'écrire des workflows à l'aide de composants graphiques. Les workflows sont représentés à l'aide du langage Scufl présenté à la section 2.2. L'environnement de Taverna fournit un répertoire permettant de contenir des services. Ces services proviennent de différentes sources en local ou à distance. Taverna travaille avec plusieurs types de services dont des services SOAP, des services RESTful, des services de grilles, des services de cloud, des scripts R, des scripts en local et des scripts en lignes de commandes. Taverna est en relation avec myExperiment [GBA+10] qui permet de partager et de ré-utiliser des workflows ainsi qu'avec les registres de services BioCalalogue et Biodiversity. Les workflows de Taverna peuvent être exécutés dans des infrastructures distantes (clusters, grids, clouds) ou utilisés comme services dans des systèmes tels que Galaxy. Taverna offre également un serveur permettant d'exécuter des workflows à partir de clients web ou clients tiers [WHF+13]. Il utilise le modèle Feta [LAWG05] pour la recherche de services. Dans Taverna, la médiation des services est gérée à l'aide

de *shims*. Taverna fournit également des mécanismes d'interaction permettant de sélectionner des données ou de paramétrer les étapes durant l'exécution des workflows [LO08].

Kepler [ABJ+04, WCA09] fournit une interface graphique pour créer des workflows dans divers domaines (par exemple en écologie, biologie, géologie, astrophysique et chimie). Comme dans Taverna, les composants, qui sont des services et des liens, sont ajoutés, étape par étape, pour former un workflow. Les workflows peuvent être capturés dans un format XML pouvant être archivé ou partagé. Kepler intègre une suite de services spéciaux pour gérer la compatibilité entre les services dont les données sont sémantiquement compatibles (même nature) et syntaxiquement incompatibles (formats différents). Une interface graphique est proposée dans Kepler pour prendre en charge *MapReduce* [DG08]. *MapReduce* est un modèle de programmation pour traiter des données du *Big Data*. Il fournit une interface permettant une parallélisation et une distribution des calculs à grande échelle. *MapReduce* utilise deux fonctions *Map* et *Reduce*. *Map* travaille sur une portion des données initiales et produit des données intermédiaires sous la forme clé-valeur. *Reduce* utilise et combine les résultats de *Map*. *MapReduce* partitionne ainsi les données d'entrée, les distribue et exécute en parallèle les programmes utilisateurs sur les blocs de données partitionnés. Plusieurs implémentations de *MapReduce* existent dont *Hadoop* qui fournit un système d'exécution et un système de fichiers distribué [DG08]. Kepler intègre *Hadoop* ; il permet de considérer les deux fonctions *Map* et *Reduce* dans son système à travers un composant nommé *MapReduce*. Lorsque le composant *MapReduce* est ajouté, les deux fonctions sont séparées et chacune représente un sous-workflow dont les entrées et sorties sont positionnées par défaut. L'utilisateur complète la description des sous-workflows en connectant des services spécifiques.

Plusieurs autres plateformes similaires aux systèmes présentés existent : PISE [Gil02], GenePattern [RLG+06], BioSide [HPRB04]. Leur principal avantage est de permettre la conduite d'expériences dans un environnement réel avec des données et des infrastructures adaptées [LBRM+13]. Cependant, les plateformes manquent de mécanismes de guidage durant la création de workflows. Par exemple, la médiation des services repose en général sur des *shims* manuels. Il n'y a pas de mécanismes adaptés de vérification de la cohérence des workflows : des erreurs et des incohérences arrivent facilement durant les étapes de construction de workflows. Les suggestions de liaisons entre services sont limitées, elles sont en général basées sur la correspondance de formats de données ou de types abstraits.

Approches de composition semi-automatique

Plusieurs travaux portent sur la composition de workflows avec des techniques automatisées [SHP03]. Il existe des approches totalement automatiques permettant de générer des compositions à partir de spécifications [BDGMC06, PH06]. Par exemple, certaines approches transforment des descriptions OWL-S en une notation Prolog [MS02] ou Petri-net [NM02] et génèrent automatiquement des plans. Nous ne nous intéressons pas directement aux approches totalement automatiques mais plutôt aux techniques semi-automatiques permettant l'assistance durant la composition de workflows [WGMK10, WGP+11]. Nous restons dans un contexte où les utilisateurs sont assistés dans la sélection et l'interconnexion de services sous forme de workflows.

Les techniques semi-automatiques reposent sur des descriptions définies sur les composants des workflows tels que les services, les entrées et sorties, les fonctions [Gil07, AHS04] ou utilisent des informations générées lors de l'exécution des workflows, par exemple des informations concernant des logs d'exécution, des historiques et des données de provenance [ZHvdA11, Bos08, CBFC12].

Kim et al. [KSG04] proposent une assistance pour la construction de workflows. Ce travail offre des mécanismes de suggestion et de vérification permettant à des utilisateurs de composer des workflows. Il utilise une base de connaissances et un algorithme de détection et de réparation d'erreurs. La base de connaissances décrit les composants utilisés dans les workflows à l'aide d'une ontologie de composants et d'une ontologie de domaine. L'ontologie de composants fournit une hiérarchie des composants, par exemple une hiérarchie entre les services, ainsi qu'une description de composants abstraits (e.g., tâches abstraites) et de composants spécifiques exécutables (e.g., tâches concrètes). L'ontologie de domaine fournit les types de données pour représenter les paramètres d'entrée et de sortie des tâches. Elle définit également des relations entre les types de données et les tâches ainsi que les correspondances entre tâches abstraites et tâches concrètes. L'algorithme, en utilisant la base de connaissances, permet d'analyser des workflows partiels dans le but de détecter les erreurs et de suggérer des actions pour réparer les erreurs détectées [KGS04]. Il utilise les techniques d'un framework d'IA [Wel99] : les composants de workflows sont vus comme des étapes d'un plan, les entrées comme des pré-conditions, les sorties comme des effets et les relations entre composants comme des liaisons causales. Les données d'entrée fournies par les utilisateurs sont vues comme des états initiaux et les résultats représentent les buts du problème de planning. Cette représentation permet de générer systématiquement des plans corrects. Sur cette base, le système de Kim et al. suit une approche interactive où les actions de l'utilisateur (par exemple

ajouter des composants, établir des liens ou spécifier des composants) sont utilisées pour raffiner les plans générés. La construction est ainsi effectuée à travers des actions primitives de l'utilisateur, accompagnées d'une détection d'erreurs et d'une suggestion d'actions de réparation. Le système assure qu'un workflow en construction soit bien formé et exécutable en considérant un ensemble de propriétés désirées. Les propriétés sont *Tasked* (le workflow contient un ou plusieurs résultats finaux), *Satisfied* (toutes les entrées sont satisfaites), *Grounded* (toutes les tâches sont exécutables), *Justified* (au moins une sortie de chaque tâche est liée), *Consistent* (chaque lien est une connexion d'une sortie vers une entrée) et *Unique* (chaque lien est unique). Chaque action est décrite en fonction des erreurs qu'elle introduit et des propriétés qu'elle respecte. Les erreurs correspondent aux propriétés insatisfaites dans un workflow. Par exemple, si un composant est ajouté à un workflow, des erreurs possibles sont le fait de ne pas vérifier les propriétés *Satisfied*, *Grounded* ou *Justified*. Les suggestions de réparation sont de nouvelles actions, par exemple ajouter un nouveau lien pour vérifier la propriété *Satisfied*. L'approche offre une stratégie de construction où les composants sont au départ abstraits et ensuite instanciés. Elle offre également une stratégie permettant une construction partant de données spécifiques pour atteindre des résultats désirés.

Dans une logique similaire, DiBernado et al. [DPW08] offrent une approche interactive de construction de workflows. Leur solution repose sur un modèle d'interaction et un algorithme de sélection de services. Le modèle d'interaction est construit sur un ensemble d'opérations. L'opération de base consiste en la consommation d'une donnée à travers des paramètres ouverts de service. Un paramètre est ouvert s'il n'est pas alimenté. Il est possible d'effectuer des opérations de composition et de décomposition de types. DiBernado et al. permettent de considérer les sous composants d'une donnée pour alimenter un service. D'autres opérations sont également possibles, par exemple masquer ou exposer des paramètres ouverts. L'approche DiBernado et al. permet une construction en avant à partir de sorties ou une construction en arrière à partir d'entrées de services. A partir d'une donnée initiale et d'une donnée finale (donnée à consommer et donnée à produire), le système de Dibernado et al. suggère les services compatibles à ajouter à un workflow courant. L'algorithme de suggestion calcule l'ensemble des services potentiels capables de consommer la donnée initiale. A partir de cet ensemble, il déduit la liste des services qui forment une chaine jusqu'à la donnée finale. Il ordonne ensuite la liste des services en fonction de la longueur des chaines. Dans leur approche, les services sont associés à des types de données abstraits. La médiation de services repose sur la correspondance entre deux types (d'entrée et de sortie). La correspondance considère les trois cas suivants : correspondance exacte, correspondance par héritage et correspondance par décomposition entre deux types. Leur approche, contrairement à beaucoup d'approches,

considère ainsi la composition des données. Elle est utilisée sur les services de la plateforme BioMoby.

Le *workflow mining* concerne l'extraction d'information sur les workflows en analysant des informations telles que les logs. Son but est de permettre l'analyse de workflows en se basant sur des informations collectées durant l'exécution. Le travail de Aalst et al. [VdAWM04] revient sur des problèmes et des approches de *workflow mining*. Il présente un format XML utilisé pour stocker et échanger des logs. Le format est indépendant des outils, il est utilisé comme entrée dans plusieurs outils [Sch02, vdAvD02]. Les informations recueillies après une analyse sont utilisées dans des travaux pour la composition de workflows. Par exemple, Zeng et al. [ZHvdA11] proposent d'utiliser la provenance pour faciliter la construction de workflows. Dans leur approche, la provenance consiste en des traces générées durant l'exécution des workflows concernant l'invocation des tâches et l'utilisation des données. A partir des traces, Zeng et al. construisent deux matrices. La première matrice représente le nombre de fois où les tâches se suivent directement (deux à deux) et la deuxième matrice représente par une valeur numérique deux tâches se suivant indirectement (avec des tâches entre elles). Une matrice de causalité, qui représente par un score les liens entre deux tâches, est calculée à partir des matrices précédentes. Cette matrice est ensuite utilisée pour calculer des recommandations de tâches sur des workflows partiels. Chaque recommandation est associée à un score de confiance.

2.3 Conclusion

Dans ce chapitre, nous avons exposé différents travaux concernant la composition de services sous forme de workflows. Dans une première partie, nous avons présenté des solutions concernant les services et le contexte en bio-informatique : caractéristiques des services, ontologies de domaine, formats de données et registres de services en bio-informatique. Dans une deuxième partie, nous avons présenté des travaux sur la composition de workflows. Deux aspects nous ont particulièrement intéressé : la médiation de services qui concerne la gestion de l'incompatibilité des données entre services et la composition de services qui concerne la construction assistée de workflows.

Les travaux présentés concentrent trois groupes de solutions. Un premier groupe fait office de ressources. Ce sont les solutions pour la description de services, workflows, données et les registres de services en bio-informatique. Comme nous le verrons, certaines de ces ressources sont utilisées par nos contributions, par exemple les services du registre EMBOSS et les données en génomique. Un

deuxième groupe correspond à des plateformes pouvant intégrer de nouvelles solutions concernant les services et les workflows. Il s'agit des plateformes telles que GenOuest, Galaxy et Taverna. Ces plateformes offrent un environnement opérationnel pour définir, utiliser et gérer des services et des workflows. Elles peuvent accueillir les approches présentées dans cette thèse. Nous avons collaboré avec la plateforme GenOuest. Le dernier groupe correspond aux approches qui vont dans le même sens que les travaux de cette thèse. Il s'agit des approches pour faciliter la médiation de services et la composition de workflows. Comme nous le verrons, nos travaux se comparent directement à ces approches.

A la lumière des éléments exposés, nous considérons que la gestion des incompatibilités des services est une étape fondamentale pour faciliter la composition de workflows, notamment en bio-informatique. La première proposition de cette thèse s'attaque à ce problème. Cette proposition est ensuite utilisé pour définir une nouvelle approche pour composer des workflows. L'environnement bio-informatique offre les ressources permettant d'appliquer et d'évaluer nos propositions. Il permet également d'ouvrir des perspectives intéressantes pour les travaux de cette thèse, par exemple l'utilisation de nos approches à travers des réseaux et des plateformes d'utilisateurs finaux. Le chapitre suivant présente les outils et méthodes sur lesquels reposent nos propositions.

Chapitre 3

Outils et Méthodes

Ce chapitre comporte deux parties indépendantes. La première partie (Section 3.1) présente des notions de typage et de système de types. L'objectif de cette partie n'est pas de s'étendre sur la théorie des types (détaillée par exemple par Pierce dans [Pie02]) mais de poser les éléments utilisés dans nos approches. Nos travaux utilisent la notion de typage à travers les solutions XML. La deuxième partie (Section 3.2) présente les *Systèmes d'Information Logiques (LIS)* et place notre travail dans le contexte de l'équipe LIS[1] où se déroule cette thèse. Nous utilisons l'approche des LIS dans la deuxième contribution de cette thèse.

3.1 Typage

Un type représente un ensemble d'objets partageant les mêmes propriétés. En programmation, les types décrivent, organisent et classent les expressions des programmes en fonction des caractéristiques pertinentes pour leur usage. Les types apportent de la sémantique aux expressions permettant la vérification mais également le partage et la réutilisation des programmes. Dans le contexte de cette thèse, les programmes sont des services et des workflows, et ce qui nous intéresse ce sont les informations sur les données qu'ils manipulent.

Le typage consiste à associer un type à des expressions et à des données. On distingue le typage dynamique et le typage statique. Dans le cas du typage dynamique, les types des expressions sont déterminés à l'exécution. Dans le cas du

1. http ://www.irisa.fr/LIS

typage statique, les types des expressions sont déterminés par un compilateur avant l'exécution. Lorsque des types sont déterminés par raisonnement, on parle de typage implicite qui est différent du typage explicite avec lequel un utilisateur indique le type des expressions. Dans notre cas, on considère un typage statique, et les types sont explicitement fournis. Dans la suite, nous verrons le typage à travers la notion de système de types ainsi que les solutions de typage proposées à travers *XML*. Nous exploitons ces solutions pour définir et utiliser des types dans nos contributions (cf. chapitre 4 et 5).

3.1.1 Système de types

Un système de types définit la façon dont des types sont exprimés, utilisés et gérés dans un contexte donné. Il fournit les mécanismes permettant de définir des types et les règles de déduction définissant des aspects tels que l'équivalence, la compatibilité et l'inférence de types. En général, les règles d'équivalence définissent les conditions pour que deux types soient les mêmes. Les règles de compatibilité déterminent les conditions pour que les valeurs d'un type soient utilisables là où un autre type est attendu. Les règles d'inférence définissent le type d'une expression en se basant sur les types de ses constituants et éventuellement de son contexte. Les mêmes règles peuvent servir à la fois à la vérification et à l'inférence.

La définition de types peut se faire selon une vision par construction (d'autres visions existent, par exemple par dénotation ou par abstraction, voir dans [Sco00]). A travers cette vision, un type est soit un type primitif, soit un type composite. Un type primitif est un type de base prédéfini et intégré au système, par exemple les types *int*, *boolean*, *float* dans les langages de programmation. Un type composite est un type défini à travers d'autres types en utilisant des constructeurs de types, par exemple *tuple*, *list*. Nous présentons, dans la suite, des exemples de définitions de types. Nous revenons sur d'autres définitions à travers les solutions *XML*.

Types de base : un type peut être l'un des types primitifs de base, $T := integer, boolean, float, ...$. Ces types dénotent des valeurs (instances) primitives, par exemple $1, false, 1.234E2$.

Constructeur de types *tuple* **:** un type *tuple* peut être défini comme suit $T := tuple(t_1T_1\ ,...,\ t_nT_n)$ où $T_1\ ,...,\ T_n$ sont des types, et $t_1\ ,...,\ t_n$ des identificateurs. Une instance du type *tuple* est un tuple d'instances des types T_i. Dans le tuple, tous les éléments sont identifiés et présents.

Constructeur de types *union* **:** Un type *union* peut être défini comme suit $T := union(t_1T_1\ ,...,\ t_nT_n)$ où $T_1\ ,...,\ T_n$ sont des types, et $t_1\ ,...,\ t_n$ des

identificateurs. Une instance du type *union* est une instance d'un des types T_i. Dans l'union, un seul élément est présent à la fois.

Constructeur de types *list* **:** Un type *list* peut être défini comme suit $T := List(T_1)$ où T_1 est un type. Une instance du type *list* est une liste d'instances du même type T_i.

Constructeur de types *function* **:** Un type *function* peut être défini comme suit $T := function(T_1, T_2)$ où T_1 et T_2 sont des types. Une instance du type *function* est une fonction dont les entrées sont du type T_1 et les sorties du type T_2.

Une expression e est dite de type T, ce qu'on note $e : T$, si elle respecte les contraintes posées par le type T. On dit également qu'elle correspond au type.

Dans un système de types, une règle de déduction formelle peut se présenter sous la forme suivante :

$$\frac{hypothesis_1 \ ... \ hyphothesis_n}{conclusion}$$

Il s'agit d'une règle formée d'un ensemble d'hypothèses (ou prémisses) et d'une conclusion. La conclusion est vérifiée si toutes les hypothèses sont vérifiées. Une règle peut ne pas admettre d'hypothèses, dans ce cas, la conclusion est un axiome, une vérité admise. A partir d'un ensemble de règles, on forme un système de déduction dans lequel une preuve formelle est un arbre dont les noeuds sont des instances de règles de déduction, les feuilles des instances d'axiomes. Nous utiliserons des règles et un système de déduction dans les travaux de cette thèse.

Voici l'exemple d'une règle qui définit un typage :

$$\frac{e_1 : function(T_1, T_2) \quad e_2 : T_1}{(e_1 \ e_2) : T_2}$$

Les hypothèses de cette règle sont $e_1 : function(T_1, T_2)$ et $e_2 : T_1$, et la conclusion est $(e_1 \ e_2) : T_2$. La règle dit que si e_1 est une fonction du type $function(T_1, T_2)$, et que e_2 est du type T_1, alors l'application e_1 sur e_2, notée $(e_1 \ e_2)$, est du type T_2.

Les systèmes de types concernent plusieurs travaux, non seulement la notion de typage, mais également des aspects plus ou moins proches, par exemple la génération de preuves [BC85], la recherche dans des librairies de fonctions [Rit91] ou la négociation de l'échange de données [Mot02]. Dans notre cas, nous nous basons sur un système de types pour la correspondance et la conversion de données entre des services dans un workflow.

Data oriented	Document oriented
<invoice> <orderDate>1999-01-21</orderDate> <shipDate>1999-01-25</shipDate> <billingAddress> <name>Ashok Malhotra</name> <street>123 Microsoft Ave.</street> <city>Hawthorne</city> <state>NY</state> <zip>10532-0000</zip> </billingAddress> <voice>555-1234</voice> <fax>555-4321</fax> </invoice>	<memo importance='high' date='1999-03-23'> <from>Paul V. Biron</from> <to>Ashok Malhotra</to> <subject>Latest draft</subject> <body> We need to discuss the latest draft <emph>immediately</emph>. Either email me at <email> mailto:paul.v.biron@kp.org</email> or call <phone>555-9876</phone> </body> </memo>

FIGURE 3.1 – Deux types de documents XML [BMC+12].

3.1.2 XML et XML-Schema

Comme évoqué dans le chapitre 2, XML est un format de données structurées. Il est standardisé par le W3C [2]. Il est dérivé du langage SGML (Standard Generalized Markup Language). XML offre un langage générique et extensible permettant de structurer une grande variété de contenus. XML est accompagné du langage *XML-Schema* pour décrire de manière formelle les types des données XML ou documents XML. Cela permet des opérations de vérification et de validation basées sur les types qui intéressent cette thèse. *XML-schema* permet de se doter de modèles décrivant la structure des documents XML dans le but de garantir la cohérence et la sécurité de leur utilisation. Historiquement, DTD (Document Type Definition) est le premier moyen utilisé pour décrire la structure des documents XML. De nombreuses alternatives sont proposées par la suite, par exemple RELAX [3] et TREX [4]. *XML-Schema* se présente comme une extension de DTD. Il est stable, il est publié comme une recommandation par le W3C. Son objectif est de fournir des mécanismes permettant de contraindre la structure des documents, de prendre en compte des types de données primitifs et de garantir la conformité des documents XML. Sa solution passe par des schémas permettant de définir et de décrire une classe de documents XML. Un schéma exprime des contraintes concernant la syntaxe, la structure et la valeur que ses instances (des documents XML) doivent respecter. *XML-Schema* fournit une spécification d'un système de types. Le système de types s'inspire des solutions existantes dans les langages tels que Java et SQL, notamment concernant les types primitifs. Il intègre des types de base et donne des mécanismes (de construction) permettant

2. http ://www.w3.org/XML/
3. http ://www.xml.gr.jp/relax
4. http ://www.thaiopensource.com/trex

de définir de nouveaux types.

Dans la suite, nous présentons la spécification du système de types de *XML-Schema*. Nous présentons ensuite comment des types pour XML peuvent être représentés avec la solution *XDuce*. Les données dans XML peuvent être très variées. Elles peuvent concerner de larges documents XML pouvant contenir des informations mixtes. Par exemple, la partie droite de la figure 3.1 montre un document XML sous forme d'un email mélangeant texte et structure. Dans notre cas, il s'agit de documents XML orientés données dont la partie gauche de la figure 3.1 montre un exemple. Dans cet exemple, les informations sont bien séparées et structurées avec des balises. Comme nous le verrons, nous séparons explicitement les composants des données utilisées à travers les services.

Système de types

Dans *XML-Schema*, un type de donnée est un triplet constitué d'un espace de valeurs, d'un espace lexical et d'un ensemble de facettes. L'espace de valeurs définit les valeurs qu'un type accepte. Chaque valeur est dénotée par un ou plusieurs littéraux. Ces littéraux forment l'espace lexical du type. Par exemple, pour le type *float*, les littéraux 120.0 et $1.2E2$ dénotent la même valeur. Les facettes caractérisent l'ensemble des valeurs (et l'espace lexical) d'un type. Elles sont de deux sortes : les facettes fondamentales et les facettes non-fondamentales. Les facettes fondamentales sont des propriétés abstraites caractérisant sémantiquement l'espace de valeurs d'un type. Elles sont définies pour tous les types. Elles sont constituées de la notion d'égalité, de la relation d'ordre, de la notion de fermeture et de cardinalité ainsi que de la nature numérique ou non des valeurs d'un type. Les facettes non-fondamentales sont des contraintes optionnelles applicables à l'ensemble des valeurs d'un type. Elles sont définies selon les types. Elles comprennent des contraintes telles que le nombre d'unités de longueur des valeurs d'un type (par exemple, l'unité de longueur pour *string* est le caractère) ou des contraintes sur les valeurs maximales ou minimales d'un type. *XML-Schema* intègre des types primitifs internes et permet de définir de nouveaux types composites à partir d'autres types [BMC+12, TBM+12, FW04].

Types primitifs internes. Les types primitifs (ou types de base) sont des types élémentaires, tels que *string*, *decimal* et *date*, inspirés des langages tels que SQL. Ils forment une hiérarchie (cf. document [BMC+12]). Chaque type primitif est défini avec une URI de référence (par exemple, *http ://www.w3.org/2001/XMLSchema#int*) et une spécification. La spécification définit l'espace des valeurs (et l'espace lexical) ainsi que les facettes du type. Par exemple, le type *string* représente les chaines de caractères en XML. Ses

```
<xsd:simpleType name="myInteger">
  <xsd:restriction base="xsd:integer">
    <xsd:minInclusive value="1"/>
    <xsd:maxInclusive value="100"/>
  </xsd:restriction>
</xsd:simpleType>
```

FIGURE 3.2 – Exemple d'un type simple défini à partir de *interger*

valeurs sont dénotées par des séquences finies composées des caractères définis dans les spécifications ISO (International Organization for Standardization). Les facettes non-fondamentales pour *string* sont : *length*, *minLength*, *maxLength*, *pattern*, *enumeration* et *whiteSpace*. *Length*, *minLength* et *maxLength* donnent des contraintes concernant la taille normale, minimale et maximale d'une chaine. *Pattern* donne une contrainte qui s'applique aux valeurs d'une chaine sous forme d'expressions régulières, *enumeration* permet de spécifier explicitement un espace de valeurs et *whiteSpace* caractérise l'utilisation du caractère *espace* selon qu'il est préservé, remplacé ou ignoré [BMC+12, TBM+12].

Types composites. Les types composites sont obtenus par construction sous forme de restrictions, d'extensions ou de composition d'autres types, les types primitifs étant les ingrédients de base. Une restriction consiste à restreindre l'espace de valeur d'un type de base pour définir un nouveau type. Le nouveau type aura un sous ensemble des valeurs du type de base. La restriction est appliquée en utilisant des facettes. Une extension consiste à étendre un type par rapport à des types de base en ajoutant d'autres éléments ou attributs. La composition est une extension associant plusieurs types. Les constructions donnent des types semblables aux types présentés à la section 3.1.1. Le modèle de XML-Schema distingue ainsi la définition de types simples et la définition de types complexes [FW04].

Une définition d'un type simple se présente sous forme de restrictions sur des types de base. De manière générale, ce sont des types primitifs de *XML-Schema* ou des types dérivés (types dérivés des primitifs) ajoutés pour une utilisation externe. Par exemple, le type *myInteger* de la figure 3.2 est une restriction sur le type de base *integer*. Ses valeurs sont comprises entre 1 et 100. Les facettes *minInclusion* et *maxInclusion* sont utilisées pour appliquer la restriction. D'autres facettes telles *enumeration* et *pattern* sont utilisables dans le même contexte.

Une définition d'un type complexe est une combinaison (y compris des restrictions et extensions) de plusieurs éléments. Les éléments peuvent être des types

```
<xsd:element name="name" type="xsd:string"/>

<xsd:element ref="text"   minOccurs="1"/>
```

Figure 3.3 – Exemple d'éléments pouvant constituer un type complexe

```
<xsd:complexType name="Person" >
     <xsd:sequence>
        <xsd:element name="name" type="xsd:string"/>
        <xsd:element name="age" type="xsd:integer"/>
        <xsd:element name="address">
            <xsd:complexType>
               <xsd:sequence>
                 <xsd:element name="street" type="xsd:string"/>
                 <xsd:element name="city" type="xsd:string"/>
                 <xsd:element name="zipCode" type="xsd:string"/>
               </xsd:sequence>
            </xsd:complexType>
       </xsd:element>
    </xsd:sequence>
</xsd:complexType>
```

Figure 3.4 – Exemple d'un type complexe

simples, des types complexes, des attributs, des groupes de types ou des groupes d'attributs. Ils peuvent être déclarés comme nécessaires ou comme optionnels. Leurs instances peuvent être destinées à être présentes une fois, n fois ou un nombre illimité de fois. Les exemples de la figure 3.3 montrent des définitions d'éléments pouvant composé un type complexe. Dans le premier cas, il s'agit d'un élément qui balise le type *xsd* : *string* par *name*. Dans le second cas, l'élément fait référence à un type défini ailleurs sans annotation.

L'exemple de la figure 3.4 montre un type définissant des personnes à partir d'une séquence composée d'un nom, un âge et une adresse, l'adresse étant une autre séquence composée d'une rue, d'une ville et d'un code postal. L'ensemble de valeurs de *Person* est l'ensemble des séquences formées des valeurs des éléments constituants le type *Person*. La figure 3.5 montre une valeur du type *Person*.

```
<person>
<name> M. X </name>
<age> 29 </age>
   <address>
      <street> Paul Bert </street>
      <city> Rennes </city>
      <zipCode> 35000 </zipCode>
    </address>
</person>
```

FIGURE 3.5 – Exemple d'une valeur du type *Person*

```
person[
      name[ M. X ]
      age[ 29 ]
      address[
           street[ Paul Bert]
           city [Rennes ]
           zipCode[ 35000 ]
             ]
       ]
```

FIGURE 3.6 – Un exemple de valeur dans XDuce

Types XML dans XDuce

XDuce [HVP00] est un langage pour traiter des documents XML. Il est construit sur une algèbre de type expressions régulières proche des langages de schémas DTD, Relax-NG et XML-Schema. XDuce est basé sur la théorie des *automates d'arbres réguliers*. Il interprète un type comme un ensemble de documents XML. Comme avec *XML-Schema*, les types définis dans XDuce correspondent à des schémas de documents XML. Dans le contexte de cette thèse, XDuce offre une autre notation des types de *XML-Schema*. Nous nous intéressons à la notation de XDuce pour sa concision. Dans la suite, nous présentons la définition de types dans XDuce.

Dans XDuce, une valeur est une séquence d'arbres XML. L'ensemble des valeurs est défini comme suit :

$$V ::= d \ \mid \ l[V] \ \mid \ V\ V \ \mid \ ()$$

d est une valeur primitive (CDATA du XML). L'expression $l[v]$ dénote la séquence v (qui est une valeur) sous le label l. Il est possible de définir une concaténation de valeurs, comme suit $v_1\ v_2$. Une séquence vide correspond à (). L'exemple de la figure 3.6 représente une valeur : une personne avec son nom, son age et son adresse. Il correspond à la valeur présentée à la figure 3.5.

Dans XDuce, un type T dénote un ensemble de valeurs et des classes de labels L sont utilisées pour former des éléments XML. Les types composites sont définis comme suit :

$$T ::= P \ \mid\ () \ \mid\ T,T \ \mid\ L[T] \ \mid\ T|T \ \mid\ T*$$

Les constructions de types considérés sont *empty*, *sequence*, *label*, *union* et *repetition*. Les types primitifs P correspondent aux types de base présents dans XML. *empty* définit un type vide () qui dénote la séquence vide. Le type T_1, T_2 dénote les valeurs qui sont la concaténation de valeurs du type T_1 avec des valeurs du type T_2. $L[T]$ définit un type label, il dénote des valeurs de la forme $l[v]$ où l est un label dans L et v une valeur du type T. $T|T$ définit un type union qui représente l'union des valeurs de différents types. Par exemple, une valeur de $T_1 \mid T_2$ est une valeur de type T_1 OU une valeur de type T_2. $T*$ définit un type répétition qui dénote une sequence de valeurs d'un même type T. Les classes de labels sont définies par la grammaire suivante.

$$L ::= l \ \mid\ \sim \ \mid\ L|L \ \mid\ L \setminus L$$

l est un label spécifique, par exemple *person*. $\sim$ est un label joker qui dénote un label quelconque et représente l'ensemble des labels. $L|L$ dénote une union de classes de labels, il permet de définir un choix entre plusieurs labels. $L \setminus L$ représente une différence de classes de labels. Il dénote une différence d'un ensemble de labels sur un autre.

```
type Address = address[street[string],
               city[string], zipCode[string]]

type person = person[name[string], age[int], addr[Address]]
```

Figure 3.7 – Exemple d'un type *Person* correspondant au schéma défini à la figure 3.4.

La figure 3.7 montre des exemples de types. Le type *Person* correspond au schéma XML défini à la figure 3.4. Il s'agit d'une sequence de noeuds associée au label *person*. Les noeuds sont des séquences contenant le nom, l'age et l'adresse. Les types de base utilisés sont *string* et *int*.

```
type Family = Person*
type Person = person[nom[string], family[Family] | nom[string]]
```

FIGURE 3.8 – Exemple de types récursifs.

Les types $T?$ et $T+$ sont également considérés, ils sont dérivés des types précédents. $T?$ définit un type optionnel qui correspond à l'union du type T au type vide, $T? = T \mid ()$. $T+$ est un type répétition avec au moins un élément, $T+ = T, T*$. Il est également possible d'associer un nom à une expression de type, comme suit $type\ X = T$. Le nom X remplace l'expression T. Par exemple, dans la définition suivante $type\ Person = person[Name, Age, Address]$, le nom $Person$ est associé à l'expression $person[Name, Age, Address]$. Les noms $Name$, Age et $Address$ remplacent des expressions de types définies ailleurs.

Il est possible de définir des types récursifs. Les expressions de type de la figure 3.8 définissent des types *Person* et *Family* qui dénotent des personnes et des familles. Une famille est une liste de personnes et une personne est définie par soit nom, soit son nom et sa famille. Les valeurs des types récursifs restent finies malgré la récursivité.

Par ailleurs, XDuce propose également une relation de sous-typage ensembliste sur les types. La relation de sous-typage définie entre deux types correspond à une inclusion entre les valeurs dénotées par les types. Elle est spécifiée à travers un ensemble de règles. De manière générale dans XDuce, un type T_1 est sous-type d'un type T_2, si et seulement si, toute valeur du type T_1 est également une valeur du type T_2. Le sous-typage utilise une relation de *sous-label*. La relation de *sous-label* organise et hiérarchise les labels. Elle permet de comparer des labels définis au niveau des types. C'est une relation réflexive et transitive. Les définitions de *sous-labels* sont fournies explicitement sur un ensemble de labels sous cette forme $subtag\ label_i <: label_j$ où $label_i$ est sous-label de $label_j$. Par exemple, $subtag\ student <: person$ permet de spécifier que *student* est sous-label de *person*. Voici un exemple de règle pour le sous-typage.

$$\frac{label_1 <: label_2}{label_1[T] <: label_2[T]}$$

Selon la règle, pour tout type T, si $label_1$ est sous-label de $label_2$, alors le type $label_1[T]$ est sous-type du type $label_2[T]$.

3.2 Les systèmes d'information logiques (LIS)

Les systèmes d'information logiques, (LIS) sont un nouveau paradigme de système d'information proposé par Ferre et Ridoux [FR04]. Ils offrent des modèles formels qui permettent la navigation, l'interrogation, la mise à jour et l'analyse de collections de données hétérogènes. Les LIS sont introduits pour palier à la rigidité des organisations hiérarchiques et le manque de progressivité des systèmes à base de requêtes. Ils réconcilient la recherche avec requêtes et la recherche par navigation pour offrir l'expressivité et le guidage dans la gestion des données hétérogènes.

Les LIS sont développés dans le cadre des travaux de l'équipe LIS [5]. À la base, les LIS reposent sur l'analyse de concepts formels (FCA) [GW12] qui cherche à regrouper sous forme de concepts des objets ayant des propriétés communes et à étudier les concepts lorsqu'ils sont formellement définis. Les systèmes d'information logiques ont, par la suite, intégré des aspects provenant de la recherche à facettes [ST09], du Web sémantique [HKR09], du traitement analytique en ligne (OLAP) [CCS93] et des langues naturelles contrôlées (CNL) [Kuh13]. Les apports des LIS concernent la gestion de données hétérogènes en général. Ils permettent donc de gérer les données sur les services et les workflows. Comme nous allons le voir dans les chapitres suivants, les mécanismes d'expression de requêtes et le guidage proposés dans les LIS peuvent être adaptés à la composition de workflows. Dans la suite, nous allons présenté le framework général des LIS, nommé ACN (Abstract Conceptual Navigation) [Fer14]. Nous allons ensuite présenter des instances des LIS.

3.2.1 Le framework ACN

Le framework ACN est fourni pour subsumer les méthodes d'accès par requêtes, par navigation et par vues interactives. Il soutient l'idée de naviguer de concept en concept où chaque concept définit une place dans la navigation et est fait d'une extension (objets partageant les propriétés du concept) et d'une intention (propriétés du concept partagées par les objets). Les composants de ACN sont les suivants :

Base de connaissance : elle contient la représentation formelle des objets. Elle est constituée, par exemple, de faits, de règles, d'axiomes ontologiques ou de taxonomies. Son contenu dépend des besoins des applications et de la disponibilité des données et des connaissances du domaine d'application.

5. http ://www.irisa.fr/LIS

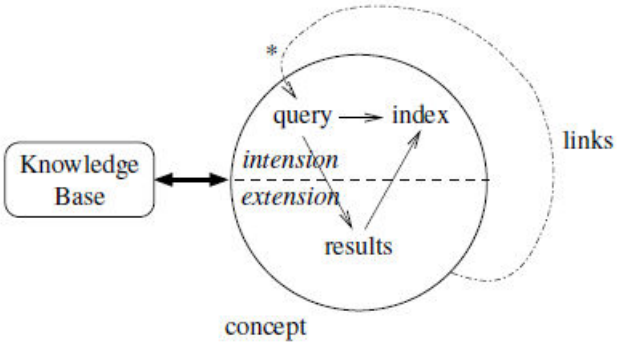

FIGURE 3.9 – Les différents composants de ACN et leurs interactions [Fer14]

Langage de requêtes : c'est un ensemble des requêtes qu'il est possible d'exprimer. Les requêtes peuvent concerner, par exemple des opérations analytiques, des mots clés, des chemins d'accès, des assertions ou des commandes. Une requête caractérise un concept, elle représente une partie des ses intentions.

Extension : elle représente l'ensemble des résultats possibles pour une requête, étant donnée une base de connaissances. Elle peut être, par exemple un ensemble de valeurs, une table ou une liste de résultats.

Index : il s'agit d'un résumé des extensions accompagné de requêtes de raffinement. Son rôle est de fournir des suggestions et un guidage.

Liens : il s'agit d'un ensemble de liens de navigation d'une requête courante vers des requêtes voisines. Les liens peuvent être dérivés de la requête, de l'extension ou de l'index d'un concept.

Dans les LIS, un utilisateur effectue une requête à travers une formule logique, et récupère une réponse dans laquelle on trouve l'extension et des informations provenant de l'index donnant des indices sur de futures requêtes à définir à travers des liens de navigation. La figure 3.9 tirée de [Fer14] illustre les interactions des composants de ACN.

3.2.2 Les instances LIS

Plusieurs instances LIS basées sur le framework ACN existent. Nous allons présenter les instances Camelis pour la navigation et l'interrogation personnelle de données, Geolis qui est appliquée à des données géographiques, Abilis qui offre un accès Web et Sewelis pour gérer des données du Web sémantique.

Camelis. Camelis [Fer09] est l'une des premières instances des LIS. Elle permet de structurer, modifier, interroger et naviguer à travers des données représentées

de manière uniforme. Les objets et les formules de Camelis peuvent être fournis de manière manuelle ou extraits automatiquement à partir de fichiers par des *parseurs*. Les fichiers peuvent être des fichiers musicaux, des images ou des références bibliographique. Camelis est assez générique pour accepter des modules logiques pour représenter de nouveaux objets ou formules logiques à intégrer dans sa base. Camelis est utilisé dans diverses applications, par exemple dans la gestion de traces d'exécution, le traitement des langues, la fouille de données.

Geolis. Geolis [BFRQ07] est proposé pour naviguer à travers des données géographiques. Les objets de Geolis intègre une notion spatiale qui consiste à décrire les objets par une forme géométrique logique. Geolis permet ainsi des relations spatiales entre les objets, par exemple les relations *est à l'interrieur*, *est traverser par*, *est à coté de*. Elle offre une navigation en traversant les relations pour passer d'un objet à un autre.

Abilis. Abilis est un serveur Web basé sur le cœur de Camelis. Il permet une notion de multi-utilisateurs à travers différents contextes. Abilis est utilisé dans le cadre de travaux collaboratifs et groupes de décision [DF08]. Une extension de Abilis est proposée. Elle intègre des aspects analyses multidimensionnelles dans le cadre de l'exploration de données géographiques [All11].

Sewelis. Sewelis [FH11] est une instance des LIS appliquée à l'exploration des données du Web sémantique. Sewelis est l'instance LIS la plus proche de cette thèse en termes des données manipulées et d'interactions offertes. Elle permet à des utilisateurs d'exprimer, étape par étape, des requêtes complexes d'interrogation et des mises à jour sur des données du Web sémantique, à travers une navigation et des suggestions. Elle est destinée à des utilisateurs finaux et offre des requêtes sûres et expressives : l'utilisateur, pour exprimer sa requête, choisit parmi plusieurs éléments suggérés. Toute sélection effectuée par l'utilisateur donne un résultat non vide. La sélection est représentée sous forme d'une requête et des transformations sont applicables pour modifier la requête. Sewelis utilise un langage de requête simple, nommé *LISQL*. Ce langage est une syntaxe de haut niveau pour le langage SPARQL. Il minimise l'utilisation des variables et facilite l'insertion des opérateurs de disjonction et de négation. Sewelis a été utilisé sur une base de données de films. Chaque film est représenté par des propriétés concernant par exemple ses auteurs, l'année de sortie, producteurs ou pays. Sewelis permet de naviguer sur ces données et de poser des questions diverses telles que *le nombre de films par auteur ?*, *les films sortis entre deux dates ?*, *les auteurs par film et par région ?*.

3.3 Conclusion

Cette partie a présenté les outils et les méthodes de notre travail. Elle a, en premier lieu, concerné les principes de typage et de système de types en général. Elle a en particulier présenté les solutions de typage dans XML et XDuce. En deuxième lieu, elle a présenté l'approche des LIS en général et des implémentations concrètes des LIS. Comme nous allons le voir dans les chapitres suivants, nos travaux utilisent les solutions présentées.

Chapitre 4

Convertibility of Service Inputs and Outputs

Résumé en français [1]

Comme nous en parlions aux chapitres précédents, l'incompatibilité des services est un problème majeur dans la composition de workflows. Face à elle, les solutions utilisant des *shims* manuels sont fastidieuses pour un utilisateur final et les solutions automatiques sont en général limitées dans les transformations de données entre services qu'elles offrent. Dans ce chapitre, nous proposons une solution pour gérer l'incompatibilité des entrées et sorties des services. L'objectif est de permettre de gérer l'incompatibilité des services durant la composition de workflows. Notre solution passe par une abstraction de types et utilise un ensemble de règles de convertibilité entre les types des entrées et sorties des services. Elle permet de générer automatiquement des *shims* pour convertir des données entre services. Nous l'avons appliqué à des services et des types en génomique et nous avons effectué des expériences et évaluations.

Notre abstraction de types s'appuie sur les données XML avec une expressivité similaire à XML-Schema. L'idée est de prendre en compte la composition et la nature des données. Pour cela, l'abstraction de types est basée sur des types primitifs et sur des constructeurs de types composites. Les types **primitifs** sont atomiques et non décomposables du point de vue de notre système. Ils sont les ingrédients de base pour construire les types composites. Ils sont

1. Chapitre est une version remaniée de l'article Ba et al. [BFD15b] en cours de publication dans le journal TLDKS

dérivés des types primitifs courants, par exemple *string*, *number*, *text*, *token*. Les constructeurs de types utilisés sont *tag*, *tuple*, *union* et *list*. Un type **tag** dénote des éléments XML, il est constitué d'un contenu et d'un concept servant de balise. Le concept peut provenir d'une ontologie ; il informe sur la nature des données. Un exemple d'un type *tag* est $Seq := sequence[String]$ pour des séquences basées sur des caractères et annotées par le concept *sequence*. Un type **tuple** dénote des séquences XML qui sont chacune une concaténation de valeurs de types quelconques. Un exemple d'un type *tuple* est $ComSeq := Seq\ specy\ version$ pour représenter une séquence complexe par une séquence brute, son espèce source et sa version. Un type **union** dénote une union de valeurs de types quelconques. Un exemple d'un type *union* est $BioSeq = ProtSeq \mid NuclSeq$ pour représenter une séquence biologique comme soit une séquence de nucléotides, soit une séquence de protéines. Un type **list** dénote des séquences non vide de valeurs d'un même type. Un exemple d'un type *list* est $SeqList = BioSeq+$ pour une liste de séquences biologiques. En plus de ces types, nous avons le type *vide*, et le type *optionnel* qui est dérivé de l'union et du type *vide*.

A partir des types abstraits, nous définissons un système de règles de convertibilité. Le système permet de vérifier la compatibilité d'une sortie d'un service par rapport à une entrée d'un autre service, et d'extraire les informations de la sortie pour alimenter l'entrée, à partir des types des entrées et sorties. Le système de règles permet, ainsi, une correspondance et une conversion de données entre des types d'entrée et sortie. Il est défini à travers la notion de convertibilité : un type A est **convertible** en un type B s'il est possible, en appliquant les règles du système, d'obtenir une fonction de A à B qui transforme toute valeur de A en une valeur de B. Les règles sont définies en combinant, deux à deux, les constructeurs de types. Elles donnent les conditions pour que deux types quelconques soient convertibles, et elles donnent en plus une preuve de la convertibilité sous forme d'une fonction de conversion. Les règles permettent la composition et la décomposition ainsi que la généralisation et la spécialisation de types. Elles utilisent des axiomes définis sur des concepts et sur des types de base. Les axiomes donnent les convertibilités entre les concepts et entre les types de base. Les règles utilisent également un ensemble d'opérations utilitaires qui travaillent sur les données, par exemple construire ou déconstruire une donnée XML. La règle suivante indique la convertibilité entre deux types tag.

$$\frac{t_a \rightarrow_t t_b \quad f_1 : A \rightarrow B}{f : t_a[A] \rightarrow t_b[B] \quad f(x) = element(t_b, f_1(content(x)))}$$

Si un concept t_a est convertible en un concept t_b, et si un type A est convertible en un type B avec pour preuve la fonction de conversion f_1, alors le type $t_a[A]$ est convertible en $t_b[B]$ avec pour preuve la fonction de conversion f. La fonction f transforme les valeurs de $t_a[A]$ en des valeurs de $t_b[B]$ en remplaçant le concept t_a par le concept t_b et en convertissant les valeurs de A en des valeurs de B par la fonction f_1. Les opérations utilitaires sont *element* et *content*. *element* permet de construire un élément XML à partir d'un concept et d'un contenu. *content* permet d'accéder au contenu d'un élément XML. Toutes les règles de convertibilité sont fournies à la section 4.3.1 de ce chapitre. Nous avons prouvé que la convertibilité est réflexive et transitive. Nous avons implémenté notre système de règles sous forme d'un programme qui décide de la convertibilité entre deux types quelconques, et génère des conversions. L'algorithme du programme procède par reconnaissance des deux types et applique les règles de manière récursive jusqu'aux axiomes. Les conversions donnent des *shims* qui sont exprimées sous forme d'un code XQuery qui convertit des documents XML.

Nous avons instancié notre approche avec des services et des types en génomique. Nous avons défini des types respectant des formats courants pour des séquences biologiques. Nous avons utilisé les types pour représenter une trentaine de services issus de plateformes de services en bio-informatique. Avec notre programme, nous avons ensuite généré un graphe de convertibilité qui représente les services et les liaisons détectées entre les services. Les liaisons matérialisent les conversions générées entre les entrées et sorties des services. Le graphe a été évalué auprès d'experts de la plateforme GenOuest [2]. Notre approche de convertibilité est adaptée. Elle prend en compte la composition des données non prise en compte avec les formats textuels et fournit automatiquement des *shims* qui sont en général obtenus de manière manuelle. Par rapport aux approches similaires, notre approche permet une expressivité plus élevée au niveau des types et des transformations de données. Les détails des expériences et évaluations sont données à la section 4.6 de ce chapitre.

2. http ://www.genouest.org

4.1 Introduction

As we argued in previous chapters, it is difficult to compose workflows because of services mismatches. Existing solutions, particularly in bioinformatics, for service mediation are not yet sufficient for easy workflow composition by domain users. In this chapter, we propose an approach to manage incompatibility of inputs and outputs for service mediation during workflow composition. Our approach relies on type abstraction. It uses a rule system defining a set of convertibility rules between abstract types. Our types precisely account for the composite structure of data relying on XML as a data model. The rules exploit composition and decomposition as well as specialization and generalization of types. They automatically generate a complete constructive specification of the conversions from output to input types of services. That specification enables to generate executable shims (converters) between input and output XML data. We report on an experiment on bioinformatics services with an implementation of our approach. We also led a survey with domain experts. Our experiments show the relevance of links established between bioinformatics services. In the following, Section 4.2 presents type abstraction, and Section 4.3 presents convertibility rules. Section 4.4 describes implementation. Section 4.5 describes instantiation and a use case in bioinformatics. Section 4.6 presents experiments. We compare our approach with related work in Section 4.7.

4.2 Abstract types

In this section, we present the language used to describe the types of data. It is defined from an open set of primitives and a fixed set of type constructors. From a semantic point of view, a type denotes a set of XML values. An XML value is a sequence of XML nodes. An XML node is either an XML element or a textual element (CDATA). An XML element is made of a tag and a content, which is a sequence of XML nodes. An XML sequence may be empty or contain a single node.

We use XML because it provides the expressiveness needed in our context, and it enables simple solutions to map to (and from) other data models. In addition, web services are typically based on XML, they are generally designed to support XML data. Our language of types follows the main XML Schema constructs, but to simplify the presentation we use regular expressions inspired by XDuce (the work of Hosoya et al. [HVP00]). XML Schema is highly expressive, it includes for example types for mixed contents, attributes and restrictions that we do not allow. As mentioned in Section 3, our data representation is data oriented. It

does not imply mixed content because the data components to use in inputs and outputs of services have to be well separated. We consider general constructs already present in languages such as XDuce. The constructs enable annotations of elements and elements that contain other elements. As we will see, they are sufficient to represent data for our context. Supporting constructs such as attributes and value restrictions is ongoing work. In the following, types are denoted by uppercase letters (e.g., T, T_1) and function $XML(T)$ defines the semantics of type T by a set of XML values.

Primitive types. $T = p$, where p is a primitive type. Primitive types are the basic ingredients to build other types. They are build-in primitives. Their structure is atomic, they are not decomposable. XML instances of a primitive type are CDATA (text). In XML Schema, primitive can be expressed by element $xsd : simpleType$.

Constructor *tag* $t[T_1]$ where t is a concept that acts as a tag. This expression denotes XML elements whose tag is t and whose content is of type T_1 : $XML(t[T_1]) = \{$<t> x_1 </t>$|\ x_1 \in XML(T_1)\}$. Concepts inform about semantics of data. They can be provided by ontologies, in that case ids of concepts are used as tags. In XML Schema, constructor *tags* can be expressed by element $xsd : element$.

Constructor *empty.* ε. The empty XML sequence : $XML(\varepsilon) = \{\varepsilon\}$.

Constructor *tuple.* T_1T_2. This type expression denotes XML sequences that are the concatenation of instances of T_1 and instances of T_2 : $XML(T_1T_2) = \{x_1x_2 \mid x_1 \in XML(T_1), x_2 \in XML(T_2)\}$. That constructor is used to define composite types and sequences. In XML Schema, constructor *tuple* can be expressed by element $xs : complexType$.

Constructor *union.* $T_1|T_2$. This type expression denotes the union of instances of T_1 and instances of T_2 : $XML(T_1|T_2) = XML(T_1) \cup XML(T_2)$. Constructor *union* can be used, for example, to consider several different types and make some treatments without distinction. In XML Schema, that constructor *tuple* can be expressed by element $xsd : choice$.

Constructor *list.* T_1+. This type expression denotes the non-empty sequences of instances of type T_1 (homogeneous lists) : $XML(T_1+) = \{x_1 \ldots x_n \mid n \geq 1 \wedge\ x_1, \ldots, x_n \in T_1\}$. In XML Schema, that constructor can be expressed by element the $minOccurs$ and $maxOccurs$ attributes.

Constructor *optional.* $T_1?$. This type expression is equivalent to $T_1|\varepsilon$.

For abbreviation it is possible to bind a name to a type expression. For example, in expression $T = T_1T_2$, there is a type name T that can be used to refer to T_1T_2. We do not allow recursive type expressions yet, it is a perspective. Figure 4.6 provides examples of type expressions.

4.3 Convertibility Rules

Representing data by considering their nature and their composition, as presented, is a first step to enhance using data between services in workflows. That is why, several solutions based on XML and on ontologies are proposed as standards to describe data types independently of services. However, representing data with expressive and standardized languages is not sufficient to resolve mismatches between input and output data of services. It is necessary to solve the $n : m$ matching problem [EXD04], namely matching and conversion between two composite structures, i.e. XML trees.

This section describes our solution to detect and resolve mismatches between complex types of inputs and outputs of services in workflows. Based on type abstraction presented above, we address matching and picking relevant data from an output of a first service to feed an input of a second service. We use a formal system that defines rules based on type theory as presented in Section 3. By using type theory in the context of workflows, we follow the same line as previous work such as the ones for describing, matching and verifying services [KLC14, CWD+06]. The novelty of our approach lies in the definition of rules to prove convertibility between complex types, and to apply the proof-as-program paradigm [BC85] to automatically derive executable converters from convertibility proofs. Proofs-as-program is a paradigm relying approaches to develop programs from proofs in constructive logic.

In order to motivate the following convertibility rules, we list a few typical cases where there is a mismatch between two types A and B ($A \neq B$), whereas there is a semantic match, i.e. A can be converted to B.

- A and B use concepts that are different but have the same meaning, for example `Integer` and `Int`.
- B may be replaced by A, for example `Float` by `Long`.
- B is a concatenation of some components of A, for example `person[name tel email]` from `person[name contact[address tel email]]`.
- A is subsumed by B, for example `protein seq` by `biological seq`.

Function	Input	Output	Description
content	XML	XML	returns the content of an XML element
element	concept, XML	XML	builds an XML element given a concept and an XML content
concat	XML, XML	XML	returns the concatenation of two XML sequences
select	XML, type, type	XML, XML	splits an XML sequence in two parts matching given types
map	converter, XML	XML	applies a converter to each node of an XML sequence and returns the concatenation of the results
choose	XML, type	XML	returns any element matching a given type from an XML sequence

Table 4.1 – Utility functions on XML values.

The approach we propose considers those kinds of mismatches. It exploits composition and decomposition as well as specialization and generalization of types. It automatically generates a complete constructive specification of the conversions from types. That specification enables to generate converters of data matching the types.

4.3.1 Rule System

Figure 4.1 lists all the rules that specify when a type A is convertible to a type B. They also define associated converters as functions from A to B. Those rules form a natural deduction system whose judgements are in the form $f : A \rightarrow B$, i.e. f is a converter from an XML value of type A to an XML value of type B, and hence A is convertible to B. A judgement $f : A \rightarrow B$ holds true if and only if it is possible to build a proof tree with that judgement at the root, and where each node instantiates a rule. The deduction system works by structural induction on couples of types (A, B), covering all combinations of type constructors for which convertibility is possible. The rules depend on conversion axioms for concepts ($t_a \rightarrow_t t_b$), and on converters between primitive types ($f_p : p_1 \rightarrow_p p_2$). Those base conversions depend on the application domain, and correspond, for instance, to well-known conversion functions (e.g., from floats to

$$\frac{f_p : p_1 \rightarrow_p p_2}{f : p_1 \rightarrow p_2 \quad f(x) = f_p(x)} \quad \text{(PRIMITIVE)}$$

$$\frac{t_a \rightarrow_t t_b \quad f_1 : A \rightarrow B}{f : t_a[A] \rightarrow t_b[B] \quad f(x) = element(t_b, f_1(content(x)))} \quad \text{(CONCEPTCHANGE)}$$

$$\frac{f_1 : A \rightarrow B}{f : t[A] \rightarrow B \quad f(x) = f_1(content(x))} \quad \text{(CONCEPTREMOVAL)}$$

$$\frac{}{f : A \rightarrow \varepsilon \quad f(x) = \varepsilon} \quad \text{(EMPTY)}$$

$$\frac{f_1 : A \rightarrow B_1 \quad f_2 : A \rightarrow B_2}{f : A \rightarrow B_1B_2 \quad f(x) = concat(f_1(x), f_2(x))} \quad \text{(CONCAT)}$$

$$\frac{f_1 : A_1 \rightarrow B}{f : A_1A_2 \rightarrow B \quad f(x) = (let\ x_1, x_2 = select(x, A_1, A_2)\ in\ f_1(x_1))} \quad \text{(LEFTSELECTION)}$$

$$\frac{f_2 : A_2 \rightarrow B}{f : A_1A_2 \rightarrow B \quad f(x) = (let\ x_1, x_2 = select(x, A_1, A_2)\ in\ f_2(x_2))} \quad \text{(RIGHTSELECTION)}$$

$$\frac{f_1 : A_1 \rightarrow B \quad f_2 : A_2 \rightarrow B}{f : A_1|A_2 \rightarrow B \quad f(x) = (case\ (x : A_1)\ then\ f_1(x) \mid (x : A_2)\ then\ f_2(x))} \quad \text{(PRECHOICE)}$$

$$\frac{f_1 : A \rightarrow B_1}{f : A \rightarrow B_1|B_2 \quad f(x) = f_1(x)} \quad \text{(LEFTPOSTCHOICE)}$$

$$\frac{f_2 : A \rightarrow B_2}{f : A \rightarrow B_1|B_2 \quad f(x) = f_2(x)} \quad \text{(RIGHTPOSTCHOICE)}$$

$$\frac{f_1 : A \rightarrow B}{f : A+ \rightarrow B+ \quad f(x) = map(f_1, x)} \quad \text{(MAP)}$$

$$\frac{f_1 : A \rightarrow B}{f : A+ \rightarrow B \quad f(x) = (let\ x_1 = choose(x, A)\ in\ f_1(x_1))} \quad \text{(CHOICE)}$$

$$\frac{f_1 : A \rightarrow B}{f : A \rightarrow B+ \quad f(x) = f_1(x)} \quad \text{(SINGLETON)}$$

FIGURE 4.1 – Convertibility rules and definitions of generated converters.

integers). By default, we assume that $t \rightarrow_t t$ for every concept t, and $f_p : p \rightarrow_p p$ with $f_p(x) = x$ for every primitive type p. The definitions of converters in rules make use of utility functions on XML values, which are described in Table 4.1. Figure 4.2 shows a convertibility proof.

Primitive rule. Rule (Primitive) allows the use of a primitive converter f_p when the two types are primitive types. That rule handles the conversion of the leaves of XML trees (built-in primitives).

Concept rules. These rules handle the conversion from and to XML elements. Rule (ConceptChange) defines converters from an XML element x to another XML element $element(t_b, f_1(content(x)))$ by applying a domain-dependent concept conversion (here, from t_a to t_b), and by recursively applying a converter f_1 to the content of x. Function *content* gives access to the content of an XML element, and function *element* builds the new element from the converted concept and converted content. Rule (ConceptRemoval) define converters from an XML element to an XML sequence by ignoring the concept, and recursively converting the content.

Empty and tuple rules. These rules handle conversions of XML sequences, i.e. constructors *empty* and *tuple*. Rule (Empty) says that any XML value x can be converted to the empty XML sequence ϵ. Rule (Concat) defines converters that first apply the converters f_1 and f_2 to the source value x, and then concatenates the two results $f_1(x)$ and $f_2(x)$ with function *concat*, hence producing an XML sequence. Rules (LeftSelection) and (RightSelection) define converters that select respectively the left and right part of the source data, an XML sequence, and convert it to the target data. This is useful when only a part of the source data is necessary to produce the target data. The selection of the parts (function *select*) is guided by the sub-types of the source sequence.

Union rules. These rules handle conversions from and to unions of types. Rule (PreChoice) defines converters that produce a target data using a different converter depending on the type of source data (A_1 or A_2). This is useful when the source data can have different structures (union type). Rules (LeftPostChoice) and (RightPostChoice) choose a converter to a target sub-type, when the target type is an union. This is useful when the target data has several acceptable structures.

List rules. The remaining rules handle conversions from and to lists. Rule (Map) define converters from a source list to a target list where a same

seq[ns[lower] species[string] version[int]]+ $\rightarrow$ seq[organism[string] ns[upper]]+
(MAP) $f(x) = map(f_1, x)$
1. seq[ns[lower] species[string] version[int]] $\rightarrow$ seq[organism[string] ns[upper]]
(CONCEPTCHANGE) $f_1(x) = element(\text{seq}, f_{1.2}(content(x)))$
1.1. seq $\rightarrow_t$ seq
1.2. ns[lower] species[string] version[int] $\rightarrow$ organism[string] ns[upper]
(CONCAT) $f_{1.2}(x) = concat(f_{1.2.1}(x), f_{1.2.2}(x))$
1.2.1. ns[lower] species[string] version[int] $\rightarrow$ organism[string]
(RIGHTSELECTION) $f_{1.2.1}(x) = (let\ x_1, x_2 = select(x)\ in\ f_{1.2.1.1}(x_2))$
1.2.1.1. species[string] version[int] $\rightarrow$ organism[string]
(LEFTSELECTION) $f_{1.2.1.1}(x) = (let\ x_1, x_2 = select(x)\ in\ f_{1.2.1.1.1}(x))$
1.2.1.1.1. species[string] $\rightarrow$ organism[string]
(CONCEPTCHANGE)
$f_{1.2.1.1.1}(x) = element(\text{organism}, f_{1.2.1.1.1.2}(content(x)))$
1.2.1.1.1.1. species $\rightarrow_t$ organism
1.2.1.1.1.2. string $\rightarrow_p$ string
(PRIMITIVE) $f_{1.2.1.1.1.2}(x) = x$
1.2.2. ns[lower] species[string] version[int] $\rightarrow$ ns[upper]
(LEFTSELECTION) $f_{1.2.2}(x) = (let\ x_1, x_2 = select(x)\ in\ f_{1.2.2.1}(x_1))$
1.2.2.1. ns[lower] $\rightarrow$ ns[upper]
(CONCEPTCHANGE) $f_{1.2.2.1}(x) = element(\text{ns}, f_{1.2.2.1.2}(content(x)))$
1.2.2.1.2. lower $\rightarrow_p$ upper
(PRIMITIVE) $f_{1.2.2.1.2}(x) = uppercase(x)$

FIGURE 4.2 – An example proof tree of convertibility between two kinds of sequence lists.

converter is applied to each element of the list. Function *map* is used to perform iteration over list elements, and concatenation of converted elements. Rule (CHOICE) defines converters that first choose an element of a list, and then recursively apply a converter to it. This is useful when a single element is expected while a list is provided. Rule (SINGLETON) defines converters that produce singleton lists from a source element, after recursively applying a converter to it. This is useful when a list is expected while a single element is provided.

For a given couple (A, B) of type expressions, several rules may be applicable. In that case, it is sufficient that one of them leads to a success to prove the convertibility from A to B. Figure 4.2 details the proof of convertibility between two kinds of biological sequence lists. In the source list, sequences are made of a nucleotide sequence, a species name, and a version number, while in the target list,

$$
\begin{aligned}
&f(x) = map(f_1, x)\\
&\text{where } f_1(x) = element(\text{seq}, concat(\\
&\qquad let\ x_1, x_2 = select(content(x), \text{ns[lower]}, (\text{species[string] version[int]}))\\
&\qquad\quad in\ let\ x_{21}, x_{22} = select(x_2, \text{species[string]}, \text{version[int]})\\
&\qquad\qquad in\ element(\text{organism}, content(x_{21})),\\
&\qquad let\ x_1, x_2 = select(content(x), \text{ns[lower]}, (\text{species[string] version[int]}))\\
&\qquad\quad in\ element(\text{ns}, uppercase(content(x_1)))))
\end{aligned}
$$

Figure 4.3 – The generated converter for example of Figure 4.2

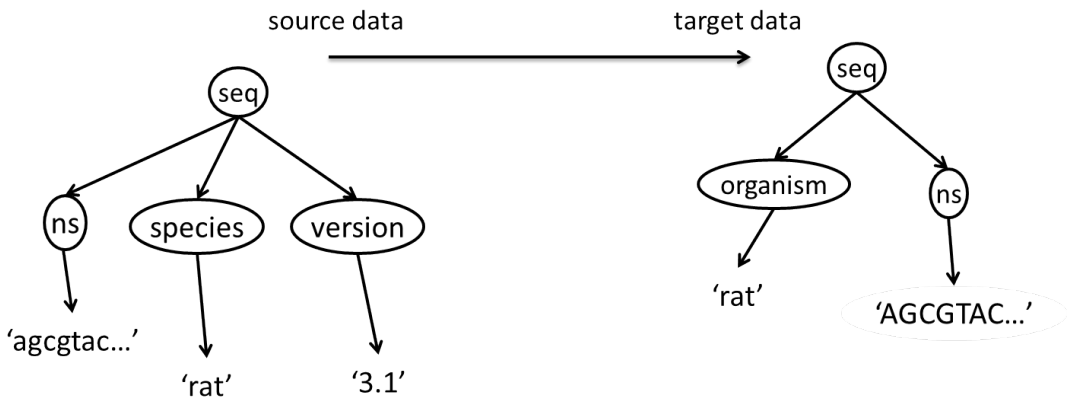

Figure 4.4 – Example of a source data and a target data to show how elements of sequence lists are transformed by converter of Figure 4.3.

sequences are made of an organism, and a nucleotide sequence. Another difference is that nucleotide sequences are lowercase in the source list (primitive type *lower*), and uppercase in the target list (primitive type *upper*). In the proof (Figure 4.2), we assume that a primitive converter is available to convert from lowercase to uppercase (see step 1.2.2.1.2.), and domain knowledge tells us that concept *species* can be replaced by *organism* (see step 1.2.1.1.1.). Each item in Figure 4.2 is the conclusion of a rule, and the sub-items are the hypotheses of the rule. At each item, the converter function is defined with calls to the converter function of sub-items. After inlining the definition of intermediate functions in the main function f, we obtain the full definition of f in Figure 4.3. Figure 4.4 shows an example of a source data matching the first biological sequence type converted to a target data matching the second biological sequence type.

Our rule system exhibits two kinds of non-determinism : (1) in the generation of converters, and (2) in the definition of converters. Firstly, given two types A and B, the system may generate several converters from A to B, i.e. several solutions to the conversion problem. This is a common feature of rule systems. For example, a converter $f : AA \rightarrow A$ can be produced by either Rule (LEFTSELECTION) or Rule (RIGHTSELECTION) : in the former, the left part of the source value is selected, while in the latter, the right part is selected. In practice, one converter must be chosen, which could be done through user interaction. Secondly, a generated converter may produce several target values for a same source value. This non-determinism comes from utility functions *select* and *choose*, and the *case* construct. Function *select* may find different ways to split the source value in two parts. Function *choose* has as many results as elements in the list. The *case* construct has two results when the two conditions are satisfied, when x matches both types A_1 and A_2. This second form of non-determinism could be used to express iteration in a workflow. For example, assuming that service S_1 produces lists of sequences, and service S_2 consumes one sequence at a time, the converter generated by Rule (CHOICE) could be a way to express that S_2 must be iterated over the results of S_1, and the output of S_2 could be considered to be the list of individual results.

4.3.2 System Properties

Two important properties of the convertibility relationship are reflexivity and transitivity. First, every type A is convertible to itself and it suffices to take the identity function as a converter. Second, for any types A, B, C, if A is convertible to B, and B is convertible to C, then A is convertible to C and it suffices to compose the two converters from A to B and from B to C to obtain a converter from A to C. We formalize those two properties in the following theorems, and

give their proofs. In those theorems, we assume that concept convertibility ($\rightarrow_t$) and primitive convertibility ($\rightarrow_p$) are reflexive and transitive. As a consequence, unlike the approach of Kaslev et al. [KLC13], we do not need rules for transitivity and reflexivity in our rule system. This makes convertibility proofs simpler and more efficient because every sub-goals in a proof of $A \rightarrow C$ only contains sub-expressions of types A and C. The transitivity rule requires the "invention" of an intermediate type B.

Theorem 1.

Let A be a type expression. There is a proof in the rule system of the judgement $f : A \rightarrow A$ where $f(x) = x$.

Proof. *We proceed by induction on type A, considering the six type constructors as different cases. For each of the type constructor, the property is verified because :*

1. *$A = p$ (primitive) : from assumption on primitives ($p \rightarrow_p p$), and by applying Rule* (Primitive).
2. *$A = t[A_1]$ (concept) : from induction hypothesis on A_1 ($A_1 \rightarrow A_1$), and assumption on concepts ($t \rightarrow_t t$), and by applying Rule* (ConceptChange).
3. *$A = \epsilon$ (empty) : from Rule* (Empty).
4. *$A = A_1A_2$ (tuple) : from induction hypothesis on A_1 ($A_1 \rightarrow A_1$) and A_2 ($A_2 \rightarrow A_2$), by applying Rule* (LeftSelection) *to the first, and Rule* (RightSelection) *to the second, and finally by applying Rule* (Concat) *to the consequences of the two previous rules.*
5. *$A = A_1|A_2$ (union) : from induction hypothesis on A_1 and A_2, by applying Rule* (LeftPostChoice) *to the first (introducing A_2), and Rule* (RightPostChoice) *to the second (introducing A_1), and finally by applying Rule* (PreChoice) *to the consequences of the two previous rules.*
6. *$A = A_1+$ (list) : from induction hypothesis on A_1, and by applying Rule* (Map).

In each case, it can be shown that the produced converter function is equivalent to the identity function. For instance, in the concept *case, assuming $f_1 : A_1 \rightarrow A_1$ is equivalent to the identity function (induction hypothesis), it can be shown that the resulting function $f(x) = element(t, f_1(content(x)))$ is equivalent to $element(t, content(x))$ which is equal to x because x has type $t[A_1]$.* ■

Theorem 2.

Let A, B, C be expression types. If there are proofs in the rule system of the judgements $f_1 : A \to B$ and $f_2 : B \to C$, then there is also a proof of the judgement $f : A \to C$ where $f(x) = f_2(f_1(x))$.

Proof. *We proceed by induction on the rules that are used at the root of the proofs of $A \to B$ and $B \to C$. As there are 13 distinct rules, there are potentially 169 distinct cases to consider. Fortunately, many cases have similar proofs and can be grouped together based on the distinction between three kinds of rules :*

- *Constructors ($\mathcal{C}$) : rules where only the target type changes between premises and conclusion (Rules* (CONCAT), (LEFTPOSTCHOICE), (RIGHTPOSTCHOICE), (SINGLETON), (EMPTY)*),*
- *Destructors ($\mathcal{D}$) : rules where only the source type changes between premises and conclusion (Rules* (CONCEPTREMOVAL), (LEFTSELECTION), (RIGHTSELECTION), (PRECHOICE), (CHOICE)*),*
- *Transformers ($\mathcal{T}$) : rules where both source and target change but use the same kind of type (Rules* (PRIMITIVE), (CONCEPTCHANGE), (MAP)*).*

Using that grouping, we arrive at 6 meta-cases described using the above group codes $\mathcal{C}$, $\mathcal{D}$, $\mathcal{T}$ and $\mathcal{X}$ to mean any rule. Applying unification constraints on the middle type B, those meta-cases then decompose themselves in 21 elementary cases :

1. *$\mathcal{X}$ - $\mathcal{C}$:*
   (a) *$\mathcal{X}$ -* (CONCAT) *: the proof of $B \to C$ by Rule* (CONCAT) *implies that $C = C_1C_2$, and that we have proofs for $B \to C_1$, and $B \to C_2$. By induction hypothesis on A, B, C_1 and A, B, C_2, we obtain $A \to C_1$ and $A \to C_2$. Then, by applying Rule* (CONCAT) *on those judgements, we finally obtain $A \to C$.*
   (b) *$\mathcal{X}$ -* (LEFTPOSTCHOICE) *: we have $C = C_1|C_2$, and $B \to C_1$. By induction hypothesis on A, B, C_1, we obtain $A \to C_1$. Then, by applying Rule* (LEFTPOSTCHOICE) *on the later, we obtain $A \to C_1|C_2$.*
   (c) *$\mathcal{X}$ -* (RIGHTPOSTCHOICE) *: similar to previous case.*
   (d) *$\mathcal{X}$ -* (SINGLETON) *: we have $C = C_1+$ and $B \to C_1$. By induction hypothesis on A, B, C_1, we obtain $A \to C_1$, from which we obtain $A \to C$ by applying Rule* (SINGLETON)*.*
   (e) *$\mathcal{X}$ -* (EMPTY) *: we have $C = \epsilon$. We directly obtain $A \to C$ by applying Rule* (EMPTY) *(everything is convertible to ϵ).*
2. *$\mathcal{D}$ - $\mathcal{X}$:*

(a) (ConceptRemoval) - $\mathcal{X}$: *we have* $A = t[A_1]$ *and* $A_1 \rightarrow B$. *By induction hypothesis on* A_1, B, C, *we obtain* $A_1 \rightarrow C$. *By applying Rule* (ConceptRemoval) *on the latter, we obtain* $A \rightarrow C$.

(b) (LeftSelection) - $\mathcal{X}$: *we have* $A = A_1A_2$, *and* $A_1 \rightarrow B$. *By induction hypothesis on* A_1, B, C, *we obtain* $A_1 \rightarrow C$. *By applying Rule* (LeftSelection) *on the latter, we obtain* $A \rightarrow C$.

(c) (RightSelection) - $\mathcal{X}$: *similar to previous case.*

(d) (PreChoice) - $\mathcal{X}$: *we have* $A = A_1|A_2$, $A_1 \rightarrow B$, *and* $A_2 \rightarrow B$. *By induction hypothesis on* A_1, B, C *and* A_2, B, C, *we obtain* $A_1 \rightarrow B$ *and* $A_2 \rightarrow B$. *By applying Rule* (PreChoice), *we obtain* $A \rightarrow C$.

(e) (Choice) - $\mathcal{X}$: *we have* $A = A_1+$ *and* $A_1 \rightarrow B$. *By induction hypothesis on* A_1, B, C, *we obtain* $A_1 \rightarrow C$. *By applying Rule* (Choice) *to the latter, we obtain* $A \rightarrow C$.

3. $\mathcal{C}$ - $\mathcal{D}$:

(a) (Concat) - (LeftSelection) : *we have* $B = B_1B_2$, *and the judgements (1)* $A \rightarrow B_1$, *(2)* $A \rightarrow B_2$, *(3)* $B_1 \rightarrow C$. *By induction hypothesis on* A, B_1, C *and judgements (1) and (3), we obtain* $A \rightarrow C$.

(b) (Concat) - (RightSelection) : *similar as previous case.*

(c) (LeftPostChoice) - (PreChoice) : *we have* $B = B_1|B_2$ *and the judgements* $A \rightarrow B_1$, $B_1 \rightarrow C$, *and* $B_2 \rightarrow C$. *By induction hypothesis on* A, B_1, C, *we obtain* $A \rightarrow C$.

(d) (RightPostChoice) - (PreChoice) : *similar to previous case.*

(e) (Singleton) - (Choice) : *we have* $B = B_1+$, *and the judgements* $A \rightarrow B_1$ *and* $B_1 \rightarrow C$. *By induction hypothesis on* A, B_1, C, *we obtain* $A \rightarrow C$.

4. $\mathcal{T}$ - $\mathcal{T}$:

(a) (Primitive) - (Primitive) : *we have* $A = p_1$, $B = p_2$, $C = p_3$, *and* $p_1 \rightarrow_p p2$ *and* $p_2 \rightarrow_p p_3$. *From assumptions on primitives, we obtain* $p_1 \rightarrow_p p_3$. *By applying Rule* (Primitive) *on the latter, we obtain* $A \rightarrow C$.

(b) (ConceptChange) - (ConceptChange) : *we have* $A = t_A[A_1]$, $B = t_B[B_1]$, $C = t_C[C_1]$, *and* $t_A \rightarrow_t t_B$, $t_B \rightarrow_t t_C$, $A_1 \rightarrow B_1$, $B_1 \rightarrow C_1$. *From assumptions on concepts, we obtain* $t_A \rightarrow_t t_C$. *By induction hypothesis on* A_1, B_1, C_1, *we obtain* $A_1 \rightarrow C_1$. *By applying Rule* (ConceptChange) *to the two latter judgements, we obtain* $A \rightarrow C$.

(c) (Map) - (Map) : *we have* $A = A_1+$, $B = B_1+$, $C = C_1+$, *and* $A_1 \rightarrow B_1$, $B_1 \rightarrow C_1$. *By induction hypothesis on* A_1, B_1, C_1, *we obtain* $A_1 \rightarrow C_1$. *By applying Rule* (Map) *to the latter, we obtain* $A \rightarrow C$.

5. $\mathcal{C}$ - $\mathcal{T}$:

   (a) (SINGLETON) - (MAP) *: we have* $B = B_1+$, $C = C_1+$, *and* $A \to B_1$, $B_1 \to C_1$. *By induction hypothesis on* A, B_1, C_1, *we obtain* $A \to C_1$. *By applying Rule* (SINGLETON) *to the latter, we obtain* $A \to C$.

6. $\mathcal{T}$ - $\mathcal{D}$:

   (a) (CONCEPTCHANGE) - (CONCEPTREMOVAL) *: we have* $A = t_A[A_1]$, $B = t_B[B_1]$, *and* $A_1 \to B_1$, $B_1 \to C$. *By induction hypothesis on* A_1, B_1, C, *we obtain* $A_1 \to C$. *By applying Rule* (CONCEPTREMOVAL) *to the latter, we obtain* $A \to C$.

   (b) (MAP) - (CHOICE) *: we have* $A = A_1+$, $B = B_1+$, *and* $A_1 \to B_1$, $B_1 \to C$. *By induction hypothesis on* A_1, B_1, C, *we obtain* $A_1 \to C$. *By applying Rule* (CHOICE) *to the latter, we obtain* $A \to C$.

In each case, it can be shown that the produced converter function is equivalent to the composition of the two converters from A *to* B, *and from* B *to* C. *For instance, in the* $\mathcal{X}$ - (CONCAT) *case, assuming* $f_1 : A \to B$, *and* $f_2 : B \to C$, *we have* $f_2(x) = concat(f_2'(x), f_2''(x))$ *where* $f_2' : B \to C_1$ *and* $f_2'' : B \to C_2$. *By application of three rules as indicated in the proof, we obtain for* $f : A \to C$, *the definition* $f(x) = concat(f_2'(f_1(x)), f_2''(f_1(x)))$. *From the definition of* f_2, *that definition can be simplified into* $f(x) = f_2(f_1(x))$, *which is indeed the composition of* f_1 *and* f_2. ■

4.4 Implementation

We have implemented our solution. Figure 4.5 shows the general approach used by a program implemented to decide convertibility between any two type expressions (output type and input type), and to generate converters from data matching the output type expression to data matching the input type expression. The algorithm is coded in Java. It is directly derived from the above rules and combines *pattern matching* on type expressions to identify constructors, and recursive calls on type sub-expressions. We use the JAXB API to support representing type expressions in XML and in JSON. JAXB classes (Java classes) are generated from a general XML schema of our type abstraction. To process convertibility in our algorithm, we use objects of the generated Java classes. They contain the type expressions. They can automatically be converted from (and to) XML documents (and JSON documents) representing type expressions and respecting the general XML schema. We also implement simple RESTful services to manage (display, add, delete, modify) properly type expressions and services that use them.

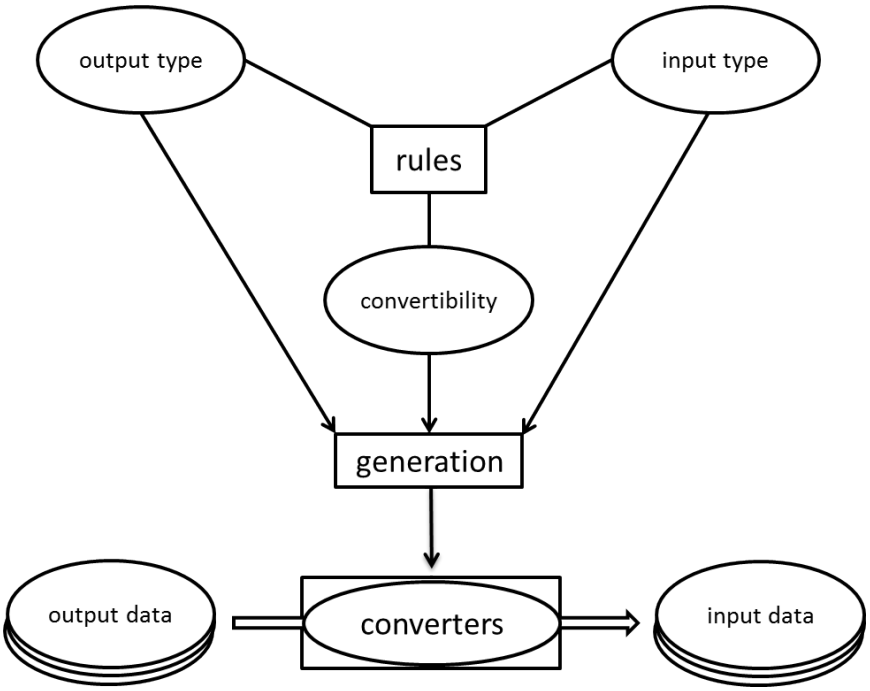

Figure 4.5 – Global schema of the program that computes convertibility.

As defined by convertibility rules, in the algorithm, recursive calls always involve smaller couples of expressions, which ensures termination of the program in all cases. The results of the program are represented by objects of a special Java class. The objects contain converter specification(s) when convertibility is detected by the program from two types, otherwise they are empty. Generated converters, from converter specification(s), are represented in XQuery, a suitable language to process XML documents, which makes the converters executable. To account for non-determinism, the result of our program is a collection of converters. Each converter will be a function from an XML value to an XML value. In the case of non-deterministic converters, only one value is produced so far. The production of several values is left to future implementation.

The computation time required to decide convertibility may be high because of the non-deterministic nature of the rule system. However, convertibility is computed once for a set of types. In practice, types are not very large, and we have not encountered any difficulty in our experiments to compute all convertibilities for a set of bioinformatics services (see Section 4.6.1). Generated converters are efficient. Most rules produce transformations that imply a constant cost per XML node, and hence a linear complexity over the input data. The two cases that may imply additional costs concern node duplication (see Rules LeftSe-

LECTION and RIGHTSELECTION) and choice handling (see Rule PRECHOICE). Node duplication corresponds to the situation where a node of the input data is converted from several nodes of the output data. In this case, the duplicated node is processed several times, thus exceeding linear complexity. However, the number of duplications is bounded by the number of constructors in the output type. Choice handling corresponds to the situation where a node can have one of two types (A_1 or A_2), and the correct type has to be identified by the converter at execution time. Type identification requires one additional node processing for each choice, thus exceeding linear complexity. However, the number of choices is equal to the number of constructor union in the input type, and the additional processing may only concern a fragment of the input data. In practice, input and output types of bioinformatics services make a limited use of duplications and choices, thus in the worst case, the complexity of converters is in the size of the input data multiplied by a small constant.

4.5 Instantiation and Use Case

This section presents how we instantiate our type abstraction and convertibility rules to bioinformatics data (Section 4.5.1). It also presents a use case that shows how our approach can be used to detect and resolve data mismatches in bioinformatics workflows (Section 4.5.2).

4.5.1 Instantiation to Bioinformatics

As presented the state-of-the-art chapter (Chapter 2), there is many available solutions to represent data in bioinformatics. For genomics data, various textual formats (e.g., FastQ, BED) [3] and XML formats (e.g., BioXSD [4], phyloXML [5]) are provided. In addition to data formats, ontologies, such as EDAM [6] have been proposed to organize and classify resources including data types and formats. Our work starts from these resources to define input and output types of services. We abstract data types by focusing on the information contents and composite structure of data. Figure 4.6 shows simple examples of types defined manually from existing bioinformatic formats. *Accession* and *SSeq* are simple types representing, respectively, an accession number and a raw sequence, defined with constructor *tag* and a primitive. In the same way, *DNASeq* (representing nucleotide sequences) and *ProtSeq* (representing amino acid sequences) are defined

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

```
> Accession = accession[string]
> SSeq = simpleSequence[string]
> ProtSeq = ns[sSeq]
> DNASeq = as[sSeq]
> Bioseq =  DNASeq | ProtSeq
> CBioseq = complexBiosequence[
      seq[Bioseq]
      species[string] source[string] name[string]
      version[string] note[string]?]
> CProtSeq = complexProteinSequence[
       seq[ProtSeq]
       species[string] source[string] name[string]
       version[string] note[string]?]
> BioseqList = CBioseq+
```

Figure 4.6 – Examples of bioinformatics types

using *SSeq*, and they specialize it using different concept. Their union forms *Bioseq*, a general type for biological sequences. *CBioseq* and *CProtSeq* are composite types holding several types through constructor *tuple*. They are biological sequences containing required (e.g., *sequence*[*Bioseq*]) and optional (e.g., *note*[*string*]?) contents, *CProtSeq* being more specific than *CBioseq*. *CBioseqList* defines a list of *CBioseq* using constructor *list*. The other types we use are defined in the same way as the above types. Concepts are inspired from the EDAM ontology.

Compared to data types and formats used on platform EMBOSS[7], our types can define accession numbers allowing to represent, for example, sequence and database references. They can represent raw sequences as in plain text format, single sequences as in gcc format, one or several sequences (e.g., alignment of sequences) as in FASTA format, as well as a simple sequence associated to its annotations and features as in EMBL format. Our types can also represent lists of files and differentiate the nature of information contained in files, for example, nucleotide sequence versus amino acid sequence. We take into account data types and formats commonly used for inputs and outputs of services on platform EMBOSS. Most platforms we visited use the same categories of data types and formats. Compared to XML formats such as BioXSD, our abstraction represents contents at a higher abstraction level. As said, we ignore, for example some type attributes and type restrictions irrelevant for current input and output data used in services.

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

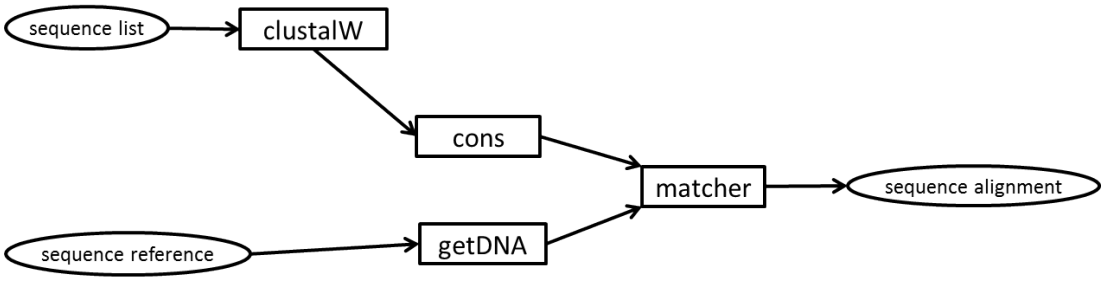

FIGURE 4.7 – Workflow at user level (w_u).

If necessary, they can easily be added.

Abstraction of types is straightforward for XML formats thanks to XML schemas. For textual formats, informal specifications must be studied to derive a structural representation. Our experiment with genomics types shows that some type expressions recur frequently, so they can easily be reused after being defined once. Since the most common data types are defined, there are increasingly less types to define. Moreover, the BioXSD initiative defines several data types for common bioinformatics web services. Specialized XML formats, such as phyloXML [HZ09] for phylogenetic data and PDBML [WIN+05] for systems biology, exist for sub-domains of bioinformatics. Moreover, XML alternatives are provided for some textual formats (e.g., GFF [DJD+01]) and some platforms define their own XML format (e.g., Uniprot XML [C+10]). For our experiment, the abstraction of types is done manually but the spreading of the above mentioned solutions will facilitate the task. We can even expect automatic or semi-automatic abstraction processes.

In our approach, adding a new service requires two steps. Firstly, identify the abstract types used as inputs and outputs of the service. Secondly, implement, if they do not already exist, the converters between XML schemas and each format, since our types define an XML. Unlike other approaches, it is not necessary to define converters for all pairs of formats, but only two for each format (from and to XML).

4.5.2 Use Case : Resolving Data Mismatches in a Bioinformatics Workflow

We now present a workflow (w), constructed with our approach. It compares a consensus sequence produced from an alignment of a list of sequences with another sequence obtained from an accession number. The workflow consumes

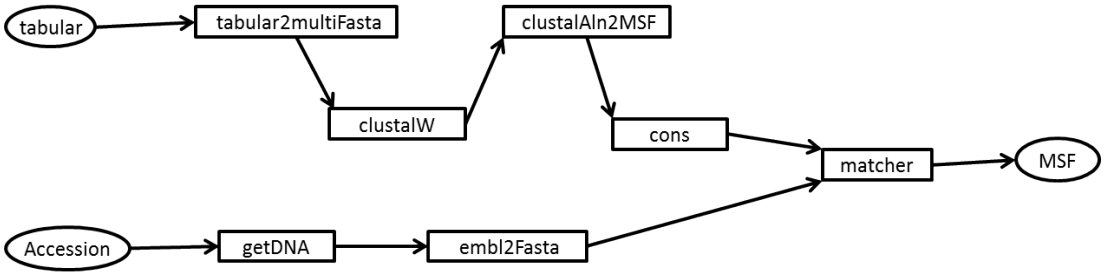

Figure 4.8 – Executable workflow (w_x).

sequence lists (tabular form) and references (accessions) to DNA sequences. It produces sequence alignments. In the following, we describe representations of workflow w, at three different levels. The workflow at user level is what the user expects to see but we show that it is underspecified, and cannot be executed as such. The workflow at execution level is fully specified and executable, but it is over-detailed for the user because it confuses domain services and shims. We finally introduce the workflow at a abstract level, from which both user level and execution level representations can be derived automatically.

Figure 4.7 shows the workflow at a user level (w_u). Boxes represent tasks and ellipses represent input and output data of the workflow. It is an interconnection of inputs and outputs of services from bioinformatics platforms. We assume that a service performs one task, it may have one or several inputs and one or several outputs. To simplify, we do not take into account parameters used to manage service behaviour (e.g., algorithm parameters). The workflow uses the following services :

— `matcher` compares two biological sequences. It takes as inputs two biological sequences in *fasta files* and returns as output a *MSF alignment*.
— `getDNA` retrieves a DNA sequence from a database. It takes an accession number and returns an *embl file* containing a DNA sequence.
— `cons` creates a consensus sequence from a multiple alignment. It takes a MSF alignment and returns a *fasta* containing a biological sequence.
— `clustalW` makes a multi-alignment of sequences. It takes a *multi Fasta* containing a list of sequences and returns an alignment.

Workflow w_u shows an ideal view where users have specified only the indispensable information to describe what has to be achieved. However, this view is not directly executable because the workflow uses services whose inputs and outputs do not immediately match. It is necessary to insert shims services to address data mismatches as generally done in the existing platforms.

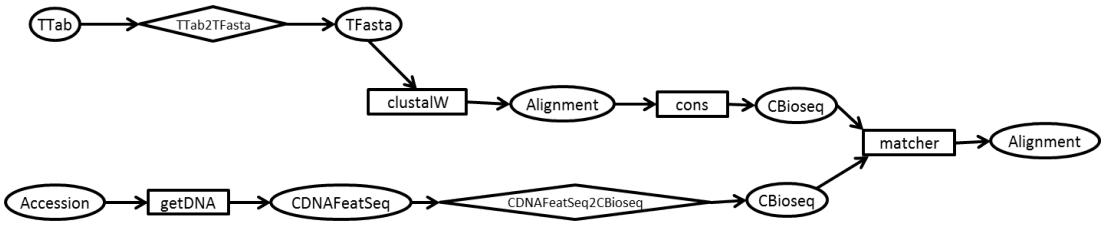

FIGURE 4.9 – Abstract workflow (w_a) in our system.

Figure 4.8 shows an executable workflow (w_x) where shims have been inserted because of the following mismatches. First mismatch (tabular2multiFasta) : the concrete lists of sequences the workflow will consume are in tabular form with additional columns, they must be adapted to feed the input of `clustalW`. Second mismatch (clustalAln2MSF) : the output of task `clustalW` must be adapted to match the input of task `cons`. The same information is used by the two tasks but it is represented by different formats. Last mismatch (embl2Fasta) : task `getDNA` provides an output that must be fed into the input of task `matcher`, the output of `getDNA` contains more information and is more specific. In existing approaches, to obtain an executable workflow, users must find and insert format converters between domain tasks for which the formats are different. Data are seen through their formats and they are not decomposable. There is no separation between data, formats and services and no separation between domain services and shim services.

Figure 4.9 shows the abstract workflow used in our system (w_a). Ellipses are data types, boxes are domain tasks, and diamonds are generated converters used as shims. Our system sees service inputs and outputs through their composite abstract types. Each service is represented with its input and output types as follows :

— *Alignment* represents the output type of the service matcher, the input type of the service cons and the output type of the service `clustalW`. It also represents the output type of the workflow.
— *CBioSeq* represents the two input types of the service `matcher` and the output type of the service `cons`.
— $TFasta = seqs[id[string]\ seq[Bioseq]]+$ represents the input type of the service `clustalW`.
— *Accession* represents the input type of the service `getDNA`, and also an input type of the workflow.
— $CDNAFeatSeq = seq[CDNASeq\ Features]$ represents the output type of `getDNA`.
— $TTab = seqs[id[string]\ C2[string]\ C3[string]\ C4[string]\ seq[Bioseq]]+$ re-

```
1  declare namespace my='www.irisa.fr/LIS/members/moba/flowlis/shims';
2
3  declare function my:content($e as node()*)
4   as node()*
5  {
6  $e/node()
7  };
8
9  declare function my:select($n1 as xs:string, $e as node()*)
10  as item()*
11 {
12 for $child in $e
13 where name($child)=$n1
14 return element {$n1}{$child/node()}
15 };
16
17 declare function my:convert($v0 as node()*)
18  as node()*
19 {
20 let $v1:=my:content($v0)
21 let $v2:=my:select('CDNASeq',$v1)
22 let $v3:=my:content($v2)
23 return $v3
24 };
```

Figure 4.10 – Example of source code of the shim *my* :converter converting data matching `CDNAFeatSeq` to data matching `CBioseq`. It is represented using XQuery. Functions *my* :content and *my* :select are utility functions.

presents an input type of the workflow.

Our system separates data types, formats and services. It detects and resolves mismatches letting users focus on domain tasks. The generated shims fix the data mismatches presented above. The first generated shim, *TTab2TFasta*, corresponds to the resolution of the `tabular2multiFasta` mismatch, by transforming each element (row) of the list (table). Transforming each element involves selecting and concatenating sub-elements (fields). The second shim, *CDNAFeatSeq2CBioseq*, fixes the `embl2Fasta` mismatch by recognizing a DNA sequence as a biological sequence after ignoring additional information. The `clustalAln2MSF` mismatch is fixed by reflexivity because *clustalAln* and *MSF* correspond to the same abstract type. By ignoring derivable and optional information, our system recognises them as equal. Figure 4.10 shows a generated XQuery source code of shim *CDNAFeatSeq2CBioseq*.

Service	Source	Inputs	Outputs	task
Blast	BioXSD	biosequence database URI	biosequence	search
Blastp	EMBL-EBI	Fasta sequence (typed) database URI	BlastResult	search
ClustalWFastaCollection	BioMoby	Fasta files	MSF	alignment
ClustalW	BioXSD	biosequence (≥ 2)	alignment	alignment
maskfeat	EMBOSS	EMBL sequence	Fasta sequence	handling

Table 4.2 – Examples of services

From our abstract workflow (w_a), it is possible to obtain the workflow at user level (w_u) by hiding intermediate types and generated shims. It is also possible to produce the executable workflow (w_x) by inserting converters between concrete formats and XML abstract types (e.g., XML from/to each of tabular, MSF, multi Fasta, embl and Fasta). Note that although the users do not have to interfere to generate the converters, they can still check them because all steps of the generation are traceable. In our approach, formats are concrete serialization of data, they are not bound to particular data or services. Their use is flexible, for example changing concrete formats do not affect service compatibility or service implementation.

4.6 Experiments

The objective of this section is to evaluate and validate our convertibility approach on real services in bioinformatics. We have shown how services and their inputs and outputs are instantiated in our approach. Now, we apply our convertibility technique to bioinformatics services and evaluate the results. Section 4.6.1 presents a graph of convertibilities where links between bioinformatics services are detected . Section 4.6.2 presents a survey with domain experts to check the relevance of graph links.

4.6.1 Convertibility between Bioinformatics Services

We selected 30 services from platforms EMBOSS [RLB00], EBI (European Bioinformatics Institute) [MVG+09], BioMoby [WL02] and services adopting the BioXSD format. Other variations and similar categories are provided in platforms but their input and output types are very similar in general, and they would add little to our experiment. Tables 4.2 and 4.3 show examples of services and types.

Type	Type in our representation	Represents
Fasta sequence	ComplexProteinSequence | ComplexBiosequence	(typed) sequences
BlastResult, Fasta files	ListOfComplexBiosequences | ListOfBiosequences	a list of sequences
EMBL sequence	AnnotatedSequence	an annotated sequence
BioXSD biosequence	Biosequence | ComplexBiosequence	one sequence
BioXSD Alignment, MSF	SequenceAlignment	an alignement of sequences
Database URI	DatabaseReference	a reference to a database

Table 4.3 – Examples of formats and types

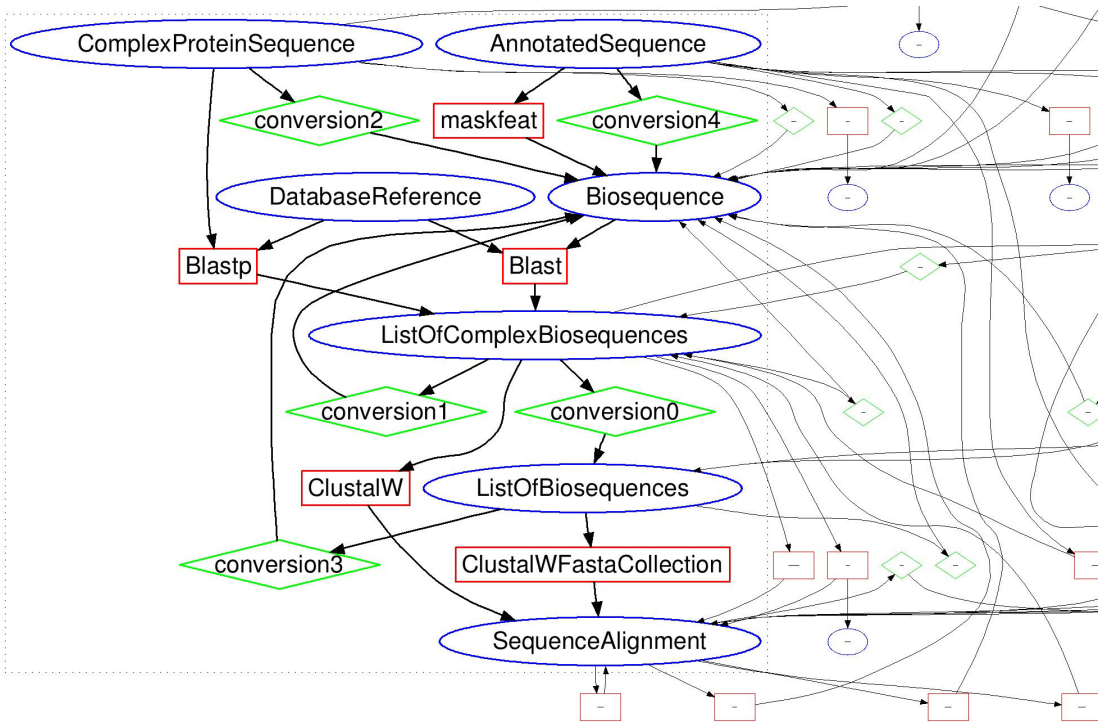

Figure 4.11 – Excerpt of the graph of links between services

From our selection of services, using our convertibility program (see Section 4.4), we automatically generated a graph of the links between services. Each connection is a convertibility, proved by our rule system, from an output type of a service to an input type of another service. Figure 4.11 shows an excerpt from the obtained graph. The complete graph and service list are available online[8]. The graph shows services, input/output types and conversions between types. Services are represented by rectangles, input/output types by ellipses and conversions by diamonds. Services are associated with their inputs and outputs respectively by incoming and outgoing arrows. Similarly, each conversion is associated with a source type and target type, it materializes an automatically detected conversion between two types. Connectivity of services in the graph materializes processing chains where output data are transformed according to the need of the service inputs.

Our program finds direct links when the services use the same abstract types.This is the case, for example, with the link between services *Blast* and *ClustalW*. Indirect links correspond to a $conversion_i$. In the following, each $conversion_i(A, B)$ specifies a function to transform data of type A to data of type B. With *conversion0(ListOfComplexBiosequences, ListOfBiosequence)*, a list of simple sequences is derived from a list of complex sequences. The function converts each element of the list and produces a new list of the converted elements. The elements of the list are converted using Rule (LEFTSELECTION) (or Rule (RIGHTSELECTION)), and the new list is produced by Rule (MAP). *Conversion1(ListOfComplexBiosequences, Biosequence)* combines rules (CHOICE) and (LEFTSELECTION) (or (RIGHTSELECTION)) to go from a list of complex protein sequences to each simple biological sequence of the list. A simple biological sequence being a component of a complex biological sequence. *Conversion2(ComplexProteinSequence, Biosequence)* shows the generalization and specialization of types. Our algorithm detects that a sequence of proteins is also a biological sequence. The conversion uses (LEFTPOSTCHOICE) (or (RIGHTPOSTCHOICE)) to obtain a complex biological sequence from the complex protein sequence and uses Rule (LEFTSELECTION) (or (RIGHTSELECTION)) to obtain a biological sequence from the complex biological sequence. Our algorithm also individually considers the elements of a list. With *conversion3(ListOfBiosequences, Biosequence)* each element of a list of sequences can be selected, it is done by rule (CHOICE). With *conversion4(AnnotedSequence, Biosequence)*, a simple sequence is derived from an annotated sequence. A composite type is decomposed and some components are selected to feed services. This conversion corres-

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

ponds to rules (LeftSelection) and (RightSelection). In above conversions, rules (ConceptChange), (ConceptRemoval) and (Primitive) are used to convert between primitives and concepts.

The complete graph contains 264 links between services out of the 900 possible links between 30 services (30x30). Our program therefore finds numerous links, but remains specific enough to be useful. Among those links, 88 are direct links, i.e., do not imply any conversion. Our convertibility relation therefore enables a three-fold increase of the number of links between services. The 30 services use 26 different types, among which 10 types are in fact ad-hoc and not decomposable (e.g., pictures, reports). Our program has identified 27 possible conversions between the 16 composite types, out of the 256 possible ones (16x16). This again shows that our approach is both productive and specific.

We presented the graph of the experiment to developers and users of the GenOuest bioinformatics platform. They pointed out that "it is remarkable that the central role of sequence alignment is so visible in the graph". They stated that "the produced graph has a pedagogical interest". Indeed, in a typical course handout [9], a graph produced by hand related to a library of bioinformatics services contains similar services and links as our graph. However, being more complete in the modeling of input and output data, our approach offers more flexibility on input and output types. Thus our algorithm provides more explicit links and differentiation between categories of services. It also reveals possible conversions between input and output data, that creates new links. Moreover, our graph is machine processable, it shows a proof of the links created between services and can quickly take into account new changes on data types and services. With a growing set of currently over 1500 available tools, it is unlikely that people can produce the graph by hand. Our main objective is to guide biologists when composing workflows. One perspective is to take into account other aspects of services. When constructing a workflow, input and output types play a central role in the selection of services by setting constraints on applicable services, but are not the only criteria for selection for biologists. It is also necessary to represent the services by their functional and non-functional properties (e.g., bioinformatics task performed, quality of results, provenance, efficiency, popularity). The GenOuest developers nevertheless mentioned that a user with domain knowledge would already find useful support in the produced graph to select services for a workflow among the possibilities given by the graph. Thus, they validated that the graph generated by our approach detects relevant information and produces,

9. Presentation of services of Wisconsin Package- Olivier Collin - CNRS Roscoff - Formation Génopole Ouest - november 2002

in a systematic way, knowledge usually acquired by experience. The costly step of the approach is the production of abstract types, currently done by hand. They highlighted some interesting perspectives. Our data abstractions could be enriched from ontologies, especially EDAM, which will significantly facilitate the type abstraction step and allow integrating others facets. In addition, data in Genomics (e.g., phylogenetic trees) and in others domains (e.g., metabolism) have to be added to the experiment.

4.6.2 Survey with Domain Experts

We conducted a more systematic evaluation of the graph produced at Section 4.6.1 by asking domain experts to judge about the relevance of generated links between services. For two services s_1 and s_2 in the graph, if the type of an output port of s_1 is convertible to the type of an input port of s_2, we suggested to experts to connect the input of s_2 to the output of s_1. We prepared 25 questions, each question being a suggestion to connect two services. Suggestions involved 18 services. The 25 questions concerned all links on ports of 3 services (output port of blastp, input port of blast and input port of cons). For each question, it was possible to give one among 6 answers : "Yes, I knew", "Yes, I discover", "It may be feasible", "It is not feasible", "No, I disagree", and "I do not know". We provided the description of services and a brief explanation of convertibility that involved the suggestion. After the 25 questions, we added general questions about user status, background knowledge, the services they know by their names, and their opinion about the relevance of suggestions. The survey was submitted to experts of bioinformatics data, services and workflows via a mailing list.

We collected responses from 6 participants. Among the participants, one did not provide any answer on suggestions, and therefore has been removed from the results on suggestions. One did not finish the survey, thus for him we just consider answered questions. All participants work in bioinformatics. Four of them are computer scientists, four are users of services and workflows, four are developers of workflows, five are developers of services, one is a developer of Workflow Management Systems (WfMS) and one is a researcher. Each participant knew on average 7 services among the 18 services proposed as suggestions. 3 of the services were known to all participants, 7 were known to no participants. The number of known services may look surprisingly low for domain experts. It can be explained by the fact that participants are actually expert in one among many bioinformatics platforms. The same task is often implemented by different services (with different names) across different platforms.

Suggested links	Cross-platforms	SL	SR	1st pers	2nd pers	3th pers	4th pers	5th pers
blastp > blast	no	5	6	--	-	++	+	o
blastp > blastp	no	5	5	o	-	++	+	o
blastp > pepcoil	yes	5	0	o	-	+	+	+
blastp > diffseq	yes	5	2	o	-	+	+	+
blastp > emma	yes	5	0	o	-	+	+	+
blastp > clustalW	yes	5	5	o	-	++	+	++
blastp > backtranseq	yes	5	0	+	-	+	+	++
blastp > clustalWFastaCollection	yes	5	2	+	-	+	+	++
blastp < **blast**	no	5	6	o	-	++	+	o
cons < **blast**	yes	0	6	+	+	+	o	o
blast < **blast**	no	6	6	++	-	++	++	o
merger < **blast**	yes	0	6	x	+	+	-	?
transeq < **blast**	yes	1	6	x	--	+	++	++
blastn < **blast**	no	6	6	x	-	++	++	o
blastx < **blast**	no	6	6	x	-	++	++	o
extractFeat < **blast**	yes	0	6	x	?	+	+	o
extractAlign < **blast**	yes	1	6	x	+	+	+	o
backtranseq < **blast**	yes	0	6	x	+	+	+	++
gassst < **cons**	yes	5	0	x	o	++	+	--
matcher < **cons**	no	3	0	x	+	o	+	o
maxAlign < **cons**	yes	0	0	x	o	+	+	o
emma < **cons**	no	0	0	x	o	+	+	o
merger < **cons**	no	0	0	x	+	+	+	o
clustalW < **cons**	yes	5	0	x	+	++	+	++
clustalWFastaCollection < **cons**	yes	2	0	x	+	++	+	++

Figure 4.12 – Reponses of 5 participants to the list of link suggestions between services.

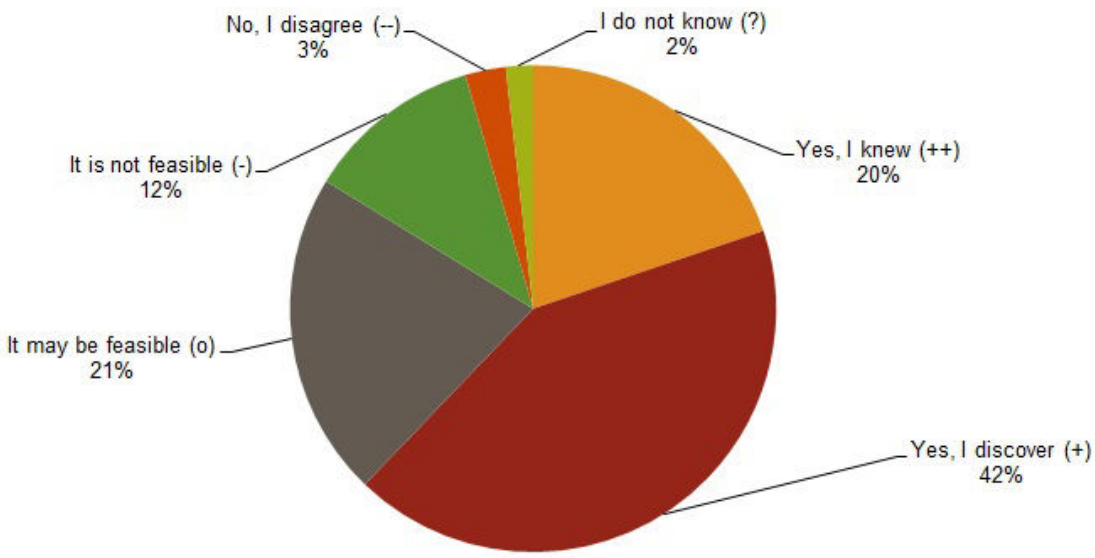

Figure 4.13 – Pie chat of all participant responses to all suggestions, per response type.

Figure 4.12 and Figure 4.13 show the results of the survey. The table of Figure 4.12 lists the responses for each participant and for each suggestion. 'Cross-platform' means services that are from different platforms. The left column of the table contains suggestions established between services, for example '**blastp** > *blast*' meant that the system suggested that service blast could consume the output of the service blastp ; '*clustalW* < **cons**' : the system suggested that clustalW could produce the input of the service cons. The second column of the table tells if connected services come from different platforms. Columns SL and SR contain the number of participants who know the tool of either the left part (SL) or the right part (SR). From column four to column eight, are the responses of the participants. The responses are represented by (++) for "Yes, I knew", (+) for "Yes, I discover", (o) for "It may be feasible", (-) for "It is not feasible", (--) for "No, I disagree" and (?) for "I do not know". Missing responses are represented by crosses. Figure 4.13 aggregates results. It provides the percentage by response type for all responses.

The results show that participants found that 83% of suggested links are acceptable. They already knew 20% and discovered 42% of suggested links. They judged that 21% of suggested links may be feasible. However, they found that 12% of suggested links as not really feasible. They are in total disagreement with 3% of suggested links. The participants have no opinion for 2% of suggested links. Despite knowing only a few (at most 15%) of the suggested services, participants still recognized most suggestions (83%) as acceptable. That paradoxical observation deserves an explanation. Many services are unknown to participants by their name, because belonging to another platform than the one they are used to, but their functions are known to them. For example, 'emma' has the same function as 'ClustalW', but only the latter is known by the participants. Thanks to the short description that we provided in the survey, and to online information, participants were able to recognize the function of unknown services, and to evaluate them accordingly. For example, the table of Figure 4.12 shows that 'emma' is evaluated very similarly to 'ClustalW'. This also explains the high proportion of "Yes, I discover" responses (42%), which is much higher than we expected. It implies two positive results. Firstly, our suggestions cover many relevant service links that would not be considered by domain experts in the construction of a workflow by lack of knowledge about services. Secondly, most of our suggestions are recognized as relevant by domain experts.

4.7 Related Work

Detecting and resolving service mismatches is prior to facilitate composing workflows. Mismatches at the signature level, between inputs and outputs of services, is a real problem when interconnecting services. Our approach addresses mismatches that occur on the structure and on the semantics of service inputs and outputs. It can be classified among works that automatically match services and generate shims for service mediation. Our approach, however, enables using existing converters as base converters between base types, and it can use semantic matching in base convertibilities between concepts.

Some approaches address service mismatches using ontologies such as EDAM [IKJ+13] and myGrid ontology [WAH+07]. They provide methods to access semantic compatibility of workflow components. For example, Lebreton et al. [LBC+10] propose to verify the semantic compatibility of service parameters for web service composition. Stroulia et al. [SW05] uses the structure of data types, messages, operations and textual descriptions to assess similarity between WSDL (Web Service Description Language) specifications. The approaches do not directly address converting data between services. They address verifying compatibility or similarity between services. However, our approach will easily benefit from works on service annotation and service matching techniques to enhance the matching.

Approaches that search shims address converting data in workflows. They generally rely on data types, formats and service descriptions. For example, Velasco-Elizondo et al. [EDG+13] use data format descriptions to automatically identify relevant shims. In the same manner, the approach of Hull et al [HSL+04] relies on the description of shims and data types to retrieve shims transforming data between services. These approaches, however, expect all shims to be manually defined by third-parties. In that context, the number of shims continues to grow and specific shims become difficult to recognize and to find. In addition, due to the lack of descriptions and information dependence between shims, it is difficult to re-use existing shims to create new ones. Our approach enables to re-use shims. The information it maintains between shims and data types as well as conversion proofs it provides through rules enable to recognize specific shims and their functions.

Techniques to retrieve shims, for example those presented above, are used to find shims in bioinformatics platforms such as Galaxy [GNTT10]. In platforms, matching between inputs and outputs is generally based on data formats and existing converters. Services, in their implementation, take into account several

formats. When a format is not provided, a format converter defined by hand is used. It has a high cost : integrating several format conversions in each tool is a burden for service developers and creating new shims is time-consuming and error prone. It also implies a strong dependency between services and formats and mixes domain tasks and tasks for the data conversion. In addition, it is more difficult to differentiate shims when they are mixed with other services. To facilitate conversion of data, separation between abstract formats and concrete formats is needed. For example, data schemas will facilitate data extraction and transformation. It is the subject of works such as Kalas et al. [KPJ+10] that provides common XML-based formats for bioinformatics data. At present, few implementations of services use these technologies. A generalization of their use would strengthen our approach, as it would facilitate the specification of abstract types. Possibilities offered by XML technologies to represent complex data and relationships associated to domain ontologies may be used to provide a pivot language for conversion between heterogeneous data formats.

In contrast to approaches that search shims, approaches that generate shims automatically provide data transformers to insert between services. They better isolate shims from domain services. They are characterized by the data complexity they support and the transformations between data they offer. Browers et al [BL04] support structural data transformations based on ontological information. Their approach links a structural type to a corresponding semantic type. The semantic type is defined with concepts of ontologies. However, the transformations that they offer rely on contextual paths. They do not allow transforming XML documents whose paths of elements are not already attached to domain concepts. They also, do not seem to take into account arbitrarily nested transformations, which is required for some complex data. In Conveyor, Linke et al. [LGG11] propose generic object-oriented type system to manage nodes, including data types, in workflows. They use interfaces, abstract classes and inheritances to express relations between types but do not mention automated methods ensuring composition and decomposition of complex data types.

Kashlev et al [KLC13] use rules to automatically insert shims as coercions into executable workflows. Their approach and the approach of Dibernado et al [DPW08] are close to our work. The first one because it uses rules, and the second one because it is applied to compose bioinformatics services. We differ from the approach of Kashlev et al. on several points. We envision to mediate data at workflow construction time, not at execution time. We allow more complex data representations, for example type constructors union, concept and list. Thus, we offer additional transformations between input and output data. For example, the approach of Kashlev et al. requires different labels for elements of

a tuple while our abstraction authorizes to define a tuple where elements use the same label. This allows us to meet some bioinformatics data representations such as representing lists of biological sequences, and enables parallel transformation on lists. We differ from the approach of Dibernado et al. [DPW08] because we allow decomposing as well as composing data. Dibernado et al. provide the data transformation during workflow composition. They allow generating shims according to connected services. However, they decompose data but do not allow to compose them.

Compared to presented approaches, our approach offers richer type abstractions and more complex transformations between data. It relies on rules that allow to systematically reason on input and output types for managing mismatches of services. In addition, our approach does not prevent to use existing shims, it provides systematic and automatic mechanisms to re-use some of them (e.g., converters for upper case, lower case or substitution). Unlike many approaches, our approach does not only provide the solution in XML, it also provides mechanisms to link existing textual representations. Furthermore, our approach may easily benefit from mechanisms of matching based on ontologies.

4.8 Conclusion

Mismatches between services make it difficult to create scientific workflows. Manually defined shims proposed to fix data mismatches are time-consuming and error prone. Existing approaches that automatically insert shims in workflows are limited in the transformations they provide. In this chapter, we presented an approach that systematically detects convertibility from output types to input types. We have defined convertibility rules that exploit (de)composition as well as specialization and generalization of types. The rules also automatically generate converters between input and output XML data that can be used as shims. Our approach is appropriate for users of platforms such as Taverna [OGA+06a] and Galaxy [GNTT10] that need more automation in service mediation. It can also be used to manage creation, re-use and conversion of parameters in workflow engines such as Bpipe [SPO12], Snakemake [KR12] that use low-level programming. In the next chapter (Chapter 5), we will see how our convertibility approach is used to help users compose workflows interactively and incrementally.

Chapitre 5

Guided Composition of Workflows

Résumé en français [1]

Dans ce chapitre, nous proposons une approche semi-automatique de composition de workflows. L'approche utilise la solution de convertibilité présentée au chapitre 4 de cette thèse. Elle est basée sur le framework des Systémes d'Information Logiques présentés à la section 3.2 du chapitre 3. Elle formalise une instance du framework LIS adaptée à la sélection et à l'interconnexion de services. Elle utilise une notion de focus pour capturer l'intention de l'utilisateur. Des suggestions et une vérification, basées sur la convertibilité des entrées et sorties des services, sont proposées durant la composition de workflows. Notre approche est appliquée à des services en bio-informatique.

Les composants sur lesquels notre approche repose sont : une base de connaissances, un workflow, des propriétés désirées, des suggestions et des transformations.

— La **base de connaissances** contient les connaissances que notre approche exploite pour raisonner. Elle est composée de services, de types et des convertibilités entre les types. Chaque service est associé à des types définissant ses entrées et ses sorties, et un graphe de convertibilité est généré à partir des services.

— Le **workflow** est la représentation d'une composition de services. Il est proche du modèle du langage de workflow Scufl présenté à la section 2.2

1. Ce chapitre est une version étendue de l'article Ba et al. [BFD15a].

du chapitre 2. Un workflow est composé de tâches, de ports d'entrée et sortie de tâches et de liaisons entre ports. Une tâche est une instance d'un service. Un port est une instance d'une entrée ou d'une sortie d'un service. Une liaison est une relation allant d'un port de sortie vers un port d'entrée. Chaque tâche et chaque port est identifié de manière unique. Un port admet le type de l'entrée ou de la sortie qu'il instancie. Nous considérons également un modèle dérivé de type *dataflow* qui est une abstraction du workflow. Le modèle *dataflow* considère principalement les liaisons entre ports : à partir d'un workflow, il garde les liaisons entre les ports des services, et pour un même service, il forme des liaisons allant de chaque port d'entrée vers chaque port de sortie.

— Les **propriétés désirées** de workflow sont les contraintes désirées que tout workflow doit respecter. Elles matérialisent un workflow bien formé et un workflow complet. Un workflow est bien formé lorsque tous ses ports d'entrée sont impliqués dans une seule liaison au maximum, que chacune de ses liaisons vérifie la convertibilité du type du port de sortie vers le type du port d'entrée, et qu'en plus sa représentation *dataflow* n'admet pas de cycle. Un workflow est complet lorsqu'il est bien formé et que tous ses ports d'entrée sont liés.

— Les **suggestions** sont des services et des liaisons recommandés automatiquement durant la composition d'un workflow. Elles sont calculées en utilisant le focus et les propriétés de workflow. Le focus est un ensemble de ports, il précise les ports qu'un utilisateur à l'intention de connecter. Des services et des liaisons sont suggérés à partir de la base de connaissances lorsque que ensemble ils connectent tous les ports du focus et qu'ils respectent les propriétés de workflow.

— Les **transformations** sont des actions qu'un utilisateur peut effectuer et qui modifient le workflow courant et, éventuellement, les suggestions. Actuellement, elles sont constituées d'ajouts et de suppressions de services et de liaisons.

Basé sur ces composants, notre système permet à un utilisateur de composer un workflow bien formé en respectant les étapes suivantes. D'abord, l'utilisateur indique des ports (le focus) qu'il souhaite connecter dans un workflow. Ensuite, le système suggère, à partir d'une base de connaissances, les services et les liaisons respectant des propriétés désirées et permettant de connecter les ports du focus. Enfin, l'utilisateur ajoute un service ou une liaison à partir des suggestions. Il peut supprimer une liaison ou un service du workflow. Les ajouts et les suppressions impliquent toujours des workflows bien formés. Le système permet des construction en avant et en arrière (de sorties vers des entrées, ou d'entrée vers des sorties).

Nous montrons à la section 5.4 que notre approche permet une composition de workflow sûre et complète. Un prototype de notre approche est disponible. Il utilise les ressources de la convertibilité du chapitre 2. Il fonctionne actuellement avec une version d'environ 120 services de la plateforme EMBOSS. Par rapport aux approches existantes, notre approche offre des suggestions à partir de plusieurs points d'un workflow. Elle considère la structure composite des types d'entrée et sortie des services. Elle garantit des liaisons entre services cohérentes et sûres par rapport à la convertibilité des données. Elle laisse à l'utilisateur le soin de diriger le processus de composition et offre des *shims* générés automatiquement entre les services.

5.1 Introduction

Heterogeneous data and services make it difficult to find and compose compatibles services. For example, the existence of numerous similar services (that does not exactly use the same parameters) make it easy to make mistakes when selecting and interconnecting services. Thus, users, particularly domain users, need more help to compose workflows. Chapter 4 provides a solution to detect and resolve service input and output mismatches. This chapter describes a new approach to help users compose services. The approach uses the convertibility solution proposed in Chapter 4, and it is based on the framework proposed by Logical Information Systems. The approach lets users, step-by-step, compose services through safe suggestions with respect to type convertibility of service inputs and outputs. Users manage workflow composition by specifying points of the workflow to complete, by receiving suggestions of components to use, and by performing insertion and removal of components. In the following, Section 5.2 defines the principle of our workflow composition approach and Section 5.3 formalizes the different components of our approach. Section 5.4 gives a formal evaluation by proving safeness and completeness. Section 5.5 presents our proof-of-concept prototype, and Section 5.6 shows a use case. Finally, Section 5.7 compares our approach with related work.

5.2 Principles of our Approach

Our approach enables composing compatible services regarding data to transfer and workflow coherence to maintain. It lets users control composition, it gives them compatible components, and it prevents them to introduce incoherences by placing guardrails. Our approach uses type convertibility presented in Chapter 4 to benefit from detection and resolution of service incompatibilities in workflows.

It is based on the framework of Logical Information Systems (LIS) presented in Chapter 3.

The LIS framework is the *building plan* of our approach. It offers a generic system model to manage heterogeneous data, by defining system components, their roles and their interactions. It directly supports knowledge base about services as any data. The guided creation of requests and the data navigation that LIS draws meets our needs : a workflow to compose is seen as a request to a knowledge base about services. Instead of the LIS extension, we use desired properties to constrain request results. For example, services to retrieve and to interconnect have their inputs and outputs convertible, new connections between services guaranty workflow coherence. Furthermore, the index of the LIS framework inspired a solution for suggestions through the notion of focus. The focus captures user intentions, it serves to set workflow points where born suggestions. As navigational links, we provide transformations that users can use to compose workflows by exploiting suggestions. Our approach benefits from LIS guidance to describe, navigate and retrieve service resources.

The LIS framework is here instantiated to data types, services and workflows. We take into account new dimensions involved by service composition. Services are computable objects to access and to interconnect as workflows. To interconnect them properly, it is necessary to retrieve the compatible ones and to specify coherent interconnections. It involves to capture user requests, to retrieve adapted components and to encompass the interconnections, step by step. Our system meets these requirements, it allows to interactively create well-defined workflows.

In our system, to compose a workflow, a user follows these 3 steps. Firstly, the user indicates the points of the workflow that he wants to augment. Secondly, our system makes suggestions related to those points, using background knowledge and preserving workflow desired properties. Thirdly, the user can select a suggestion to extend the workflow. He can also remove a component of the workflow at any time. The composition steps can be applied backward or forward in the dataflow (from inputs to outputs, or from outputs to inputs) at any time.

In the following, we introduce the components used in our approach. Figure 5.1 shows a schematic representation of the components. Proofs of the preservation of desired properties during composition can be found Section 5.4 as a formal validation of our approach.

Knowledge Base : it is composed of service descriptions as well as their type signature abstractions and detected convertibilities. Services have

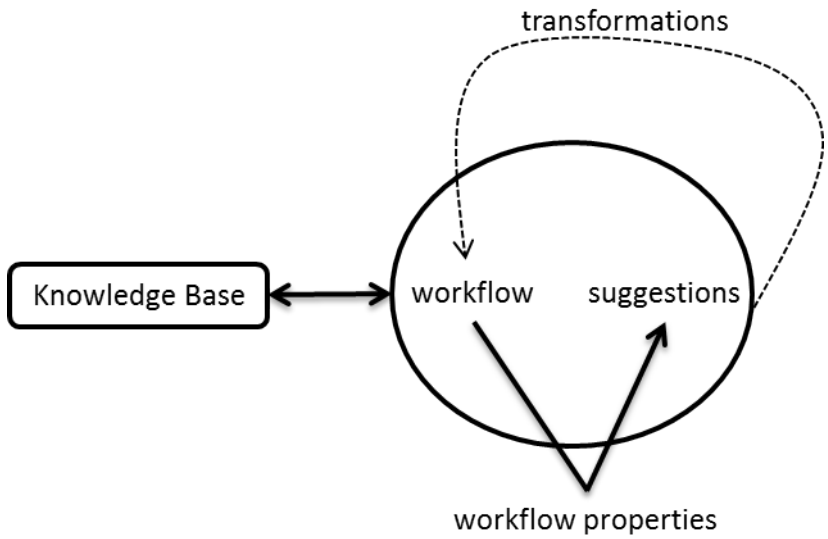

Figure 5.1 – A schematic representation of the components of our approach.

ports, each port is described by a type. We formalize the contents of our knowledge base in Section 5.3.1.

Workflow : it is composed of tasks, ports of tasks, and data links between ports. Tasks are instances of services. They have concrete ports that are instances of ports of services. The workflow model is formalized in Section 5.3.2.

Workflow Properties : they define constraints on the workflow that have to be preserved during the composition. For example, there cannot be several links to a same task input. Properties are formalized in Section 5.3.3.

Suggestions : they are services and links proposed to users during composition. They depend of the focus specified by the user. The focus consists of points of the workflow at which links and services can be inserted. The suggestions are calculated from the focus and the current workflow using type convertibility. They guarantee the workflow properties. The suggestion mechanism is formalized in Section 5.3.4.

Transformations : they define authorized modifications of the workflow components. The main transformations are insertion and removal of services and links. They are formalized in Section 5.3.5.

5.3 Formal Models

In this section, we provide formal definitions of the components of our approach introduced above.

5.3.1 Knowledge Base

The knowledge base is defined through a set of types $\mathcal{T}$, a set of ports Π, a library of domain services Σ and a set of convertibilities Λ.

Definition 1 (Abstract port).

An abstract port $\pi = (name, type) \in \Pi$ is defined by its name and its type $\in \mathcal{T}$.

Definition 2 (Service).

A service $\sigma = (name, \Pi^{in}, \Pi^{out}) \in \Sigma$ is defined by its *name* as well as a set of abstract input ports Π^{in}, and a set of abstract output ports Π^{out}.

Definition 3 (Convertibility).

A convertibility $\lambda = (type_1, type_2) \in \Lambda$ is defined by a couple of types verifying that $type_1$ is convertible to $type_2$.

Services, input ports and output ports corresponds to resources already presented in Section 4.5.1. Convertibilities are correspond to the results obtained in convertibility graph in Section 4.6.1. In the following, we use the 'dot' notation to note tuple components in definitions, for example $\sigma.\Pi^{out}$ is the set of output ports of service σ.

5.3.2 Workflow

A workflow is defined by instantiating services and ports, and by linking them. Our workflow model is close to the language Scufl for representing scientific workflows.

Definition 4 (workflow).

A workflow is a 3-tuple $W = (P, T, L)$, where

- P is a set of concrete ports, namely instances of abstract ports. Each concrete port $p = (id, \pi)$ is uniquely identified by id and refers to the port π it instantiates. In the following, we use term 'port' to refer both to concrete and abstract ports, the context will disambiguate. We note $W.IP$ and $W.OP$, respectively, the set of input ports and the set of output ports of the tasks of the workflow,
- T is a set of tasks, namely instances of services. Each task $t = (id, \sigma, P^{in}, P^{out})$ is an instance of the σ service, it is uniquely identified by id. It defines input ports and output ports ($P^{in}, P^{out} \subset P$) that are, respectively, instances of the input and output ports of σ ($\sigma.\Pi^{in}$, $\sigma.\Pi^{out}$).
  T contains two special tasks (t_{setter} and t_{getter}) handling the global inputs and outputs of the workflow. Task t_{setter} does not have inputs and its outputs are the external inputs of the workflow, it enables users to set the source data of the workflow. Task t_{getter} does not have outputs and its inputs are the external outputs of the workflow, it enables users to get the results of the workflow.
- L is a set of links between ports. Each link $l = (p_{out}, p_{in})$ is defined from an output port to an input port.

Ports of a workflow are ports of the tasks composing the workflow, including the ports of tasks t_{getter} and t_{setter}. Figure 5.2 shows an example workflow. Boxes are tasks, circles are ports, and edges are links. The workflow enables to merge two biological sequences, the merge produces a new biological sequence that is matched with a given biological sequence.

A workflow can be converted in a executable languages. For example in Scufl, our tasks correspond to Scufl processor entities, our ports correspond to ports of processors and our links to links defined in Scufl. We choose to be independent from specific languages at composition level. This allows to be free against evolutions of workflow languages and to avoid certain restrictions made by some languages, for example concerning complexity and usability. This also enables to choose the workflow languages at execution level, once the composition is completed. It is efficient to serialize a workflow in different languages according, for example, to execution environments and criteria. A workflow W can be abstracted as a dataflow graph. The dataflow graph $DF(W)$ has ports of W as vertices, and data dependencies as edges. There is an edge from p_1 to p_2 (flow from p_1 to

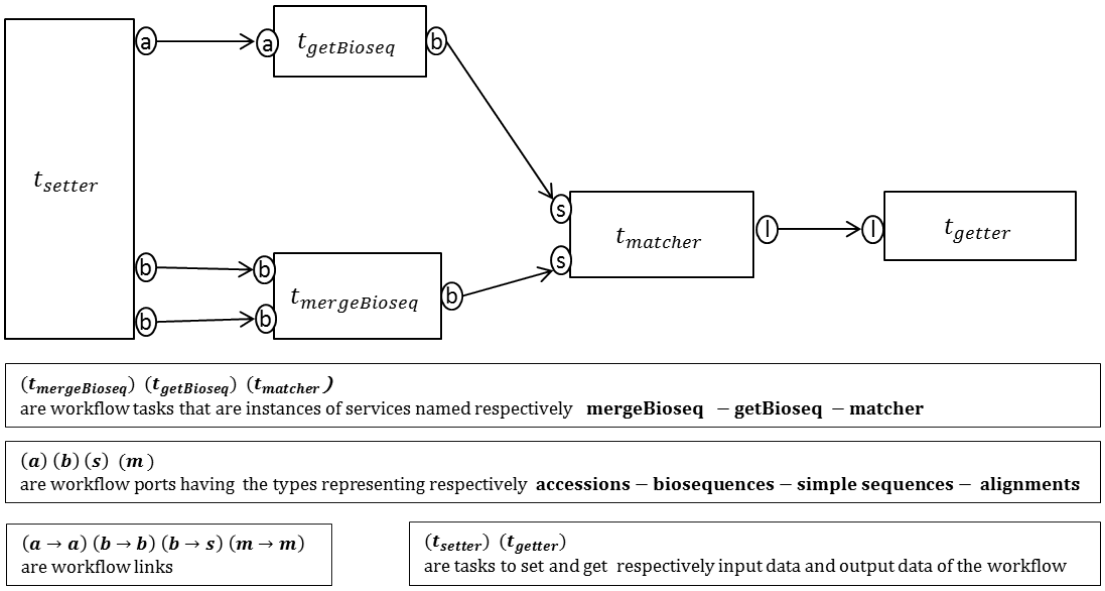

Figure 5.2 – An example workflow.

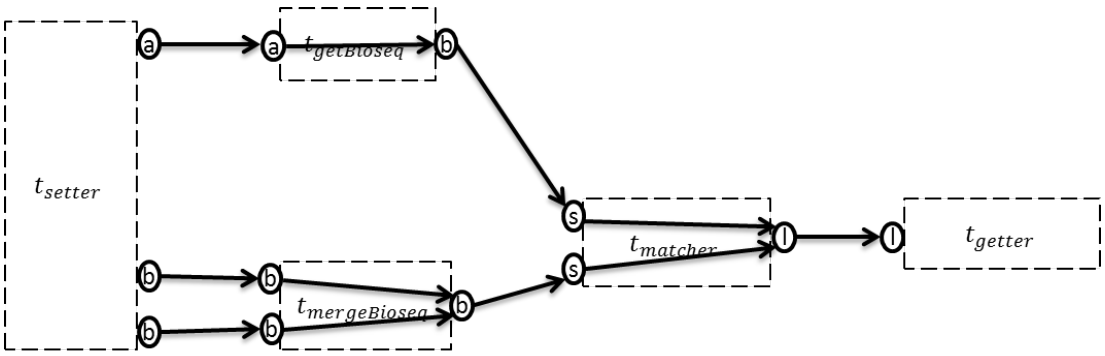

Figure 5.3 – An example dataflow.

p_2) if there is a link from p_1 to p_2, or there is a task of which p_1 is an input, and p_2 is an output. Figure 5.3 shows the dataflow graph derived from the example workflow of Figure 5.2.

Definition 5 (dataflow graph).

Given a workflow $W = (P, T, L)$, its dataflow graph $DF(W)$ is defined by the graph $G = (V, E)$, where $V = W.P$ and
$E = W.L \cup \{(p_{in}, p_{out}) \mid \exists t \in W.T : p_{in} \in t.P^{in} \wedge p_{out} \in t.P^{out}\}$

5.3.3 Workflow Properties

A workflow should be well-formed at every composition step. At the end of the composition, the workflow should be fully-defined. To define these properties, we use port dependencies from the dataflow graph.

Definition 6 (dependency).

Port p_2 depends on port $p1$ if and only if there is a path from p_1 to p_2 in the dataflow graph $DF(W)$.

Definition 7 (well-formed workflow).

A workflow is well-formed if and only if (1) no port of the workflow depends upon itself, i.e., there is no circuit in $DF(W)$, (2) every input port of the workflow is, at most, the target of one link and (3) every link (p_{out}, p_{in}) verifies convertibility, i.e., $(p_{out}.\pi.type, p_{in}.\pi.type) \in \Lambda$.

In a well-formed workflow, there is no cycle, an input only consumes data from one output, and there is no data incompatibilities (data consumed by an input must be convertible from the output data to match the input type). A data from an output may however be fed in several inputs. As in many existing models of scientific workflows, we do not allow expressing directly complex constructs such as loop in workflow definitions. There is approach that enables complex constructs, but they have to deal with lower level programming of workflows, it is not what we want for our final users.

Definition 8 (fully-defined workflow).

A workflow is fully-defined if and only if it is well-formed, and every input port is the target of a link.

Well-formedness allows for partial workflows with correct links between ports. A fully-defined workflow can be executed/evaluated. Indeed, a task can be executed only if all its inputs are linked to an output. A fully-defined workflow can be abstracted as a new service, which can then be used in future workflows.

5.3.4 Focus and Suggestions

Given a workflow $W = (P, T, L)$, a focus $F \subseteq P$ is a set of ports of the tasks composing the workflow. The focus defines points of the workflow from which tasks and links can be added. We define $F^{in} = F \cap W.IP$ the input ports of the focus and $F^{out} = F \cap W.OP$ the output ports of the focus. Asking for suggestions with reference to the focus is equivalent to requesting services that can be connected to all focus ports. Suggestions concern on the one hand services whose input and output types are convertible to/from the port types of the focus, and on the other hand links verifying convertibility between ports of the workflow. Suggestions must guarantee that the workflow after connection remains well-formed.

The services are suggested from input ports F^{in} and output ports F^{out} of a given focus F as defined by function $sugg^{\sigma}$ that returns services and detected links.

Definition 9 (service suggestions).

Provided that $for\ all\ p_o \in F^{out}, for\ all\ p_i \in F^{in}$, p_o does not depend on p_i, and p_i does not depend on any port of the workflow, the suggestion of services is defined as follows :

$$\begin{aligned} sugg^{\sigma}(W, F) = \{(\sigma, \varphi^{in}, \varphi^{out}) \mid \quad & \sigma \in \Sigma \wedge \\ & \varphi^{in} \in F^{out} \rightarrow \sigma.\Pi^{in} \wedge \varphi^{in}\ is\ injective \wedge \\ & \forall p \in F^{out} : (p.\pi.type,\ \varphi^{in}(p).type) \in \Lambda \wedge \\ & \varphi^{out} \in F^{in} \rightarrow \sigma.\Pi^{out} \wedge \\ & \forall p \in F^{in} : (\varphi^{out}(p).type,\ p.\pi.type) \in \Lambda\} \end{aligned}$$

In the definition of $sugg^{\sigma}$, detected convertibilities between the ports of the focus and ports of a service are described by mappings (φ^{in}) from the set of output ports of the focus to the set of input ports of the service, and (φ^{out}) from the set of input ports of the focus to the set of output ports of the service. The two mapping define links between ports of the focus and ports of suggested services. Mapping φ^{in} prevents links from one output to several inputs of a same service in suggestions. The situation where a same data is consumed in several inputs of a service is not common, thus we prefer adding those links separately. φ^{in} is injective to avoid suggesting multiple links on an input of a service. Mapping φ^{out} avoids multiple links on an input, and enables to consume a data in several services. Furthermore, for a suggestion to be valid, it is necessary that (1) no input port of the focus depends on a port of the workflow, and (2) no output port of the focus depends on an input port of the focus. The first condition prevents multiple links on input ports, and the second prevents circular dependencies.

The suggestion of links is defined by function $sugg^{l}$ that computes all possible links between ports of the workflow.

Definition 10 (link suggestions).

The suggestion of links is defined as follows :

$$\begin{aligned} sugg^{l}(W) = \{(p_{out}, p_{in}) \mid \; & p_{out} \in W.OP \wedge p_{in} \in W.IP \wedge \\ & p_{in} \text{ does not depend on any port} \\ & p_{out} \text{ does not depend on } p_{in} \wedge \\ & (p_{out}.\pi.type,\ p_{in}.\pi.type) \in \Lambda\} \end{aligned}$$

The suggestion of links does not depend on the focus. However, it could be possible to use the focus to filter suggested links, and it would be based on the same properties as the suggestion of services : no multiple links to inputs, no circuits, convertibility of data types.

5.3.5 Transformations

The main transformations during workflow construction are insertion and removal of services and links. The services and links to insert in the workflow are chosen among the suggestions. The tasks and links to remove are chosen in the current workflow. We prove in Section 5.4 that every insertion of suggested service or link, and every removal produces a well-formed workflow, when starting with a well-formed workflow.

We assume that function $inst^{\pi}$ defines the instantiation of ports and that function $inst^{\sigma}$ defines the instantiation of services. As said in Section 5.3.2, concrete ports are instantiated from abstract ports and tasks are instantiated from services. $inst^{\pi}$ creates new port(s) from given port(s) and $inst^{\sigma}$ creates a new task from a given service. A service can be instantiated several times, it produces tasks that are different in the workflow.

The insertion of a service, defined by function $insert^{\sigma}$, corresponds to the addition of a new task to the workflow, and the addition of links between ports of the task and ports of the focus. The task is instantiated from a service of the suggestions. Function $insert^{\sigma}$ uses the focus and returns the new workflow after insertion.

Definition 11 (task insertion).

Let $(\sigma, \varphi^{in}, \varphi^{out}) \in sugg^{\sigma}(W, F)$, insertion is defined by $insert^{\sigma}(W, F, (\sigma, \varphi^{in}, \varphi^{out})) = W'$, where

$$\begin{aligned} t &= inst^{\sigma}(\sigma), \\ W'.P &= W.P \cup t.P^{in} \cup t.P^{out}, \\ W'.T &= W.T \cup \{t\}, \\ W'.L &= W.L \cup \{(p, p') \mid p \in F^{out} \wedge p' = inst^{\pi}(\varphi^{in}(p))\} \\ &\qquad \cup \{(p', p) \mid p \in F^{in} \wedge p' = inst^{\pi}(\varphi^{out}(p))\} \end{aligned}$$

The insertion of a link is defined by function $insert^{l}$ that takes a current workflow and a suggested link, and provides the new workflow.

Definition 12 (link insertion).

Let $(p_{out}, p_{in}) \in sugg^{l}(W)$, insertion is defined by $insert^{l}(W, (p_{out}, p_{in})) = W'$, where

$$W'.P = W.P, \quad W'.T = W.T, \quad W'.L = W.L \cup \{(p_{out}, p_{in})\}$$

From a practical point of view, links actually contain converters when needed. As shown in Chapter 4, those converters are automatically generated. Note that users may have to make a choice, when several converters are generated. The removal of links and tasks from a workflow is, respectively, defined by functions $remove^{l}$ and $remove^{t}$.

Definition 13 (link removal).

$remove^l(W, (p_{out}, p_{in})) = W'$, where

$$W'.P = W.P, \quad W'.T = W.T, \quad W'.L = W.L \setminus \{(p_{out}, p_{in})\}$$

Definition 14 (service removal).

$remove^t(W, t) = W'$, where

$$\begin{aligned} &W'.P = W.P \setminus t.P^{in} \cup t.P^{out}, \\ &W'.T = W.T \setminus \{t\}, \\ &W'.L = W.L \setminus \{(p_{out}, p_{in}) \in W.L \mid p_{out} \in t.P^{out} \vee p_{in} \in t.P^{in}\} \end{aligned}$$

The removal of a task implies the removal of all links connected to the task.

5.4 Properties of our Approach

At each step of the composition of a workflow with our approach, users benefit suggestions of links and services. Instead of suggesting in a open manner without any control, we prevent suggesting incompatible workflow components (services and links that violate the workflow well-formedness). This enables to let the users select and assemble the workflow components without worrying about data incompatibilities and incoherence that we do automatically manage. Thus, we save the users incompatibility and incoherence errors that can consume time during workflow composition process. In the following, we show that our approach enables to keep workflow under composition well-formed regardless transformations made by a user, and that it is possible to compose any well-formed workflow by a finite sequence of transformations. We prove safeness and completeness : the workflow obtained after applying a transformation on a well-formed workflow is well-formed, and it is possible to compose any well-formed workflow. We provide lemmas and their proofs for each transformation. Safeness and completeness proofs are a preliminary evaluation of our approach. As discussed in Chapter 6, a perspective is to evaluate usability of our approach, compared to existing approach.

Lemma 1 (insertion of a service).

Let W be a well-formed workflow and F a focus. If a suggested service $x \in sugg^{\sigma}(W, F)$ is inserted in the workflow W, then the new workflow $W' = insert^{\sigma}(W, F, x)$ is well-formed.

Proof. *The proof is based on conditions involved by considering a suggestion* $x = (\sigma, \varphi^{in}, \varphi^{out})$*, and the insertion of* x *with function* $insert^{\sigma}$ *:*

— $x \in sugg^{\sigma}(W, F)$ *implies the following conditions (from definition 9) :*

A1. $\forall p_{in} \in F^{in}, \forall p_{out} \in W.OP : (p_{out}, p_{in}) \notin W.L$

A2. $\forall p_o \in F^{out}, \forall p_i \in F^{in} : p_o$ *does not depend on* p_i

A3. $\varphi^{in} \in F^{out} \rightarrow \sigma.\Pi^{in}$

A4. φ^{in} *is injective*

A5. $\forall p \in F^{out} : (p.\pi.type, \varphi^{in}(p).type) \in \Lambda$

A6. $\varphi^{out} \in F^{in} \rightarrow \sigma.\Pi^{out}$

A7. $\forall p \in F^{in} : (\varphi^{out}(p).type, p.\pi.type) \in \Lambda$

— $insert^{\sigma}(W, F, x) = W'$ *implies the following conditions (from definition 11) :*

B1. $t = inst^{\sigma}(\sigma)$

B2. $W'.P = W.P \cup t.P^{in} \cup t.P^{out}$

B3. $W'.T = W.T \cup \{t\}$

B4. $W'.L = W.L \cup \{(p, inst^{\pi}(\varphi^{in}(p))) \mid p \in F^{out}\} \cup \{(inst^{\pi}(\varphi^{out}(p)), p) \mid p \in F^{in}\}$

With the previous conditions, it suffices to prove that (1) W' *is a workflow , and prove that* W' *is well-formed : (2) no port of* W' *depends upon itself, (3) each input port of* W' *is, at most, the target of one link, and (4) each link* (p_{out}, p_{in}) *verifies convertibility* $(p_{out}.\pi.type, p_{in}.\pi.type) \in t.P^{out}\}$.

1. W' *is a workflow :* W *is a workflow and, from (B2) and (B3) a new task and new ports (defined with the task) are added to* W*. From (B4) new links are added to* W*, and each link is defined from an output port to an input port.*

2. *No port of* W' *depends on itself : If a port depends on itself after insertion, then there is an input port* $p_{in} \in F^{in}$ *and an output port* $p_{out} \in F^{out}$ *such that* p_{out} *depends on* p_{in} *in* W*, which is not possible according to (A2).*

3. *Each input port of W' is, at most, the target of one link : If there is an input port p_{in} that is target of two links after insertion, then either p_{in} is involved in a link in W, or two links involving p_{in} are inserted. p_{in} cannot be involved in a link in W according to (A1), and it is not possible to insert two links involving p_{in} according to (A3) and (A4).*
4. *Each link (p_{out}, p_{in}) establishes that $(p_{out}.\pi.type, p_{in}.\pi.type) \in \Lambda$: It is true according to (A5) and (A7).* ■

Lemma 2 (insertion of a link).

Let W be a well-formed workflow. If a suggested link $x \in sugg^l(W)$ is inserted in the workflow W, then the new workflow
$W' = insert^l(W, x)$ is well-formed.

Proof. *We use the same reasoning as the proof of Lemma 1. The proof is based on conditions involved by considering a suggestion $x = (p_{out}, p_{in})$, and the insertion of x with function $insert^l$:*

— *$(p_{out}, p_{in}) \in sugg^l(W)$ implies the following conditions (from definition 10) :*
  A1. $p_{out} \in W.OP$
  A2. $p_{in} \in W.IP$
  A3. $\forall p \in W.OP : (p, p_{in}) \notin W.L$
  A4. p_{out} does not depend on p_{in}
  A5. $(p_{out}.\pi.type, p_{in}.\pi.type) \in \Lambda$

— *$insert^l(W, (p_{out}, p_{in})) = W'$ implies the following conditions (from definition 12) :*
  B1. $W'.P = W.P$
  B2. $W'.T = W.T$
  B3. $W'.L = W.L \cup \{(p_{out}, p_{in})\}$

With the previous conditions, it suffices to prove that (1) W' is a workflow, and prove that W' is well-formed : (2) no port of W' depends upon itself, (3) each input port of W' is, at most, the target of one link, and (4) each link (p_{out}, p_{in}) verifies convertibility $(p_{out}.\pi.type, p_{in}.\pi.type) \in t.P^{out}$.

1. *W' is a workflow. Because W is a workflow, and with (B1) and (B2) no task and port is added to W. With (B3) a link is added to W, and the link is defined from an output port to an input port.*
2. *No port of W' depends upon itself. If a port depends to itself after insertion of (p_{out}, p_{in}), then p_{out} depends on p_{in} in W, which is not possible according to (A4).*

3. *Each input port of W' is, at most, the target of one link. Because if there is an input port p_{in} that is target of two links after insertion, then either p_{in} is involved in a link in W, or two links involving p_{in} are inserted. p_{in} cannot be involved in a link in W according to (A3), and it is no possible to insert two links involving p_{in} according to (A4).*
4. *Each link (p_{out}, p_{in}) establishes that $(p_{out}.\pi.type, p_{in}.\pi.type) \in \Lambda$ according to (A5).* ■

Lemma 3 (removal of tasks and links).

Let W be a well-formed workflow. If a task or a link is removed from the workflow W, then the new workflow W' is well-formed.

Proof. *Removal of tasks and links can only remove dependencies between ports. Hence, no circular dependency can appear, no input port can become the target of several links.* ■

Definition 15 (empty workflow).

The empty workflow is the workflow that has no task, hence no ports, and no links, apart the special tasks t_{getter} and t_{setter} and their ports.

Theorem 3 (safeness).

Starting with the empty workflow, every sequence of transformations leads to a well-formed workflow.

Proof. *The empty workflow is well-formed because it does not contain links (e.g., no dependencies). Lemmas 1, 2 and 3 prove that each transformation on a well-formed workflow provides a well-formed workflow. Thus, by induction, any sequence of transformations leads to a well-formed workflow.* ■

Theorem 4 (completeness).

For every well-formed workflow W, there is a finite sequence of transformations starting with the empty workflow.

Proof. *We prove the theorem by induction on well-formed workflows with decreasing size, i.e. number of tasks and links. If W is the empty workflow, then no insertion at all is required to reach W. Otherwise, W has at least one task or one link. Suppose that $W = (P, T, L)$ has one link $x = (p_{out}, p_{in})$, and let us define a smaller workflow $W' = remove^l(W, x) = (P, T, L \setminus \{x\})$. By Lemma 3, W' is well-formed. Therefore, by induction hypothesis, there exists a finite sequence of suggestion insertion from the empty workflow to W'. It remains to be proved that (1) x is a suggested link of W', and (2) that $W = insert^l(W', x)$.*

1. *$x \in sugg^l(W')$: Port p_{out} is obviously an output port, and p_{in} an input port. p_{out} cannot depend on p_{in} in W' because otherwise, there would be a circular dependency in W going through p_{out} and p_{in}, which would contradict the well-formedness of W. Finally, link x is well-typed because it belongs to W, and W is well-formed.*
2. *$W = insert^l(W', x)$: We have $insert^l(W', x) = (P, T, (L \setminus \{x\}) \cup \{x\}) = (P, T, L) = W$.*

The remaining case to be considered is when W is not empty and has no link ($L = \emptyset$). W has at least one task t. Let us define a smaller workflow $W' = remove^t(W, t) = (P \setminus (t.P^{in} \cup t.P^{out}), T \setminus \{t\}, \emptyset)$. From Lemma 3, W' is well-formed. From inductive hypothesis, there exists a finite sequence of suggestion insertion from the empty workflow to W'. It remains to be proved that (1) $t.\sigma$ is a suggested service of W' for some focus F, and that (2) $W = insert^\sigma(W', F, x)$.

1. *$x = (t.\sigma, \varphi^{in}, \varphi^{out}) \in sugg^\sigma(W', F)$ for some focus F : Workflow W adds no link to W' (W has no link), we can set $F = \emptyset$, and φ^{in}, φ^{out} are empty mappings. From that, every service in Σ, and $t.\sigma$ in particular, is a suggested service.*
2. *$W = insert^\sigma(W', F, x)$: We have $insert^\sigma(W', F, x) = ((P \setminus (t.P^{in} \cup t.P^{out})) \cup (t.P^{in} \cup t.P^{out}), (T \setminus \{t\}) \cup \{t\}, \emptyset \cup \emptyset \cup \emptyset) = (P, T, \emptyset) = W$.*

The proof is based on a sequence of insertions that first inserts all tasks, and then inserts all links. Of course, this is in general neither the only sequence, nor the most plausible one. It simply makes the proof easier. ■

5.5 Implementation

We developed a proof-of-concept prototype. The prototype is named *FlowLis*. FlowLis uses the resources and convertibility program presented in Section 4.4. We added models for workflow and dataflow as well as implementation for the properties, suggestions and transformation presented in Section 5.3. The implementation uses XML and JSON technologies to contain data. It re-uses JGraphT and JGraphX APIs to manage the workflow and the dataflow representations.

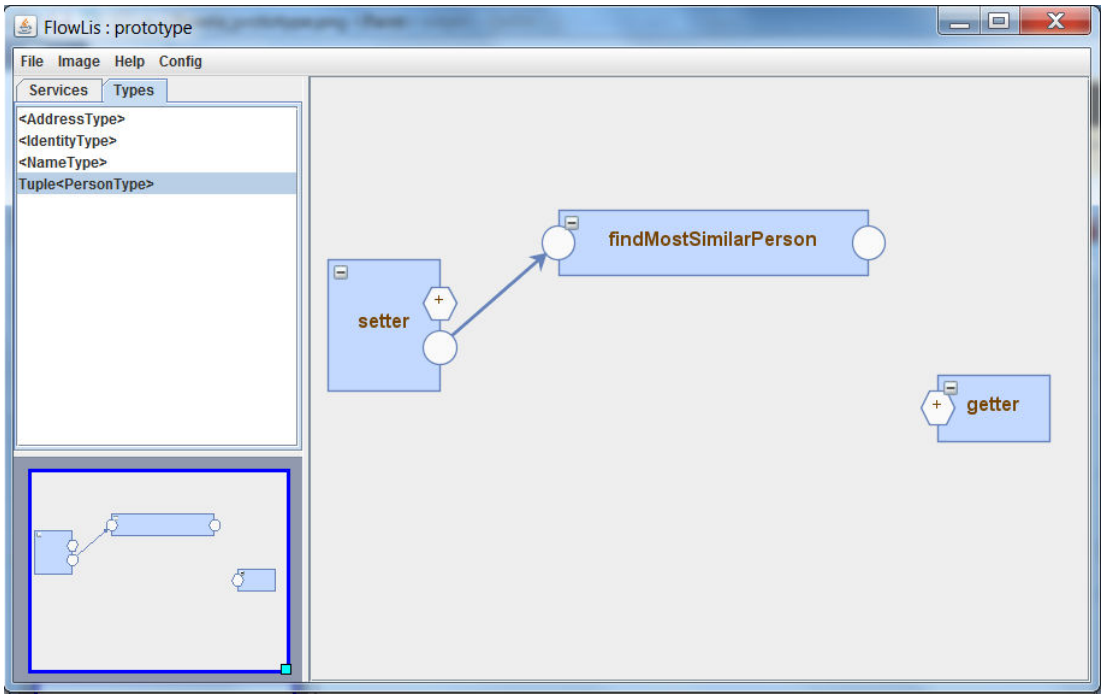

FIGURE 5.4 – GUI of FlowLis.

FlowLis provides real environment to consider tasks and ports from existing service resources to draw well-formed workflows. Its knowledge base, represented in XML, is composed of the service and convertibility informations presented in Chapter 4. It provides suggestion engines and transformations mechanisms that users can handle. Figure 5.4 shows the GUI of FlowLis. On the left, it has a panel to list services and types. On the right, a panel displays the workflow under construction. A workflow is initially composed of tasks *getter* and *setter*. New tasks can be added to the workflow by drag-and-drop from the list of services. New ports and their types can be added to the tasks *getter* and *setter* from the list of types. It enables to consider new data types to consume or to produce. At bottom left, a outline panel is provided to zoom-in or zoom-out the components of a workflow. A menu enables to configure the system and its resources.

Figure 5.5 shows a JSON-formatted example of a service description as it is managed in our system. The service is named *findMostSimilarPerson*, it takes *PersonType* as type for its input and output ports. Each port is identified by a *paramName*. The links serve to access *PersonType* definition and others facets of the services. Figure 5.6 shows the JSON-formatted definition of *PersonType* as it is managed in our system. *PersonType* is a tuple composed of types *IdentityType*, *NameType*, *AddressType* and *PhoneType*.

```
{
    "name": "findMostSimilarPerson",
    "inputs": [
        {
            "paramName": "param0",
            "type": {
                "label": "PersonType",
                "link": "fp:types/PersonType"
            }
        }
    ],
    "outputs": [
        {
            "paramName": "param0",
            "type": {
                "label": "PersonType",
                "link": "fp:/types/PersonType"
            }
        }
    ],
    "link": "fp:services/findMostSimilarPerson"
}
```

Figure 5.5 – A JSON formatted service description as it can be managed into FlowLis.

Services and types descriptions can be added from existing service wrappers by means of transducers. They can be managed by means of XML and JSON files, or through RESTful services presented in Section 4.4. Actually, we define by hand the input and output types for the services we use because they do not have formal type definitions (they use textual formats in their inputs and outputs). As presented in Section 4.5.1, the type definitions, respecting existing formats, are used to set inputs and outputs of a new service to consider.

In our system, we separate users that configure the system from final users that draw workflows. Configuring the system implies adding new services, defining new types or new transducers as well as setting the others aspects of the system. Users that draw workflows work with resources already present in the knowledge base, they do not have to worry about configurations.

FlowLis is still rudimentary but it is reactive and already complete for the composition of well-formed workflows. Actually, it contains services and data types extracted from the EMBOSS libraries [RLB00] (services similar to those used in Chapter 4). yet the workflows are not executable, the generation of executable workflows is left to future work.

```
{
    "constr": "tuple",
    "label": "PersonType",
    "elmts": [
        {
            "constr": "tag",
            "label": "identity",
            "content": { "label": "IdendityType" }
        },
        {
            "constr": "tag",
            "label": "name",
            "content": { "label": "NameType" }
        },
        {
            "constr": "tag",
            "label": "address",
            "content": { "label": "AddressType" }
        },
        {
            "constr": "tag",
            "label": "mobile",
            "content": { "label": "PhoneType" }
        }
    ]
}
```

Figure 5.6 – A JSON formatted type expression as it can be managed into FlowLis.

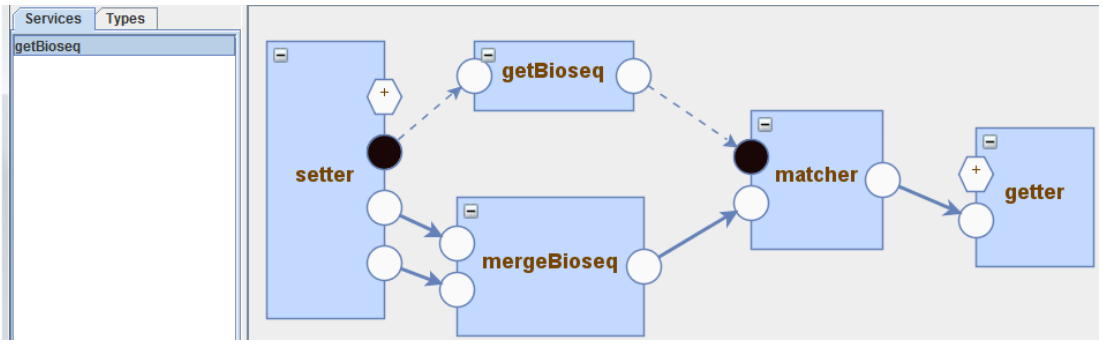

Figure 5.7 – Example of a step during the composition of the workflow at Figure 5.2, where an instance of service *getBioseq* is inserted to a partial workflow.

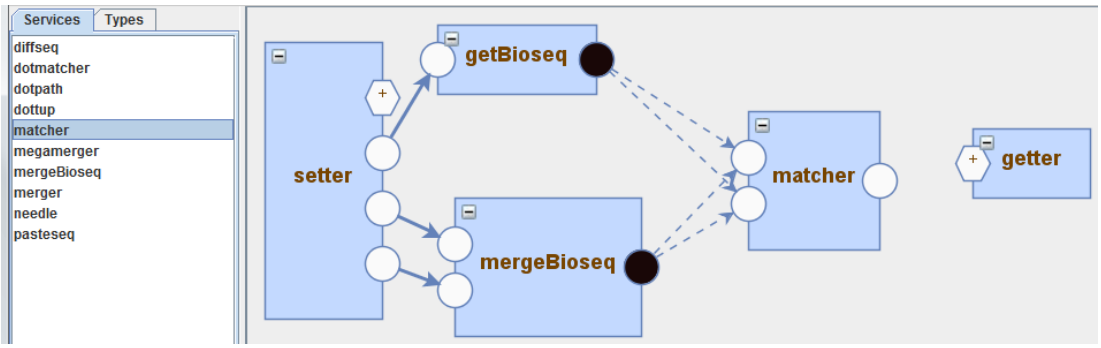

Figure 5.8 – Example of a step during the composition of the workflow of Figure 5.2, where an instance of service *matcher* is inserted into a partial workflow.

5.6 Use Case

We use the workflow depicted in Figure 5.8 to show how services are composed in our approach. The workflow uses data types and services similar to those presented above. As already mentioned, the workflow enables to merge two sequences, the merge produces a sequence that is matched with a given sequence. Services are represented by boxes, their input and output ports are represented by circles. Array represents links between ports. Services, types and convertibilities are defined in the knowledge base of FlowLis.

Figures 5.8 and 5.7 show two examples of service insertion to compose the workflow of Figure 5.2. In each figure, the left part shows a list of suggested services and the right part shows a partial workflow. In partial workflows, black circles represent the focus. Dashed arrows connect the instances of services to be inserted (an instance of *matcher* is added to the partial workflow of figure 5.8 and an instance of *getBioseq* is added to the partial workflow of figure 5.7). Plain

arrows and other connected tasks form the partial workflows, composed of services and links already inserted. Partial workflows contain tasks *getter* and *setter*. As mentioned, ports of tasks *getter* and *setter* and their associated types can be added by users. By default, each port of tasks *setter* and *getter* has the same type as the type of the port it is connected to. The two figures are independent, they are two possible states of the composition of the workflow. Note that each partial workflow is well-formed.

In Figure 5.8, the focus is composed of two output ports of tasks *getBioseq* and *mergeBioseq* that have the same type *Bioseq*. Considering the focus, the user is interested in services having at least two input ports such that each input is convertible from *Bioseq*. According to our system, suggested services at the left of Figure 5.8 verify that condition. Service *matcher* is one of them and can be inserted by selecting the links to accept. The resulting workflow is well-formed.

In Figure 5.7, we suppose that the user decides that the type *Accession* defines the type of the output port of task *setter*. *Accession* is a type that defines reference for biological sequences. Hence, the focus is composed of an output port of task *setter* and an input port of task *matcher*. The two ports of the focus have types *Accession* and *Seq*. According to our system, service *getBioseq* is the only one in the knowledge base that verifies that condition, its output type is *Bioseq* that is convertible to *Seq*. If service *getBioseq* is inserted can be inserted, the resulting workflow becomes fully-defined.

In practice, ports composing the focus are added one by one. For example, in Figure 5.8, if we suppose that the user initially adds the output port of *getBioseq*, the initial suggestions contain more services, for example services providing one output. When the user adds the output port of *mergeBioseq*, some services are filtered out from the suggestions. More generally, any service not compatible with a port p is also not compatible with $F \cup \{p\}$. Thus, in order to obtain the new suggestion list, after addition of a new port to the focus, it is sufficient to remove, from the list of suggestions, services incompatible with the new port. It is not necessary to recompute the suggestions from the knowledge base. The more ports are added to the focus, the more services are filtered out from the suggestions. In Figure 5.7, after insertion of service *getBioseq*, suggestions become empty because the input ports of the focus become connected. The connected input ports are removed from the focus and new suggestions are computed. In workflows, each link represents one conversion. If there were multiple conversions detected in a link, the user is invited to choose one.

5.7 Related Work

Our workflow composition approach is interactive. It integrates users in the process, gives him simple ways to compose workflows and lets him conduct composition while preparing next steps. It offers mechanisms for suggestion and verification of workflow composition during the steps. In addition, in contrast to most existing approach in bioinformatics, our approach uses shims that are systematically generated from the abstraction of input and output data of services. It promotes separation between formats and data types, and separation between domain services and services for data conversion.

Platforms, such as Galaxy [GNTT10] and Taverna [OGA+06a], provide operational contexts to create, discover, enact, reproduce and share workflows. They offer a graphical user interface (GUI) to draw workflows. They use ontologies, such as myGrid [WAH+07] and EDAM [IKJ+13] ontologies, to support automation and reasoning. For example, Dhamanastra et al. [DCZ+12] use service annotations to support suggestions during composition of services in Galaxy. Semantic annotations are used to support service discovery in Taverna [LAWG05]. In platforms, however, users manage conversion of data between services using manual shim that convert formats [RLS+06]. Furthermore, they do not provide support to verify workflow composition.

Wang et al. [WMK+08] provide a framework for service discovery and service composition. As in operational platforms, they provide a GUI and a multifaceted UI Wizard to facilitate selection and composition. They work directly with standard Web Services, and they use semantic annotations from ontologies. Our approach uses service descriptions that consider each function individually, and semantics is provided by our type abstraction that describes data composition and nature. Their approach provides data matching but only considers exact and subsumed matches. Our approach better exploit data structure composition to assist users in type and link verifications. In addition, their approach seems entangled in specific technologies, prejudicing its general use. Wang et al. among others provide control constructs to formalize control structures such as iterative loops, conditional, exception handling. As said, we do not allow those structures yet.

Sirin et al. [SHP03] show the benefits of using semantic descriptions for service selection and composition. They allow users to select services during the workflow composition by proposing mechanisms to filter non-compatible services but they do not verify workflow composition. Kim et al. [KSG04] provide an interactive approach using AI planning techniques. They propose an error scan algorithm

detecting errors and suggesting actions to fix them. Their suggestion mechanism relies on desirable properties of the workflow. The desirable properties concern satisfaction of inputs, usability of data, redundancy of links, executable components, acyclic workflows. We use the same properties in our approach through well-formedness and completeness. They filter and order suggested actions by first considering actions that fix more errors than they cause. Their approach, however, does not, as we can, restrict suggestions to the particular points of the workflow a user is managing. In addition, their approach repairs errors whereas our approach prevents them.

Similarly to Kim et al., Dibernado et al. [DPW08] propose an interactive model for interconnecting services. They, in particular, enable to consider partial data to suggest services to be connected. Services are suggested when they initiate a chain from considered data to a workflow result. Their approach ranks the suggested services according to the length of the path to reach the result. As we discuss it in perspectives, their ranking algorithm could be adapted for our approach. However, as in many existing approaches, their approach does not exploit composite structure of data as much as our approach. It is based on the *has-A* relation of ontologies and input data cannot be recomposed from several parts of output data. To automatically achieve conversion between input and output data of services, they need conversion between data formats to be explicitly defined in ontologies. In addition, they do not offer the filtering capabilities as our focus does (for example, suggest services consuming or/and producing a given set of types).

Besides, Sheng et al [SQV+14] provide a survey of service compositions used both in industrial and research areas. In their paper, service composition life cycle is devised into definition, selection, deployment and execution phases. We are concerned by the definition and selection phases in research service composition, precisely in bioinformatics. We have integrated selection and definition phases through a guided approach of composition. They point out requirements for expressiveness, correctness, automation and selectability for selection and composition aspects. In expressiveness, our approach supports complex data representation, and enables sequence and concurrent workflow constructs. In correctness, our approach checks composition validity through a property of well-formedness that guarantees correctness of the links. In automation, our approach automatically manages compatibility between input and output data of the services. It also provides an interactive model enabling users to define workflows step by step. In selectability, our approach suggests services based on the focus points indicated by the user. The suggested service are compatible at signature level. Furthermore, as discussed in perspectives, we plan to use information from *workflow mining* to

enhance suggestion and verification in workflows.

5.8 Conclusion

In this chapter, we described an approach to guide users compose workflows. The approach is interactive, it captures user intentions, provides suggestions and enables users to define well-formed workflows. It uses convertibility approach provided in Chapter 4 to detect and resolve data mismatches between services. We formalize the components of our approach by applying the LIS framework. We provide proofs of safeness and completeness of user guidance. We also describe our prototype FlowLis and define use case. Compared to existing works, our approach offers more possibilities for suggestions. It enables suggestions from multiple ports, and it takes into account the composite structure of input and output data. In addition, our approach prevents errors by keeping workflows well-formed. It gives simple ways to compose workflows and lets users conduct composition while placing guardrail to prevent them from errors such as incompatibility of services and workflow incoherence. Our approach shows simple ways to help users, specially domain expert users. Even if our approach lacks some features, for example it does not yet take into account other service facets and does not provide ranking for suggestions, we discuss some interesting tracks in perspectives to complete it. A significant advantage is the extensibility of the components of our approach. Each component is extensible to take into account new aspects. For example, suggestions may take into account informations according to other properties of services (e.g., QoS, provider). LIS mechanisms can be used to refine suggestions by navigation and filtering operations.

Chapitre 6

Conclusion générale

6.1 Conclusion

Dans cette thèse, nous avons proposé de faciliter la composition de workflows en passant par deux contributions majeures. La première contribution permet de gérer la convertibilité des sorties vers les entrées des services. La deuxième contribution propose de guider les utilisateurs dans la composition de workflows cohérents par la suggestion de services compatibles.

Notre solution de convertibilité permet de détecter et de résoudre les incompatibilités entre les entrées et sorties de services. Elle passe par une abstraction des types des entrées et sorties des services. Elle utilise un ensemble de règles de convertibilité des types de sortie vers des types d'entrée. L'abstraction de types s'appuie sur le modèle de données XML. Elle prend en compte la composition et la nature des données. Elle permet les types primitifs et les types composites. Les règles de convertibilité donnent les conditions pour que deux types quelconques soient convertibles, et elles donnent en plus une preuve de la convertibilité sous forme d'une fonction de conversion. Elles permettent la composition et la décomposition ainsi que la généralisation et la spécialisation de types. Les règles permettent de générer automatiquement des convertisseurs pour le transfert des données entre services dans les workflows.

Notre approche pour guider les utilisateurs dans la composition de workflows utilise notre solution de convertibilité. Elle est basée sur le modèle des Systèmes

d'Information Logiques (LIS) qui permettent de guider un utilisateur dans l'expression de requêtes et dans la navigation sûre à travers des données représentées avec une logique uniforme. Notre approche adapte les LIS à la composition de workflows. Pour cela, nous formalisons les workflows, les suggestions de services et la composition de workflow. Elle capture l'intention de l'utilisateur à l'aide d'une notion de focus et permet des suggestions cohérentes durant la composition de workflows. Notre approche permet de garantir des workflows bien formés.

Nous avons instancié et évalué nos solutions avec des services et des types en génomique. Nous avons défini des types respectant des formats courants pour des séquences biologiques. Avec un programme défini à partir de nos règles de convertibilité, nous avons généré un graphe de convertibilité qui représente les services et les liaisons détectées entre les services, les liaisons matérialisant les conversions générées entre les entrées et sorties des services. Le graphe, évalué par des experts, montre que notre approche est adaptée. Elle prend en compte la composition de données non prise en compte avec les formats et fournit automatiquement des convertisseurs qui sont en général obtenus de manière manuelle. Notre approche de composition de workflows permet à un utilisateur de composer, étape par étape, un workflow à travers des suggestions et en empêchant les incohérences. Un prototype pour la composition de workflows est disponible. Par rapport aux approches existantes, notre approche offre des suggestions à partir de plusieurs points d'un workflow. Elle considère la structure composite des types d'entrée et sortie des services.

6.2 Perspectives

Le travail effectué ouvre un certain nombre de perspectives. Elles concernent des questions directes posées par les solutions mises au point, par exemple comment traiter les cas de conversions multiples ? Comment prendre en compte d'autres types complexes ? Comment être plus expressifs dans la définition des workflows ? Les perspectives concernent aussi l'amélioration de nos approches, par exemple, par l'intégration d'ontologies et l'exploitation des informations de log ainsi que par la gestion des suggestions durant la composition de workflows.

6.2.1 Non-déterminisme

Une perspective de notre travail concerne la gestion du non-déterminisme dans la convertibilité. Jusque là, nous proposons de gérer les cas de conversions multiples par une intervention utilisateur, c'est-à-dire, présenter les convertisseurs à l'utilisateur pour qu'il choisisse les convertisseurs les mieux adaptés à sa situation.

C'est une solution qui fonctionne dans notre contexte actuel puisque les cas de conversions multiples sont rares. Les types ont une taille raisonnable et les sous types sont annotés de manière pragmatique avec des concepts spécifiques. Cependant, gérer les conversions multiples par le biais de l'utilisateur implique des difficultés dans certaines situations, par exemple, dans le cas où il faut convertir un type de grande taille (e.g., schéma d'une base de donnée textuelle) en type de petite taille (e.g., un élément susceptible d'être présent plusieurs fois dans la base). Dans ce cas, le nombre de choix possibles peut très vite exploser mettant l'utilisateur dans l'impossibilité de choisir. Ainsi, il est nécessaire de réfléchir à des solutions pour faciliter la gestion des cas multiples. Pour cela, il serait intéressant d'utiliser les informations concernant les types et les convertisseurs ainsi que les preuves de convertibilité accompagnant les types et les convertisseurs. Il serait intéressant de compter sur des informations provenant d'ontologies de domaine [IKJ+13, BD13] (voir la section 6.2.4) et de techniques de *workflow mining* [VdAvDH+03] (voir la section 6.2.5). Ces informations peuvent permettre de classer les convertisseurs générés, par ordre de pertinence, selon des critères à définir. Elles peuvent permettre d'ignorer ou de filtrer des convertisseurs en fonction des usages.

De manière globale, nous pensons que l'intervention de l'utilisateur n'est pas suffisante mais est nécessaire dans notre contexte. Il est toujours possible d'éliminer des convertisseurs inadaptés, redondants ou incohérents par filtrage mais il restera toujours des cas nécessitant une intervention extérieure pour compléter la spécification d'une conversion à appliquer. Par exemple, dans le cas où deux parties similaires d'une donnée provenant d'un premier service sont à exploiter de manière exclusive (ou non) pour alimenter d'autres services, il faut une intervention extérieure pour savoir ce qu'il faut faire. Ainsi, il faut non seulement des techniques permettant de filtrer et de classer les convertisseurs pertinents en fonction des usages mais il faut également des techniques interactives souples permettant à l'utilisateur de comprendre et de régler les situations que le système ne peut pas résoudre par lui même.

6.2.2 Abstraction de types

Notre abstraction de types est assez expressive pour couvrir une bonne partie des types de données actuellement utilisés dans les plateformes en bioinformatique. Cependant, il existe des types encore non considérés, par exemple les types récursifs pour les arbres (e.g., les données phylogéniques [HZ09] en bioinformatique qui représentent des individus en fonction d'autres individus). La prise en compte d'attributs dans la définition des types peut également être utile, par exemple pour séparer unités de mesure et valeurs ou pour gérer certains pa-

ramètres des services. L'intégration des types récursifs est assez directe en ce qui concerne l'abstraction de types, elle suivra les solutions apportées dans les langages tels que *XDuce* [HVP00]. Il faut, cependant, réfléchir à l'impact de leur intégration sur la génération des convertisseurs. Une première idée peut s'inspirer des règles de sous-typage et de filtrage (*pattern matching*) proposées dans les solutions *XDuce*, notamment pour les types récursifs.

Par ailleurs, hormis l'enrichissement des constructions de types, c'est toute une politique de gestion des types qu'il est intéressant de développer. En effet, la situation actuelle en bio-informatique favorise la création et la multiplication des formats personnalisés indépendants [WvdVW+09]. Ces formats (types concrets), maintenus par des utilisateurs isolés et dépourvus de schémas (types abstraits) explicites et de documentations formelles, sont difficiles à ré-utiliser. C'est une situation qu'il serait intéressant de résoudre à travers une politique adaptée de gestion de types de données dont l'idée de base pourrait être de promouvoir la réutilisation des types existants en évitant au maximum de ré-inventer. Il conviendra de réfléchir à une approche de gestion des types de données comme c'est le cas actuellement pour la gestion des services et des workflows. Une telle approche peut permettre de composer des types à partir de types de base pour répondre à de nouveaux besoins. Elle pourra s'appuyer sur des mécanismes d'annotation et de filtrage de types [Her12].

6.2.3 Expressivité des workflows

L'expressivité des workflows est un aspect important à considérer dans les perspectives [SQV+14]. Pour le moment, notre approche offre des workflows à travers des structures de séquences de services simples et de séquences de services en parallèles. Comme beaucoup de travaux en bio-informatique, nous nous sommes limités à des structures s'accommodant avec la vision flot de données. Les services dans un workflow forment un graphe de dépendances où les services dépendants sont exécutés dans l'ordre des dépendances et les services indépendants sont exécutés en parallèle. Des structures de contrôle plus élaborées sont possibles, par exemple l'itération, les structures de choix, les variables [vDATHKB03].

Une question que nous nous posons (qui s'est posée, de manière plus générale, avec la mise en place du langage Scufl [LBW+04]) est de savoir quelles structures sont vraiment pertinentes dans notre contexte d'utilisation. Du point de vue de la programmation orientée service toutes les structures sont pertinentes puisqu'il faut envisager tous les besoins et toutes les situations [vDATHKB03]. Cependant, dans notre cas, l'utilisateur final n'a pas réellement besoin de certaines complexités dans la définition des workflows. Des expressions trop complexes rendent les

workflows difficiles à mettre en place et à exploiter de manière automatisée. Elles peuvent même nuire à l'intérêt d'utiliser les workflows. En plus, elles risquent de nuire à l'efficacité de notre système en termes de suggestions et de vérifications. En tenant compte de cela, une perspective serait d'intégrer les structures de workflows nécessaires mais en les accompagnant de mécanismes adaptés permettant à des utilisateurs finaux de facilement les utiliser. Cela pousse à réfléchir à des solutions alliant expressivité et simplicité. Les premières pistes peuvent s'orienter vers des workflows pour définir des workflows. C'est-à-dire, proposer des approches qui s'appuient sur toutes les compétences (informaticiens, bioinformaticiens ou biologistes) pour élaborer des workflows avec différents niveaux d'intervention et différents rôles dédiés. Cela peut s'aligner avec des réflexions sur l'intégration des environnement *Big Data* comme c'est le cas dans des systèmes tels que Kepler [WCA09].

6.2.4 Intégration d'ontologies

Bien que nos approches soient actuellement basées sur la sémantique apportée par les types abstraits des entrées et sorties de services, l'objectif de notre travail, à terme, est d'exploiter toute information concernant les services pour faciliter la composition de workflows. Les ontologies apportent beaucoup d'informations utiles sur les services et leurs domaines d'utilisation. Ces informations concernent d'autres facettes des services, par exemple la nature des services, les algorithmes, les fonctions, les fournisseurs. Elles concernent également des aspects tels que la composition et le fonctionnement des services ainsi que les infrastructures utilisées. Les ontologies telles qu'EDAM [IKJ+13], les langages de description de services tels que WSMO [RKL+05] et les formats de données tels que BioXSD [KPJ+10] peuvent être exploitées sans beaucoup de difficultés dans nos approches. Il serait intéressant de les utiliser pour améliorer la convertibilité des entrées et sorties des services, la suggestion et la vérification durant la composition de services. Elles peuvent également permettre des mécanismes pour le filtrage, le classement et la navigation à travers les ressources concernant les types de données, les services et les workflows. L'utilisation de ces solutions peut passer par les LIS [FR04] qui offrent déjà beaucoup des mécanismes souhaités. Il serait souhaitable d'adapter ces mécanismes à un système de gestion de workflows. Un tel système pourrait permettre à un utilisateur final de ne pas se confronter à des tâches complexes pour définir ou gérer des workflows. Il serait souhaitable, qu'au pire, les utilisateurs aient juste à choisir entre plusieurs voies pertinentes avec assez d'informations pour décider.

6.2.5 Workflow mining

Les données historiques et traces collectées durant le cycle de vie des workflows sont très précieuses. C'est pourquoi, des techniques de *workflow mining* sont utilisées sur ces données pour améliorer les conditions de composition et de gestion des workflows [Bos08, ZHvdA11]. Une idée qu'il serait intéressant de développer, à travers les techniques de *workflow mining*, est de s'informer sur des aspects tels que le comportement des utilisateurs, les tâches généralement associées et les paramètres habituellement appliqués lors de la construction et de l'exécution des workflows. Ces informations permettront d'améliorer les mécanismes de notre système, par exemple, un modèle comportemental issu d'une analyse d'un réseau d'utilisateurs va permettre d'anticiper les besoins d'un utilisateur particulier. L'analyse du nombre de fois que des services fonctionnent ensembles va permettre d'évaluer la pertinence d'une liaison (convertibilité, conversion) entre des services particuliers. L'analyse du nombre de fois qu'un service a fonctionné dans un contexte donné va permettre d'évaluer la pertinence de services similaires. Les analyses vont être à la base de mécanismes sophistiqués pour classer, suggérer ou vérifier durant la sélection et la composition de services. Il faut, cependant, avoir mis en place des techniques permettant, d'une part, de collecter les données et, d'autre part, de les analyser. Pour l'aspect collecte de données, on pourrait compter sur des solutions existantes, par exemple les langages tels que CWL [1], les formats pour représenter des logs [VdAWM04] et les solutions de gestion de provenance des données [BBD+09], évoquées dans le chapitre 2. Concernant l'aspect analyse, on pourrait compter sur des techniques statistiques, de fouilles de données et l'utilisation des LIS.

6.2.6 Suggestion

Durant la composition de workflows, les suggestions de services et de liaisons peuvent être nombreuses malgré les contraintes du focus. Il est nécessaire de mettre en place des techniques de classement et de filtrage des suggestions. Les techniques peuvent s'inspirer de solutions utilisées dans certaines approches. Par exemple, DiBernado et al. [DPW08] proposent de suggérer à partir des services qui forment le plus court chemin entre une donnée initiale et une donnée finale. Cette solution peut être adaptée à notre cas. Cependant, il faut tenir compte de la notion de focus. Il s'agira d'analyser des plus courts chemins à travers les ports du focus. Il serait également intéressant de réfléchir sur le fait que former le plus court chemin ne rend pas forcément un groupe de services plus adapté que d'autres. Des services très adaptés peuvent ne pas former un chemin court. Des critères supplémentaires peuvent permettre d'améliorer le classement des sug-

1. http ://common-workflow-language.github.io/

gestions [WGMK10]. Comme nous l'avons discuté dans les parties précédentes, les informations sur la description des ressources (Section 6.2.4) et des informations provenant de techniques de *workflow mining* (Section 6.2.5) peuvent être intéressantes pour le classement des ressources.

Par ailleurs, à travers le focus, il est question de requêtes implicites évoquant des services et des liaisons respectant les contraintes du focus. Ces requêtes implicites peuvent être élargies pour permettre de modifier les contraintes (augmenter ou relâcher) dans le but d'orienter les suggestions. Elles peuvent impliquer des vues additionnelles par des opérations telles que la négation, l'union ou la différence. C'est toute une théorie sur la suggestion qui peut être développée. Elle concernera les bases sur lesquelles une suggestion est faite, ce qu'il faut suggérer et comment il faut le suggérer.

Bibliographie

[ABJ+04] Ilkay Altintas, Chad Berkley, Efrat Jaeger, Matthew B. Jones, Bertram Ludäscher et Steve Mock : Kepler : An extensible system for design and execution of scientific workflows. In International Conference on Scientific and Statistical Database Management (SSDBM), pages 423–424. IEEE Computer Society, 2004.

[ACKM03] Gustavo Alonso, Fabio Casati, Harumi Kuno et Vijay Machiraju : Web Services : Concepts, Architectures and Applications. Springer, 2003.

[ACMS08] Vikas Agarwal, Girish Chafle, Sumit Mittal et Biplav Srivastava : Understanding approaches for web service composition and execution. In R. K. Shyamasundar, éditeur : Bangalore annual Compute conference, page 1. ACM, 2008.

[AFG+03] M. Addis, J. Ferris, M. Greenwood, P. Li, D. Marvin, T. Oinn et A. Wipat : Experiences with e-Science workflow specification and enactment in bioinformatics. UK e-Science All Hands Meeting, pages 459–467, 2003.

[AHS04] Sudhir Agarwal, Siegfried Handschuh et Steffen Staab : Annotation, composition and invocation of semantic web services. Web Semantics : Science, Services and Agents on the World Wide Web, 2(1):31–48, 2004.

[AK07] José Luis Ambite et Dipsy Kapoor : Automatically composing data workflows with relational descriptions and shim services. In Karl Aberer, Key-Sun Choi, Natasha Fridman Noy, Dean Allemang, Kyung-Il Lee, Lyndon J. B. Nixon, Jennifer Golbeck, Peter Mika, Diana Maynard, Riichiro Mizoguchi, Guus Schreiber et

Philippe Cudré-Mauroux, éditeurs : The Semantic Web, International Semantic Web Conference, volume 4825 de LNCS, pages 15–29. Springer, 2007.

[All11] Pierre Allard : Logical modeling of multidimensional analysis of multivalued relations - Application to geographic data exploration. Thèse de doctorat, Thèse de l'Université de Rennes 1 - École doctorale MATISSE, 12 décembre 2011. supervised by S. Ferré and O. Ridoux.

[Alo02] Gustavo Alonso : Myths around web services. IEEE Data Engineering Bulletin, 25(4):3–9, 2002.

[AM07] Gennady Agre et Zlatina Marinova : An INFRAWEBS approach to dynamic composition of semantic web services. Cybernetics and Information Technologies, 7(1):45–61, 2007.

[BBD+07] Duncan A Brown, Patrick R Brady, Alexander Dietz, Junwei Cao, Ben Johnson et John McNabb : A case study on the use of workflow technologies for scientific analysis : Gravitational wave data analysis. In Workflows for e-Science, pages 39–59. Springer, 2007.

[BBD+09] Zhuowei Bao, Sarah Cohen Boulakia, Susan B. Davidson, Anat Eyal et Sanjeev Khanna : Differencing provenance in scientific workflows. In Yannis E. Ioannidis, Dik Lun Lee et Raymond T. Ng, éditeurs : International Conference on Data Engineering (ICDE), pages 808–819. IEEE, 2009.

[BC85] Joseph L. Bates et Robert L. Constable : Proofs as programs. ACM Transactions on Programming Languages and Systems (TOPLAS), 7(1):113–136, 1985.

[BD13] Mouhamadou Ba et Gayo Diallo : Large-scale biomedical ontology matching with servomap. IRBM, 34(1):56–59, 2013.

[BDGMC06] Daniela Berardi, Giuseppe De Giacomo, Massimo Mecella et Diego Calvanese : Automatic web service composition : Service-tailored vs. client-tailored approaches. AI for Service Composition, page 63, 2006.

[BFD15a] Mouhamadou Ba, Sébastien Ferré et Mireille Ducassé : Safe suggestions based on type convertibility to guide workflow composition. In Mohand-Said Hacid Zbigniew W. Ras Stefano Ferilli Floriana Esposito, Olivier Pivert, éditeur : International Symposium on Foundations of Intelligent Systems (ISMIS), LNCS 9384. Springer, 2015.

[BFD15b] Mouhamadou Ba, Sébastien Ferré et Mireille Ducassé : Solving data mismatches in bioinformatics workflows by generating data converters. 2015. accepted for publication.

[BFRQ07] Olivier Bedel, Sébastien Ferré, Olivier Ridoux et Erwan Quesseveur : GEOLIS : a logical information system for geographical data. Revue Internationale de Géomatique, 17(3-4):371–390, 2007.

[BG07] Roger Barga et Dennis Gannon : Scientific versus business workflows. In Workflows for e-Science, pages 9–16. Springer, 2007.

[BL04] Shawn Bowers et Bertram Ludäscher : An ontology-driven framework for data transformation in scientific workflows. In Erhard Rahm, éditeur : International Workshop on Data Integration in the Life Sciences (DILS), volume 2994 de LNCS, pages 1–16. Springer, 2004.

[BMC+12] Paul Biron, Ashok Malhotra, World Wide Web Consortium et al. : XML schema definition language (XSD) 1.1 part 2 : Datatypes. W3C Recommendation, 2012.

[Bos08] Aishwarya Bose : Effective web service discovery using a combination of a semantic model and a data mining technique. Mémoire de D.E.A., Queensland University of Technology, 2008.

[BTN+10] Jiten Bhagat, Franck Tanoh, Eric Nzuobontane, Thomas Laurent, Jerzy Orlowski, Marco Roos, Katy Wolstencroft, Sergejs Aleksejevs, Robert Stevens, Steve Pettifer, Rodrigo Lopez et Carole A. Goble : Biocatalogue : a universal catalogue of web services for the life sciences. Nucleic Acids Research, 38(Web-Server-Issue):689–694, 2010.

[C+10] UniProt Consortium et al. : The universal protein resource (uniprot) in 2010. Nucleic Acids Research, 38(Database-Issue):142–148, 2010.

[CBFC12] Sarah Cohen-Boulakia, Christine Froidevaux et Jiuqiang Chen : Reecriture de workflows scientifiques et provenance. In Proc. of the 28th Journees de Bases de Donnees Avancees, 2012.

[CCS93] E.F. Codd, S.B. Codd et C.T. Salley : Providing OLAP (On-line Analytical Processing) to User-Analysts : An IT Mandate. Codd & Date, Inc, San Jose, 1993.

[Chi02] Marina Chicurel : Bioinformatics : Bringing it all together technology feature. Nature, 419(6908):751–757, 2002.

[CHS+03] Peter A Covitz, Frank Hartel, Carl Schaefer, Sherri De Coronado, Gilberto Fragoso, Himanso Sahni, Scott Gustafson et Kenneth H

Buetow : caCORE : A common infrastructure for cancer informatics. Bioinformatics, 19(18):2404–2412, 2003.

[CIJ+00] Fabio Casati, Ski Ilnicki, Li-jie Jin, Vasudev Krishnamoorthy et Ming-Chien Shan : Adaptive and dynamic service composition in eFlow. In Benkt Wangler et Lars Bergman, éditeurs : International Conference on Advanced Information Systems Engineering, volume 1789 de LNCS, pages 13–31. Springer, 2000.

[CKM+03] Francisco Curbera, Rania Khalaf, Nirmal Mukhi, Stefan Tai et Sanjiva Weerawarana : The next step in web services. Communications of the ACM, 46(10):29–34, 2003.

[Cun02] Hamish Cunningham : Gate, a general architecture for text engineering. Computers and the Humanities, 36(2):223–254, 2002.

[CWD+06] Zhiwei Chen, Jian Wu, ShuiGuang Deng, Ying Li et Zhaohui Wu : Describing and verifying web service using type theory. In International Conference on CSCW in Design (CSCWD), pages 746–750. IEEE, 2006.

[DCBE+07] Susan B. Davidson, Sarah Cohen-Boulakia, Anat Eyal, Bertram Ludäscher, Timothy M. McPhillips, Shawn Bowers, Manish Kumar Anand et Juliana Freire : Provenance in scientific workflow systems. IEEE Data Eng. Bull., 30(4):44–50, 2007.

[DCZ+12] Alok Dhamanaskar, Michael E Cotterell, Jie Zheng, Jessica C Kissinger, Christian J Stoeckert Jr et John A Miller : Suggestions for galaxy workflow design using semantically annotated services. In Maureen Donnelly et Giancarlo Guizzardi, éditeurs : International Conference on Formal Ontology in Information Systems (FOIS), volume 239, pages 29–42. IOS Press, 2012.

[DF08] Mireille Ducassé et Sébastien Ferré : Fair(er) and (almost) serene committee meetings with logical and formal concept analysis. In P. Eklund et O. Haemmerlé, éditeurs : Proceedings of the International Conference on Conceptual Structures, LNAI 5113, pages 217–230. Springer, July 2008.

[DG08] Jeffrey Dean et Sanjay Ghemawat : Mapreduce : simplified data processing on large clusters. Communications of the ACM, 51(1): 107–113, 2008.

[DJD+01] Robin D. Dowell, Rodney M. Jokerst, Allen Day, Sean R. Eddy et Lincoln Stein : The distributed annotation system. BMC Bioinformatics, 2:7, 2001.

[DMK+05] Steffen Durinck, Yves Moreau, Arek Kasprzyk, Sean Davis, Bart De Moor, Alvis Brazma et Wolfgang Huber : BioMart and Bio-

conductor : a powerful link between biological databases and microarray data analysis. Bioinformatics, 21(16):3439–3440, 2005.

[DPW08] Michael DiBernardo, Rachel Pottinger et Mark Wilkinson : Semi-automatic web service composition for the life sciences using the biomoby semantic web framework. Journal of Biomedical Informatics, 41(5):837–847, 2008.

[DS05] Schahram Dustdar et Wolfgang Schreiner : A survey on web services composition. International journal of web and grid services, 1(1):1–30, 2005.

[DSW06] Marlon Dumas, Murray Spork et Kenneth Wang : Adapt or perish : Algebra and visual notation for service interface adaptation. In Schahram Dustdar, José Luiz Fiadeiro et Amit P. Sheth, éditeurs : International Conference Business Process Management (BPM), volume 4102 de LNCS, pages 65–80. Springer, 2006.

[ECH10] Emad Elabd, Emmanuel Coquery et Mohand-Said Hacid : Selecting web services for choreography implementation : Compatibility checking approach with access control. In Knowledge Systems Institute Graduate School, éditeur : International Conference on Software Engineering and Knowledge Engineering, pages 235–240, 2010.

[EDG+13] Perla Velasco Elizondo, Vishal Dwivedi, David Garlan, Bradley R. Schmerl et José Maria Fernandes : Resolving data mismatches in end-user compositions. In Yvonne Dittrich, Margaret M. Burnett, Anders I. Mørch et David F. Redmiles, éditeurs : International Symposium on End-User Development (IS-EUD), volume 7897 de LNCS, pages 120–136. Springer, 2013.

[EGK+02] John Ellson, Emden Gansner, Lefteris Koutsofios, Stephen C North et Gordon Woodhull : Graphviz : open source graph drawing tools. In Graph Drawing, pages 483–484. Springer, 2002.

[EXD04] David W. Embley, Li Xu et Yihong Ding : Automatic direct and indirect schema mapping : Experiences and lessons learned. ACM SIGMod Record, 33(4):14–19, 2004.

[Fer09] Sébastien Ferré : Camelis : a logical information system to organise and browse a collection of documents. International Journal of General Systems, 38(4):379–403, 2009.

[Fer14] Sébastien Ferré : Reconciling Expressivity and Usability in Information Access - From Filesystems to the Semantic Web. Habilitation thesis, Matisse, Univ. Rennes 1, 2014. Habilitation à Diriger des Recherches (HDR), defended on November 6th.

[FFST11] Dieter Fensel, Federico Michele Facca, Elena Simperl et Ioan Toma : Semantic Web Services. Springer Science & Business Media, 2011.

[FH11] S. Ferré et A. Hermann : Semantic search : Reconciling expressive querying and exploratory search. In L. Aroyo et C. Welty, éditeurs : International Semantic Web Conference, LNCS 7031, pages 177–192. Springer, 2011.

[FR04] S. Ferré et O. Ridoux : An introduction to logical information systems. Information Processing & Management, 40(3):383–419, 2004.

[FW04] David C Fallside et Priscilla Walmsley : Xml schema part 0 : Primer second edition. W3C recommendation, 2004.

[GBA+10] Carole A Goble, Jiten Bhagat, Sergejs Aleksejevs, Don Cruickshank, Danius Michaelides, David Newman, Mark Borkum, Sean Bechhofer, Marco Roos, Peter Li et al. : myexperiment : a repository and social network for the sharing of bioinformatics workflows. Nucleic acids research, 38(suppl 2):W677–W682, 2010.

[GDE+07] Yolanda Gil, Ewa Deelman, Mark Ellisman, Thomas Fahringer, Geoffrey Fox, Dennis Gannon, Carole Goble, Miron Livny, Luc Moreau et Jim Myers : Examining the challenges of scientific workflows. IEEE computer, 40(12):26–34, 2007.

[GEW01] Michael Girdley, Sandra L Emerson et Rob Woollen : J2EE Applications and BEA WebLogic Servers. Prentice Hall PTR, 2001.

[GF12] Michael Y. Galperin et Xosé M. Fernández-Suarez : The 2012 nucleic acids research database issue and the online molecular biology database collection. Nucleic Acids Research, 40(Database-Issue):1–8, 2012.

[Gil02] Don Gilbert : Pise : Software for building bioinformatics webs. Briefings in bioinformatics, 3(4):405–409, 2002.

[Gil07] Yolanda Gil : Workflow composition : Semantic representations for flexible automation. In Workflows for e-Science, pages 244–257. Springer, 2007.

[GKA+11] Sveinung Gundersen, Matús Kalas, Osman Abul, Arnoldo Frigessi, Eivind Hovig et Geir Kjetil Sandve : Identifying elemental genomic track types and representing them uniformly. BMC Bioinformatics, 12:494, 2011.

[GNTT10] Jeremy Goecks, Anton Nekrutenko, James Taylor et The Galaxy Team : Galaxy : a comprehensive approach for supporting ac-

cessible, reproducible, and transparent computational research in the life sciences. Genome Biology, 11(8):R86, 2010.

[GRH+05] Belinda Giardine, Cathy Riemer, Ross C Hardison, Richard Burhans, Laura Elnitski, Prachi Shah, Yi Zhang, Daniel Blankenberg, Istvan Albert, James Taylor et al. : Galaxy : a platform for interactive large-scale genome analysis. Genome research, 15(10):1451–1455, 2005.

[GSH+08] Carole Goble, Robert Stevens, Duncan Hull, Katy Wolstencroft et Rodrigo Lopez : Data curation + process curation = data integration + science. Briefings in bioinformatics, 9(6):506–517, 2008.

[GSN+01] Carole A. Goble, Robert Stevens, Gary Ng, Sean Bechhofer, Norman W. Paton, Patricia G. Baker, Martin Peim et Andy Brass : Transparent access to multiple bioinformatics information sources. IBM Systems Journal, 40(2):532–551, 2001.

[GW12] Bernhard Ganter et Rudolf Wille : Formal Concept Analysis : Mathematical Foundations. Springer Science & Business Media, 2012.

[GWS+02] R. Mark Greenwood, Chris Wroe, Robert Stevens, Carole A. Goble et Matthew Addis : Position paper : Are bioinformaticians doing e-business ? In Brian Matthews, F. Robert A. Hopgood et Michael D. Wilson, éditeurs : The Web and the Grid : from e-science to e-business (EuroWeb), Workshops in Computing. BCS, 2002.

[GWS03] Carole Goble, Chris Wroe et Robert Stevens : The mygrid project : services, architecture and demonstrator. In UK e-Science All Hands Meeting, pages 595–602, 2003.

[Had06] Marc J Hadley : Web application description language (wadl). Rapport technique, 2006.

[HBN10] Michael M Hoffman, Orion J Buske et William Stafford Noble : The genomedata format for storing large-scale functional genomics data. Bioinformatics, 26(11):1458–1459, 2010.

[Her12] Alice Hermann : Création et mise à jour d'objets dans une base de connaissances. Thèse de doctorat, Thèse de l'INSA Rennes - École doctorale MATISSE, 17 décembre 2012. supervised by M. Ducassé and S. Ferré.

[HKR09] P. Hitzler, M. Krötzsch et S. Rudolph : Foundations of Semantic Web Technologies. Chapman & Hall/CRC, 2009.

[HLL+09] Mohand-Said Hacid, Freddy Lécué, Alain Léger, Christophe Rey et Farouk Toumani : Les web services sémantiques, automate et intégration i. introduction, éléments et scénarios, découverte de services web. TSI. Technique et science informatiques, 28(2):229–262, 2009.

[HPRB04] M Hallard, P Picouet, LF Rodriguez et S Bigaret : Bioside : faciliter laccés des biologistes aux ressources bioinformatiques. JOBIM 2004 : 5èmes journèes ouvertes biologie informatique mathèmatique, 27-30 juin, Montrèal, Canada, page 64, 2004.

[HSL+04] Duncan Hull, Robert Stevens, Phillip Lord, Chris Wroe et Carole Goble : Treating shimantic web syndrome with ontologies. In AKT Workshop on Semantic Web Services (AKT-SWS04), pages 1–4. The Open University, Milton Keynes, UK, 2004.

[HVP00] Haruo Hosoya, Jerome Vouillon et Benjamin C. Pierce : Regular expression types for XML. In Martin Odersky et Philip Wadler, éditeurs : International Conference on Functional Programming (ICFP), pages 11–22. ACM, 2000.

[HWS+06] Duncan Hull, Katy Wolstencroft, Robert Stevens, Carole A. Goble, Matthew R. Pocock, Peter Li et Tom Oinn : Taverna : a tool for building and running workflows of services. Nucleic Acids Research, 34(Web-Server-Issue):729–732, 2006.

[HZ09] Mira V Han et Christian M Zmasek : phyloXML : XML for evolutionary biology and comparative genomics. BMC bioinformatics, 10(1):356, 2009.

[IKJ+13] Jon C. Ison, Matús Kalas, Inge Jonassen, Dan M. Bolser, Mahmut Uludag, Hamish McWilliam, James Malone, Rodrigo Lopez, Steve Pettifer et Peter M. Rice : EDAM : an ontology of bioinformatics operations, types of data and identifiers, topics and formats. Bioinformatics, 29(10):1325–1332, 2013.

[KGS04] Jihie Kim, Yolanda Gil et Marc Spraragen : A knowledge-based approach to interactive workflow composition. In International Conference on Automatic Planning and Scheduling (ICAPS), 2004.

[KGS05] Matthias Klusch, Andreas Gerber et Marcus Schmidt : Semantic web service composition planning with owls-xplan. In AAAI Fall Symposium on Semantic Web and Agents, 2005.

[KLC13] Andrey Kashlev, Shiyong Lu et Artem Chebotko : Coercion approach to the shimming problem in scientific workflows. In IEEE

International Conference on Services Computing, pages 416–423. IEEE, 2013.

[KLC14] A. Kashlev, S. Lu et A. Chebotko : Typetheoretic approach to the shimming problem in scientific workflows. IEEE Transactions on Services Computing, PP(99):1–1, 2014.

[KNB+09] Woralak Kongdenfha, Hamid R. Motahari Nezhad, Boualem Benatallah, Fabio Casati et Régis Saint-Paul : Mismatch patterns and adaptation aspects : A foundation for rapid development of web service adapters. IEEE T. Services Computing, 2(2):94–107, 2009.

[KNT10] Toshiaki Katayama, Mitsuteru Nakao et Toshihisa Takagi : Togows : integrated soap and rest apis for interoperable bioinformatics web services. Nucleic acids research, 38(suppl 2):W706–W711, 2010.

[KPJ+10] Matús Kalas, Pål Puntervoll, Alexandre Joseph, Edita Bartaseviciute, Armin Töpfer, Prabakar Venkataraman, Steve Pettifer, Jan Christian Bryne, Jon C. Ison, Christophe Blanchet, Kristoffer Rapacki et Inge Jonassen : BioXSD : the common data-exchange format for everyday bioinformatics web services. Bioinformatics, 26(18), 2010.

[KR12] Johannes Köster et Sven Rahmann : Snakemake - a scalable bioinformatics workflow engine. Bioinformatics, 28(19):2520–2522, 2012.

[KSG04] Jihie Kim, Marc Spraragen et Yolanda Gil : An intelligent assistant for interactive workflow composition. In Jean Vanderdonckt, Nuno Jardim Nunes et Charles Rich, éditeurs : International Conference on Intelligent User Interfaces, pages 125–131. ACM, 2004.

[Kuh13] T. Kuhn : A survey and classification of controlled natural languages. Computational Linguistics, 2013.

[KVBF07] Jacek Kopecký, Tomas Vitvar, Carine Bournez et Joel Farrell : SAWSDL : semantic annotations for WSDL and XML schema. IEEE Internet Computing, 11(6):60–67, 2007.

[KZB+10] W James Kent, Ann S Zweig, G Barber, Angie S Hinrichs et Donna Karolchik : BigWig and BigBed : enabling browsing of large distributed datasets. Bioinformatics, 26(17):2204–2207, 2010.

[LAWG05] Phillip W. Lord, Pinar Alper, Chris Wroe et Carole A. Goble : Feta : A light-weight architecture for user oriented semantic ser-

vice discovery. In Asunción Gómez-Pérez et Jérôme Euzenat, éditeurs : The Semantic Web : Research and Applications, European Semantic Web Conference (ESWC), volume 3532 de LNCS, pages 17–31. Springer, 2005.

[LBC+10] Nicolas Lebreton, Christophe Blanchet, Daniela Barreiro Claro, Julie Chabalier, Anita Burgun et Olivier Dameron : Verification of parameters semantic compatibility for semi-automatic web service composition : a generic case study. In Gabriele Kotsis, David Taniar, Eric Pardede, Imad Saleh et Ismail Khalil, éditeurs : International Conference on Information Integration and Web-based Applications and Services (iiWAS), pages 845–848. ACM, 2010.

[LBRM+13] Yvan Le Bras, Aurélien Roult, Cyril Monjeaud, Mathieu Bahin, Olivier Quénez, Claudia Heriveau, Anthony Bretaudeau, Olivier Sallou et Olivier Collin : Towards a life sciences virtual research environment. In Journées Ouvertes en Biologie, Informatique et Mathématiques (JOBIM), 2013.

[LBW+04] Phillip Lord, Sean Bechhofer, Mark D Wilkinson, Gary Schiltz, Damian Gessler, Duncan Hull, Carole Goble et Lincoln Stein : Applying semantic web services to bioinformatics : Experiences gained, lessons learnt. In Sheila A. McIlraith, Dimitris Plexousakis et Frank van Harmelen, éditeurs : The Semantic Web - International Semantic Web Conference (ISWC), volume 3298 de LNCS, pages 350–364. Springer, 2004.

[LFJ07] Xitong Li, Yushun Fan et Feng Jiang : A classification of service composition mismatches to support service mediation. In International Conference on Grid and Cooperative Computing (GCC), pages 315–321. IEEE Computer Society, 2007.

[LGG11] Burkhard Linke, Robert Giegerich et Alexander Goesmann : Conveyor : a workflow engine for bioinformatic analyses. Bioinformatics, 27(7):903–911, 2011.

[LH07] Holger Lausen et Thomas Haselwanter : Finding web services. In European Semantic Technology Conference (ESTC). Citeseer, 2007.

[LHW+09] Heng Li, Bob Handsaker, Alec Wysoker, Tim Fennell, Jue Ruan, Nils Homer, Gabor Marth, Goncalo Abecasis, Richard Durbin et al. : The sequence alignment/map format and SAMtools. Bioinformatics, 25(16):2078–2079, 2009.

[Li11] Heng Li : Tabix : fast retrieval of sequence features from generic TAB-delimited files. Bioinformatics, 27(5):718–719, 2011.

[LLF+09] Cui Lin, Shiyong Lu, Xubo Fei, Darshan Pai et Jing Hua : A task abstraction and mapping approach to the shimming problem in scientific workflows. In IEEE International Conference on Services Computing (SCC), pages 284–291. IEEE Computer Society, 2009.

[LO08] Anders Lanzén et Tom Oinn : The Taverna interaction service : enabling manual interaction in workflows. Bioinformatics, 24(8): 1118–1120, 2008.

[LRPF04] Rubén Lara, Dumitru Roman, Axel Polleres et Dieter Fensel : A conceptual comparison of WSMO and OWL-S. In Liang-Jie Zhang, éditeur : European Conference on Web Services (ECOWS), volume 3250, pages 254–269. Springer, 2004.

[MBD+02] Katerina Michalickova, Gary D Bader, Michel Dumontier, Hao Lieu, Doron Betel, Ruth Isserlin et Christopher WV Hogue : SeqHound : biological sequence and structure database as a platform for bioinformatics research. BMC bioinformatics, 3(1):32, 2002.

[MBH+04] David Martin, Mark Burstein, Jerry Hobbs, Ora Lassila, Drew McDermott, Sheila McIlraith, Srini Narayanan, Massimo Paolucci, Bijan Parsia, Terry Payne et al. : Owl-s : Semantic markup for web services. W3C member submission, 22:2007–04, 2004.

[MDZ+07] Philip Maechling, Ewa Deelman, Li Zhao, Robert Graves, Gaurang Mehta, Nitin Gupta, John Mehringer, Carl Kesselman, Scott Callaghan, David Okaya et al. : Scec cybershake workflows-automating probabilistic seismic hazard analysis calculations. In Workflows for e-Science, pages 143–163. Springer, 2007.

[MGN+10] Hervé Ménager, Vivek Gopalan, Bertrand Néron, Sandrine Larroudé, Julien Maupetit, Adrien Saladin, Pierre Tufféry, Yentram Huyen et Bernard Caudron : Bioinformatics applications discovery and composition with the mobyle suite and mobylenet. In Zoé Lacroix et Maria-Esther Vidal, éditeurs : International Workshop on Resource Discovery, volume 6799 de LNCS, pages 11–22. Springer, 2010.

[MJRS+09] Marco Mesiti, Ernesto Jiménez-Ruiz, Ismael Sanz, Rafael Berlanga-Llavori, Paolo Perlasca, Giorgio Valentini et David Manset : Xml-based approaches for the integration of heterogeneous bio-molecular data. BMC bioinformatics, 10(Suppl 12):S7, 2009.

[MLK14] Aravind Mohan, Shiyong Lu et Alexander Kotov : Addressing the shimming problem in big data scientific workflows. In IEEE

International Conference on Services Computing (SCC), pages 347–354. IEEE, 2014.

[MM04] Nikola Milanovic et Miroslaw Malek : Current solutions for web service composition. IEEE Internet Computing, 8(6):51–59, 2004.

[Mot02] Rod Moten : Using type theory as a language for negotiation objects in online exchanges. Rapport technique, Technical Report 02-10, Rensselaer Polytechnic Institute, 2002. Online at http ://www. cs. rpi. edu/tr/02-10. pdf, 2002.

[MS02] Sheila A. McIlraith et Tran Cao Son : Adapting golog for composition of semantic web services. In Dieter Fensel, Fausto Giunchiglia, Deborah L. McGuinness et Mary-Anne Williams, éditeurs : International Conference on Principles and Knowledge Representation and Reasoning (KR), pages 482–496. Morgan Kaufmann, 2002.

[MS10] Thorsten Möller et Heiko Schuldt : OSIRIS next : Flexible semantic failure handling for composite web service execution. In IEEE International Conference on Semantic Computing (ICSC), pages 212–217. IEEE Computer Society, 2010.

[MSZ01] Sheila A McIlraith, Tran Cao Son et Honglei Zeng : Semantic web services. IEEE intelligent systems, 16(2):46–53, 2001.

[MVG+09] Hamish McWilliam, Franck Valentin, Mickael Goujon, Weizhong Li, Menaka Narayanasamy, Jenny Martin, Teresa Miyar et Rodrigo Lopez : Web services at the european bioinformatics institute-2009. Nucleic Acids Research, 37(Web-Server-Issue):6–10, 2009.

[MWT+10] Paolo Missier, Katy Wolstencroft, Franck Tanoh, Peter Li, Sean Bechhofer, Khalid Belhajjame, Steve Pettifer et Carole A. Goble : Functional units : Abstractions for web service annotations. In World Congress on Services, SERVICES, pages 306–313. IEEE Computer Society, 2010.

[NBCT06] Hamid R Motahari Nezhad, Boualem Benatallah, Fabio Casati et Farouk Toumani : Web services interoperability specifications. IEEE Computer, 39(5):24–32, 2006.

[NL05] Pieter BT Neerincx et Jack AM Leunissen : Evolution of web services in bioinformatics. Briefings in Bioinformatics, 6(2):178–188, 2005.

[NM02] Srini Narayanan et Sheila A McIlraith : Simulation, verification and automated composition of web services. In David Lassner,

Dave De Roure et Arun Iyengar, éditeurs : International Conference on World Wide Web, pages 77–88. ACM, 2002.

[NMM+09] Bertrand Néron, Hervé Ménager, Corinne Maufrais, Nicolas Joly, Julien Maupetit, Sébastien Letort, Sébastien Carrère, Pierre Tufféry et Catherine Letondal : Mobyle : a new full web bioinformatics framework. Bioinformatics, 25(22):3005–3011, 2009.

[OGA+06a] T. Oinn, M. Greenwood, M. Addis, J. Ferris, K. Glover, C. Goble, D. Hull, D. Marvin, P. Li et P. Lord : Taverna : Lessons in creating a workflow environment for the life sciences. Concurrency and Computation : Practice and Experience, 18(10):1067–1100, 2006.

[OGA+06b] Thomas M. Oinn, R. Mark Greenwood, Matthew Addis, M. Nedim Alpdemir, Justin Ferris, Kevin Glover, Carole A. Goble, Antoon Goderis, Duncan Hull, Darren Marvin, Peter Li, Phillip W. Lord, Matthew R. Pocock, Martin Senger, Robert Stevens, Anil Wipat et Chris Wroe : Taverna : lessons in creating a workflow environment for the life sciences. Concurrency and Computation : Practice and Experience, 18(10):1067–1100, 2006.

[Pap03] Mike P Papazoglou : Service-oriented computing : Concepts, characteristics and directions. In International Conference on Web Information Systems Engineering (WISE), pages 3–12. IEEE Computer Society, 2003.

[Pas12] Michael Pasternak : Restful service description language, 2012. US Patent App. 13/656,032.

[Pel03] C. Peltz : Web services orchestration and choreography. IEEE Computer, 36(10):46–52, 2003.

[PF02] Shankar R Ponnekanti et Armando Fox : Sword : A developer toolkit for web service composition. In International Conference on World Wide Web, volume 45, 2002.

[PH06] Minh Phan et Fumio Hattori : Automatic web service composition using congolog. In International Conference on Distributed Computing Systems Workshops (ICDCS Workshops), pages 17–17. IEEE Computer Society, 2006.

[Pie02] Benjamin C Pierce : Types and programming languages. MIT press, 2002.

[PIK+10] Steve Pettifer, Jon Ison, Matúš Kalaš, Dave Thorne, Philip McDermott, Inge Jonassen, Ali Liaquat, José M Fernández, Jose M Rodriguez, David G Pisano et al. : The embrace web service collection. Nucleic acids research, 38(suppl 2):W683–W688, 2010.

[PKPS02] Massimo Paolucci, Takahiro Kawamura, Terry R. Payne et Katia P. Sycara : Semantic matching of web services capabilities. In Ian Horrocks et James A. Hendler, éditeurs : The Semantic Web - International Semantic Web Conference (ISWC), volume 2342, pages 333–347. Springer, 2002.

[PL09] Trevor Paterson et Andy Law : An xml transfer schema for exchange of genomic and genetic mapping data : Implementation as a web service in a Taverna workflow. BMC bioinformatics, 10(1):252, 2009.

[Pre04] Chris Preist : A conceptual architecture for semantic web services. In Sheila A. McIlraith, Dimitris Plexousakis et Frank van Harmelen, éditeurs : The Semantic Web - Third International Semantic Web Conference (ISWC), volume 3298 de LNCS, pages 395–409. Springer, 2004.

[PSS04] Massimo Paolucci, Naveen Srinivasan et Katia Sycara : Expressing wsmo mediators in owl-s. In Martin David, Lara Rubén et Takahira Yamaguchi, éditeurs : Workshop on Semantic Web Services. Citeseer, CEUR-WS, 2004.

[PVDH07] Mike P Papazoglou et Willem-Jan Van Den Heuvel : Service oriented architectures : approaches, technologies and research issues. VLDB journal, 16(3):389–415, 2007.

[Rit91] Mikael Rittri : Using types as search keys in function libraries. Journal of Functional Programming, 1(01):71–89, 1991.

[RKL+05] Dumitru Roman, Uwe Keller, Holger Lausen, Jos de Bruijn, Rubén Lara, Michael Stollberg, Axel Polleres, Cristina Feier, Cristoph Bussler et Dieter Fensel : Web service modeling ontology. Applied ontology, 1(1):77–106, 2005.

[RKO+03] Anthony Rowe, Dimitrios Kalaitzopoulos, Michelle Osmond, Moustafa Ghanem et Yike Guo : The discovery net system for high throughput bioinformatics. Bioinformatics, 19(suppl 1):i225–i231, 2003.

[RLB00] Peter Rice, Ian Longden et Alan Bleasby : Emboss : the european molecular biology open software suite. Trends in genetics, 16(6): 276–277, 2000.

[RLG+06] Michael Reich, Ted Liefeld, Joshua Gould, Jim Lerner, Pablo Tamayo et Jill P Mesirov : Genepattern 2.0. Nature genetics, 38(5):500–501, 2006.

[RLS+06] Uwe Radetzki, Ulf Leser, S. C. Schulze-Rauschenbach, J. Zimmermann, Jens Lüssem, Thomas Bode et Armin B. Cremers :

Adapters, shims, and glue - service interoperability for in silico experiments. Bioinformatics, 22(9):1137–1143, 2006.

[RMB+10] Martin G Reese, Barry Moore, Colin Batchelor, Fidel Salas, Fiona Cunningham, Gabor T Marth, Lincoln Stein, Paul Flicek, Mark Yandell, Karen Eilbeck et al. : A standard variation file format for human genome sequences. Genome Biology, 11(8):R88, 2010.

[Rom08] Paolo Romano : Automation of in-silico data analysis processes through workflow management systems. Briefings in Bioinformatics, 9(1):57–68, 2008.

[RSBM+10] Philippe Rocca-Serra, Marco Brandizi, Eamonn Maguire, Nataliya Sklyar, Chris Taylor, Kimberly Begley, Dawn Field, Stephen Harris, Winston Hide, Oliver Hofmann et al. : Isa software suite : supporting standards-compliant experimental annotation and enabling curation at the community level. Bioinformatics, 26(18):2354–2356, 2010.

[SBH06] Søren J Sørensen, Mette Burmølle et Lars H Hansen : Making bio-sense of toxicity : new developments in whole-cell biosensors. Current opinion in biotechnology, 17(1):11–16, 2006.

[Sch02] Guido Schimm : Process miner - a tool for mining process schemes from event-based data. In Sergio Flesca, Sergio Greco, Giovambattista Ianni et Nicola Leone, éditeurs : European Conference on Logics in Artificial Intelligence, volume 2424 de LNCS, pages 525–528. Springer Berlin Heidelberg, 2002.

[Sco00] Michael Lee Scott : Programming Language Pragmatics. Morgan Kaufmann, 2000.

[SGBB01] Robert Stevens, Carole Goble, Patricia Baker et Andy Brass : A classification of tasks in bioinformatics. Bioinformatics, 17(2): 180–188, 2001.

[SGL07] Amit P Sheth, Karthik Gomadam et Jon Lathem : SA-REST : semantically interoperable and easier-to-use services and mashups. IEEE Internet Computing, 11(6):91–94, 2007.

[SHP03] Evren Sirin, James Hendler et Bijan Parsia : Semi-automatic composition of web services using semantic descriptions. In Workshop on Web Services : Modeling, Architecture and Infrastructure, pages 17–24, 2003.

[SK03] Biplav Srivastava et Jana Koehler : Web service composition-current solutions and open problems. In International Conference on Automated Planning and Scheduling-Workshop on Planning for Web Services, volume 35, pages 28–35, 2003.

[SKH+06] Philipp N. Seibel, Jan Krüger, Sven Hartmeier, Knut Schwarzer, Kai Löwenthal, Henning Mersch, Thomas Dandekar et Robert Giegerich : XML schemas for common bioinformatic data types and their application in workflow systems. BMC Bioinformatics, 7:490, 2006.

[SPO12] Simon P Sadedin, Bernard Pope et Alicia Oshlack : Bpipe : a tool for running and managing bioinformatics pipelines. Bioinformatics, 28(11):1525–1526, 2012.

[SPW+04] Evren Sirin, Bijan Parsia, Dan Wu, James Hendler et Dana Nau : Htn planning for web service composition using shop2. Web Semantics : Science, Services and Agents on the World Wide Web, 1(4):377–396, 2004.

[SQV+14] Quan Z Sheng, Xiaoqiang Qiao, Athanasios V Vasilakos, Claudia Szabo, Scott Bourne et Xiaofei Xu : Web services composition : A decade's overview. Information Sciences, 280:218–238, 2014.

[SRO03] Martin Senger, Peter Rice et Tom Oinn : Soaplab-a unified sesame door to analysis tools. In UK e-Science All Hands Meeting, volume 18, pages 509–513. Citeseer, 2003.

[ST09] G. M. Sacco et Y. Tzitzikas, éditeurs. Dynamic taxonomies and faceted search. The information retrieval series. Springer, 2009.

[Ste02] Lincoln Stein : Creating a bioinformatics nation. Nature, 417(6885):119–120, 2002.

[SW05] Eleni Stroulia et Yiqiao Wang : Structural and semantic matching for assessing web-service similarity. International Journal of Cooperative Information Systems, 14(4):407–438, 2005.

[TBM+12] Henry S Thompson, David Beech, M Maloney et al. : XML schema definition language (XSD) 1.1 part 1 : Structures, 2012.

[TDGS14] Ian J Taylor, Ewa Deelman, Dennis B Gannon et Matthew Shields : Workflows for e-Science : scientific workflows for grids. Springer Publishing Company, Incorporated, 2014.

[TKW+10] Sayed Gholam Hassan Tabatabaei, Wan Kadir, MN Wan, Suhaimi Ibrahim et Amir Vahid Dastjerdi : AIMO translator : Bridging the gap between semantic web service discovery and composition. In International Conference on Internet and Web Applications and Services, pages 268–273. IEEE, 2010.

[Vas01] Clemens F Vasters : BizTalk Server 2000 : A Beginner's Guide. McGraw-Hill Professional, 2001.

[vDATHKB03] Wil MP van Der Aalst, Arthur HM Ter Hofstede, Bartek Kiepuszewski et Alistair P Barros : Workflow patterns. Distributed and parallel databases, 14(1):5–51, 2003.

[vdAvD02] W.M.P. van der Aalst et B.F. van Dongen : Discovering workflow performance models from timed logs. In Yanbo Han, Stefan Tai et Dietmar Wikarski, éditeurs : International Conference on Engineering and Deployment of Cooperative Information Systems, volume 2480 de LNCS, pages 45–63. Springer Berlin Heidelberg, 2002.

[VdAvDH+03] Wil MP Van der Aalst, Boudewijn F van Dongen, Joachim Herbst, Laura Maruster, Guido Schimm et Anton JMM Weijters : Workflow mining : a survey of issues and approaches. Data & knowledge engineering, 47(2):237–267, 2003.

[VDAVH04] Wil Van Der Aalst et Kees Max Van Hee : Workflow management : models, methods, and systems. MIT press, 2004.

[VdAWM04] Wil Van der Aalst, Ton Weijters et Laura Maruster : Workflow mining : Discovering process models from event logs. IEEE Transactions on Knowledge and Data Engineering, 16(9):1128–1142, 2004.

[VIK+10] Marko Vujasinovic, Nenad Ivezic, Boonserm Kulvatunyou, Edward Barkmeyer, Michele Missikoff, Francesco Taglino, Zoran Marjanovic et Igor Miletic : Semantic mediation for standard-based B2B interoperability. IEEE Internet Computing, 14(1):52–63, 2010.

[WAH+07] Katy Wolstencroft, Pinar Alper, Duncan Hull, Chris Wroe, Phillip W. Lord, Robert D. Stevens et Carole A. Goble : The myGrid ontology : bioinformatics service discovery. International Journal of Bioinformatics Research and Applications, 3(3):303–325, 2007.

[WCA09] Jianwu Wang, Daniel Crawl et Ilkay Altintas : Kepler + hadoop : A general architecture facilitating data-intensive applications in scientific workflow systems. In Ewa Deelman et Ian J. Taylor, éditeurs : Workshop on Workflows in Support of Large-Scale Science, WORKS '09, pages 12 :1–12 :8. ACM, 2009.

[WE87] Michael S Waterman et Mark Eggert : A new algorithm for best subsequence alignments with application to trna-rrna comparisons. Journal of molecular biology, 197(4):723–728, 1987.

[Wel99] Daniel S Weld : Recent advances in AI planning. AI Magazine, 20(2):93–123, 1999.

[WGMK10] Rui Wang, Sumedha Ganjoo, John A Miller et Eileen T Kraemer : Ranking-based suggestion algorithms for semantic web service composition. In World Congress on Services, (SERVICES), pages 606–613. IEEE Computer Society, 2010.

[WGP+11] Rui Wang, Chaitanya Guttula, Maryam Panahiazar, Haseeb Yousaf, John A Miller, Eileen T Kraemer et Jessica C Kissinger : Web service composition using service suggestions. In World Congress on Services, (SERVICES), pages 482–489. IEEE Computer Society, 2011.

[WHF+13] Katherine Wolstencroft, Robert Haines, Donal Fellows, Alan R. Williams, David Withers, Stuart Owen, Stian Soiland-Reyes, Ian Dunlop, Aleksandra Nenadic, Paul Fisher, Jiten Bhagat, Khalid Belhajjame, Finn Bacall, Alex Hardisty, Abraham Nieva de la Hidalga, Maria P. Balcazar Vargas, Shoaib Sufi et Carole A. Goble : The Taverna workflow suite : designing and executing workflows of web services on the desktop, web or in the cloud. Nucleic Acids Research, 41(Webserver-Issue):557–561, 2013.

[WIN+05] John D. Westbrook, Nobutoshi Ito, Haruki Nakamura, Kim Henrick et Helen M. Berman : Pdbml : the representation of archival macromolecular structure data in xml. Bioinformatics, 21(7):988–992, 2005.

[WL02] Mark D. Wilkinson et Matthew Links : Biomoby : An open source biological web services proposal. Briefings in bioinformatics, 3(4): 331–341, 2002.

[WMK+08] Zhiming Wang, John A Miller, Jessica C Kissinger, Rui Wang, Douglas Brewer et Cristina Aurrecoechea : Ws-biozard : A wizard for composing bioinformatics web services. In IEEE Congress on Services, Part I, SERVICES I, pages 437–444. IEEE Computer Society, 2008.

[WvdVW+09] Ingo H. C. Wassink, Paul E. van der Vet, Katy Wolstencroft, Pieter B. T. Neerincx, Marco Roos, Han Rauwerda et Timo M. Breit : Analysing scientific workflows : Why workflows not only connect web services. In IEEE Congress on Services, Part I, SERVICES I, pages 314–321. IEEE Computer Society, 2009.

[WVM09] Mark D Wilkinson, Benjamin Vandervalk et Luke McCarthy : SADI semantic web services – 'cause you can't always GET what you want ! In Markus Kirchberg, Patrick C. K. Hung, Barbara Carminati, Chi-Hung Chi, Rajaraman Kanagasabai, Emanuele Della Valle, Kun-Chan Lan et Ling-Jyh Chen, éditeurs :

IEEE Asia-Pacific Services Computing Conference (APSCC), pages 13–18. IEEE, 2009.

[WW08] C. Walker et D. W. Walker : Integration and data sharing between ws-based workflows. In IEEE International Conference on Web Services (ICWS), pages 667–674. IEEE Computer Society, 2008.

[YS97] Daniel M Yellin et Robert E Strom : Protocol specifications and component adaptors. ACM Transactions on Programming Languages and Systems (TOPLAS), 19(2):292–333, 1997.

[ZHvdA11] Reng Zeng, Xudong He et Wil MP van der Aalst : A method to mine workflows from provenance for assisting scientific workflow composition. In World Congress on Services (SERVICES), pages 169–175. IEEE Computer Society, 2011.

[ZW97] Amy Moormann Zaremski et Jeannette M Wing : Specification matching of software components. ACM Transactions on Software Engineering and Methodology (TOSEM), 6(4):333–369, 1997.

Table des figures

2.1 Un service nommé *matcher* et des descriptions dans EMBL-EBI. . 24
2.2 Exemple d'une règle dans Snakemake 33
2.3 Exemple de définition d'une tâche et d'une structure de construction dans Bpipe 34

3.1 Deux types de documents XML [BMC+12]. 46
3.2 Exemple d'un type simple défini à partir de *interger* 48
3.3 Exemple d'éléments pouvant constituer un type complexe 49
3.4 Exemple d'un type complexe 49
3.5 Exemple d'une valeur du type *Person* 50
3.6 Un exemple de valeur dans XDuce 50
3.7 Exemple d'un type *Person* correspondant au schéma défini à la figure 3.4. 51
3.8 Exemple de types récursifs. 52
3.9 Les différents composants de ACN et leurs interactions [Fer14] . . 54

4.1 Convertibility rules and definitions of generated converters. 64
4.2 An example proof tree of convertibility between two kinds of sequence lists. 66
4.3 The generated converter for example of Figure 4.2 67
4.4 Example of a source data and a target data to show how elements of sequence lists are transformed by converter of Figure 4.3. . . . 67
4.5 Global schema of the program that computes convertibility. 73
4.6 Examples of bioinformatics types 75
4.7 Workflow at user level (w_u). 76
4.8 Executable workflow (w_x). 77

4.9 Abstract workflow (w_a) in our system. 78
4.10 Example of source code of the shim my :converter converting data matching `CDNAFeatSeq` to data matching `CBioseq`. It is represented using XQuery. Functions my :content and my :select are utility functions. 79
4.11 Excerpt of the graph of links between services 81
4.12 Reponses of 5 participants to the list of link suggestions between services. 85
4.13 Pie chat of all participant responses to all suggestions, per response type. 85

5.1 A schematic representation of the components of our approach. . . 95
5.2 An example workflow. 98
5.3 An example dataflow. 98
5.4 GUI of FlowLis. 108
5.5 A JSON formatted service description as it can be managed into FlowLis. 109
5.6 A JSON formatted type expression as it can be managed into FlowLis. 110
5.7 Example of a step during the composition of the workflow at Figure 5.2, where an instance of service *getBioseq* is inserted to a partial workflow. 111
5.8 Example of a step during the composition of the workflow of Figure 5.2, where an instance of service *matcher* is inserted into a partial workflow. 111

Index

ADN, 5, 6, 20
Amazon, 12

BAM, 18
BED, 18
Big Data, 35
Bio-informatique, 6, 7
BioCatalogue, 7, 21
Biologie, 6, 7
BioMoby, 16, 21
BioPortal, 18
BioXSD, 19
BPEL, 24
BPML, 25

Camelis, 52
CDuce, 19
Cohérence, 8
Compatibilité, 8, 26
Composition de services, 8, 26, 31, 89
Connexion, 8, 80
Convertibilité, 58
Correspondance, 8, 30

DDBJ, 7
Distribuée, 6
DOM, 19

e-Science, 5, 7
EBI, 7
EDAM, 17
EMBL-EBI, 7, 22
EMBOSS, 7, 21, 23, 78
Environnements virtuels de recherche, 7
EXI, 19
Expériences, 6

FASTA, 18
Format de données, 18, 72

Galaxy, 7, 21, 33
Gate, 21
GenOuest, 7, 21, 33
Geolis, 53
GraphViz, 31
Guidage, 8

Hétérogène, 6, 8
Hadoop, 35
HATEOAS, 13
HTML, 13
HTTP, 13

IA, 36
in silico, 6
in situ, 6

Java, 12
JavaScript, 19
JAXB, 70

JSON, 19, 70

Kepler, 35

LIS, 92
logs, 38

Médiation de services, 26, 58
MapReduce, 35
Mobyle, 21, 34
myExperiment, 7
myGrid, 16

NGS, 18

OASIS, 20
Ontologie de domaine, 16
Ontologie de myGrid, 16
OWL, 14
OWL-S, 14

Perl, 12, 33
Python, 12, 33

QoS, 12

R, 33
Règle, 61
Ressource, 6
REST, 12, 13, 21, 70
RSDL, 13

SA-REST, 14
SAWSDL, 14
SAX, 19
Scufl, 25
Seekda, 21
Semi-automatique, 36, 91
Services web, 5, 12
Services web sémantiques, 14
Shell, 33
shims, 27, 61
SOA, 12
SOAP, 12, 21
SOC, 5
Suggestion, 98, 120
Système de règles, 60

Taverna, 34
TIC, 5, 7
Transformation, 8
Type, 58

UDDI, 13, 20
UniProt, 7
URI, 13

W3C, 14
WADL, 13
WfMC, 6
Workflow, 6, 24, 91
Workflow mining, 38
WS-CDL, 25
WSDL, 13
WSMO, 15

XDuce, 59
XML, 19, 58, 70
XML-Schema, 44, 58
XPDL, 25

INSA de RENNES
Service des Formations

AVIS DU JURY SUR LA REPRODUCTION DE LA THESE SOUTENUE

Titre de la thèse:
Composition guidée de services - Application aux workflows d'analyse de données en bio-informatique

Nom Prénom de l'auteur : BA MOUHAMADOU

Membres du jury :
- Monsieur HACID Mohand-Said
- Madame COHEN-BOULAKIA Sarah
- Monsieur LAVENIER Dominique
- Monsieur LO Moussa
- Madame DUCASSE Mireille
- Monsieur FERRÉ Sébastien

Président du jury : D. LAVENIER.

Date de la soutenance : 04 Décembre 2015

Reproduction de la these soutenue

Thèse pouvant être reproduite en l'état
~~Thèse pouvant être reproduite après corrections suggérées~~

Fait à Rennes, le 04 Décembre 2015

Signature du président de jury

Le Directeur,

M'hamed DRISSI

INSTITUT NATIONAL des SCIENCES APPLIQUÉES - RENNES -

Résumé

Dans les domaines scientifiques, particulièrement en bioinformatique, des services élémentaires sont composés sous forme de workflows pour effectuer des expériences d'analyse de données complexes. À cause de l'hétérogénéité des ressources, la composition de services est une tâche difficile. Les utilisateurs, en composant des workflows, manquent d'assistance pour retrouver et interconnecter les services compatibles. Les solutions existantes utilisent des services spéciaux définis de manière manuelle pour gérer les conversions de formats de données entre les entrées et sorties des services dans les workflows. Cela est pénible pour un utilisateur final. Gérer les incompatibilités des services avec des convertisseurs manuels prend du temps et est lourd. Il existe des solutions automatisées pour faciliter la composition de workflows mais elles sont généralement limitées dans le guidage et l'adaptation des données entre services.

La première contribution de cette thèse propose de détecter systématiquement la convertibilité des sorties vers les entrées des services. La détection de convertibilité repose sur un système de règles basé sur une abstraction des types d'entrée et sortie des services. L'abstraction de types permet de considérer la nature et la composition des données d'entrée et sortie. Les règles permettent la décomposition et la composition ainsi que la spécialisation et la généralisation de types. Elles permettent également de générer des convertisseurs de données à utiliser entre services dans les workflows.

La deuxième contribution propose une approche interactive qui permet de guider des utilisateurs à composer des workflows en fournissant des suggestions de services et de liaisons compatibles basées sur la convertibilité de types d'entrée et sortie des services. L'approche est basée sur le modèle des Systèmes d'Information Logiques (LIS) qui permettent des requêtes et une navigation guidées et sûres sur des données représentées avec une logique uniforme. Avec notre approche, la composition de workflows est sûre et complète vis-à-vis de propriétés désirées.

Les résultats et les expériences, effectués sur des services et des types de données en bio-informatique, montrent la pertinence de nos approches. Nos approches offrent des mécanismes adaptés pour gérer les incompatibilités de services dans les workflows, en prenant en compte la structure composite des données d'entrée et sortie. Elles permettent également de guider, étape par étape, des utilisateurs à définir des workflows bien formés à travers des suggestions pertinentes.

Abstract

In scientific domains, particularly in bioinformatics, elementary services are composed as workflows to perform complex data analysis experiments. Due to the heterogeneity of resources, the composition of services is a difficult task. Users, when composing workflows, lack assistance to find and interconnect compatible services. Existing solutions use special services manually defined to manage data format conversions between the inputs and outputs of services in workflows, it is difficult for an end user. Managing service incompatibilities with manual converters is time-consuming and heavy. There are automated solutions to facilitate composing workflows but they are generally limited in the guidance and the data adaptation between services they offer.

The first contribution of this thesis proposes to systematically detect convertibility from outputs to inputs of services. Convertibility detection relies on a rule system based on an abstraction of input and output types of services. Type abstraction enables to consider the nature and the composition of input and output data. Rules enable decomposition and composition as well as specialization and generalization of types. They also enable to generate data converters to use between services in workflows.

The second contribution proposes an interactive approach that enables to guide users to compose workflows by providing suggestions of compatible services and links based on convertibility of input and output types of services. The approach is based on the framework of Logical Information Systems (LIS) that enables safe and guided requests and navigation on data represented with a uniform logic. With our approach, composition of workflows is safe and complete w.r.t. desired properties.

The results and experiences, conducted on bioinformatics services and datatypes, show the relevance of our approaches. Our approaches offer adapted mechanisms to manage service incompatibilities in workflows, by taking into account the composite structure of inputs and outputs data. They enable to guide, step by step, users to define well-formed workflows through relevant suggestions.

N° d'ordre : 15ISAR 28 / D15 - 28

Institut National des Sciences Appliquées de Rennes
20, Avenue des Buttes de Coëmes • CS 70839 • F-35708 Rennes Cedex 7
Tel : 02 23 23 82 00 - Fax : 02 23 23 83 96