Concevoir et partager des workflows d'analyse de données. Application aux traitements intensifs en bioinformatique

François Moreews

UNIVERSITÉ DE RENNES 1

THÈSE / UNIVERSITÉ DE RENNES 1
sous le sceau de l'Université Européenne de Bretagne

pour le grade de

DOCTEUR DE L'UNIVERSITÉ DE RENNES 1

Mention : Informatique

École doctorale Matisse

présentée par

François MOREEWS

préparée à l'unité de recherche IRISA – UMR6074
Institut de Recherche en Informatique et Système Aléatoires
composante universitaire : ISTIC

Concevoir et partager des workflows d'analyse de données. Application aux traitements intensifs en bioinformatique

Thèse soutenue à Rennes
le 11/12/2015

devant le jury composé de :

David GROSS-AMBLARD
Professeur à l'Université de Rennes 1 / *Président*

Sarah COHEN-BOULAKIA
Maître de conférence à l'Université de Paris-Sud / *Rapporteur*

Christophe ANTONIEWSKI
Directeur de Recherche au CNRS, Institut de Biologie Paris Seine / *Rapporteur*

Jean-François GIBRAT
Directeur de Recherche à l'INRA de Jouy-en-Josas, responsable scientifique de l'Institut Français de Bioinformatique / *Examinateur*

Sébastien FERRÉ
Maître de conférence à l'Université de Rennes 1 / *Examinateur*

Dominique LAVENIER
Directeur de Recherche au CNRS à l'IRISA / *Directeur de thèse*

Remerciements

Ce travail n'aurait pu être réalisé sans le soutien de plusieurs personnes.

Je tiens à remercier en tout premier lieu, pour sa disponibilité et sa confiance, Dominique Lavenier, directeur de recherche au CNRS et responsable du projet Genscale, qui a dirigé cette thèse.

Je remercie chaleureusement Sarah Cohen-Boulakia, maître de conférence à l'Université de Paris-Sud, et Christophe Antoniewski, directeur de recherche au CNRS à l'Institut de Biologie Paris Seine, d'avoir bien voulu accepter la charge de rapporteur. Je tiens également à adresser tous mes remerciements à David Gross-Amblard, professeur à l'Université de Rennes 1, Jean-François Gibrat, Directeur de recherche à l'INRA, et Sébastien Ferré, maître de conférence à l'Université de Rennes 1, qui me font l'honneur de juger ce travail.

Je tiens également à remercier Sandrine Laggarigue, professeur à Agrocampus Ouest pour son soutien. Un grand merci à Erwan Drezen pour ses apports méthodologiques. Je remercie Aymeric Antoine-Lorquin, Olivier Sallou et Jonathan Piat pour leurs contributions et Olivier Collin pour la qualité de l'environnement technique mis à disposition par la plate-forme Genouest.

Je voudrais également témoigner toute ma reconnaissance à ceux et celles qui ont eu la patience de m'entourer tout au long de ce travail, Carolina, Alice, Thomas mais aussi Marie-Claude et Alain.

Ce travail est dédié à mon frère Vincent.

Table des matières

Introduction

Contexte

La reproductibilité est la capacité d'une expérience à être répétée à l'identique par son auteur ou par un tiers. Elle assure que les résultats seront équivalents à des moments différents et dans des conditions différentes. C'est un des principes fondamentaux qui conditionnent le caractère scientifique d'une étude et la capacité à généraliser ses conclusions.

L'accroissement de la reproductibilité scientifique est un enjeu essentiel pour accélérer la productivité scientifique. Pour la favoriser, les analyses de données, qui représentent la partie *in silico* des protocoles expérimentaux, doivent être formalisées. Ces processus peuvent être implémentés sous forme de chaînes de traitements. En pratique, celles-ci sont souvent de nature hétérogène et peu structurées, limitant la capacité d'échanges et la maintenabilité. Les systèmes de gestion de workflows (Workflow Management Systems ou WfMS) scientifiques ont émergé depuis une dizaine d'années et proposent de mieux formaliser les chaînes de traitements en mettant en avant des propriétés de répétabilité, de maintenabilité et de capacité à l'échange. Certains de ces environnements bénéficient d'une très large base d'utilisateurs. Ainsi, en bioinformatique, un WfMS comme Galaxy tire parti d'une communauté très active et rend accessible des traitements d'analyse de données reproductibles couvrant une large partie du domaine d'application.

Cependant, en dépit d'un indéniable succès, le public concerné reste essentiellement celui des scientifiques consommateurs de données. Faire cohabiter concepteurs de méthodes et utilisateurs autour d'une même plate-forme d'analyse de données reste un défi. En effet, les WfMS sont aujourd'hui essentiellement des moyens de mise à disposition d'outils et d'orchestration, employés par les scientifiques du domaine d'application. Il s'agit, par exemple, des bio-analystes et des biologistes dans le cas d'analyses bioinformatiques. Ces environnements ne sont pas orientés vers la conception de composants dont l'intégration reste coûteuse, technique et non automatisée. Cela a pour conséquence de limiter leur adoption par les concepteurs d'outils, les développeurs et spécialistes de l'algorithmique. Cependant, ces concepteurs pourraient avoir un rôle essentiel à jouer dans l'intégration des méthodes d'analyses dans les WfMS. Leur implication accélérerait la diffusion de nouvelles méthodes et de prototypes d'intérêt. Par ailleurs, les dynamiques d'échanges et d'interactions entre consommateurs de données et producteurs de processus serait amplifiées. Des initiatives récentes montrent que l'idée d'un tel environnement

pivot est une préoccupation partagée [1, 2].

Par ailleurs, nous constatons qu'aujourd'hui, les scientifiques qui réalisent des analyses de données exploitant le calcul intensif disposent de nombreuses options quant au choix de l'infrastructure de calcul. L'éventail des possibilités est large et inclut l'usage de ressources internes en passant par des plates-formes académiques mutualisées ou les prestataires de *Cloud Computing*. Ces choix sont influencés par des facteurs comme les caractéristiques des projets, les partenariats, la disponibilité des machines, la puissance nécessaire ou la taille des données. En fonction de ces multiples paramètres, il peut être opportun de pouvoir déplacer les chaînes de traitements d'une plate-forme d'exécution à une autre, ou de les déployer temporairement, le temps d'une analyse. Aujourd'hui, ces possibilités sont limitées par les méthodes proposées pour déployer les chaînes de traitements. L'autonomie de l'utilisateur dans le choix de l'infrastructure mais aussi dans la personnalisation des environnements d'analyse est, de ce fait, restreinte. Même si des projets innovent pour trouver des solutions opérationnelles [3, 4], les WfMS ne prennent pas encore suffisamment en compte la nécessité de favoriser le *nomadisme des chaînes de traitements* .

En outre, il est évident que dans de multiples domaines scientifiques, les besoins en calcul augmentent, poussés par un accroissement massif de la production de données. Ainsi, l'arrivée des nouvelles méthodes de séquençage en génomique, mais aussi, plus généralement, de différentes technologies à haut-débit en biologie, provoque un goulot d'étranglement dans l'analyse de données en bioinformatique. Dans ce contexte, la diffusion de l'usage d'environnements de traitements de données comme les WfMS est une solution intéressante et déjà éprouvée pour simplifier l'exploitation des infrastructures de calcul. Cependant, ces logiciels ne proposent pas de mécanismes génériques et accessibles, permettant de tirer parti efficacement des caractéristiques spécifiques des infrastructures de calcul. Ainsi l'accès au parallélisme reste très limité, alors que dans de nombreux cas, la mise en œuvre de techniques simples pourrait garantir l'accélération des traitements. Il ne s'agit pas seulement d'obtenir des résultats plus rapidement mais également de faire diminuer les coûts des traitements, en exploitant mieux les capacités des machines disponibles. De plus, en délivrant plus rapidement des résultats exploitables, les méthodes d'analyse des données massives pourraient être raffinées, débouchant sur des résultats d'analyse plus pertinents.

Ces éléments montrent que de multiples efforts méthodologiques restent encore à entreprendre pour améliorer et généraliser la capture, l'exécution et la diffusion des workflows d'analyse de données.

Motivations

Une de nos principales motivations est de contribuer à l'élaboration de systèmes de gestion de workflows scientifiques adaptés à la conception et à la diffusion d'analyses de données et tout particulièrement dans le domaine de la bioinformatique.

Les facilités d'échanges permises par internet et les pratiques de développement *open-source* démultiplient l'accès aux outils d'analyse de données. Mais leurs usages restent

encore souvent restreints à des spécialistes de par la complexité de mise en œuvre et l'hétérogénéité des implémentations. Dans des domaines scientifiques où les analyses de données *in silico* sont prépondérantes, il est essentiel de favoriser le partage de ces processus. Il s'agit d'une démarche d'*Open Science* capable d'accélérer la recherche là où la culture de l'ouverture et de l'échange est déjà forte. Pour cela, nous pensons que les WfMS peuvent être un point d'entrée facilitant la capture des méthodes d'analyse de données. En effet, ils constituent le socle d'environnements émergents, candidats de choix pour la conception et la diffusion normalisée des processus d'analyse de données. Nous constatons que, depuis une dizaine d'années, la publication des chaînes de traitements sucite un vif intérêt. Un des moyens proposé est la création de répertoires web visant à agréger et mettre à disposition les workflows scientifiques à l'échelle d'une communauté [5]. Néanmoins, cet objectif ambitieux fait face à de multiples difficultés: pour que les workflows diffusés couvrent réellement le domaine applicatif, il faut répertorier et classifier les chaînes de traitements, déterminer des patrons de conception invariants, puis être capable d'intégrer ces caractéristiques dans une spécification suffisamment expressive et intelligible pour être manipulable et largement exploitable. Cette démarche laborieuse n'a jamais été complètement réalisée en bioinformatique.

En outre, l'activité scientifique est caractérisée par la mise au point de nouvelles techniques et méthodologies. De ce fait, les méthodes d'analyse de données *in silico* font appel à un assemblage d'outils non seulement préexistants mais aussi en cours d'élaboration. Il s'agit donc fréquemment de prototypes expérimentaux qui passent par différentes phases de mise au point. Bien que de nombreux environnements de gestion de workflows soient disponibles, ceux-ci sont orientés vers la mise à disposition de composants préexistants et à leur orchestration. Ils négligent la conception de nouveaux composants. Le cycle de vie des méthodes d'analyse est peu pris en compte et, de ce fait, le passage des phases de conception aux phases de mise en production reste coûteux. Les WfMS pourraient prendre en compte cette dimension et non pas se limiter à la mise à disposition de composants. A défaut, les outils les plus innovantes ne seront pas référencées dans les répertoires et la dynamique communautaire sera limitée par une faible implication des scientifiques concepteurs. Par ailleurs, aujourd'hui, la propriété de reproductibilité, souvent mise en avant dans le cadre d'une approche workflow, n'est pas complètement effective. Pour garantir la reproductibilité, la diffusion des protocoles doit être associée à une réelle capacité à rejouer l'exécution de l'analyse telle qu'elle a été initialement exploitée. Pour cela, une des approches possibles consiste à formaliser une spécification de workflows échangeables et déployables efficacement. Nous pensons que la mise en place des conditions techniques nécessaires au *nomadisme des chaînes de traitements* est un préalable à une diffusion efficace des workflows. Ainsi, l'utilisateur d'un WfMS pourra exécuter un workflow, extrait d'un répertoire communautaire, sur une infrastructure de *cloud computing* de son choix, sans se soucier des aspects techniques du déploiement, qui aujourd'hui nécessitent un support technique important.

Une deuxième motivation est la recherche de méthodes capables de lever les verrous de la capture des chaînes de traitements d'analyse de données à haut-débit. Pour cela,

la phase de création des processus d'analyse de données exploitant le calcul intensif doit d'abord être formalisée. Nous serons alors en mesure de proposer une démarche de conception plus efficace. Ensuite, nous devons définir une spécification de workflows adaptée, dans laquelle l'orchestration des tâches prend en compte le recours au calcul intensif. Cet aspect est crucial pour des domaines comme la bioinformatique. En effet, depuis 2005 et l'apparition des nouvelles technologies de séquençage (NGS pour Next-Generation Sequencing), les besoins en calcul et de stockage des traitements ont connu une augmentation considérable. Dans le même temps, les bioinformaticiens concevant des chaînes de traitements continuent à utiliser des méthodes basées sur le scripting et la ligne de commande. Les processus d'analyse sont donc de plus en plus complexes à écrire et à maintenir manuellement. En effet, l'évolution rapide des composants logiciels métiers mais également les contraintes causées par l'exploitation des environnements de calcul intensif provoquent des difficultés nouvelles. Ainsi, l'usage de couches logicielles supplémentaires et de patrons de conception additionnels particuliers sont actuellement nécessaires. C'est pourquoi, aujourd'hui, le développement de chaînes de traitements de données et l'optimisation de leurs performances s'avèrent des tâches techniques particulièrement laborieuses. Dans cette situation, le recours à des environnements intégratifs de conception comme les WfMS peut être une solution pour simplifier la conception et l'exécution des chaînes de traitements d'analyse intensive de données. Cela paraît d'autant plus adapté, qu'avec la nécessité d'analyser des masses de données toujours plus importantes, sont apparus des WfMS capables d'exécuter les processus sur des infrastructures distribuées dédiées au calcul intensif. Cependant, ces environnements n'intégrent pas de mécanismes génériques capables d'accélérer efficacement l'exécution des chaînes de traitements.

Objectifs

Les objectifs généraux de ce travail concernent l'intégration dans les WfMS de processus de conception des workflows efficaces, adaptés aux analyses de données haut-débit telles qu'on les conçoit en bioinformatique. Il s'agit également de faciliter leur diffusion et leur usage.

En premier lieu, nous nous intéresserons à la conception d'environnements de gestion de workflows scientifiques, basés sur des Architectures Orientées Service (SOA). Nous chercherons à définir un formalisme facilitant la capture des chaînes de traitements d'analyse de données scientifiques et plus particulièrement celles qui sont basées sur des composants hétérogènes. Les formalismes employés devront garantir le découplage entre spécification et implémentation et favoriser l'indépendance des modèles capturés vis-à-vis des environnements d'exécution logiciels et matériels. Nous voulons définir une spécification de workflow à la fois adaptée à la conception, à l'exécution et au déploiement des chaînes de traitements.

Nous désirons également favoriser l'intégration du calcul intensif dans les workflows. Pour cela, des mécanismes particuliers comme la reprise sur erreur, l'accès au parallélisme, et le déplacement des données doivent être intégrés aux WfMS. Même si des progrès importants ont été réalisés, il reste de nombreuses pistes à explorer pour améliorer la capture des spécifications et permettre la génération d'implémentations capables de tirer parti des caractéristiques des infrastructures de calcul.

Par ailleurs, il faut adapter les WfMS aux changements et à l'évolution des environnements d'exécution logiciels et matériels. Aujourd'hui, c'est le choix du WfMS qui définit le type d'infrastructure de calcul accessible. Il s'agit d'un frein à l'adoption de l'approche workflow, dans un contexte où de nouveaux types d'infrastructures mais aussi de modes d'accès au calcul apparaissent, portés par l'engouement pour le *Big Data*. La capacité à exécuter la même spécification sur différents types d'environnements d'exécution favoriserait l'indépendance du concepteur de workflows vis-à-vis des plates-formes techniques. La pérennisation des spécifications serait également renforcée. Enfin, l'élaboration de répertoires de workflows ouverts, et réellement reproductibles serait rendue possible. De ce fait, il est essentiel de définir une spécification de workflows indépendante des environnements d'exécution logiciels et matériels.

Enfin, le WfMS doit devenir un outil de conception d'application. Les WfMS scientifiques possèdent de bonnes capacités de mise à disposition et d'orchestration de composants préexistants. Ils ciblent donc des scientifiques consommateurs de données pour qui le WfMS remplit la fonction d'automatisation des analyses. La prise en compte du cycle de développement des applications scientifiques, et notamment du processus de prototypage et de raffinement d'une spécification, étendrait l'audience des WfMS aux producteurs de composants: les scientifiques spécialistes de l'algorithmique et les développeurs. Ces concepteurs maîtrisent la programmation et les infrastructures de calcul et se passent des WfMS. Pour qu'ils perçoivent mieux l'intérêt de modifier leurs pratiques de travail, ces environnements pourraient prendre en charge les tâches techniques de conception les plus laborieuses.

Afin de réaliser ces objectifs, nous définirons l'approche méthodologique suivante:

Tout d'abord, nous formaliserons un modèle DataFlow adapté à la conception graphique et au calcul intensif. Nous exploiterons la capacité de représentation des modèles DataFlow dans le cadre de l'élaboration d'un langage graphique permettant la spécification simplifiée et une exécution efficace des workflows. Pour cela, nous nous inspirerons des propriétés et patrons de conception des langages fonctionnels. Les modèles de calcul DataFlow permettent l'expression de multiples niveaux de parallélisme. Nous chercherons donc à exploiter cette caractéristique pour définir un moteur d'exécution parallélisé intégrant les spécificités du calcul intensif et des architectures orientées service des WfMS.

Ensuite, nous voulons rendre possible la simplification de la phase de design des workflows par l'introduction dans l'environnement logiciel de fonctionnalités prenant en

charge les aspects purement techniques de la conception. L'intégration de ces fonctionnalités doit permettre à l'utilisateur de mieux se focaliser sur la conception et l'orchestration de composants d'analyse des données. Pour cela, nous suivrons une approche orientée modèle, démarche inspirée de l'ingénierie dirigée par les modèles (MDE pour Model Driven Engineering). Elle sera basée sur l'analyse du domaine de la bioinformatique en tant que méta-modèle.

Nous proposerons une structuration des phases de conception facilitant l'isolation des composants associés au métier. Les chaînes de traitements seraient initialement simplifiées lors de la capture. L'environnement d'exécution pourrait ensuite se charger d'injecter des composants techniques et de définir leurs paramétrages suivant des règles liées au contexte d'exécution. Ainsi, le découplage entre spécifications de workflows et environnements logiciels et matériels sera formalisé.

Dans un deuxième temps, nous appliquerons une démarche similaire pour modéliser les dépendances logicielles des workflows et produire une spécification autonome, indépendante d'environnements logiciels prédéfinis. Ici, la modélisation ne sera pas basée sur le domaine d'application mais sur des représentations génériques et opérationnelles des dépendances et des contextes d'exécution attendus.

Contributions

Les contributions essentielles de cette thèse concernent la définition d'une méthodologie de conception orientée modèle rendant possible la création rapide de workflows scientifiques *Ad hoc* et leur diffusion.

Tout d'abord, nous proposons une méthode de capture de prototypes de workflows et de transformation en implémentations permettant l'analyse haut-débit.

Par ailleurs, nous présentons également un mode de représentation des dépendances logicielles dans la spécification des workflows qui est particulièrement adapté a l'échange et la reproductibilité. Nous listons ci-dessous plus précisément nos contributions.

Une méthode orientée modèle dans la conception des workflows scientifiques

La contribution essentielle de cette thèse est de proposer une méthode orientée modèle dans la conception des workflows scientifiques. Nous mettons notamment en évidence son intérêt pour:

— l'extraction simplifiée du parallélisme;

— la génération de différentes implémentations et vues à partir d'un même modèle;

— la représentation des dépendances logicielles des workflows.

Nous intégrons la connaissance du domaine métier, ici la bioinformatique, par la création d'un méta-modèle partiel.

Ce méta-modèle partiel est constitué à partir d'ontologies. Nous nous focalisons sur l'exploitation d'une ontologie décrivant les types de données et les processus en bioinformatique, EDAM (EMBRACE Data and Methods). Le méta-modèle produit dote les

fonctions définies par l'utilisateur (UDF) de propriétés exploitables concrètement au sein d'un modèle de calcul DataFlow.

C'est la connaissance du méta-modèle intégré à l'environnement qui fait émerger des fonctionnalités avancées, facilitant la conception de workflows.

Ce travail met particulièrement en évidence l'intérêt d'une typologie des classes de composants dans un WfMS. Nous cherchons à capturer dans l'environnement de conception des éléments du méta-modèle du domaine scientifique d'applications. Il s'agit ici de méthodes liées aux types des données métiers nommées DFF (Data Format Function). En les exploitant, nous guidons la conception et permettons l'automatisation de la génération du parallélisme de données gros grain.

Définition d'un langage DataFlow adapté à la conception rapide de workflows intensifs

Un des apports majeurs de ce travail est la définition des bases formelles d'un langage graphique DataFlow générique, particulièrement adapté aux analyses de données intensives.

Ce langage permet de spécifier de façon simplifiée un workflow basé sur la composition d'acteurs hétérogènes sans état. La spécification produite est interprétable par un moteur de workflows. Pour cela nous avons exploité des principes et patrons de conception issus des langages fonctionnels. Ceux-ci facilitent la hiérarchisation des fonctionnalités et permettent ainsi une prise en main intuitive des outils de conception, sans limiter l'expressivité du langage.

L'apport des langages fonctionnels est ici notamment caractérisé par l'utilisation de la gestion implicite des itérations et l'emploi d'un système de jetons de données abstraits référençant des collections (*monades collection*) et associé à des fonctions de filtrage (*listes en compréhension*).

Modèle de calcul générique exploitant les niveaux extractibles de parallélisme dans les workflows

Une autre contribution de cette étude est la création d'un modèle de calcul original. Ce modèle de calcul s'inspire des modèles DataFlow historiques en les adaptant aux architectures orientées service dédiées au calcul intensif. Ce modèle de calcul est associé à un algorithme d'ordonnancement et permet l'exploitation de plusieurs niveaux de parallélisme extractible à partir d'une spécification DataFlow: le parallélisme de tâches (itération, dépendances de données) et le parallélisme de données gros grain.

La particularité de notre modèle de calcul est qu'il exploite les niveaux de parallélisme les plus communs dans les workflows bioinformatiques. D'autre part, il est adapté aux infrastructures distribuées comme les clusters de calcul. C'est une approche générique et automatisable permettant l'introduction du parallélisme dans l'exécution des workflows et donc la diminution des temps d'analyses. Le moteur de workflows basé sur ce modèle est performant pour l'analyse des grands jeux de données. Par ailleurs, la capture d'une spécification de workflow adaptée à ce moteur est particulièrement rapide.

Génération d'implémentations multiples à partir de la capture d'une unique spécification de workflows

Nous avons montré le potentiel d'une approche orientée modèle pour adapter les workflows à différents environnements d'exécution. Pour cela, nous abordons l'exploitation de la transformation de modèles. Par cette méthode, il est possible, à partir d'un même modèle de worfklow, de générer différentes implémentations, chacune adaptée à un contexte d'exécution particulier. Il peut s'agir d'une spécification exécutable par un moteur de workflows ou de code interprétable dans un langage *general purpose*. Cette capacité à s'adapter à différents contextes d'exécution favorise l'indépendance entre spécification et implémentation et la pérennisation des processus capturés.

Application à la conception d'un WfMS

Nous montrons l'intérêt des formalismes élaborés en les implémentant au sein d'un nouveau *WfMS intégratif* orienté service qui associe un environnement de conception, un répertoire de workflows et un moteur d'exécution. Nous mettons notamment en évidence l'intérêt de cet environnement pour la conception rapide de workflows intensifs scientifiques. Ce WfMS rend possible la capture rapide de composants hétérogènes et leur orchestration. Nous étudions des exemples d'intégration de méthodes d'analyse de données bioinformatiques, en cours d'élaboration. Ces méthodes sont capturées sous forme de workflows puis converties semi-automatiquement en implémentations adaptées à un contexte d'exécution intensif.

Cet environnement logiciel est capable, à partir d'une même spécification, de produire différents types d'implémentations et modes de mise à disposition. Il peut ainsi concilier différents usages pour plusieurs publics tout en garantissant la normalisation et la pérennisation des processus d'analyse. Grâce à ce type de plates-formes, le workflow devient lui même un service exploitable dans différents contextes applicatifs et contextes d'exécution. Nous pensons donc que ce nouveau type de WfMS favorise l'usage et la dissémination des workflows.

Formalisation d'une spécification autonome de workflows

Une autre contribution de cette thèse est la définition d'une spécification DataFlow autonome, suffisante pour permettre le déploiement des workflows indépendamment de l'infrastructure de calcul. Le couplage entre spécifications de workflows et dépendances des composants permet, dans le cas du modèle de *cloud computing* CaaS (Container as a Service), la diffusion et l'échange de chaînes de traitements de données directement interprétables et réutilisables. La spécification proposée, A-SCDFM (Autonomous Semi-Concrete DataFlow Model), rend possible la création de répertoires communautaires de workflows intrinsèquement diffusables et ré-exécutables. Pour cela nous suggérons l'élaboration conjointe de répertoires de workflows basées sur A-SCDFM et de répertoires de containers incluant les dépendances logicielles. Cette architecture pourrait déboucher sur une accélération de la diffusion et de la mise à disposition des chaînes de traitements

de données, notamment intensives dans des contextes de développement communautaire, comme on le trouve en bioinformatique.

Vue d'ensemble de la dissertation

Dans la première partie, nous présenterons un état de l'art formel concernant les modèles de calculs exploitables dans les moteurs de workflows. Nous décrirons plus particulièrement les modèles utilisés dans certains WfMS scientifiques. Nous aborderons la notion de langages graphiques, et précisons certaines caractéristiques des langages fonctionnels et leurs liens avec les langages graphiques et les modèles de calcul DataFlow.

Nous détaillerons l'approche orientée modèle et en particulier les architectures orientées modèle (MDA pour Model Driven Architecture), et l'intérêt qu'elles présentent pour la conception de chaînes de traitements. Nous aborderons notamment la notion de méta-modèle et montrons comment des ontologies liées au domaine d'application sont exploitées aujourd'hui dans l'ingénierie dirigée par les modèles.

Dans la deuxième partie, nous présenterons un état de l'art technique des WfMS scientifiques et des plates-formes d'exécution adaptés au calcul intensif. Nous nous intéresserons particulièrement aux clusters de calcul et à l'utilisation de différents modèles de *cloud computing*. Nous décrirons l'audience actuelle des WfMS scientifiques en bioinformatique et les freins à l'adoption de ces outils par les concepteurs d'applications. Nous détaillerons les architectures logicielles et les mécanismes actuellement utilisés dans ces environnements pour gérer ou produire des workflows impliquant du calcul intensif. Nous caractériserons également ces systèmes par rapport à leurs capacités à permettre la création de répertoires de workflows communautaires.

La troisième partie décrira nos propositions méthodologiques. Nous présenterons un nouveau modèle de calcul DataFlow, UDFAS (URI based DataFlow for Asynchronous Services), adapté à l'éxécution des workflows intensifs au sein d'un système de gestion de workflows. Nous exposerons en détail ses patrons de conception inspirés des langages fonctionnels, notamment le recours à un système de jetons de données abstraits. Ce modèle permet l'expression de différents niveaux de parallélisme implicites. Nous présenterons également l'algorithme d'ordonnancement associé. Nous proposerons ensuite une démarche de capture de workflows basée sur une typologie des acteurs permettant d'isoler les éléments que l'on peut intégrer automatiquement pour simplifier la conception. Cette méthode permet la génération semi-automatique du parallélisme de données gros grain et rend possible la création rapide de workflows intensifs. Nous présentons les méthodes de transformation de modèles qui permet de générer différentes implémentations fonctionnelles, adaptés à de multiples environnements d'exécution. Nous décrivons la notion de spécification de modèle de workflow semi-concret (SCDFM pour Semi-Concrete DataFlow Model), qui facilite l'indépendance vis-à-vis des environnements d'exécution. Par l'association entre modèle de workflow SCDFM et représentation des dépendances des composants, nous pouvons proposer un mécanisme d'injection des configurations

facilitant le déploiement sur les clusters et les plates-formes PaaS. Par ailleurs, nous proposons de définir les dépendances logicielles des composants à partir de containers et de les représenter directement dans un modèle de workflow. La spécification DataFlow générique et autonome obtenue, nommée A-SCDFM (Autonomous Semi-Concrete DataFlow Model), permet un déploiement extrêmement simplifié sur toute plate-forme de *cloud computing* CaaS (Container as a Service).

Dans la quatrième partie, nous aborderons la description de l'implémentation des artefacts logiciels conçus pour valider l'intérêt de nos propositions méthodologiques. Cette description concerne (i) l'implémentation d'un WfMS incluant un mécanisme de parallélisme semi-automatique dirigé par l'utilisateur et permettant la capture simplifiée de composants hétérogènes, (ii) l'implémentation du Modèle de calcul UDFAS en tant que cœur du moteur de workflows de notre WfMS, (iii) l'implémentation du couplage entre spécification de workflows SCDFM et dépendances des composants pour simplifier le changement d'environnement d'exécution et automatiser le déploiement sur les plates-formes CaaS.

Dans la cinquième partie, nous exposerons nos résultats. Nous évaluerons les performances de notre modèle de calcul UDFAS et son adaptation au calcul intensif. Nous illustrerons l'usage des fonctionnalités de conception et d'exécution de workflows issues de nos propositions méthodologiques et implementées dans notre WfMS. Ces illustrations, PrimerFinder et Orthocis, sont des cas d'utilisation concrets du domaine de la bioinformatique, issus de collaborations scientifiques. PrimerFinder illustre notre méthode de capture et de conception ainsi que la génération rapide du parallélisme dans un contexte de prototypage d'application scientifique. Orthocis met en lumière l'indépendance des spécifications workflows produites vis-à-vis des infrastructures d'exécution et les capacités de déploiement automatique associées. D'autre part, le rôle d'un répertoire de workflows dans le cadre du développement rapide d'applications sera évalué.

Enfin, la dernière partie inclut la discussion et la conclusion de cette étude. Nous reviendrons sur les travaux effectués et synthétiserons les apports de cette thèse. L'extension du modèle de calcul UDFAS sera abordée et nous discuterons de la création de répertoires de workflows communautaires basés sur des spécifications autonomes A-SCDFM. Pour terminer, nous présenterons les perspectives d'applications de nos travaux dans le contexte de la conception de WfMS pour l'Open Science et proposerons des pistes pour de futures recherches.

Chapitre 1

Etat de l'art formel

Dans ce premier chapitre, nous nous concentrerons sur l'analyse des méthodes d'exécution et de conception des workflows. Nous étudierons ici les principes qui sont mis en œuvre aujourd'hui dans les applications gérant des chaînes de traitements. Ainsi, nous pourrons avoir une vision, la plus complète possible, des formalismes dont tirent parti, même implicitement, les WfMS. D'autre part, nous élargirons notre état des lieux à des méthodologies issues du génie logiciel et dont pourraient bénéficier ces environnements. De cette façon, nous pourrons définir le cadre formel qui nous permettra de faire des propositions basées sur des assises théoriques solides. Tout d'abord, nous mettrons à jour les fondements algorithmiques qui sous-tendent l'élaboration de moteurs d'exécution de workflows. Nous examinerons les méthodes d'orchestration qui déterminent les performances de l'exécution. Nous nous pencherons sur la notion de *modèle de calcul* et considérerons les différents choix possibles pour l'implémentation d'un moteur de workflows efficace. Nous détaillerons les modèles les mieux adaptés à l'exécution des workflows d'analyse de données à haut-débit. Ensuite, nous aborderons les caractéristiques des langages graphiques qui peuvent être employés dans les outils de conception des WfMS. Nous déterminerons quels sont les critères essentiels à considérer lorsqu'on élabore un tel langage. Nous étudierons en particulier comment concilier la facilité de conception et l'expressivité. De plus, nous décrirons les approches méthodologiques établies qui peuvent faciliter la mise en œuvre de la capture des spécifications et, plus largement, peuvent mieux structurer la démarche de conception de workflows. Ainsi nous présenterons les méthodes de conception orientées modèle et leur couplage à l'usage des ontologies dans le cadre général de la conception d'applications. L'ensemble de ces éléments nous servira à mieux cerner comment les WfMS rendent possible la conception et l'exécution des workflows d'analyse de données intensive et comment ils peuvent y être mieux adaptés.

1.1 Modèles de calcul

Les WfMS sont bâtis autour d'un artefact logiciel qui permet d'orchestrer l'exécution des différents composants qui forment un workflow que l'on nomme *moteur de*

workflows. Nous allons tout d'abord nous attacher à décrire les bases formelles qui sous-tendent l'élaboration de ces moteurs. Les moteurs de workflows utilisent des modèles de représentation des processus. Ces *modèles de calcul* (MoC pour Model of Computation) constituent le cœur algorithmique d'un moteur de workflows. Nous décrirons dans cette section les principaux MoC utilisés dans les WfMS.

1.1.1 Définition

Modèle de calcul

D'après Edwards et al [6], un modèle formel de conception doit posséder une sémantique précise et non ambiguë concernant (i) des relations implicites ou explicites entre entrées et sorties et éventuellement des variables d'état, (ii) des propriétés, (iii) des fonctions de "coût", (iv) des contraintes.
La définition formelle suivante d'un MoC communément admise est:

Définition 1.1 *Un MoC est une description mathématique qui possède une syntaxe et des règles permettant le calcul du comportement d'un système à un haut niveau d'abstraction. Il est utilisé pour spécifier la sémantique du calcul et de la concurrence.*

On peut citer différents exemples de MoC comme les machines à états finis, les systèmes d'équations différentielles, les DataFlows ou les réseaux de Pétri.

Un MoC a différents usages. Il rend possible la capture sans ambiguïté d'une fonctionnalité nécessaire ou l'utilisation d'outils associés à ce modèle. Un MoC facilite la vérification de la validité d'une spécification fonctionnelle par rapport à des propriétés. Enfin, une partie des spécifications peut être générée à partir de celui-ci.

Ainsi, pour concevoir une application qui réalise une certaine fonction, un concepteur peut vouloir explorer différentes implémentations possibles dans le but de maximiser une fonction de coût. Dans ce cas, des modèles de calcul peuvent être utilisés. Un MoC doit à la fois avoir les algorithmes de génération et de validation adaptés et être assez efficace pour le domaine d'application.

Lorsque l'on utilise un langage pour décrire un comportement, "le MoC détermine l'expressivité du langage tandis que la syntaxe affecte la compacité, la modularité et la réutilisabilité " [6].

Par ailleurs, on peut se poser la question de la relation entre MoC et représentation graphique d'un algorithme.

Ainsi, les *modèles graphiques de calcul* sont des représentations graphiques abstraites du mode d'exécution d'un système. Dans la littérature on peut voir les MoC parfois assimilés à leur représentation graphique. Nous préférerons différencier les MoC et leurs représentations. Nous désignerons donc ici par modèle graphique de calcul la représentation graphique d'un MoC.

1.1.2 La représentation des Workflows

Nous passons ici en revue certaines classes de MoC utilisées pour formaliser les workflows, que ce soit dans les domaines scientifiques ou dans le monde de l'entreprise. Nous

présenterons tout d'abord les méthodes algèbriques, basées sur des langages textuels et caractérisées par une grande expressivité. Nous aborderons ensuite les méthodes basées sur les graphes et généralement associées à des langages graphiques.

1.1.2.1 Méthodes algèbriques de manipulation de processus

π calcul

Le π calcul est une algèbre de processus qui décrit des systèmes configurables. Le π calcul est basé sur le concept de mobilité, qui inclut la communication et le changement.

La relation entre différents processus est la communication. Elle peut induire au cours du temps un changement dynamique de la structure qui associe des processus.

Ainsi un processus peut inclure dynamiquement d'autres processus qu'il reçoit par communication. La communication est basée sur la notion de nom. Un nom est un terme qui désigne indifféremment des liens, des références, ou des identifiants associés à un espace de nommage particulier.

Le π calcul possède un grand pouvoir d'expressivité. C'est un langage théorique permettant d'isoler des propriétés essentielles et d'étudier la programmation parallèle et distribuée. Il peut être vu comme une algèbre utile pour décrire des processus et comment ils communiquent de façon formelle et pour raisonner sur le comportement d'un système. Le π calcul trouve aujourd'hui des applications dans la gestion des processus métiers (BPM pour Business Process Management). Il permet ainsi d'exprimer les patrons classiques présents dans les workflows [7]. La nécessité d'utiliser la notion de mobilité dans un système de gestion de workflows pourrait justifier l'usage du π calcul, par exemple dans la représentation de traitements inter-organisations. Nous ne considérerons pas ce formalisme, puisque nous ne nous plaçons pas dans ce cas de figure.

Méthodes algèbriques relationnelles

Différents modèles de calcul basés sur des méthodes algèbriques relationnelles ont été conçus pour l'analyse de données:

— L'algèbre d'Ogasawara est une méthode algèbrique dédiée aux workflows et centrée sur les données implémentant des patrons de conception adaptés à la gestion du parallélisme [8], implémenté dans l'outil CHIRON [9].

— Pig Latin [10] est un langage orienté relationnel permettant de représenter des opérations de type *map*, *sort* et *reduce* sur des fonctions définies par l'utilisateur (UDF pour User Defined Functions). L'exécution s'effectue sur le framework MapReduce [11] Hadoop. Le modèle de données est un modèle relationnel imbriqué. Chaque opération est compilée en MapReduce et les opérations sont chaînées simplement. En bioinformatique, Pig Latin est exploité pour l'analyse de séquences haut-débit, notamment via la librairie SeqPig [12] qui intègre des fonctions dédiées à l'analyse de séquences. SeqPig définit des fonctions d'importation et d'exportation associées à des formats utilisés en bioinformatique mais également des UDF pour le traitement et l'alignement de séquences.

— Plusieurs langages reprenant la syntaxe de SQL sont intégrés à des frameworks d'analyse intensive de données sur clusters base sur le parallélisme de données gros grain. Tenzin [13], Shark [14] et HiveQL [15] sont des langages étendant la syntaxe déclarative de SQL et qui permettent l'exécution sur un environnement de type MapReduce. Hive supporte les UDF. Spark SQL [16] reprend la syntaxe SQL et permet l'exécution sur le framework Spark [17]. A la différence de MapReduce dans lequel les données sont systématiquement écrites sur le système de fichiers, Spark permet de garder les données en mémoire. Spark permet directement l'intégration de fonctions codées par l'utilisateur sans faire appel à un mécanisme d'extension de type UDF. L'outil SparkSeq [18] est un exemple d'exploitation de Spark pour le traitement des données NGS en bioinformatique.

Les méthodes algèbriques relationnelles présentent un intérêt certain pour élaborer des moteurs de workflows d'analyse de données dotés d'une grande expressivité.

1.1.2.2 Méthodes basées sur les graphes

Nous abordons ici les modèles de calcul basées sur les graphes qui constituent un formalisme précis pour modéliser des entités associées par des relations (cf. annexe B). Ils présentent naturellement des aptitudes à la représentation graphique. Cela en fait des modèles de choix pour l'élaboration d'interfaces visuelles de conception de workflows comme on en trouve dans les WfMS.

Modèles de files d'attente

Les modèles de files d'attente (queuing systems) sont utilisés par exemple pour calculer des délais d'attente dans un système et optimiser cette attente et les ressources afférentes. Un système de queue est déterminé par les propriétés de la population d'arrivée comme sa taille qui peut être finie ou infinie et son taux d'émission, qui peut suivre une distribution (poisson, exponentielle...). Les propriétés de la file d'attente à considérer sont sa taille maximum et la stratégie rattachée à la fourniture du service, par exemple FIFO (First In First Out). Dans un modèle de queue, les nœuds représentent des opérateurs complexes associés à des taux d'arrivée par unité de temps ou des décisions. Les arêtes représentent des événements, des jetons ou des requêtes.

Les automates finis

Les automates finis ou machines à états finis (FSM pour Finite State Machines) permettent de définir des séquences de contrôle dans des tâches contrôle-intensives. Une FSM est conçue comme une machine abstraite qui peut se trouver dans un seul état à la fois, parmi une liste d'états prédéfinis. La FSM passe d'un état à un autre suivant une transition déclenchée par un événement ou condition. Une FSM possède un état initial et chaque état est attaché à des entrées/sorties. Dans certaines classes de FSM, on associe des actions aux états qui sont déclenchées lorsqu'on entre dans un état ou lorsque on le quitte. Le modèle FSM n'est pas Turing complet. Une implémentation de FSM a une mémoire limitée à son nombre d'états. Un diagramme états-transitions

(State Transition Diagram ou STD) peut être utilisé pour représenter un FSM. Le STD est un modèle graphique faisant partie d'UML et utilisé pour décrire le comportement de systèmes. Dans le STD, on définit un graphe orienté dont les nœuds représentent des états. Un état peut être rattaché à une action d'entrée et une action de sortie. Il existe au moins un état de départ et un nombre quelconque d'états d'arrivée. Les arêtes représentent des transitions entre les états. Une transition peut être associée à un événement déclencheur, une condition à remplir, une liste d'actions à réaliser. Les événements et conditions peuvent être combinés par des opérateurs AND et OR. Lors d'un changement d'état, les actions suivantes sont exécutées:

— l'action de sortie de l'état de départ;

— la séquence d'actions de la transition;

— l'action de l'état d'arrivée.

Les réseaux de Pétri

Un réseau de Pétri est un MoC qui décrit un système pouvant avoir des processus concurrents, asynchrones, distribués, ou non déterministes. C'est un graphe biparti orienté constitué de deux types de nœuds: les places et les transitions. Des règles d'exécutions définissent la transmission de jetons par les arêtes d'une place à l'autre (franchissement). Ces règles ne sont appliquées que lorsqu'une place d'entrée contient un jeton. Les propriétés des réseaux de Pétri ont été très étudiées, permettant l'exploitation de méthodes d'analyse puissantes notamment pour la validation des modèles représentés [19]. De ce fait, on peut les classer dans les méthodes formelles. Les réseaux de Pétri sont utilisés depuis longtemps dans les systèmes de gestion de workflows [19], notamment pour le BPM (Business Process Management). Le langage de workflows YAWL [20] est basé sur les réseaux de Pétri, mais les étend pour faciliter la modélisation de workflows complexes. YAWL est un langage basé sur les réseaux de Pétri comme représentation intermédiaire et vise à simplifier leur usage. Il s'agit d'un formalisme utilisable dans le BPM mais qui semble avoir peu d'usage dans les WfMS scientifiques orientés analyse intensive de données.

Les réseaux de processus de Khan

Les réseaux de processus de Khan (ou KPN pour Kahn Process Networks) sont des modèles permettant de représenter la concurrence des traitements [21]. Un KPN est un graphe orienté où (i) les nœuds représentent des processus de calcul indépendants et concurrents, (ii) les arêtes représentent des queues FIFO de communication entre processus. Les processus lisent et écrivent des éléments atomiques de données ou jetons (tokens) depuis un canal d'entrée vers un canal de sortie. L'écriture vers un canal de sortie est non bloquante pour le processus, qu'elle réussisse ou non. Par contre, la lecture depuis un canal est bloquante. De ce fait, l'exécution d'un processus ne se poursuit que si le canal entrant contient suffisamment de jetons de données. Dans un KPN, il n'y a pas d'ordonnancement global, le contrôle de l'exécution est complètement distribué entre les processus individuels. Comme l'échange de données a été distribué entre les queues

FIFO des canaux, il n'y a pas de nécessité d'un espace mémoire commun accessible par de multiples processus.

D'autre part, l'exécution d'un KPN est déterministe. En effet, pour une séquence de jetons définis, rattachée à un canal, les exécutions associées à un nœud produiront toujours la même séquence de jetons de sortie quels que soient l'ordre et les durées d'exécution des processus.

Les processus d'un modèle KPN sont monotoniques, ils ne nécessitent qu'une partie d'un flux de données pour produire une partie d'un flux de sortie. La Monotonicité est une propriété intéressante car elle permet le parallélisme. Dans un KPN, chaque processus peut être représenté comme un FSM ayant l'un des deux états, "actif" ou "en attente" de jetons d'entrée.

Les nœuds qui n'ont pas d'arêtes entrantes se comportent comme des sources de données et ceux qui n'ont pas d'arêtes sortantes se comportent comme des puits de données. On peut utiliser ces nœuds particuliers pour représenter respectivement les données d'entrée et les données de sortie du modèle. Les modèles KPN sont particulièrement adaptés à la représentation du traitement de flux de données. L'exécution se termine lorsque tous les processus sont bloqués, en attente de lecture de jetons d'entrée. Cela peut se produire même s'il subsiste des jetons dans certains canaux.

Les modèles DataFlow

Un graphe DataFlow est un graphe orienté dont les arêtes représentent les dépendances de données entre des opérations représentées par les nœuds. L'application est représentée comme un graphe orienté où:

— les nœuds ou acteurs représentent les traitements ou fonctions;

— les arêtes représentent les canaux de communication entre fonctions qui correspondent à des tampons FIFO (first-in, first-out).

Les jetons de données sont produits et consommés sur les arêtes. L'exécution est guidée par les données. En effet, un acteur ne peut être exécuté que s'il reçoit suffisamment de jetons de données sur ses arêtes entrantes. Cette condition est définie par des règles de déclenchement.

Les modèles DataFlow sont en grande partie assimilables à un cas particulier de KPN où les nœuds sont des acteurs ayant des règles de déclenchement (firing) liées à la présence de jetons de données. Néanmoins, les DataFlows possèdent un ordonnanceur global alors que les KPN n'en ont pas. Différentes classes de modèles DataFlow correspondent à des règles spécifiques de production et de consommation de jetons de données.

Par ailleurs, l'ordre d'exécution des acteurs ne fait pas partie de la spécification mais est généralement déterminé par un compilateur ou par la plate-forme d'exécution. L'exécution est itérative et le cœur d'une boucle peut être itéré un grand nombre de fois.

Les modèles DataFlow sont apparus pour les architectures hardwares parallèles mais sont aujourd'hui à la base des moteurs de workflows des WfMS scientifiques.

Nous reviendrons donc en détail sur cette famille de modèles sur laquelle nous centrerons notre étude.

1.1.3 Validation et exécution d'un MoC

Après la phase de conception, l'instance d'un MoC doit être validée suivant des règles cherchant notamment à garantir la terminaison de l'exécution du modèle en détectant par exemple les risques de deadlocks. La validation du MoC peut éventuellement exploiter la simulation ou prendre en compte la compatibilité des données si le modèle exploite un typage des données.

Puis vient l'étape d'exécution du MoC. Dans le moteur de workflows, elle inclut l'ordonnancement des fonctions rattachées aux acteurs. Pour exécuter l'ensemble de processus définis dans le workflow, nous devons déterminer quand et où exécuter les processus, ce qui correspond à l'ordonnancement réalisé par un programme dédié (ordonnanceur ou scheduler).

Dans le cas d'un modèle DataFlow, les processus sont interdépendants et correspondent aux fonctions des acteurs associés aux valeurs des entrées courantes. Pour un modèle DataFlow générique, l'ordonnancement ne peut être décidé que durant l'exécution du DataFlow. On parle alors d'ordonnancement dynamique [22].

Lors de l'exécution, un ordonnanceur dynamique évalue les entrées des arêtes de chaque nœud pour vérifier s'il est exécutable suivant les conditions du modèle. Il choisit un élément de la plate-forme d'exécution puis ordonne l'exécution sur cette cible de la fonction associée au nœud du graphe, peuplée avec les valeurs courantes des entrées.

1.2 L'expressivité dans les langages graphiques

Nous abordons ici la notion formelle d'expressivité d'un langage, appliquée tout particulièrement aux langages graphiques qui trouvent un usage dans la conception de workflows. Un langage de programmation est Turing-complet s'il hérite des caractéristiques d'une machine de Turing. La complétude de Turing décrit la propriété d'un langage à exprimer n'importe quel algorithme. En matière d'implémentation, le langage doit permettre de définir la lecture et l'écriture de variables, la définition de conditions et d'itérations.

L'expressivité dans un langage de workflows, en dehors de la notion de complétude de Turing, peut être évaluée par la notion de couverture des besoins. Ainsi nous pouvons considérer la capacité du langage à représenter des motifs de structure ou de contrôle, simples ou complexes. Il s'agit ici d'une démarche basée sur l'observation des besoins et des usages.

Dans le domaine du BPM, ces motifs font l'objet d'un inventaire et d'une catégorisation sous le terme de "workflow patterns" [23]. Ils référencent des motifs de contrôle simples permettant de modéliser le routage séquentiel, parallèle et conditionnel, des motifs de synchronisation (ET, OU, XOR...). On peut lister des "workflow patterns" généraux et spécialisés, puis déterminer si l'on peut les implémenter avec un MoC ou une algèbre. Mais l'on doit garder à l'esprit que le choix du formalisme de représentation doit être guidé par la facilité des utilisateurs ciblés à implémenter les patrons de conception. En effet, la seule notion d'expressivité des langages ne permet pas de com-

parer des langages dédiés aux workflows sans prendre en compte l'effort fourni par le concepteur. Kiepusewski appelle cela la pertinence (suitability) [24]. Que l'on parle des langages textuels ou des langages visuels de programmation à base de diagrammes, des choix sont opérés qui peuvent privilégier la conception intuitive ou bien l'expressivité.

Par ailleurs, l'expressivité dans les langages graphiques doit être appréhendée de façon particulière.

Les langages aux grammaires textuelles ont un mode de lecture préétabli, suivant linéairement une séquence d'éléments syntaxiques. Par contre, les langages aux grammaires graphiques n'ont pas d'ordre de lecture préétabli et décrivent visuellement leurs éléments sous la forme de propriétés et de relations.

D'après Bertin [25], un élément d'une représentation graphique peut avoir deux rôles. Tout d'abord, elle peut servir de mémoire artificielle ou de dispositif mnémonique. Ensuite, elle peut être un outil de découverte. Dans ce cas, les similarités et différences entre représentations peuvent suggérer des relations et hypothèses non évidentes dans les données. Bertin a défini la notion de *variables visuelles* et les a classées en groupes (forme, taille, valeur, grain, couleur, orientation, coordonnées). Chaque type de variable permet d'exprimer les caractéristiques particulières d'une information, valeur continue, discrète, évolutive dans le temps, combinaisons. Par la suite, Card et Mackinlay ont complèté cette liste de variables visuelles [26] en ajoutant deux types de variables. Une connexion est un lien visuel permettant de représenter des relations entre entités. Une inclusion est un élément graphique dans un autre permettant de rendre compte de la hiérarchie, du groupement ou de l'héritage.

Dans le domaine de la représentation des concepts, Mackinlay propose un critère qui détermine l'efficacité d'un langage visuel. Il met en avant qu'une représentation visuelle est plus efficace qu'une autre si l'information qu'elle véhicule est plus directement perçue. Il formalise la notion d'efficacité (effectiveness) d'un langage ou *éfficacité opérationnelle*. Celle-ci décrit la facilité qu'a une audience cible à décrire une information. L'*éfficacité opérationnelle* s'applique aux langages de programmation, qu'ils soient graphiques ou non.

En programmation, les langages à usage général sont des langages permettant d'écrire des applications dans une large variété de domaines d'application à la différence des langages dédiés qui incluent des éléments destinés à offrir des fonctionnalités spécifiques à un domaine d'application particulier. Pour un langage dédié, on peut préciser la notion d'*éfficacité opérationnelle*:

Définition 1.2 *L'éfficacité opérationnelle d'un langage dédié représente la facilité qu'a une audience cible à décrire les entités et les relations d'un domaine applicatif particulier.*

Mais comment le concepteur d'un langage visuel peut-il accroître son *éfficacité opérationnelle*? Mackinlay propose plusieurs principes pour augmenter l'*éfficacité opérationnelle* d'un langage. Tout d'abord, les propriétés de la représentation graphique ou variables visuelles doivent correspondre strictement à des propriétés des données. Ensuite, la partie la plus importante de l'information doit être représentée de la façon la plus évidente ou la plus majoritaire. Les informations doivent être hiérarchisées par ordre

d'importance. Cela justifie de proposer des représentations différentes d'une même information. Chaque représentation, qualifiée de vue ou de perspective, ciblera une audience particulière ou cherchera à mettre en avant certains groupes de propriétés ou mécanismes.

Il est particulièrement important de comprendre comment l'on peut intégrer efficacement dans les WfMS des fonctionnalités de design visuel des workflows et de leurs composants. La personnalisation dans les systèmes de conception basés sur des diagrammes a été étudiée par MacLean *et al.* [27]. Ces auteurs mettent en avant le concept de "montagne de la personnalisation". La figure 1.1 montre le niveau de savoir faire nécessaire à la réalisation d'un certain degré de personnalisation d'un processus au sein d'un système.

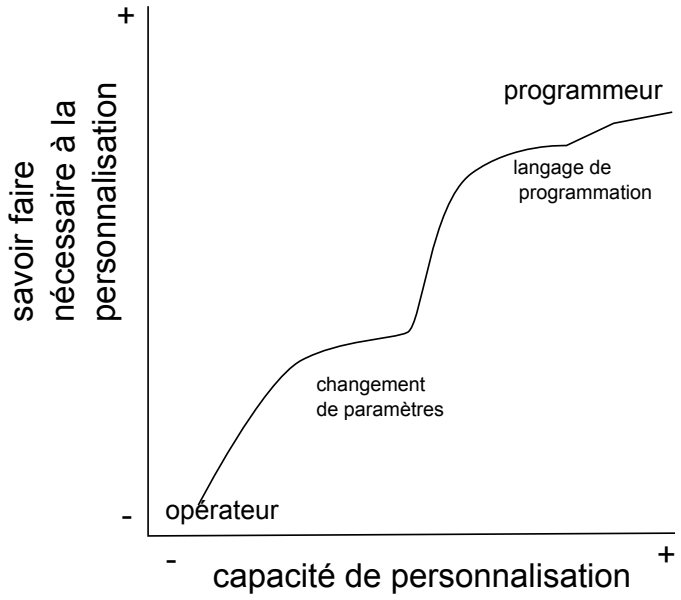


FIGURE 1.1: La montagne de la personnalisation (tailorability mountain) d'après MacLean

D'autre part, Lieberman propose de séparer les activités de conception des utilisateurs finaux en deux classes distinctes suivant un degré de savoir-faire croissant [28]. La première classe représente la paramétrisation ou l'adaptation qui sont des activités donnant le choix à l'utilisateur entre des comportements alternatifs déjà disponibles dans le système. La deuxième classe est constituée de la création et la modification de programmes. Elle concerne les activités qui nécessitent de créer un artefact logiciel ou de modifier un artefact préexistant, incluant les macros, le scripting, et la programmation visuelle. Dertouzos propose, quant à lui, le concept de "système à pente douce" [29]. Dans de tels systèmes, le gain de personnalisation doit s'obtenir de façon proportionnel au savoir-faire nécessaire. Pour réaliser une adaptation, l'utilisateur ne doit pas être confronté à un saut de complexité pour obtenir un gain de puissance de personnalisation.

1.3 Les langages DataFlow

Nous allons centrer notre étude sur les modèles DataFlow. En effet la plupart des moteurs de workflows adaptés à l'analyse de données sont basés sur ces modèles. Nous commencerons par décrire leurs différents variants puis nous nous intéresserons à leurs capacités à faciliter la conception graphique. Enfin nous mettrons en lumière les liens entre langages DataFlow et langages fonctionnels.

1.3.1 Une famille de modèles

Nous décrirons ici en détail les principaux modèles de calcul qui dérivent des principes généraux du DataFlow. En effet, aujourd'hui, les moteurs de workflows des WfMS scientifiques son basés sur des modèles de ce type. Il est donc essentiel de mieux comprendre comment ils sont conçus.
Dans un Synchronous Data Flow (SDF) [30], le nombre des jetons de données consommés et produits par un acteur est connu au moment de l'exécution. On peut ainsi déterminer de façon statique la séquence d'exécutions des nœuds (acteurs). Dans un SDF, l'ordre d'exécution des nœuds n'est pas préétabli. De ce fait le SDF ne permet pas l'utilisation de variables globales. Un autre modèle, nommé DataFlow cyclo-statique (CSDF pour Cyclo-Static DataFlow) permet à l'application de changer de comportement suivant un cycle pré-établi [31].

Quant au modèle DataFlow booléen (BDF pour Boolean DataFlow) [32], c'est une généralisation du SDF. Dans le BDF, les jetons sont typés en deux classes: les jetons représentant les données elles-mêmes et des jetons booléens de contrôle, associés à la gestion des conditions. Pour cela, le modèle BDF introduit l'ajout d'acteurs "switch" et "select" (de-multiplexeur et multiplexeur). On peut alors implémenter des structures de contrôle de type *if-then-else* et *do-while*. Le BDF, plus général que le SDF, permet la représentation de conditions, d'itérations et de la récursivité.

Ainsi, l'acteur "switch" consomme un jeton de contrôle et copie un jeton de données d'une arête entrée vers l'arête de sortie adéquate, choisi en fonction de la valeur booléenne du jeton de contrôle.

L'acteur "select" consomme un jeton de contrôle et copie un jeton de données à partir d'une arête d'entrée, choisi en fonction de la valeur booléenne du jeton de contrôle, vers l'arête de sortie.

Les acteurs "switch" et "select" ne sont pas compatibles avec le modèle SDF puisque le nombre de jetons produits ou consommés n'est pas fixé à l'avance.

En outre, plusieurs modèles dérivés du SDF intègrent la notion de hiérarchie, comme l'Interface-Based Synchronous DataFlow [33]. Un acteur peut ici être composite et représenter un sous-graphe. Les jetons de données entrant d'un acteur composite C au niveau hiérarchique n deviennent les jetons d'entrée d'un des acteurs du sous-graphe représentant C au niveau $n-1$. Les jetons de sortie suivent le chemin inverse, du niveau $n-1$ au niveau n.

L'intérêt est multiple. Tout d'abord, à chaque niveau hiérarchique, les propriétés du modèle peuvent être évaluées et contrôlées sans prendre en compte l'implémentation

des acteurs composites. Ensuite, d'un point de vue du concepteur, ces niveaux forment des frontières qui peuvent correspondre à des vues différentes du modèle, facilitant sa compréhension et l'organisation en composants 1.2.

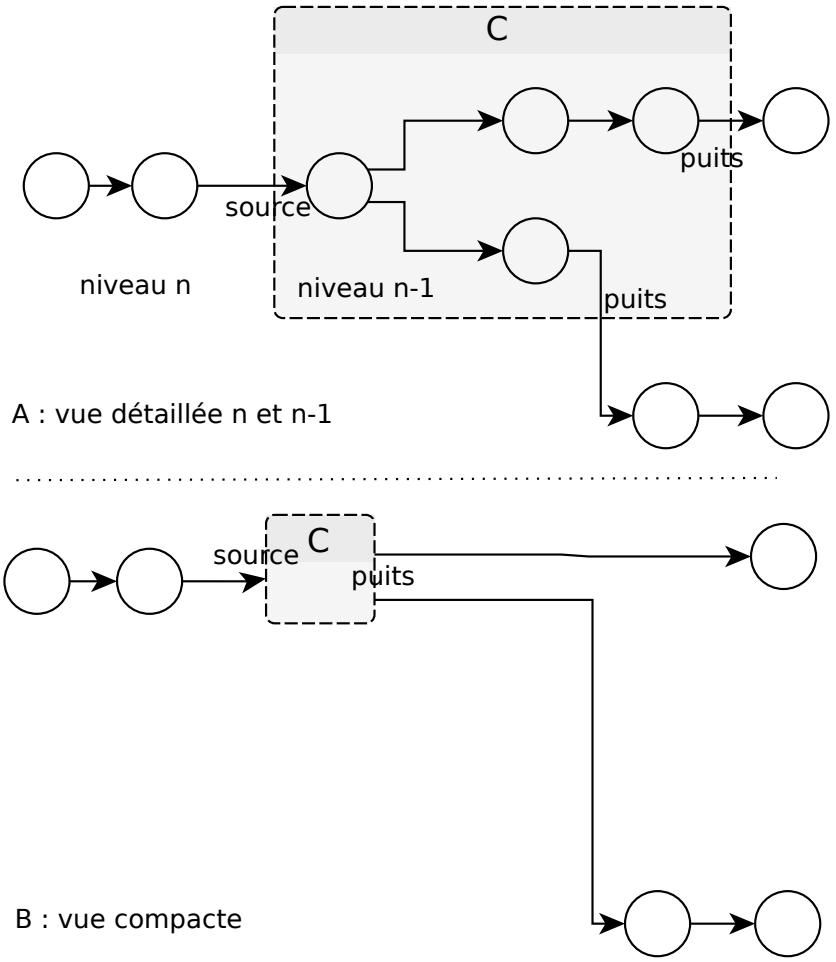


FIGURE 1.2: Acteur composite dans un modèle DataFlow hiérarchique. La hiérarchie structure les représentations graphiques du modèle en vues complémentaires et facilite l'évaluation des propriétés du MoC. L'acteur composite C peut être représenté en tant que sous graphe (A) ou simple acteur (B)

D'autres modèles DataFlow peuvent être employés. Il s'agit, par exemple, des modèles DataFlow dynamiques (Dynamic DataFlow Graphs). Dans ceux-ci, les taux de production et de consommation des jetons de données peuvent varier de façon non prévisible au moment de la phase de compilation [34]. Les modèles DataFlows dynamiques sont caractérisés par une grande expressivité mais l'on ne peut pas toujours se prému-

nir efficacement des risques d'erreurs à la compilation ou l'exécution (buffer underflow, deadlock). On peut obtenir un modèle DataFlow dynamique au comportement déterministe si l'on n'autorise que certaines structures de graphes et types d'acteurs comme dans le cas des graphes à flot régulier (ou RSFG pour Regular Stream Flow Graph) [35]. Il est possible de représenter des modèles DataFlow dynamiques de différentes façons [36] et notamment en combinant explicitement des graphes DataFlows et des automates finis [37, 38, 39].

1.3.2 La nature graphique des langages DataFlow

Nous avons porté notre attention sur une famille de MoC particuliers, les modèles DataFlow, qui constituent la base de nombreux moteurs de workflows des WfMS scientifiques. Une des caractéristiques essentielles des programmes DataFlow est qu'ils peuvent être exprimés graphiquement [40].

Les langages visuels DataFlows (DFVPL pour DataFlow Visual Programming Languages) se sont développés depuis les années 90 avec comme objectif initial l'expression du parallèlisme [41]. Par la suite, une autre utilité des DFVPL a émergée: le génie logiciel. L'adaptation au génie logiciel et au suivi du cycle de vie des applications, de la conception à la mise en production, constitue un des avantages des DFVPL. On peut citer notamment LabView [42], un langage commercial DataFlow toujours utilisé à l'heure actuelle.

Une des raisons de la popularité des langages graphiques DataFlow est qu'ils peuvent être compris par des non programmeurs comme par exemple les clients ou les utilisateurs finaux qui peuvent ainsi mieux dialoguer avec les programmeurs. [43]. Les langages graphiques DataFlow associent généralement étroitement langage et environnement de conception et d'exécution. Il s'agit donc de véritables *frameworks*. De ce fait, les DFVPL sont particulièrement adaptés au prototypage rapide [40]. On retrouve aujourd'hui une partie de ces caractéristiques dans les WfMS scientifiques orientés DataFlow.

1.3.3 Langages fonctionnels et DataFlow

Nous décrivons ici les relations qui existent entre les modèles DataFlow et une famille de langages proches, les langages fonctionnels. Dans cette section, nous mettons l'accent sur leurs points communs et les caractéristiques qui les prédisposent à l'élaboration de langages graphiques. Notre objectif est ici de mettre en lumière que leurs similitudes rendent possible l'adaptation des patrons de conception fonctionnels, particulièrement riches et expressifs, au contexte de la conception graphique de modèle DataFlow.

1.3.3.1 Les langages Fonctionnels

Les langages Fonctionnels possèdent certaines propriétés caractéristiques comme:

— la concision;

— la Composabilité;

— l'aptitude au parallélisme;

— la transparence référentielle;

— l'évaluation par nécessité.

La **concision** correspond à une grande expressivité associée à une syntaxe compacte. Les langages fonctionnels ont une grande aptitude à la concision et permettent de très hauts niveaux d'abstraction. Une seule variable graphique au sens de Bertin [25], peut représenter une fonction dont l'implémentation complexe est masquée.

La **Composabilité** concerne les relations entre les composants. Un langage est composable s'il fournit des composants qui peuvent être sélectionnés et assemblés selon différentes combinaisons pour satisfaire à des exigences spécifiques. La Composabilité d'un langage est liée aux propriétés d'autonomie et d'absence d'état des composants ou fonctions. Chaque composant ou fonction peut être représenté visuellement comme une boîte avec ses paramètres d'entrée et ses sorties (ports I/O).

L'aptitude au parallélisme est permise par l'absence d'effet de bord et de structure de contrôle impérative. Le parallélisme peut être implicite et pris en charge par un compilateur.

La **transparence référentielle** est une propriété d'une expression d'un programme qui représente le fait qu'une expression peut être substituée par sa valeur sans altérer le programme. Ainsi on peut mettre en œuvre des méthodes de tolérance aux pannes en exécutant les fonctions avec les mêmes entrées.

L'**évaluation par nécessité** ou *lazy evaluation* est une stratégie qui retarde l'évaluation d'une expression. La technique d'implantation permet que l'exécution d'une portion de code ne soit réalisée que lorsqu'elle est réellement nécessaire.

Nous décrivons ici certaines des propriétés syntaxiques des langages d'orientation fonctionnelle. Une description plus complète est disponible en annexe (C).

— **Une fonction d'ordre supérieur** est une fonction qui prend une ou plusieurs fonctions comme entrée et/ou renvoie une fonction.

— **Une liste en compréhension** est une construction syntaxique qui définit une nouvelle liste à partir d'un liste existante filtrée. Les listes de compréhension sont utilisées pour définir par une syntaxe très compacte des sous-ensembles d'éléments.

— **Une variable immuable** est une variable dont l'état ne peut pas être modifié après sa création. Dans de nombreux langages fonctionnels, les variables sont immuables par défaut. Cela permet notamment l'exécution des fonctions sans effet de bord. Cela rend possible la parallélisation de données sans verrou. Dans certain cas, le compilateur peut analyser le code, déterminer les fonctions *a priori* les plus lentes, puis définir automatiquement une exécution parallèle. Cela est beaucoup plus complexe dans un langage impératif où chaque fonction peut modifier l'état associé à sa portée extérieure, ce qui peut alors influencer le comportement d'autres fonctions.

— **Les monades** sont des patrons de conception de haut niveau qui permettent d'encapsuler des traits impératifs pour respecter la contrainte d'absence d'état des langages purement fonctionnels. Ces traits peuvent être, par exemple, les effets de

bord ou les exceptions. On peut utiliser une monade pour encapsuler un objet de type prédéfini dans un objet conteneur ayant d'autres attributs afin de gérer des opérations initialement non permises dans le langage de l'implémentation. L'usage des monades présente plusieurs intérêts dont l'analyse simplifiée du comportement d'un programme mais aussi la possibilité d'optimisations dont la parallélisation.

1.3.3.2 La nature fonctionnelle des langages DataFlow

On peut retrouver aujourd'hui certaines des caractéristiques fonctionnelles étudiées précédemment dans des environnements de traitement massivement parallèle des données de type DISCS (Data-Intensive Scalable Computing Systems) comme MapReduce [11] dont les opérations *map* et *reduce* sont directement inspirées des fonctions *map* et *reduce* présentes dans les langages fonctionnels. Des principes d'orientation fonctionnels sont également exploités dans les langages DataFlow, qui nous intéressent ici tout particulièrement de par leur usage dans les WfMS scientifiques. L'intérêt de la programmation fonctionnelle dans l'élaboration d'outils de conception DataFlow est mis à jour depuis longtemps [44]. Il est établi que les langages DataFlow ont de nombreuses propriétés similaires avec les langages fonctionnels [45, 46].

Néanmoins, on a constaté que des programmes complexes qui peuvent être implémentés dans un langage fonctionnel ne peuvent l'être dans un langage DataFlow [40]. La récursivité est par exemple absente de la plupart des langages DataFlows [47]. On pourrait donc considérer ces langages comme des langages fonctionnels, limités à des patrons de conception particuliers. C'est généralement vrai, même si plusieurs langages DataFlow permettent l'emploi d'une syntaxe impérative, notamment pour exprimer les boucles.

Les études sur les modèles DataFlow ont tout d'abord concerné la mise au point d'architectures matérielles hautement parallèles [45]. Ces premiers travaux qui ont mis en lumière un ensemble de propriétés que ces MoC partagent avec les langages fonctionnels comme:

— la composabilité;
— la transparence référentielle;
— des variables immuables;
— l'évaluation par nécessité;
— l'expression simplement du parallélisme potentiel.

Les langages DataFlow permettent l'expression simple du parallélisme potentiel. Ces modèles rendent possible l'ordonnancement implicite des calculs, soulageant le programmeur de la gestion explicite du parallélisme en exploitant l'analyse des dépendances de données. Ce sont les recherches sur l'exploitation du parallélisme qui sont à l'origine des premiers modèles DataFlows [40]. Elles concernaient initialement la programmation des processeurs parallèles puis l'architecture matérielle d'un "ordinateur DataFlow". Les architectures matérielles DataFlow sont des architectures qui diffèrent des architectures classiques de von Neumann. L'architecture de von Neumann distingue différentes parties d'un ordinateur entre:

— l'unité de traitement qui effectue les opérations de base;

— l'unité de contrôle qui gère l'ordonnancement des opérations;

— la mémoire qui contient les données et le programme qui informe l'unité de contrôle des calculs des données à traiter et des calculs à effectuer;

— les entrées et les sorties, qui gèrent la communication avec l'extérieur;

— un registre qui contient l'adresse mémoire de l'instruction qui s'exécute ou va exécuter, appelé le compteur ordinal.

Les architectures matérielles DataFlow n'ont pas de compteur ordinal. L'ordre d'exécution de l'instruction ne peut donc pas être préétabli du fait que l'exécution des instructions est déterminée uniquement lorsque les données d'entrée des instructions sont disponibles. Aujourd'hui, les architectures DataFlow matérielles sont toujours mises en œuvre dans le traitement du signal ou la gestion des réseaux. Les architectures DataFlow sont également purement logicielles et peuvent être, par exemple, exploitées dans la conception de moteurs de bases de données, de calculs parallèles ou de workflows.

On a constaté que le parallélisme utilisé dans les premières architectures matérielles DataFlow qui opéraient à un grain très fin, au niveau de chaque instruction pouvait avoir des performances décevantes, notamment pour les algorithmes qui n'expriment qu'un faible degré de parallélisme. De meilleures performances pouvaient être obtenues à travers la conception d'architectures exploitant les avantages d'un parallélisme plus gros grain [48] et la conception d'architectures matérielles hybrides DataFlow / Von Neuman. Ces architectures matérielles hybrides sont encore étudiées aujourd'hui [49]. Dans les langages DataFlows, on ne trouve généralement pas de construction de concurrence explicite comme des assignements parallèles ou des boucles. L'introduction de ce type de construction serait en contradiction avec les capacités d'expression implicite du parallélisme qui constitue une des caractéristiques les plus séduisantes des modèles DataFlows [40].

Par ailleurs, l'étude de la génération du parallélisme pour les architectures hardwares DataFlow a donné lieu à des méthodes d'optimisation. Ainsi, Johnston [40] expose une méthode de fusion des nœuds produisant des fragments séquentiels dans le graphe DataFlow. L'auteur se réfère notamment aux travaux précurseurs de Bic [48] qui expérimentait une approche dite "Threaded DataFlow". Celle-ci exploite le fait qu'un programme décrit par un graphe DataFlow comporte souvent des sous-graphes composés de séquences d'exécutions temporelles. Les tâches assignées aux nœuds de ces séquences ne pourront jamais s'exécuter en parallèle à cause des dépendances de données. Par conséquent, il ne sert à rien de les assigner à des processeurs différents. Il est nettement plus efficace de placer l'ensemble de la séquence d'instructions dans un seul processus d'exécution, de sorte que la sortie d'un nœud acteur initial peut être utilisée immédiatement par le suivant dans la séquence. L'adaptation de cette approche aux infrastructures actuelles de calcul mériterait d'être etudiée.

Dans cette partie, nous avons détaillé les similitudes entre DataFlow et langages fonctionnels et mieux appréhendé leurs caractéristiques. Nous disposons ainsi d'un cadre théorique pour évaluer les moteurs de workflows et mieux comprendre comment concevoir un MoC performant, adapté aux WfMS.

1.4 Approche de conception orientée modèle

Après avoir détaillé les modèles de calculs qui constituent la base algorithmique des moteurs de workflows et en particulier les DataFlows, nous abordons ici les phases en amont de l'exécution qui concernent la capture des spécifications. Il s'agit de l'étape de la conception du workflow dans les WfMS. Nous allons présenter dans cette section les approches de conception orientées modèle qui sont utilisées en génie logiciel. Même si elles sont encore peu utilisées pour la conception de workflows, leur emploi mérite d'être étudié car elles peuvent structurer et faciliter le design des chaînes de traitements, notamment si l'on vise l'indépendance des workflows vis-à-vis des plates-formes d'exécution. Il s'agit d'approches de conception qualifiées de descendantes. Par une succession d'étapes où des niveaux de détails supplémentaires sont ajoutés, on peut ainsi atteindre un niveau de description suffisant d'un système ou d'un processus. Des modèles abstraits puis de plus en plus concrets sont construits, jusqu'à la production d'un artefact ou l'exécution d'un processus.

L'ingénierie orientée modèle (MDE pour Model Driven Engineering) met la notion de modèle au centre des méthodes de conception. Le MDE est un type de méthode descendante où les différents modèles sont systématiquement interconnectés et qui définit des règles de transformation automatisable entre modèles. Dans le cadre du MDE, la transformation de modèles constitue l'opération essentielle qui permet de spécifier et de mettre en œuvre la production de modèles cibles à partir de modèles sources.

1.4.1 Architectures Orientées Modèle

L'architecture orientée modèle (MDA pour Model-Driven Architecture) est une méthode d'ingénierie orientée modèle qui définit précisément des modèles par rapport aux informations qu'ils apportent sur la description d'une plate-forme (application, environnement ou système). Le MDA peut être considéré comme un sous-ensemble du MDE restreint à la gestion des dépendances d'un logiciel vis-à-vis d'une plate-forme d'exécution. Le MDA cherche à séparer par des modèles successifs les aspects concernant la conception fonctionnelle liée au métier des considérations techniques associées à l'implémentation ou l'architecture. Partant des spécifications métiers, par la définition d'une succession de modèles de plus en plus concrets, on va modéliser le système ou l'application jusqu'à obtenir une implémentation exécutable. Le MDA définit les modèles suivants (cf. figure 1.3):

— Le CIM (Computation Independent Model). Le CIM est le modèle du domaine d'application qui donne une vue du système du point de vue du métier. Le CIM ne contient pas de spécification technique et constitue un référentiel.

— Le PIM (Platform Independent Model). Le PIM est indépendant de toute technologie ou langage lié au déploiement. Il correspond au modèle de conception métier spécifique au système. Le PIM ne contient pas de détail sur son utilisation sur la plate-forme.
— Le PSM (Platform Specific Model). Le PSM est le modèle qui décrit les éléments spécifiques à la plate-forme de déploiement et doit permettre de produire l'implémentation de l'artefact désiré (système, programme).

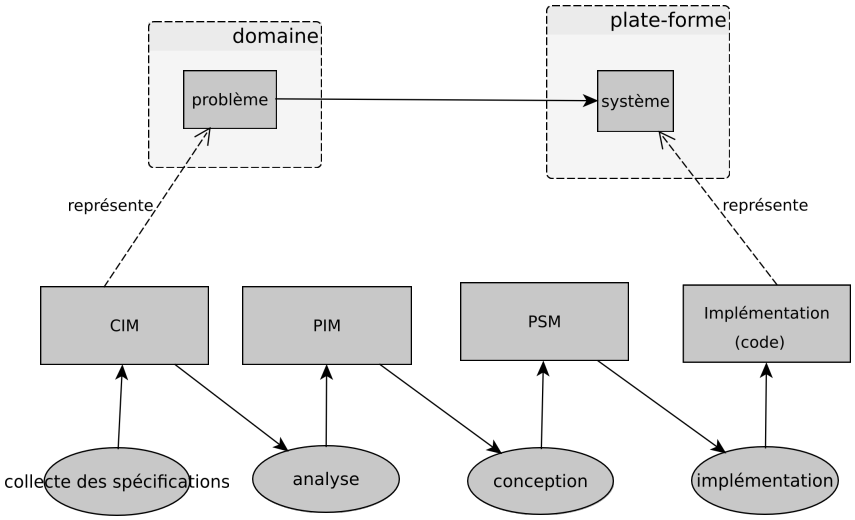


FIGURE 1.3: Les modèles du MDA.

Des transformations qui instancient les différents modèles définissent une traçabilité stricte entre les différents modèles du processus MDA. Cette traçabilité doit notamment permettre des transformations inverses (cf. figure 1.4).

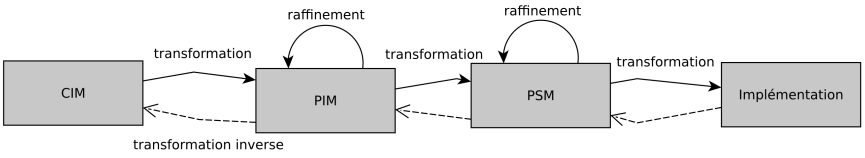


FIGURE 1.4: les transformations du MDA.

Les méta-modèles en MDA

Un méta-modèle est un modèle qui définit les entités, leurs propriétés, leurs relations, et la sémantique associée au modèle. Il décrit la structure à laquelle les modèles doivent

se conformer. De plus, le méta-modèle précise des règles de cohérence et rend ainsi possible la validation des modèles par les outils dédiés. MOF (Meta-Object Facility) est un standard développé par l'OMG (Object Management Group) [50] permettant de représenter des méta-modèles. MOF utilise une architecture de référence à 4 niveaux de modélisation orientée objet.

Ces niveaux sont:

— M3, le méta méta-modèle, niveau le plus abstrait spécifié par le MOF. M3 décrit le méta-modèle M2;

— M2, les méta-modèles d'une discipline ou domaine particulier;

— M1, les modèles manipulés, instances de M2;

— M0, le niveau le plus concret, correspondant aux artefacts produits ou à l'exécution.

Le niveau n est instancié à partir du modèle fourni par un niveau $n+1$ et le niveau supérieur M3 s'autodéfinit. UML peut servir à représenter les modèles notamment les niveaux M1 et M2. Si on applique cette représentation à la conception d'un programme on peut définir:

— la description de la grammaire du langage au niveau M2;

— la description du programme dans un langage au niveau M1;

— l'exécution du programme au niveau M0.

Dans le cas des chaînes de traitements, on peut, par exemple, définir les niveaux de modélisation suivants:

— M2 concerne les méta-modèles qui incluent la description des données et des méthodes exploitées dans les chaînes de traitements;

— M1 correspond aux modèles graphiques ou textuels de représentation des processus incluant le modèle de calcul;

— M0 est le niveau de l'implémentation et de l'exécution des workflows.

Finalement, si nous considérons le workflow comme une simple application, nous pouvons conclure que le MDA et la notion de méta-modèle présentent un intérêt pour mieux formaliser la démarche de conception. L'usage du MDA pour produire des spécifications génériques (PIM), indépendantes des implémentations dédiées aux plates-formes d'exécution (PSM), nous paraît particulièrement intéressant si nous pouvons l'adapter à la création de workflows dans les WfMS scientifiques d'analyse de données.

1.5 Ontologies et MDA

Nous nous penchons maintenant sur les travaux qui associent l'usage du MDA et des ontologies. Les ontologies sont de longue date employées en bioinformatique et il nous paraît naturel d'établir un parallèle entre l'usage des méta-modèles et des ontologies. Nous abordons ici l'idée de l'élaboration d'un méta-modèle servant à guider la conception, dans lequel on intègre des données provenant des ontologies décrivant les données et processus du domaine métier.

1.5.1 L'apport des ontologies dans la conception logicielle

Le Web Sémantique a mis en avant la notion d'ontologie. Une définition d'ontologie [51] communément admise est la suivante:

Définition 1.3 *Une ontologie est une spécification explicite et formelle de conceptualisations partagées*

Il s'agit d'une forme modèle qui possède des propriétés particulières. Elle représente une information partagée au sein d'un groupe qu'il soit étendu ou non. C'est une spécification qui peut être partielle et qui se caractérise par la description et la conceptualisation de choses du monde réel. C'est un modèle descriptif et partagé représentant la réalité par un ensemble de concepts, leurs relations et leurs contraintes.

Des communautés d'intérêt, comme des groupes de chercheurs partageant un même object d'étude, sont souvent à l'origine de l'élaboration d'une ontologie. C'est une ensemble d'individus partageant une expertise dans un espace sémantique et concerné par l'élaboration d'un vocabulaire commun, qu'il maintient et fait évoluer suivant l'état de l'art. Une ontologie est une connaissance collective dont la qualité est garantie par une communauté d'experts qui est préexistante à l'élaboration d'une application. Cette connaissance peut être théoriquement réutilisée dans le processus de conception de l'ensemble des artefacts qui manipulent les mêmes entités du monde réel.

On s'intéresse depuis peu à l'intégration des ontologies dans la conception logicielle notamment dans le MDA. Cette intégration est basée sur une perception complémentaire de ces deux types de modèles, precisée par Aßman. L'auteur oppose le rôle *descriptif* des ontologies au rôle *prescriptif* des modèles manipulés dans l'approche MDA [52]. Le MDA fournit une méthodologie qui prescrit des types de modèles adaptés aux différentes phases de conception d'un système et à leurs relations. Il se focalise sur la maximisation de la réusabilité, de la portabilité et de l'interopérabilité. Le MDA cherche a séparer les aspects métiers des aspects techniques, notamment par la recherche de l'indépendance par rapport aux plates-formes. Cependant, il n'aborde que très peu l'intégration de la sémantique du domaine métier associée à l'artefact à produire. Cette sémantique peut être informellement représentée dans le processus de conception par de la documentation ou sous-jacente dans le choix des entités et relations contenues dans les modèles.

Une intégration de la sémantique plus structurée peut être bénéfique au MDA notamment en limitant les ambiguïtés. Lors de l'élaboration d'applications complexes, différents intervenants sont généralement impliqués, chacun spécialiste d'aspects métiers ou techniques. Dans ce cas, une intégration formelle de la sémantique peut limiter les mauvaises interprétations des modèles. Il est admis que les ontologies peuvent améliorer l'approche MDA en permettant une représentation explicite et non ambiguë du vocabulaire et des objets du domaine métier, de la discipline. L'usage des ontologies associées à des espaces de nommage particuliers pourrait faciliter la communication entre multiples intervenants lors des différentes phases du cycle de vie des applications comme lors de la collecte des

besoins, de l'analyse et de la conception. Elles pourraient être utilisées dans la validation de modèles, permettre une meilleure expressivité ou faciliter l'automatisation de certaines transformations.

Pour promouvoir cette intégration, L'OMG a proposé l'ODM (Ontology Definition Metamodel) [53], une spécification qui met en correspondance les éléments de syntaxe des langages associés au Web Sémantique, OWL (Web Ontology Language) et RDF (Resource Description Framework), avec des représentations UML, un langage graphique de conception communément employé, notamment en MDA. Si l'on s'intéresse au cas particulier des systèmes distribués, on peut considérer que l'usage d'ontologies présente un intérêt pour faciliter l'interopérabilité. Ainsi les Architectures Orientées Service (SOA) dans lesquelles les notions de réutilisation et de découverte sont primordiales peuvent bénéficier de façon évidente de l'apport des ontologies. Dans ce type d'applications, la définition de vocabulaire peut faciliter la composition et la recherche de compatibilité entre composants. Ces principes ont, d'ailleurs, déjà été appliqués en bioinformatique, par exemple dans le framework de composition de service Bio-jETI [54] et le WfMS Wings [55].

1.5.2 Formalisation de l'intégration des ontologies en MDA

L'intégration des ontologies en MDA a été étudiée sous l'angle de:

— l'aide à la conception;

— la vérification de modèles;

— la synthèse d'implémentations.

Ainsi, Zivkovic [56] met en avant l'intérêt des ontologies de domaine pour guider la conception d'un modèle. Celles-ci permettraient d'orienter l'utilisateur en proposant des informations sur les éléments à considérer lors de chaque étape de conception.

En outre, l'auteur évoque l'emploi des ontologies pour la vérification de cohérence entre un modèle conçu par l'utilisateur et des modèles de référence. Il suggère notamment l'application à la composition de workflows en BPM.

D'autre part, certains travaux sur le MDA traitent de l'intégration d'ontologies formalisant les dépendances des plates-formes pour la gestion des configurations. Ainsi Wagelaar[57] met en avant une *ontologie de plate-forme* qui expliciterait suffisamment finement les spécificités du contexte de déploiement pour rendre possible la génération de PSM compatibles avec non seulement la plate-forme ciblée et ses variants mais aussi avec d'autres plates-formes non envisagées initialement.

Une formalisation générale de l'intégration des ontologies dans un processus de conception MDA a été proposée par Aßman [52]. Elle structure les niveaux d'intégration des ontologies dans un processus de conception logicielle et définit une terminologie (cf. figure 1.5).

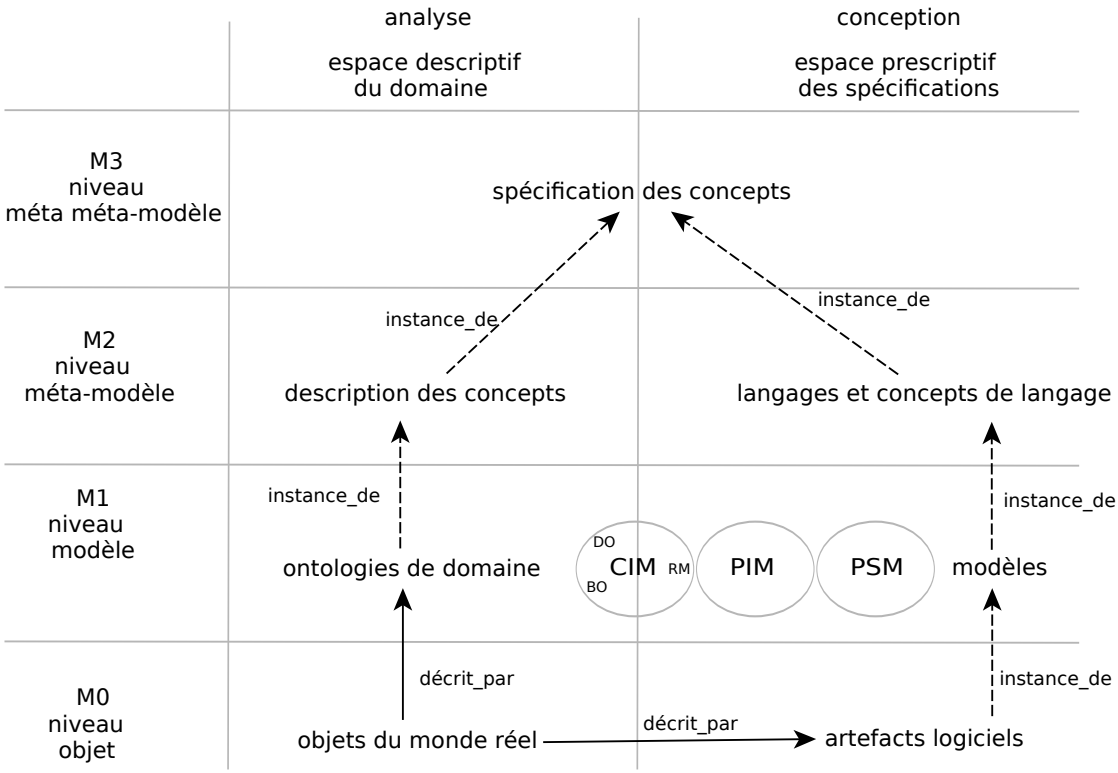


FIGURE 1.5: la méta-pyramide représentant l'intégration des ontologies au MDA d'après Aßman

Le modèle CIM (Computation Independent Model) du MDA (cf. figure 1.3) est généralement le modèle référentiel donnant le point de vue du métier (niveau M1) . Il est subdivisé ici en 3 sous-modèles ou compartiments:

Deux dans l'espace descriptif et un dans l'espace prescriptif:

— le CIM-BO (Business Ontology) contient les ontologies représentant une description des règles métiers;

— le CIM-DO (Domain Ontology) contient les ontologies représentant une description du domaine d'activité;

— le CIM-RM (requirements) décrit la spécification des exigences.

Cette représentation dite "ontology-aware meta-pyramid" permet de formaliser concrètement l'intégration des ontologies dans le processus de conception MDA. Le concepteur peut commencer l'analyse en collectant des ontologies du domaine et des modèles d'analyses standardisées préexistantes. Ces éléments sont ensuite raffinés en des modèles de conception puis on intègre les exigences (RM) pour produire un modèle CIM complet. Le CIM est alors raffiné en PIM puis par une succession de PSM, une implémentation est générée.

Le formalisme développé par Aßman est proposé de façon générale pour la conception d'applications. Mais nous pensons qu'il pourrait être adapté pour structurer une approche MDA utilisant les ontologies pour guider la conception de workflows.

1.6 Résumé

Dans ce chapitre, nous avons exposé un état de l'art des méthodes de conception et d'exécution des workflows.

— Nous avons tout d'abord présenté les grandes familles de modèles de calcul (MoC) et tout particulièrement les modèles DataFlow, qui sont prépondérants dans les moteurs de workflows des WfMS scientifiques destinés à l'analyse de données. Nous avons constaté que les travaux précurseurs sur les architectures matérielles DataFlow trouvent de nouvelles applications avec la nécessité de traiter des données massives dans les worfklows.

  Par ailleurs, nous avons mis l'accent sur les relations entre DataFlows et langages d'orientation fonctionnels. Nous considérons que les similarités entre ces deux types de langages nous permettent de disposer d'un cadre théorique élargi. Celui-ci rend possible l'élaboration de modèles de calcul mieux adaptés aux chaînes de traitements d'analyse de données. Il ne faut pas oublier que les modèles DataFlow n'ont pas été pensés initialement pour les moteurs de workflows. Ils convient donc de les revisiter en prenant en compte les contraintes d'un nouveau contexte: celui des systèmes de gestion de workflows.

— Nous avons aussi détaillé la notion de langage graphique et d'expressivité d'un langage. Les principes de conception qui en découlent sont employables dans les WfMS pour faciliter la capture des spécifications des workflows. Nous retirendrons

que l'expressivité d'un langage de capture de worfklow n'est pas incompatible avec sa facilité d'utilisation, même si pour garantir son *éfficacité opérationnelle*, il est nécessaire d'appliquer des principes de hiérachisation.

— De plus, nous avons présenté les méthodologies orientées modèle et l'usage qu'elles font des ontologies. L'approche MDA permet de formaliser une démarche de conception exploitant successivement différents modèles et méta-modèles strictement définis. Il existe peu d'exemples d'usage du MDA associés aux ontologies pour la conception de workflows mais nous pensons que cette approche présente un fort potentiel et qu'il est possible de s'inspirer des travaux concernant la conception logicielle en général.Nous pensons en particulier à l'usage des ontologies de domaine pour guider la conception [56].

Dans le prochain chapitre, nous étudierons les systèmes de gestion de workflows scientifiques. Nous verrons comment les WfMS qui exploitent des moteurs basés sur des modèles DataFlow facilitent la capture et l'exécution des chaînes de traitements. Nous comparerons les caractéristiques de ces environnements logiciels et leur adaptation à la conception et à l'analyse intensive de données.

Chapitre 2

Etat de l'art des techniques et usages

Dans ce deuxième chapitre, nous présentons les qualités et limites des WfMS scientifiques utilisés pour l'analyse de données massives. Nous détaillons les aspects techniques relatifs à la conception de ces environnements et étudions leurs usages actuels.

Nous comparerons les caractéristiques de certains d'entre eux, utilisés à l'heure actuelle en science, en particulier en bioinformatique, sans chercher l'exhaustivité mais en mettant l'accent sur les plus représentatifs. Nous détaillerons en particulier les WfMS basés ou inspirés des patrons de conception des Architectures Orientées Service (SOA pour Service Oriented Architecture). Leur adaptation à la conception de workflows et notamment au prototypage sera étudiée en détail. Nous évaluerons le degré d'autonomie laissé au concepteur en ce qui concerne non seulement l'orchestration mais aussi la création et de nouveaux composants.

Nous tenterons de mieux appréhender comment ces éléments impactent l'audience potentielle de ces environnements. Nous présenterons ensuite les infrastructures de calcul intensif qui sont aujourd'hui mises à disposition via les environnements de gestion de workflows. Nous détaillerons en particulier les modes de *cloud computing* qui semblent aujourd'hui les mieux adaptés aux besoins des scientifiques et comment les WfMS répondent aux besoins de simplification du déploiement.

Ensuite, afin de mieux cerner si une exécution efficace des chaînes de traitements conçue dans un WfMS est possible, nous rappellerons succinctement les différents types de parallélisme exploitables, en mettant l'accent sur les plus accessibles. Ainsi, nous pourrons mieux appréhender comment les WfMS facilitent la création d'implémentations performantes et s'ils tirent parti efficacement du parallélisme extractible des workflows.

Finalement, Nous identifierons les caractéristiques qui déterminent l'adaptation des WfMS à la conception, l'exécution et à la diffusion des chaînes de traitements d'analyse intensive de données. Ainsi, nous serons à même de déterminer les principaux manques de ces environnements sur lesquels nous considérons qu'il est important de se pencher.

Terminologie

Nous précisons ci-dessous les définitions de termes que nous utilisons de façon récurrente et qui concernent l'objet de cette étude.

— **Composant**: on dénomera composant des unités élémentaires d'un logiciel, assemblées de façon faiblement couplée, dans le cadre ou non des architectures orientées service (SOA) et des WfMS.

— **Chaîne de traitements**: nous qualifierons de chaîne de traitements le produit résultant de l'assemblage de composants. Une chaîne de traitements peut être implémentée dans un langage quelconque et peut ou non nécessiter l'usage d'un moteur de workflows.

— **Acteur**: nous désignerons par acteur un composant unitaire utilisé dans une démarche SOA ou dans un système de gestion de workflows.

— **Workflow**: le terme de workflow sera employé pour parler du produit résultant de l'assemblage d'acteurs, dans le contexte des architectures orientées service et des systèmes de gestion de workflows. La spécification d'un workflow suit un formalisme particulier. Son exécution nécessite un moteur de workflows.

— **Moteur de workflows**: un moteur de workflows est un artefact logiciel qui permet l'exécution (ou orchestration) d'une spécification de workflows suivant un modèle de calcul particulier.

— **Haut-débit**: l'expression de "haut-débit" qualifie des processus exploitant efficacement de grandes quantités de données.

— **Wrapper**: un wrapper est un composant adaptateur qui permet de faire fonctionner des composants ayant des interfaces incompatibles. Le wrapper encapsule le composant dont il normalise l'interface.

— **Descripteur de déploiement**: nous emploierons le terme de descripteur de déploiement pour définir un fichier de configuration permettant le déploiement d'un artefact ou composant dans un environnement logiciel.

2.1 Les systèmes de gestion des workflows scientifiques d'analyse de données

2.1.1 Définitions

La gestion des processus métiers (BPM pour Business Process Management) est une discipline qui vise à modéliser les flux d'activités métiers dans l'entreprise. La Workflow Management Coalition (WfMC) est une organisation qui fédère notamment des éditeurs et utilisateurs de solutions BPM. Elle a pour mission de promouvoir l'usage des workflows et d'établir une terminologie commune et des standards. Elle propose des définitions de workflow et de wfms (WfMS pour Workflow Management System) [58]:

Définition 2.1 *"Un workflow correspond à l'automatisation d'un processus métier dans sa globalité ou en partie, durant lequel des documents, informations et tâches sont passés d'un participant à un autre suivant des règles procédurales".*

Définition 2.2 *"Un WfMS est un système qui définit complètement, gère et exécute des workflows à travers l'invocation de logiciels dont l'ordre d'exécution est lié à une représentation informatique de la logique du workflow."*

Plus précisément, le wfms est considéré comme un système qui définit, crée et gère l'exécution de workflows à travers l'usage d'une application. Cette application utilise un ou plusieurs moteurs de workflows, qui sont capables d'interpréter la définition du processus, d'interagir avec les participants du workflow et invoquer les applicatifs appropriés. Même si nous ne traitons pas ici en détail les aspects d'interactions entre intervenant et processus et que nous nous concentrons sur les processus d'analyse de données, ces définitions très larges issues du domaine du BPM sont applicables aux domaines scientifiques. Nous considérons que la généricité des principes développés dans le BPM justifie de chercher à transposer certains de ses formalismes.

2.1.2 Typologie des WfMS

Afin d'identifier les WfMS les plus à même de faciliter la conception et l'exécution des workflows d'analyse intensive de données, nous allons commencer par établir un rapide panorama des très nombreux types de WfMS qui existent.

Certains des WfMS sont adaptés à l'intégration des processus métiers dans une organisation et possèdent, par exemple des capacités à formaliser les interactions entre participants ou entre entités. D'autres sont dédiés à des traitements de données de type ETL (Extract, Transform, Load) ou à la fouille de données. De multiples travaux cherchent à comparer les caractéristiques des WfMS scientifiques [59, 60]. Nous pouvons nous référer à la classification de Yu et Buyya [60]. Même si elle s'applique initialement au contexte des grilles, elle permet de caractériser les WfMS scientifiques en ce qui concerne le mode de conception, d'ordonnancement, de gestion des erreurs et de déplacement des données (cf. figure D.1 en annexe).

Nous cherchons ici à identifier les types de WfMS capables de gérer au mieux des workflows scientifiques d'analyse intensive de données et notamment ceux que l'on trouve en bioinformatique. Pour cela nous pouvons prendre en compte des critères particuliers que nous avons identifiés comme l'expressivité du langage de capture des spécifications, la prise en compte du prototypage, l'autonomie du concepteur ou la capacité à produire des implémentations performantes.

Nous n'évoquerons ici que certains WfMS. En effet, l'étude exhaustive des très nombreux environnements disponibles aujourd'hui dépasse le cadre de cette étude. Nous présenterons des applications, soit caractéristiques d'une architecture logicielle spécifique, soit particulièrement adaptées à la conception et à l'exécution des chaînes de traitements d'analyse de données.

TABLE 2.1: Présentation des WfMS scientifiques et frameworks principaux adaptés à l'analyse de données

outil	url	publications
SHIWA	shiwa-workflow.eu	[61]
CHIRON	sourceforge.net/projects/chironengine	[9]
MOBYLE	mobyle.pasteur.fr	[62]
GALAXY	galaxyproject.org	[63]
TAVERNA	taverna.org.uk	[64]
KEPLER	kepler-project.org	[65]
PEGASUS	pegasus.isi.edu	[66]
SWIFT	ci.uchicago.edu/swift	[67]
KNIME	knime.org	[68]
WINGS	wings-workflows.org	[69, 70]
MOTEUR2	modalis.polytech.unice.fr/moteur2	[71]

outil	parallélisme de tâches	parallélisme de données	interopérabilité des workflows	orienté domaine
SHIWA	délégué	non	Oui (Méta-worfklow)	générique
CHIRON	oui	oui, approche des bases de données	non	générique
MOBYLE	Oui, restreint.	non	non	Bioinformatique
GALAXY	Oui , restreint à une seule liste d'entrées	Oui mais restreint à certains formats et non paramétrable	non	bioinformatique
TAVERNA	pas de parallélisation	pas de parallélisation. A implementer dans les acteurs	non	Mixte. Intêret essentiellement générique
KEPLER	oui	non	non	générique
PEGASUS	Oui avec DAGman	non	non	générique
SWIFT	oui	non	non	générique
KNIME	oui	Oui mais orienté matrices et non fichiers	non	Data-mining, statistiques bioinformatique, chimie
WINGS	oui	oui mais par intégration manuelle	non	générique
MOTEUR2	oui	non	non	générique

FIGURE 2.1: Comparatif des fonctionnalités des WfMS scientifiques et frameworks adaptés à l'analyse de données (1)

outil	interface	composition et capture graphique	génération de code	infrastructure supportée	localisation des données
SHIWA	concepteur	déléguée	non	déléguée	déléguée
CHIRON	concepteur	non	non	Cluster	non
MOBYLE	Utilisateur,web	non	non	Cluster	oui, monosite
GALAXY	utilisateur et concepteur (workflow mais pas composant), web	orchestration et paramétrage des acteurs	non	Cluster, cloud	oui, monosite
TAVERNA	concepteur	oui	non	PC , cluster par acteur avec API	non
KEPLER	concepteur	oui mais très complexe	non	PC (multithreading), cluster par acteur avec API	non
PEGASUS	prototype (java web start)	non	non	Cluster, Grille et cloud	oui
SWIFT	non	non	non	cluster, grille	oui
KNIME	concepteur	oui	non	PC (multithreading), Cluster	non
WINGS	concepteur	oui	oui	Cluster, Grille et cloud via Mesos ou PEGASUS	non
MOTEUR2	non	non	non	Grille (notamment via P-GRADE)	oui

FIGURE 2.2: Comparatif des fonctionnalités des WfMS scientifiques et frameworks adaptés à l'analyse de données (2).

Néanmoins, nous avons réalisé une étude comparative synthétique des fonctionnalités des principaux WfMS scientifiques et frameworks utilisés pour l'analyse de données (cf. tables 2.1, 2.1, 2.2). Pour un comparatif plus détaillé, on peut se référer à des revues reprenant notamment d'autres critères [72], [73].

2.1.3 Clients lourds

Tout d'abord, nous allons décrire un premier type d'environnements logiciels. Ce sont les WfMS basés sur des clients lourds qui sont utilisés pour orchestrer des services adaptés au calcul intensif. Ce type d'environnements donne généralement accès à des Web Services basés sur les spécifications SOAP/WSDL [74] mais également à des services ne suivant pas de norme de spécification (Rest, services locaux encapsulant des lignes de commande...).

Taverna est un exemple de ce type de WfMS [64]. C'est un outil graphique de conception et d'exécution de workflows, orienté service, qui facilite l'orchestration des services distants ou locaux. Les architectures des WfMS basées sur des Web Services de type SOAP/WSDL distribués, illustrées en bioinformatique par le couple BioMoby/Taverna [75], n'ont eu qu'un succès limité pour plusieurs raisons. Tout d'abord l'absence de sémantique associée aux services [76] limite la capacité de découverte des services. On peut néanmoins pallier ce manque par l'ajout de descriptions sémantiques au Web services via des technologies comme OWL-S. Celles-ci permettraient de faciliter la découverte automatique de services et leur composition dans le cadre d'un modèle d'exécution distribuée entre domaines [77]. Par ailleurs, c'est seulement lorsque l'organisation qui donne l'accès aux services garantit la disponibilité et la maintenance des outils [78] que l'on peut envisager l'usage de Web Services externes et distribués au sein d'un processus de production d'analyse de données. Celui-ci nécessite en effet un niveau élevé de qualité de service (Quality of Service ou QoS).

L'adaptation au calcul intensif implique en outre l'intégration dans ces environnements de middlewares spécialisés dans le déplacement des données et la soumission des jobs. Ainsi, des couches applicatives dédiées comme OPAL2 [79] permettent la mise à disposition de groupes d'opérations de Web Services permettant d'exécuter une commande via l'ordonnanceur d'un cluster (cf. figure 2.3).

Ces middlewares sont adaptés aux services ayant des durées importantes d'exécution et sont basés sur un patron de conception asynchrone associant une opération de soumission, une autre donnant l'état d'avancement et une troisième permettant d'avoir accès aux résultats.

Ainsi, lorsqu'ils sont associés aux interfaces de programmation (API pour Application Programming Interface) d'OPAL2 ou de PBS, encapsulées dans les acteurs, les WfMS Taverna et Kepler [65], peuvent exécuter des processus sur des clusters. Triana [80], quant à lui, est un WfMS qui inclut directement des fonctionnalités similaires et peut exploiter notamment des services exposés par des middlewares dédiés aux grilles (GridFTP, GAT) et à des systèmes de soumission Peer-to-Peer. Wings [81, 69] se différencie des précédents WfMS en intégrant notamment des fonctionnalités de gestion de la provenance et un intéressant générateur de code.

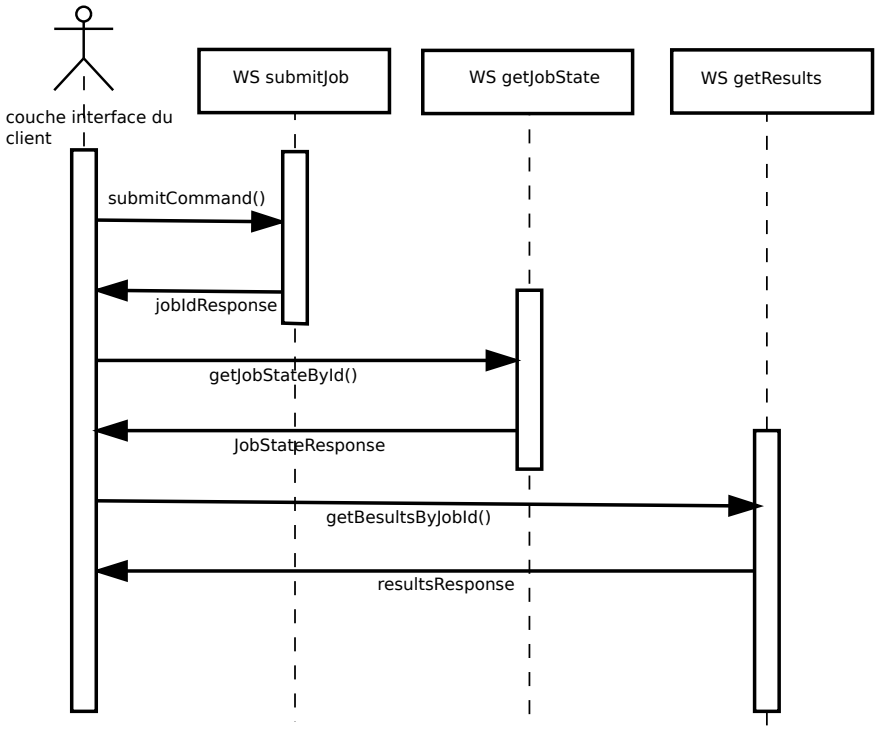


FIGURE 2.3: Diagramme de séquences UML représentant les différents appels d'opérations de Web Services permettant l'invocation asynchrone de méthodes à longue durée d'exécution. Le middleware OPAl2 implémente ce patron de conception.

Cette classe de logiciels de gestion de workflows comprend des outils souvent à vocation générique. Si l'on observe les workflows définis dans ces environnements, on constate que de nombreux acteurs doivent être définis laborieusement par le concepteur pour une seule exécution intensive.

On peut ainsi être contraint de définir successivement un acteur d'upload, de soumission, d'attente, puis de download. Le graphe DataFlow résultant contient de nombreux nœuds qui correspondent à des acteurs techniques et de ce fait, ne représente pas clairement les étapes majeures de l'analyse. Kepler permet cependant le masquage de cette complexité du DataFlow en encapsulant une partie du graphe dans des acteurs composites. Par ailleurs, dans ce type de WfMS, même si la création de composants est souvent accessible aux concepteurs de workflows, elle reste néanmoins technique et complexe.

2.1.4 Environnements SaaS

Abordons maintenant le type de WfMS le plus populaire à l'heure actuelle en bioinformatique. Il s'agit des WfMS client/serveur basés sur les technologies du web. Ceux ci intègrent des API de soumission sur des ordonnanceurs et sont particulièrement adaptés aux traitements sur clusters. On peut citer les applications open-source Mobyle [62] et Galaxy [63] dans cette catégorie. L'outil commercial IGOR [82] fait également partie de cette classe de WfMS. En bioinformatique, Galaxy a été progressivement adopté par de nombreuses organisations pour la mise à disposition de composants et de chaînes de traitements en tant qu'environnement de production basé sur des infrastructures HPC. Galaxy intègre une large collection de composants d'analyses bioinformatiques. Ceux-ci sont prédéfinis sous la forme d'acteurs disponibles pour l'exécution directe ou l'orchestration par le biais d'un MoC d'inspiration DataFlow. L'intérêt de l'approche workflow est ici amplifié par la disponibilité d'un grand nombre de composants intégrés par une communauté active. Galaxy, comme de nombreux WfMS scientifiques, propose de simplifier les spécifications en utilisant une interface d'édition d'un graphe et permet de soumettre les traitements sur des clusters via l'accès à l'ordonnanceur. Ainsi, un workflow peut être conçu et exécuté par le biais d'une interface web mais c'est également possible via des API. Galaxy, par exemple, propose une API rest, BioBlend, [83] pour coder des tâches de soumission. Un workflow peut ainsi être soumis puis monitoré par une série de services. Cependant, les potentialités des API des WfMS, notamment pour la construction d'environnements d'analyse *Ad hoc*, n'ont pas encore été évaluées.

On peut considérer que l'association des technologies des Web Services et de middlewares adaptés au calcul intensif peut être une solution intéressante pour implémenter des WfMS dans le cadre d'une architecture SOA fortement ouverte et évolutive. Cependant, aujourd'hui, c'est une toute autre architecture qui a été choisie. En effet, les environnements comme Galaxy privilégient l'internalisation des fonctions de découverte, de composition et d'exécution. Ils s'affranchissent complètement des spécifications et technologies des Web Services. On abandonne la distribution des services et les implémentations de la plupart des acteurs doivent être déployés indépendamment sur l'infrastructure (script, binaires et dépendances). L'abandon de la distribution des services a plusieurs avantages. Tout d'abord, les déplacements de données d'entrée et de sortie

entre composants sont limités, ce qui constitue une solution aux problèmes des flux massifs de données engorgeant les réseaux durant les traitements. En outre, ces architectures centralisent la gestion de la qualité de service, ce qui facilite l'obtention d'un haut niveau de QoS.

Néanmoins, on peut considérer que l'abandon des Web Services est une régression en ce qui concerne l'interopérabilité. La compatibilité directe des composants entre différents WfMS est perdue ainsi que le principe de la découverte de services. Cette disparition est néanmoins compensée à un autre niveau. En effet les composants sont souvent largement diffusés au sein d'un répertoire web partagé, parfois disponible sous forme de machines virtuelles pré-configurées (Virtual Appliances), et utilisables par l'instance du WfMS que l'on exploite.

2.1.5 WfMS et environnements intégratifs

Dans les plates-formes logicielles comme Galaxy ou Mobyle, la gestion des workflows est associée à d'autres fonctionnalités, permettant le travail collaboratif, la gestion des données et la visualisation des données. La conception et l'exécution des workflows sont des fonctionnalités au cœur d'un riche environnement intégratif d'analyse, dédié à une communauté ou à un domaine scientifique.

Une des conséquences de ce choix d'architecture est que le WfMS peut potentiellement exploiter des informations issues d'autres fonctionnalités de l'environnement. Ainsi, une partie de la connaissance du domaine scientifique autour des données et des traitements y est capturée, formellement ou non. Celle-ci peut être considérée comme une source de méta-données exploitables.

Ainsi, on peut noter que ces plates-formes disposent de leur propre système interne de classification des composants mais également de typage des données permettant des conversions et facilitant la compatibilité entre composants. Ces éléments, souvent enfouis et non exposés aux utilisateurs, peuvent être ré-interprétés, dans le cadre d'une approche orientée modèle (cf. section 1.4), comme des fragments d'un méta-modèle associé au domaine scientifique. Ces constatations nous amènent à qualifier ces logiciels de *systèmes de gestion de workflows intégratifs*:

Définition 2.3 *Un WfMS intégratif est un environnement logiciel d'analyse de données qui intègre un WfMS et capture une partie de la connaissance d'un domaine ou d'une discipline.*

Nous avons vu que la formalisation de la connaissance du domaine d'application offre des opportunités d'exploitation (cf. section 1.5) pour l'aide à la conception et l'adaptation des applications à différentes plates-formes d'exécution. Cela nous amène à considérer les WfMS que nous qualifions d'intégratifs comme étant la classe de WfMS la plus propice à l'élaboration d'une démarche efficace de conception de workflows.

Pour cela, mais aussi parce que ces environnements ont trouvé une large audience en bioinformatique, nous les considérons comme les plus prometteurs. Nous poursuivrons donc cette étude en nous consacrant uniquement aux WfMS intégratifs.

Catégorisation des fonctions techniques

Nous continuons l'étude des WfMS intégratifs en évaluant comment ceux-ci simplifient le processus de conception des workflows.

Nous savons que le déplacement de données, le parallélisme, l'ordonnancement des tâches et, d'une façon plus générale, le "plumbing" représente la part majoritaire du code produit par les concepteurs de chaînes de traitements [84]. La capture d'un workflow directement à partir d'un enchaînement de commandes se traduit par un graphe contenant une majorité d'acteurs aux fonctions techniques. Le concepteur se consacre donc essentiellement à l'intégration laborieuse du "plumbing". Dans le contexte des WfMS, la sur-représentation des acteurs techniques dans les workflows est un écueil récurrent et bien connu, parfois appelé problème des *shims* ou adaptateurs, particulièrement prévalent dans les outils d'orchestration de services [85, 86]. Les WfMS intégratifs sont des environnements qui prennent particulièrement bien en charge les fonctionnalités techniques des chaînes de traitements (cf. figure 2.2) de telle sorte que le concepteur puisse mieux se focaliser sur l'exploitation des acteurs d'analyse de données ou composants *métiers*.

	script	WfMS WS/WSDL	WfMS intégratifs
ordonnancement des tâches	concepteur	système	système
gestion des dépendances des tâches	concepteur	système	système
déplacements des données	concepteur	concepteur	système
normalisation des interfaces des commandes	concepteur	concepteur	système

TABLE 2.2: Comparaison de la prise en charge des fonctions techniques des chaînes de traitements suivant les types d'environnements

Nous mettons ici en évidence une caractéristique de cette classe d'environnements logiciels. Les WfMS intégratifs prennent en charge de nombreux aspects techniques de la conception comme:

— l'ordonnancement des tâches;

— la gestion des dépendances de tâches;

— le déplacement des données;

— la normalisation des interfaces des composants.

C'est la prise en charge par l'environnement logiciel d'aspects techniques de l'implémentation qui simplifie la conception. Ce principe, mis en œuvre dans les WfMS intégratifs doit pouvoir s'appliquer à d'autres aspects techniques de la conception des chaînes de traitements.

On peut penser, par exemple, à l'exploitation du parallélisme ou à l'adaptation au changement de plates-formes d'exécution.

2.2 L'usage des WfMS scientifiques dans les organisations

Nous décrivons dans cette partie le contexte de l'usage des WfMS en bioinformatique en nous basant à la fois sur la littérature et sur des observations directes du fonctionnement de plusieurs laboratoires et plates-formes de services. Nous nous demandons quels types de processus sont formalisables aujourd'hui dans les WfMS et qui sont les usagers de ces environnements.

2.2.1 Émergence et gradient de formalisation des chaînes de traitements scientifiques

Tout d'abord, nous nous posons la question de savoir si les WfMS sont aujourd'hui capables de capturer une part importante des processus d'analyse de données relatifs à un domaine ou une discipline. Pour cela, nous nous référons à la classification de Mentzas

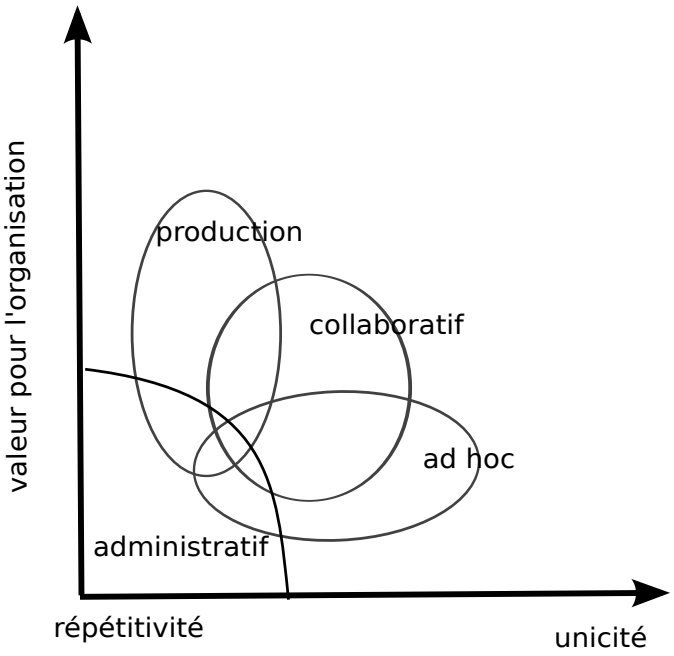


FIGURE 2.4: Classification des workflows en BPM d'après Mentzas et Halaris [87].

et Halaris [87] qui concerne le domaine du BPM. Elle s'applique aux workflows produits au sein d'une organisation quelconque. De par son caractère général, il est possible de s'en inspirer pour étudier les workflows scientifiques exploités par un laboratoire ou un institut. Les auteurs y distinguent des workflows de *production*, des workflows *administratifs*, *collaboratifs* ou *Ad hoc* (cf. figure 2.4).

Les workflows de *production* représentent les processus les plus stables et les plus structurés dans une organisation. Ils sont corrélés à une part importante de la valeur produite par l'organisation. On les oppose aux workflows *Ad hoc* qui représentent des

processus rares et/ou moins formalisés et dynamiques. Dans l'approche BPM, on considère que lorsque des processus ne sont pas du tout formalisés, l'organisation perd sa capacité à les rationaliser, les améliorer et les généraliser. Pour faire émerger ces processus, des outils et méthodes dédiés peuvent être mis en œuvre. Ainsi Aalts a récemment proposé une approche dénommée *Process Mining* permettant de faire émerger de nouveaux workflows a priori *Ad hoc* à partir de l'analyse des traces des activités comme les logs [88]. Il s'agit ici d'identifier des variants à partir d'un workflow stable et structuré.

En science, il existe une classe de processus qui partage certaines propriétés des workflows *Ad hoc* du BPM. En effet, la recherche scientifique est par nature exploratoire et les méthodes d'analyse de données évoluent constamment. Donc toute méthode innovante commence par être d'usage minoritaire, non formalisée et changeante au cours du temps. Certaines de ces méthodes vont disparaître et d'autres deviendront des méthodes de référence au sein d'une communauté disciplinaire. Ainsi en bioinformatique et notamment dans le domaine du séquençage haut-débit, les outils changent très régulièrement et les chaînes de traitements doivent suivre cette évolution.

Par ailleurs, l'aspect exploratoire de l'analyse de données nécessite une personnalisation fine des composants. Ces méthodes rares pourront ne pas figurer dans un WfMS et donc ne pas être accessibles à une audience élargie, d'autant plus que le coût d'intégration dans le WfMS est important (cf. figure 2.5). Par ailleurs, nous constatons que les implémentations exploitant du parallélisme à grain fin n'apparaissent généralement que dans un second temps et souvent uniquement pour les méthodes reconnues (de type *production*).

Ces observations nous amènent à formuler les hypothèses suivantes:

Le caractère dynamique, émergent, personnalisé et éphémère des applications scientifiques d'analyse de données doit être pris en compte dans les WfMS.

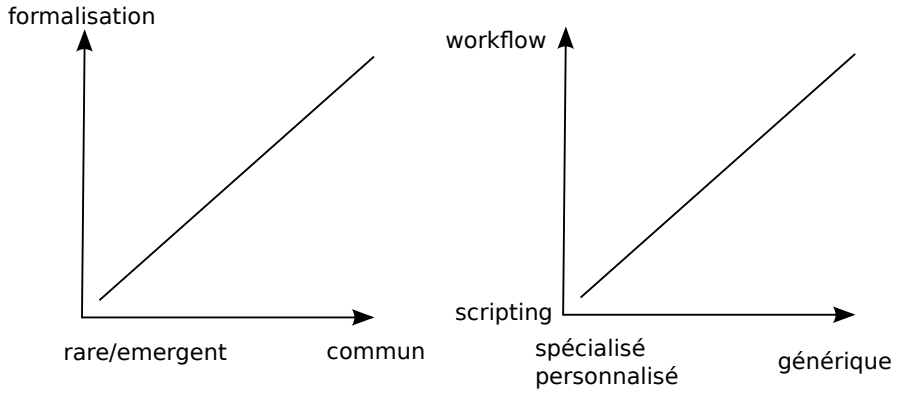


FIGURE 2.5: Classification intuitive des chaînes de traitements scientifiques suivant leur degré de formalisation ou modes d'implémentation.

Nous montrerons ultérieurement comment les WfMS répondent aujourd'hui à ces besoins.

2.2.2 Les intervenants des WfMS

Toujours afin de mieux cerner comment sont utilisés les WfMS, nous abordons maintenant la question de leur audience dans le contexte de la bioinformatique. Nous allons décrire les utilisateurs actuels de ces systèmes. Nous cherchons à établir si l'ensemble des scientifiques impliqués dans la production et l'usage des applications d'analyse de données exploitent ces environnements ou s'ils sont réservés à un groupe particulier d'usagers. Pour répondre à ces questionnements, nous nous basons sur la spécification de référence des architectures orientées service [89]qui propose des rôles génériques suivants:

— le *médiateur* facilite l'interaction et la connectivité dans l'offre et l'usage des services;

— le *consommateur* interagit avec un service pour satisfaire un besoin;

— le *fournisseur* offre un service.

Un participant peut jouer plusieurs rôles en fonction du contexte. Considérons ici plus précisément l'audience des scientifiques et développeurs travaillant en rapport avec les chaînes de traitements haut-débit en bioinformatique. En se basant sur l'observation des pratiques de terrain dans différents laboratoires et plates-formes de services consacrées à l'analyse de données (cf. tableau D.1 en annexe), on peut segmenter les utilisateurs en plusieurs groupes distincts par leurs pratiques. On peut ensuite définir différents cas d'utilisation (cf. figure 2.6) dans lesquels les rôles des intervenants peuvent être identifiés.

Les *médiateurs* sont les ingénieurs dédiés à l'infrastructure et à la gestion du WfMS. Ils assurent la maintenance de l'environnement.

Les *consommateurs de services* sont les biologistes et les bio-analystes. En l'absence de WfMS, ils participent aux spécifications lors d'une phase de collecte de besoins et réceptionnent les résultats. En utilisant des WfMS, leur rôle peut s'étendre à la composition de services. Ils perçoivent directement un bénéfice personnel à l'usage du WfMS. Celui-ci leur permet d'être plus autonomes et leur donne accès à des outils qu'ils ne peuvent manipuler sans intermédiaire.

Les *fournisseurs de services* sont les informaticiens, chercheurs et ingénieurs, qui sont responsables de l'implémentation des composants accessibles, par exemple, sous forme de ligne de commande. Le bénéfice direct résultant de l'intégration de leur composant peut être une meilleure visibilité de leur travail. Cependant, le WfMS s'intègre mal à leurs méthodes de développement. En effet, les WfMS basés sur des modèles DataFlow ne gèrent pas toujours les cas complexes nécessitant de l'expressivité (condition, boucles, fonctions...). Par ailleurs, l'intégration de nouveaux composants suit une procédure de déploiement faisant intervenir un *médiateur*, limitant l'usage pour le prototypage (cf. section 2.3.1). Même si le WfMS simplifie la soumission sur l'infrastructure d'exécution, la contrepartie est souvent une personnalisation limitée du contexte d'exécution et les

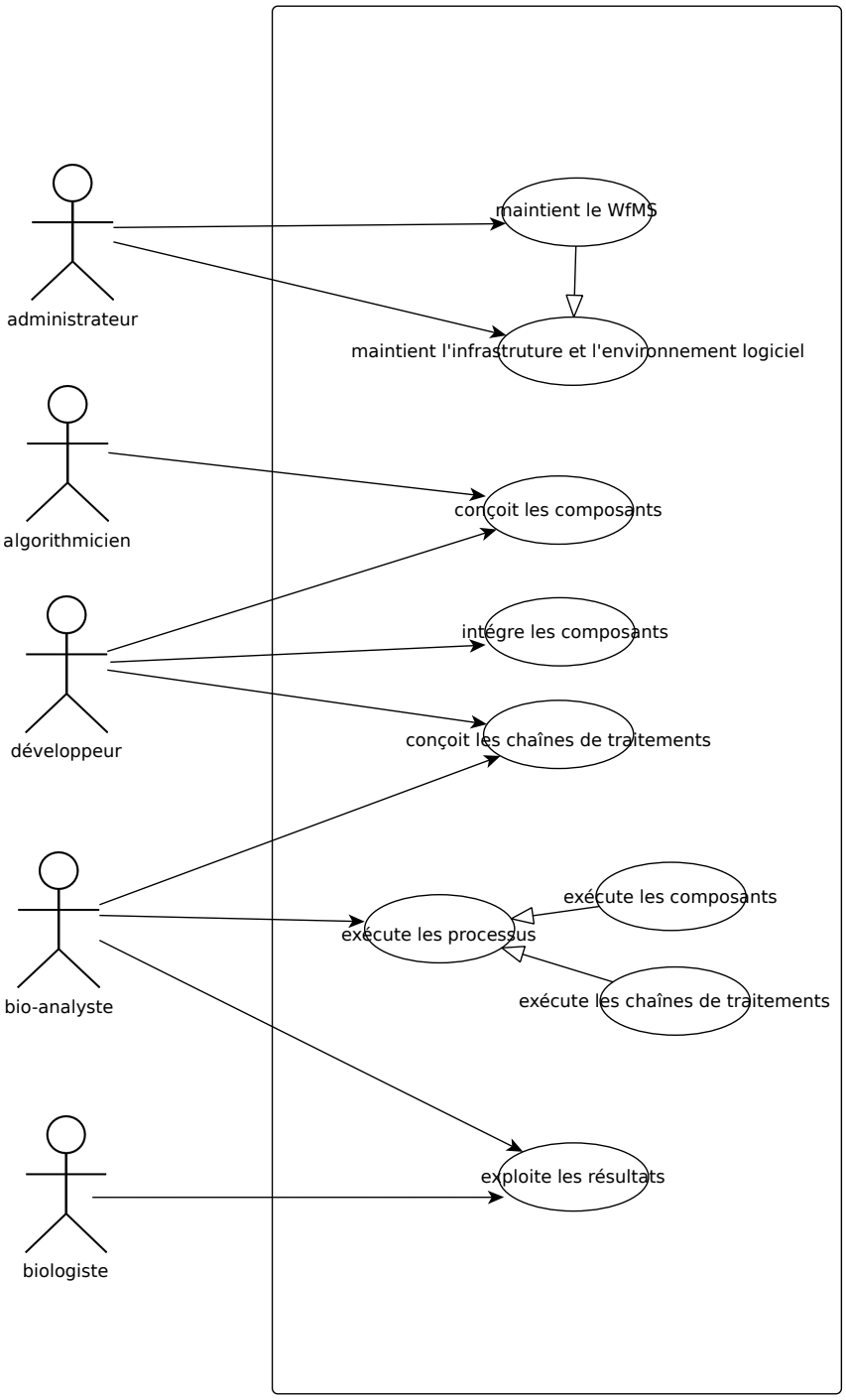


FIGURE 2.6: Cas d'utilisation des chaînes de traitements bioinformatiques (avec ou sans WfMS).

tâches laborieuses d'implémentation du parallélisme restent à la charge du fournisseur de service (cf. section 2.7.2). Nous remarquons donc que les scientifiques producteurs de composants, qui maîtrisent des langages de programmation n'utilisent que rarement les WfMS de leur propre chef, préférant des méthodes d'implémentation basées sur le codage de script et l'utilisation de lignes de commande. Le coût important d'intégration et le manque de personnalisation des composants ainsi que l'expressivité limitée dans l'orchestration reste un facteur limitant.

Nous pensons que l'adoption des WfMS par cette audience nécessite la prise en compte de caractéristiques spécifiques dans les modèles, notations et langages utilisés dans les WfMS.

Cette audience doit pouvoir retirer de l'intégration de leurs chaînes de traitements dans un WfMS un bénéfice directement perceptible. Celui-ci peut correspondre, par exemple, à la prise en charge par l'environnement de certains aspects techniques laborieux de la conception comme la parallélisation et la soumission des tâches, tout en gardant un degré de personnalisation suffisant.

Nous constatons que les fournisseurs de services sont très minoritairement des usagers des WfMS intégratifs. Ce groupe se compose des développeurs, algorithmiciens, et ingénieurs proposant de nouveaux composants et workflows Ad hoc.

Nous faisons l'hypothèse que des WfMS mieux conçus pour cette audience devraient tout à la fois diminuer le coût d'intégration des workflows, augmenter la couverture fonctionnelle des WfMS et favoriser la diffusion de l'innovation auprès des scientifiques consommateurs de services.

2.3 WfMS et prototypage

2.3.1 Prototypage de workflows et conception de composants dans les WfMS

Les WfMS intégratifs comme Galaxy et Mobyle rendent possible l'orchestration de composants. Néanmoins ils restent aujourd'hui principalement utilisés comme des répertoires d'outils intégrés sous forme de services où chaque tâche est manuellement exécutée. Leurs capacités d'orchestration restent secondaires même si elles séduisent les utilisateurs avancés. Concernant Galaxy, l'expressivité du langage DataFlow est, à l'heure actuelle, encore malheureusement réduite notamment par l'absence de gestion des conditions et d'acteurs composites. Cependant, malgré la faible expressivité de son moteur de workflows, Galaxy possède certaines caractéristiques intéressantes pour la conception et le prototypage de workflows. Un utilisateur peut par exemple convertir un historique de ses activités en un workflow linéaire éditable et re-exécutable. Il est libre de déterminer l'enchaînement des tâches et leurs relations. Néanmoins, il ne peut faire appel qu'à des composants préexistants dans l'environnement. Les WfMS intégratifs existants proposent des capacités de personnalisation des composants encore très limitées. Défi-

nir de nouveaux composants est laborieux et reste une tâche technique nécessitant des compétences particulières et relèvent d'un *médiateur*, comme un administrateur de la plate-forme logicielle. Lors du prototypage d'une chaîne de traitements, les suites d'essais, de tâtonnements ou de raffinements nécessaires à la mise au point de l'application se traduisent par des changements d'implémentation. Avec ce type d'environnement, il reste bien plus facile de s'affranchir du WfMS et de réaliser l'intégration a posteriori à partir d'une version stable de l'implémentation.

Dans les WfMS intégratifs, la création de composants n'est pas directement accessible au concepteur de workflows. Cela limite l'intérêt du WfMS pour le prototypage des chaînes de traitements.

Une des raisons de cette limitation est la complexité et l'absence de généricité des méthodes de déploiement des composants applicables aux plates-formes d'exécution. Un composant défini par l'utilisateur a potentiellement de nombreuses dépendances, parfois mal formalisées et incompatibles. Cette complexité est accrue lorsque le nombre d'utilisateurs et de composants est élevé et se pose alors des problèmes de respect de l'intégrité du système.

La possibilité de créer des composants et de pouvoir directement les exploiter est déterminante pour faciliter le prototypage. Par conséquent, il est important de disposer d'une méthodologie générique de déploiement des composants et de leurs dépendances au sein du WfMS.

2.3.2 Intégration de composants hétérogènes

Dans Les WfMS intégratifs, la conception de composants est une tâche technique qui nécessite la définition d'une couche d'intégration spécifique. Ainsi Galaxy et Mobyle utilisent des techniques d'intégration de composants adaptées aux applications hétérogènes préexistantes dites *legacy* [90] pour lesquelles il est difficile de modifier le code.

Chaque composant est décrit dans un descripteur de déploiement XML ou JSON qui spécifie comment invoquer l'outil, comment représenter ses paramètres et quelles sont les sorties attendues. Ce système permet la mise à disposition d'outils sous forme de service, accessible par une interface web ou via une API. La création d'un nouveau composant nécessite l'écriture du descripteur, voir éventuellement d'un wrapper, dans le cas d'un outil en ligne de commande. Dans cette approche d'intégration, le composant est initialement considéré a minima comme une boîte noire, qualifiée uniquement par ces entrées et sorties. Cependant, le descripteur peut préciser des propriétés du composant et des méta-données permettant la gestion du typage et la classification du composant, mais aussi la personnalisation du contexte d'exécution.

Finalement, la technicité nécessaire à l'intégration d'outils métiers nous montre que ces environnements ne peuvent pas être qualifiés de *systèmes de conception à pente douce* (cf. section 1.2). L'utilisateur est ici confronté à une augmentation brutale de la complexité pour avoir à disposition de nouveaux composants.

Néanmoins, on peut noter que de récentes initiatives de contributeurs tentent d'orienter Galaxy vers la conception et le prototypage en proposant des fonctionnalités d'aide à la création de composants via des interfaces web ou l'intégration d'outils interactifs [91] comme IPython [92].

2.4 Catégorisation des composants logiciels

Nous avons vu que la capacité d'un intervenant à formaliser une chaîne de traitements dans un WfMS peut être impactée par différents facteurs comme l'expressivité du langage utilisé et la possibilité de créer simplement de nouveaux composants. Un autre facteur important est le type de composants que sont capables de gérer ces environnements.

Il n'est pas exclu que l'adaptation des WfMS intégratifs à des spécifications largement admises comme les Web Services puisse faciliter l'ouverture de ces environnements et leur interopérabilité. Il existe de nombreux travaux sur le sujet.

Ainsi, par exemple, la spécification BIOXSD a été proposée pour définir des formats de données normalisés associés aux entrées et sorties des opérations des web-services [84] et faciliter l'interopérabilité et la composition de services. Cependant, nous avons vu que nos environnements cibles, les WfMS intégratifs, ne sont pas basés sur les Web Services, n'ayant pas la contrainte de la distribution des exécutions entre domaines. Cela nous amène à ne pas approfondir ici l'étude de l'orchestration de composants Web Services.

Nous considérerons directement les niveaux des composants hétérogènes sous-jacents. Nous nous concentrerons sur l'intégration et l'orchestration de composants préexistants ou en cours de création, comme des binaires ou des scripts, au niveau interne d'une organisation scientifique. Ces composants cibles incluent des outils critiques nécessitant un environnement d'exécution intensif personnalisé. Le dénominateur commun de ces composants hétérogènes est qu'ils sont invocables en ligne de commande.

Dans le contexte de l'analyse de données à haut-débit, une des fonctions primordiales des WfMS est de faciliter la capture des chaînes de traitements intensifs à base de composants instanciables par des lignes de commande. Ces composants sont de nature hétérogène, et leur implémentation est, soit préexistante, soit simultanée à la conception du workflow.

Les composants à états sont des composants dont les résultats d'exécution dépendent du contexte dans lequel ils sont invoqués. Considérons un répertoire de services basés sur des composants hétérogènes exécutables en ligne de commande. Lorsque le comportement de certains composants est déterminé par des données non explicitement décrites par la syntaxe de la commande, nous pouvons considérer ces composants comme des composants à états. Il peut s'agir par exemple de l'usage d'une source de données, fichiers ou bases de données, exploités en interne par le composant et pouvant évoluer sans que l'on puisse contrôler cette évolution au niveau du moteur de workflows.

Dans le cadre de plates-formes logicielles basées sur des architectures orientées service, ce type de services pose des problèmes important d'intégration. Ainsi, le SOA met en avant la nécessité de définir des services autonomes et sans état [93]. Cela se justifie par le fait que les possibilités de changement de comportement d'un service suivant ses états sont masquées et ne peuvent donc pas être gérées, limitant la réutilisation du service. Dans le cas de la composition de services, la complexité d'un workflow découle de la complexité des acteurs. Même si l'on pouvait déléguer la gestion des états des acteurs à la couche d'orchestration (moteur de workflows), il est difficile de prévoir le comportement d'un tel workflow ayant une forte complexité cyclomatique [1]. De façon générale, il est admis que, dans le cadre d'une architecture orientée service, une complexité cyclomatique importante réduit la maintenabilité et la testabilité du système [94].

Ainsi dans un WfMS, la capacité à réutiliser un workflow et les capacités de reprise sur erreur sont liées à l'absence d'état des acteurs. De ce fait, les MoC des WfMS scientifiques sont généralement conçus pour des acteurs sans état.

Nous constatons, d'une part, qu'il est possible de modifier certains composants à états pour en faire des composants sans état moyennant un effort d'implémentation. D'autre part, nous remarquons que les composants à états ne constituent pas une part importante des acteurs exploités dans les chaînes de traitements en bioinformatique.

Les WfMS basés sur des MoC DataFlow sont théoriquement limités à l'usage de composants sans état mais c'est le concepteur de composants qui doit prendre en compte cette contrainte.

Ces conclusions nous amènent à ne considérer dans la suite de cette étude que des composants sans état. Il s'agit d'un limitation intrinsèque à la plupart des WfMS scientifiques et en particulier à ceux qui sont basés sur un modèle DataFlow.

Nous abordons maintenant la personnalisation des composants dans les environnements de gestion de workflows. L'intégration de *fonctions définies par l'utilisateur* (UDF pour User Defined Function) est une fonctionnalité issue du domaine des bases de données relationnelles (DBMS pour Database Management System) et permet la personnalisation des environnements par l'utilisateur.

De façon plus générale, dans un contexte où les fonctions et opérateurs sont généralement fournies par le système, une fonction définie par l'utilisateur et intégrable à l'environnement est qualifiée d'UDF.

Dans les DBMS, la conception des UDF s'effectue au sein d'un cadre contraint imposant un langage prédéfini et des patrons de conceptions spécifiques. C'est grâce à ces contraintes que l'invocation de l'UDF pourra bénéficier d'optimisations (parallélisation, réorganisation des opérateurs).

1. La complexité cyclomatique ou complexité de McCabe est une métrique indépendante du langage qui permet de mesurer la complexité d'un composant d'un programme. Elle mesure le nombre de chemins indépendants dans le composant modélisé sous la forme d'un graphe.

Les environnements DISCS (DISCS pour Data-Intensive Scalable Computing Systems) comme MapReduce, quant à eux, rendent possible l'intégration d'UDF suivant des patrons de conception bien plus libre. Dans ce cas, en l'absence d'une description des propriétés des UDF, si ces fonctions sont de nature hétérogène, elles seront généralement vues comme des boîtes noires dans l'environnement sans possibilité d'optimisation interne à l'UDF. [95]. Dans un WfMS, un acteur d'un workflow peut encapsuler un service qui gère l'exécution d'une méthode d'analyse de données. Cette méthode etant de nature hétérogène et librement personnalisable, nous considérons que l'on peut également la qualifier d'UDF. Le tableau 2.3 récapitule les propriétés des UDF entre DBMS, DISCS et WfMS.

	DBMS	DISCS	WfMS
codage des UDF	controlé	libre	libre
propriétés des UDF	connues	inconnues	inconnues
couplage des UDF	fort	faible	faible

TABLE 2.3: Comparaison des propriétés des UDF entre DBMS, DISCS et WfMS

Nous considérons dans notre étude la définition suivante d'un acteur UDF:

Un acteur UDF est un acteur d'un workflow qui encapsule un service gérant l'exécution d'une méthode de nature hétérogène et librement personnalisable par le concepteur.

Aujourd'hui, la création et l'intégration d'acteurs d'UDF dans les WfMS restent complexe. Cela limite l'adaptation de ces environnements à la conception et au prototypage de workflows.

2.5 Les types de sources de données

Nous venons de décrire les types de composants exploitables dans les WfMS intégratifs. Ces composants d'analyse consomment et produisent des données. Nous nous penchons maintenant sur les caractéristiques de ces données que doivent manipuler efficacement les moteurs de workflows. Nous allons décrire succinctement les propriétés des données manipulées par les chaînes de traitements d'analyse intensive de données en bioinformatique. Il s'agit ici d'établir les contraintes que les données font peser sur les WfMS. Nous ne traiterons ici que des données stockées dans des fichiers et dont l'analyse pose des problèmes de passage à l'échelle. L'usage de données distribuées stockées dans des bases de données posent d'autres problèmes comme l'interopérabilité que nous n'aborderons pas ici.

Aujourd'hui, les entités biologiques et leurs relations sont décrites dans des ontologies, associées par exemple aux séquences [96] ou aux annotations de gènes [97]. Ces ontologies

sont regroupées dans OBO (Open Biological and Biomedical Ontologies) Foundry. Il existe un lien souvent strict entre types de données et formats de fichiers. La communauté bioinformatique ne s'est intéressée que très récemment à la description systématique et uniforme des formats de fichier. L'ontologie EDAM [98] référence aujourd'hui les formats majoritaires exploités en bioinformatique. On peut donc espérer disposer d'une liste de formats de fichiers relativement complète et dont la maintenance et l'évolution serait garanties par certains acteurs de la communauté disciplinaire. Du point de vue du modèle d'exécution, ces formats représentent les types structurés qui constituent les entrées et sorties des nombreux outils d'analyse de données du domaine. On peut citer des formats texte largement diffusés comme le format de séquence FASTA qui représente une collection de séquences d'ADN, d'ARN ou de protéines ou les formats matriciels d'annotation GFF et BED.

L'arrivée de données massives de séquençage a donné lieu à la création de formats de séquences de lectures (reads) comme FASTQ. Puis face à la taille des données, des formats binaires d'encapsulation ou d'archivage ont été mis au point, permettant de stocker les lectures ou les alignements (BAM, SRA, containeurs HDF5). Ils sont alors associés à des lignes de commande permettant l'extraction ou le requêtage. Un jeu de données de séquences brutes NGS au format FASTQ peut prendre plusieurs centaines de Gigas octets et un alignement BAM plusieurs Gigas octets, mais la taille est variable en fonction du type de plate-forme technologique de production. Chaque étape de traitement peut à son tour produire des quantités de données de cet ordre. Dans ces conditions, malgré les recherches actuelles sur la compression [99], le rapatriement de ces données et leur localisation pour l'exécution d'une chaîne de traitements constituent une tâche délicate. Ces phases peuvent représenter une part importante du code des chaînes de traitements, lorsqu'elles y sont intégrées. Elles nécessitent alors des contrôles d'intégrité, de la reprise sur erreur et l'usage de protocoles de transfert multiples.

Nous retiendrons donc que dans le cas des chaînes de traitements haut-débit en bioinformatique, les données sont généralement:

— de taille importantes;

— potentiellement distribuées;

— structurées ou partiellement structurées;

— dans des fichiers;

— dans un format décrit par une ontologie.

Un WfMS dédié au calcul intensif doit être capable de gérer de façon efficace des données ayant ces caractéristiques.

Nous constatons que les environnements actuels de conception de workflows gèrent les types de données de deux manières distinctes. Ils peuvent considérer les composants manipulant un type de données particulier (conversion, filtre validation, découpage, fusion...) comme des UDF quelconques. Les WfMS intégratifs peuvent également masquer ces composants à l'utilisateur et rendre leur usage implicite. Celui-ci n'a parfois qu'à

définir par annotation le type d'une donnée pour se voir proposer des services compatibles impliquant une conversion de format. Ainsi, Galaxy offre un mélange des deux approches. Cependant, seulement certains types de données sont gérés et les mécanismes actuels d'extension du modèle de données sont rudimentaires. Il est nécessaire de modifier Galaxy et de coder de nouvelles classes pour intégrer de nouveaux types de données.

Un autre point important est l'utilisation de références de données par le moteur de workflows. Ainsi, le choix fait dans le moteur initial de Taverna, FreeFluo, de manipuler directement les données qui sont alors copiées de façon multiples lors de l'orchestration le rend inadapté à la manipulation des masses de données. Les WfMS intégratifs, quant à eux, peuvent utiliser des mécanismes de référence, bien plus efficaces.

2.6 Exploitation des infrastructures d'exécution par les WfMS

Après avoir étudié en détail les architectures des WfMS et leurs usages, nous décrivons dans cette section comment les WfMS tirent aujourd'hui parti des différentes infrastructures d'exécution.

2.6.1 Les infrastructures de calcul et leurs modes de mise à disposition

Nous décrivons tout d'abord les différents types d'environnements d'exécution et modes de mise à disposition qui peuvent être exploités par le biais d'un WfMS. Il est possible d'installer un WfMS sur un simple ordinateur personnel suffisamment puissant ou une unique machine de calcul. Néanmoins, lorsque l'on manipule des données volumineuses ou que l'on doit exécuter des analyses nécessitant une grande puissance de calcul, on trouve dans les laboratoires et les plates-formes de service des infrastructures basées sur des architectures de calcul distribuées associées à des WfMS.

Les clusters de calcul sont les plus répandus. Un cluster est un groupe de machines homogènes co-localisées et interconnectées. Des outils logiciels permettent une gestion centralisée des machines ou nœuds du cluster afin d'assurer efficacement la répartition de la charge, l'attribution des ressources et de garantir la disponibilité.

La grille de calcul (grid computing) est, quant à elle, une infrastructure qui agrège des ordinateurs ou des clusters. Ces ressources peuvent être hétérogènes, distribuées, partagées et localisées sur des sites différents. Les recherches sur les grilles de calcul ont été menées pour régler des problèmes de passage à l'échelle d'applications scientifiques.

Par la suite, le concept de *cloud computing* a émergé. Il provient de l'association du modèle de la grille avec les technologies de virtualisation dans le contexte de l'internet haut-débit. La simplicité d'utilisation qui découle de l'usage des architectures orientées services constitue une des clefs du succès du *cloud computing*. En bioinformatique, son association aux WfMS, constitue une des solutions proposées pour remédier au problème

de la reproductibilité des analyses de données [100]. Le *cloud computing*, en tant que service, possède plusieurs modes de mise à disposition. Les plus connus sont l'IaaS, le PaaS et le SaaS mais il en existe bien d'autres. Nous décrivons dans les paragraphes suivants les différents modèles de *cloud computing* qui sont associés à l'exploitation de services d'analyse de données par le biais d'un WfMS.

Infrastructure as a Service

Le modèle *Infrastructure as a Service* (IaaS) offre un accès *à la demande* à des machines généralement virtuelles, associées à un hyperviseur. Il s'agit d'un outil de virtualisation qui permet à plusieurs systèmes d'exploitation de travailler sur une même machine physique. Un système de gestion intégrant plusieurs hyperviseurs permet de redimensionner l'infrastructure virtuelle en fonction de l'évolution des besoins. Les utilisateurs déploient alors sur ces machines des images de systèmes d'exploitation dans lesquelles sont installées leurs propres applications. L'IaaS offre par exemple la possibilité de déployer un cluster de machines virtuelles, d'y installer un ordonnanceur de tâches avec ou sans l'aide d'une application qui automatise ces tâches.

Ainsi des outils comme Starcluster [101] ou Cloudman [102] simplifient le déploiement d'un cluster virtuel sur l'infrastructure commerciale Amazon's Elastic Compute Cloud (EC2) [103].

D'autres outils comme OpenNebula [104] ou OpenStack [105] sont des solutions open-source qui offrent la possibilité à un laboratoire ou une plate-forme de service de gérer leur propre infrastructure virtuelle ou *cloud*, accessible en mode IaaS. Le scientifique peut alors y déployer des machines et éventuellement les regrouper en clusters virtuels, pour exploiter efficacement l'ensemble des cœurs de calcul disponibles.

Du point de vue des chercheurs en bioinformatique, l'IaaS est un modèle qui offre un intérêt immédiat [106]. Il est adapté au développement, au test et à des pratiques encore souvent majoritaires de conception des chaînes de traitements implémentées sous forme de scripts et de lignes de commande. L'autonomie du concepteur est dans ce cas très élevée. Dans le cas de l'IaaS, l'ensemble des dépendances des composants d'une analyse *in silico* devra être installé manuellement. Cela requiert donc un niveau important de technicité ou l'aide d'un médiateur.

Platform as a Service

Dans le modèle *Platform as a Service* (PaaS), c'est toute une plate-forme qui est mise à disposition, comprenant, système d'exploitation, langage de programmation et environnement d'exécution. Des bases de données et serveur web peuvent aussi y figurer. Dans le cas du PaaS, les utilisateurs ne fournissent que leurs propres applications qui seront hébergées sur la plate-forme. Du point de vue du concepteur de workflows ou d'outils d'analyse intégrables au WfMS en tant que service atomique, le WfMS est un élément d'une plate-forme d'exécution environné en mode PaaS. Dans ce modèle, le déploiement des dépendances des composants d'une chaîne de traitements seront mis en œuvre non pas par l'utilisateur mais par un médiateur, généralement un administrateur système. Des images pré-peuplées de collections d'outils métiers appelées *Virtual*

Appliances peuvent également être utilisées. Elles sont mises à disposition dans un répertoire web nommé *Virtual Appliance Marketplace*. Les utilisateurs ont alors à leur disposition des images de systèmes d'exploitation sur lesquelles est déjà déployés l'ensemble des logiciels en rapport avec une thématique donnée, ce qui limite la phase de personnalisation au minimum. Le modèle PaaS, couplé à une *Virtual Appliance Marketplace*, est aujourd'hui exploité pour la mise en œuvre d'environnements d'exécution pour des WfMS. Il est par exemple proposé par l'Institut Français de Bioinformatique (IFB's Bioinformatics Appliances) [107] dans le cadre de sa plate-forme de *cloud computing*.

Software as a Service

Dans le modèle Software as a Service (SaaS), le service fourni correspond à l'accès à une application, généralement mise à disposition par le web.

D'un point de vue de l'utilisateur qui exécute un workflow, les WfMS donnent accès à une application métier en mode SaaS. Pour cela, les WfMS disposent de fonctionnalités de génération d'interfaces web à partir des *descripteurs de déploiement*.

Containers as a Service

Une forme récente de *cloud computing*, dérivée du PaaS est particulièrement adaptée aux Architectures Orientées Services. Il s'agit du modèle émergeant du *Containers as a Service* ou CaaS [108]. Le CaaS exploite des environnements de gestion de containers Linux comme Docker [109]. La technologie des containers n'est pas une technologie de virtualisation, puisqu'il n'y a pas d'hyperviseur. Elle apporte une capacité de personnalisation et d'isolation "grain fin" au niveau des processus, en exploitant des technologies récentes incluses dans le noyau linux (Cgroups, Unionfs).

Une fois les applicatifs installés dans un container avec leurs dépendances, ils peuvent être déployés de façon extrêmement simplifiée sur toute infrastructure CaaS. Docker est une solution technique qui rend particulièrement facile l'usage de la puissante technologie de gestion des containers Linux. Docker permet un déploiement à la demande, adapté à un contexte où le développement et le mise en production sont integrés et réalisés par les mêmes intervenants. Cette approche est communement qualifiée de *Dev-Ops*.

L'usage du modèle CaaS est illustré par la plate-forme de *cloud computing* Google Container Engine [110]. Ce service commercial propose à l'utilisateur de déployer des *containers* Docker sur une infrastructure PaaS particulière qui facilite la gestion fine du cycle de vie de multiples occurrences de containers. Ici les containers encapsulent les applicatifs et l'ensemble de leurs dépendances logicielles et peuvent être directement définis par l'utilisateur. Dans la mesure ou les nœuds de calcul partagent des caractéristiques minimales (gestionnaire Docker et noyaux linux compatibles), le passage d'une infrastructure CaaS à une autre est transparent. Le container est *plate-forme agnostique*.

Le modèle CaaS est mis en avant pour ses capacités de performances, d'isolation et de dimensionnement dynamique [111], mais également parce qu'il facilite la réutilisation des composants dans le cadre du SOA [108]. Il donne la possibilité de personnaliser un environnement d'exécution à partir d'images préexistantes, provenant par exemple d'un

répertoire d'images de containers, ou d'exploiter des images créées par le concepteur, dans un contexte d'autonomie totale de celui-ci vis-à-vis des autres intervenants. Grâce au SOA et aux containers, il devient aisé d'intégrer des composants hétérogènes construits, par exemple, par des développeurs indépendants les uns des autres, dans des langages de leur choix.

Ce modèle de *cloud computing* émergeant est actuellement très populaire et commence à être exploité pour faciliter le déploiement des composants des WfMS [112] mais, pour l'instant, son usage n'est ni formalisé ni généralisé. Ces éléments nous montrent que les caractéristiques du CaaS en font un modèle de grand intérêt qui semble particulièrement adapté à la conception communautaire de répertoires d'applications. De ce fait, il nous semble essentiel d'étudier l'exploitation des architectures CaaS dans le cadre de la conception de workflows. L'intérêt du CaaS est d'autant plus fort lorsque les utilisateurs peuvent contribuer à l'édification d'un répertoire communautaire d'applications. En effet, chaque nouveau composant élaboré peut être facilement diffusé sous forme d'image de container qui constitue un équivalent d'un format générique de déploiement et d'échange de composants dans le monde Linux.

Le cloud hybride

Nous avons passé en revue le IaaS, le PaaS, le SaaS et le CaaS. Nous terminerons par la présentation du *cloud hybride* qui est un mode de mise à disposition mixant ressources internes et externes et qui peut s'adapter à l'ensemble des modèles précédents. Dans ce modèle, l'organisation fournit un environnement qui permet de gérer des ressources pouvant être internes ou bien externes. On peut ainsi élaborer une infrastructure centrée sur des ressources initialement internes (cluster hébergeant des machines virtuelles) mais qui pourra permettre la montée en charge par un déploiement dans un cloud externe par le mécanisme du *cloud bursting* [113]. Hormis la grille, le cloud hybride est le type d'infrastructure qui offre le plus de contraintes (multiples sites, distribution des processus et des données, nécessité du déploiement des composants et dépendances associées, authentification multiple) De ce fait, une application comme un WfMS conçu pour le cloud hybride, en relâchant certaines contraintes, peut convenir à des infrastructures plus simples comme un cluster de calcul unique, voire une seule machine multi-cœurs suffisamment puissante.

Il s'agit donc d'un modèle complexe à mettre en œuvre mais qui débouche sur une grande flexibilité d'usage. Il n'existe pas actuellement à notre connaissance de WfMS directement adapté à ce modèle.

Infrastructures cibles

Cette rapide typologie des usages du *cloud computing* pour l'analyse de données met en lumière deux contextes d'exécution convenant particulièrement aux WfMS. Ce sont les suivants:

— *Les clusters de calculs en mode PaaS.* Les clusters sont les infrastructures les plus classiques pour le traitement intensif des données bioinformatiques et leur exploitation en mode PaaS est commune. Lorsque l'on cherche à mettre à disposition des

composants orchestrables pré-établis via un WfMS, le choix d'une infrastructure PaaS pré-peuplée est une solution appropriée.

— *Le modèle CaaS.* Le modèle émergeant CaaS présente des caractéristiques qui le prédispose au prototypage de logiciels et à l'élaboration de répertoires communautaires d'applications hétérogènes. Nous avons vu que l'usage du CaaS n'est pas encore formalisé dans les WfMS. Nous avons donc l'opportunité de réfléchir à l'articulation entre les WfMS et ce nouveau modèle.

2.6.2 Adaptation des WfMS aux changements d'infrastructures de calcul

Après avoir décrit les infrastructures de calcul, nous allons maintenant étudier comment les WfMS actuels exploitent les infrastructures de calcul et notamment comment ils tirent profit des modèles PaaS et CaaS.

Si l'on considère le développement des architectures SOA et la diffusion des technologies du *cloud computing*, nous constatons qu'il est de plus en plus aisé et peu coûteux de modifier la localisation d'exécution des processus d'analyse de données. Cette décision peut être motivée par de nombreux paramètres comme le coût, la sécurité, la rapidité d'exécution ou la disponibilité. Cependant, des obstacles s'opposent encore à l'exploitation via les WfMS d'infrastructures d'exécution personnalisées, mises en place à la demande.

Dans le cadre de l'usage d'un cluster classique, les composants exploitables au sein des acteurs du workflow sont déployés manuellement ou via des outils d'automatisation d'installation, via un gestionnaire de packages et des outils de gestion de configuration de machines.

Dans le cas d'un cluster utilisé en mode PaaS, il est également possible de mettre à disposition des images virtuelles pré-peuplées de composants métiers, les *Virtual Appliances*, [3] qui pourront être directement sélectionnées par l'utilisateur du WfMS.

Ces deux méthodes sont disponibles si l'on veut utiliser les composants ou workflows du WfMS Galaxy mis à disposition dans un répertoire dédié, nommé Galaxy ToolShed [114]. Celui-ci contient notamment des descripteurs de déploiement associés à des procédures d'installation.

L'intérêt du PaaS est donc essentiellement sa capacité à rendre exploitable des composants pré-existants par le WfMS. Dès lors que l'on considère non plus la seule mise à disposition de composants pré-existants, mais également la conception d'acteurs, les tâches de déploiement deviennent laborieuses et le recours à un médiateur (administrateur système) est souvent nécessaire.

Concernant le modèle CaaS, Galaxy permet depuis 2014 l'intégration de contextes d'exécution à base de containers [112] mais l'écriture des descripteurs de composants reste une tâche technique. Cela a pour conséquence de limiter l'intérêt du WfMS pour le prototypage de workflows. Par ailleurs, le fait qu'il n'existe pas de répertoire de containers à destination de la communauté disciplinaire ni de protocole de création d'acteurs à base

de containers formalisé ne facilite pas la généralisation de cette nouvelle méthode de gestion des dépendances des composants.

Ainsi, même s'il existe des travaux précurseurs concernant la gestion des dépendances des workflows, et en premier lieu l'exploitation du modèle émergeant CaaS, on constate aujourd'hui le manque d'un formalisme, capable de *favoriser le nomadisme des chaînes de traitements* d'analyse de données en associant modèle de workflow générique et gestion des dépendances des composants.

La capacité limitée des WfMS à simplifier le déploiement des dépendances des composants sur une infrastructure de calcul cible affecte directement la capacité de réutilisation des workflows diffusés. Pour les WfMS intégratifs, c'est aujourd'hui un facteur limitant la diffusion de l'usage des workflows [115] et donc un obstacle à la reproductibilité et aux potentialités d'échanges mises en avant par l'approche workflow.

2.7 Performances des moteurs de workflows

Après avoir décrit les architectures de calcul et les modes de *cloud computing* qui présentent le plus d'intérêt pour l'exploitation d'un WfMS et dans quelle mesure ces environnements sont adaptés au besoin de changement d'infrastructures de calcul, nous allons étudier comment ces environnements sont capables de tirer profit efficacement de la puissance de calcul mise à leur disposition. Pour cela, nous abordons l'évaluation des performances de leurs moteurs de workflows. Celles-ci dépendent fortement de leurs capacités à exploiter le parallélisme disponible.

2.7.1 Les types de parallélismes accessibles dans les WfMS

Afin d'être capable de mieux évaluer les performances des moteurs de workflows, nous faisons ici un bref rappel concernant le parallélisme, dont il existe de multiples niveaux. La taxonomie de Flynn [116] décrit les modèles de parallélisme dans les architectures matérielles de bas niveau (cf. tableau E.1 en annexe). Plus récemment, Sima et al [117] proposent une autre classification qui est applicable au niveau algorithmique du calcul. Celle-ci est basée sur le mode d'obtention du parallélisme et met en évidence deux catégories majeures de parallélisme, le parallélisme de tâches et le parallélisme de données (cf. figure E.1 en annexe).

La performance de l'implémentation d'un workflow complexe composé de nombreuses tâches est directement liée à la capacité du moteur de workflows à extraire le parallélisme à partir de la spécification. Nous nous intéresserons donc au parallélisme exploitable au niveau applicatif. Nous allons donc détailler les différentes catégories de parallélisme exploitées dans les applications.

Tout d'abord, le parallélisme au niveau des instructions rend compte de l'exécution concurrente de plusieurs instructions d'un même flot. Celles-ci doivent être indépendantes de telle sorte que le résultat du calcul de chacune d'elles ne dépende pas de

l'exécution d'une autre. Le parallélisme d'instruction est produit et géré par le hardware ou le compilateur. Les dépendances de contrôles et de données restreignent son usage.

Ensuite, il existe le parallélisme de tâches ou de threads qui concerne plusieurs threads ou séquences d'instructions d'une même application qui peuvent être exécutées de façon concurrente. Il est généré par le compilateur ou l'utilisateur et géré par le compilateur ou le hardware. Le parallélisme de tâches permet de bénéficier des caractéristiques des processeurs multi-threads et multi-cœurs. Ce type de parallélisme est néanmoins limité par le coût de communication et de synchronisation et par les patrons de conception des algorithmes. L'usage de patrons de conception issus des langages fonctionnels facilite l'extraction de cette catégorie de parallélisme (cf. section 1.3.3.1).

Au niveau d'un processeur, le parallélisme de données est caractérisé par l'exécution d'un même jeu d'instructions sur plusieurs données de façon concurrente. Dans ce contexte, il est alors limité par la taille mémoire et par l'irrégularité des données traitées. Au niveau d'une infrastructure de calcul distribuée comme un cluster, le parallélisme de données dépendra de la bande passante du réseau et pourra nécessiter la persistance des paquets de données. Le contexte de l'intégration de composants *legacy* dans un WfMS, limite l'exploitation du parallélisme de flot d'instructions.

Le parallélisme de données et le parallélisme de tâches constituent les formes de parallélisme les plus accessibles par les WfMS.

2.7.2 Exploitation du parallélisme dans les WfMS actuels

Tous les WfMS étudiés ici autorisent la création de tâches atomiques qui peuvent elles-mêmes intégrer du parallélisme. Ainsi, quand il est disponible, le parallélisme de données gros grain repose sur l'implémentation de chaque composant et n'est pas accessible de façon générique au concepteur de workflows.

Ces WfMS n'offrent pas de mécanisme général d'extraction du parallélisme de données gros grain. Celui-ci nécessite le codage de composants dédiés. La conception de composants exploitant du parallèlisme est effectuée de façon totalement découplée des systèmes de gestion de workflows. On retrouve ici une caractéristique des moteurs de workflows orientés DataFlow des WfMS conçus pour l'intégration de composants préexistants et hétérogènes: l'implémentation du MoC DataFlow est dissociée de l'implémentation des acteurs. Cette contrainte du couplage faible limite l'accès aux niveaux les plus fins du parallélisme, mais il est établi conceptuellement que les modèles DataFlow permettent d'exprimer les niveaux de parallélisme à plus gros grains [40].

Concernant l'intégration du parallélisme, une des pistes étudiée aujourd'hui concerne l'exploitation au sein des WfMS de langage textuel dédié permettant d'exprimer le parallélisme dans les DataFlows. Ainsi des expérimentations d'intégration avec Galaxy du framework SWIFT [67] qui dispose d'un langage de script parallèle ont été menées

[118] mais cela ne concerne pas l'intégration des composants *legacy*, qui correspondent à une part importante des composants utilisés en bioinformatique. On peut également citer l'usage d'un framework similaire, PEGASUS [66], pour mapper des tâches sur les ressources d'une grille ou d'un cluster à partir d'un outil de composition graphique [70].

Un des WfMS les plus avancés concernant l'intégration de la parallélisation est Knime [68]. Knime est un client lourd qui permet de représenter graphiquement les traitements sous la forme de modèles DataFlows. Knime permet la soumission sur des clusters HPC et est capable d'exploiter les caractéristiques de ce type d'infrastructure ainsi que le parallélisme des processeurs multi-cœurs.

Knime propose un mécanisme d'extraction du parallélisme de données gros grain, accessible par la configuration d'un acteur [119]. Ce mécanisme est basé sur des collections

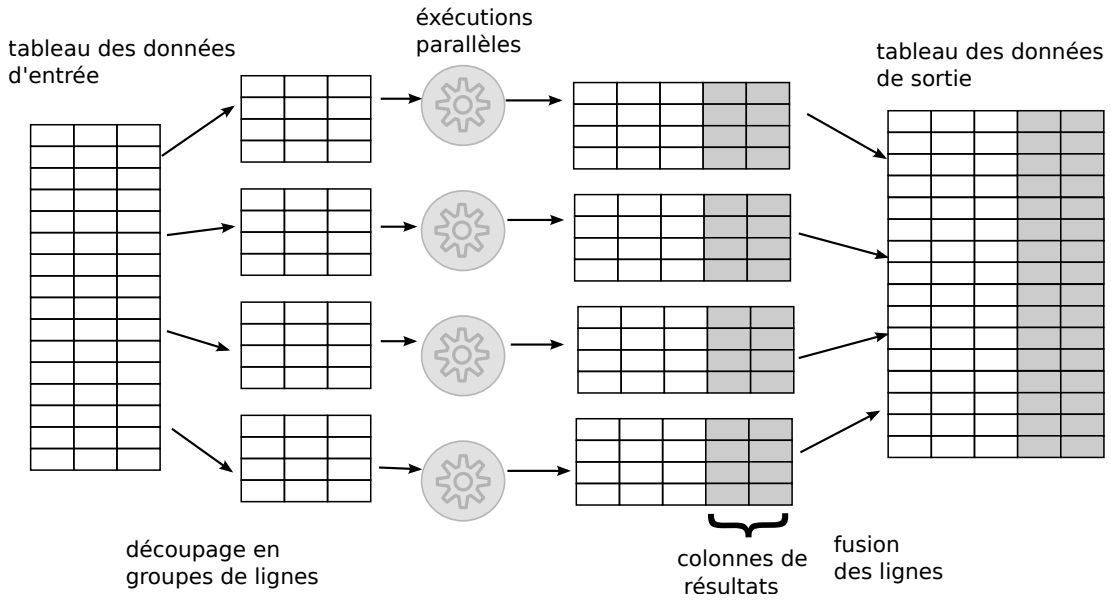


FIGURE 2.7: Parallélisation de données gros grain basée sur des tableaux. Le WfMS Knime intégre cette méthode qui permet de bénéficier du parallélisme des processeurs multi-cœurs et de nœuds des clusters HPC

de données qui doivent être explicitement découpées en sous-groupes pour permettre leur traitement en parallèle. Après configuration manuelle, Knime automatise la génération des exécutions parallèles issues d'un acteur ou d'un sous-workflow. Les résultats sont fusionnés automatiquement(cf. figure 2.7). Ce WfMS est conçu pour traiter des tableaux de données, stockés localement sur disque. Il ne dispose pas d'une fonctionnalité intégrée de référencement des données distribuées. On peut tenter de remédier à cela en introduisant l'usage de chemins de fichiers dans les tableaux de données mais pour être généralisée, cette approche nécessiterait probablement la modification coûteuse de tous les acteurs qui consomment et produisent ce type de données. D'autre part, pour gérer les grandes quantités de données produites par des exécutions distribuées, l'introduction

supplémentaire d'acteurs de déplacement de données reste nécessaire.

Les WfMS scientifiques étudiés, représentatifs des usages actuels en bioinformatique, même s'ils présentent des caractéristiques intéressantes de mise à disposition de chaînes de traitements, restent imparfaitement adaptés à la conception de workflows intensifs. Cette adaptation nécessite notamment une meilleure intégration de fonctionnalités dédiées comme l'extraction du parallélisme.

2.8 Résumé

Nous avons établi dans ce chapitre un état de l'art technique décrivant les WfMS et leurs contextes actuels d'utilisation. Nous avons mis en évidence un type d'environnements de gestion de workflows (WfMS) scientifique particulier basé sur des architectures orientées service que nous proposons de nommer *WfMS intégratifs*. Ils constituent une classe de WfMS émergentes, des plates-formes logicielles qui associent la gestion de workflows à un environnement d'analyse de données qui capture une partie de la connaissance d'un domaine disciplinaire. Nous considérons que nos travaux doivent s'appliquer à ce type de WfMS, particulièrement prometteur. Nous avons évalué les caractéristiques de ces environnements selon leurs capacités à (i) intégrer un langage graphique de composition ayant un niveau d'expressivité suffisant, (ii) simplifier la création des composants et de leurs relations, (iii) prendre en compte le caractère dynamique de la conception, (iv) s'adapter à de multiples environnements d'exécution, (v) faciliter la production d'implémentations performantes. Nous avons ainsi pu mettre en lumière certains manques ou défauts de formalisme sur lesquels nous comptons focaliser nos travaux.

— En premier lieu, les WfMS intégratifs présentent aujourd'hui des limitations pour l'adaptation au protypage des chaînes de traitements d'analyse intensive de données.

  D'une part, en bioinformatique, l'usage des WfMS intégratifs se développe aujourd'hui mais leur audience est principalement constituée des utilisateurs finaux (biologistes, bio-analystes) et non des concepteurs des composants et des workflows, c'est-à-dire des scientifiques fournisseurs de services. En effet, ces environnements ne prennent pas suffisamment en compte les phases de prototypage des composants et des chaînes de traitements pour intéresser cette audience. Les composants sont considérés comme déjà disponibles dans le système et la conception de nouveaux acteurs nécessite l'intervention d'un médiateur.

  D'autre part, les WfMS actuels permettent une simplication de la conception de workflows en gérant une partie de leurs aspects techniques comme le déplacement et le routage des données et la soummision des jobs. Lorsque les workflows sont basés sur des modèles DataFlow, certains niveaux de parallélisme peuvent être exploités, par exemple, à partir de l'analyse des dépendances de données. Cependant, aucun mécanisme simple permettant la mise en œuvre rapide du parallélisme de données gros grain n'est disponible.

— En deuxième lieu, la conception actuelle des WfMS intégratifs limite la diffusion

des chaînes de traitements.

La capacité de déploiement simplifié sur une infrastructure de calcul cible constitue un des éléments essentiels permettant l'indépendance des spécifications de workflows vis-à-vis des infrastructures d'exécution (cf. section 2.6.2). Elle faciliterait l'édification de répertoires de workflows communautaires en favorisant le nomadisme des chaînes de traitements. Les mécanismes de déploiement existant font appel à des médiateurs (administrateurs systèmes) ou dans le cas du Pass à des *Virtual Appliances.* Ces mécanismes doivent être mieux formalisés pour permettre le déploiement rapide sur des infrastructures de type cluster de calcul et plus largement du PaaS et du CaaS. Le modèle CaaS émergent est celui qui semble le plus adapté à la mise en œuvre d'un couplage entre spécifications de workflows et dépendances des composants. Aujourd'hui, son usage commence à être envisagé dans les WfMS. Il nous semble qu'il pourrait faciliter une large diffusion de l'usage des WfMS et favoriser l'élaboration de répertoires communautaires de workflows.

Nous décrirons dans le chapitre suivant nos propositions méthodologiques visant à adapter les wfms intégratifs à la conception, à l'exécution et au déploiement des chaînes de traitements d'analyse intensive de données.

Chapitre 3

Méthodologies

3.1 Synthèse des objectifs

Nous avons mis en évidence différents types de WfMS scientifiques. Les environnements que nous avons nommés *WfMS intégratifs* constituent la cible de nos travaux. Ils sont dédiés à l'analyse de données et associent la gestion de workflows avec une partie de la connaissance du domaine.

Partant du constat que les moteurs DataFlow de ces WfMS n'exploitent pas encore efficacement les infrastructures de calcul, nous jugeons important de proposer un modèle de calcul formalisé et performant, adapté à leurs architectures orientées service.

D'autre part, en analysant les usages actuels des WfMS en bioinformatique, nous avons constaté que les scientifiques, développeurs et concepteurs de méthodes d'analyse qui représentent les producteurs de services, utilisent peu ces environnements. Ceux-ci s'adressent plutôt aux *utilisateurs finaux* consommateurs de services, les biologistes ou les bio-analystes. La prise en charge par l'environnement logiciel d'aspects techniques supplémentaires comme la génération du parallélisme, mais également l'adaptation à différentes infrastructures de calcul seraient des fonctionnalités d'intérêt pour atteindre cette audience tout comme l'intégration dynamique de composants. En effet, la création d'acteurs reste plus laborieuse dans un WfMS que dans un environnement classique. Ainsi, des efforts de simplification de la capture des spécifications doivent être accomplis et passer de la mise à disposition à la conception de services.

Par ailleurs, aujourd'hui, la part de la connaissance du domaine disciplinaire intégrée dans ces WfMS n'est pas explicitée formellement et cela limite son usage par l'environnement. Néanmoins, elle peut être mise en lumière sous forme d'un méta-modèle. En effet, nous pensons qu'une meilleure exploitation de cette connaissance est possible grâce à une approche orientée modèle et peut déboucher sur une méthode de conception plus efficace des workflows guidée par l'utilisateur.

Nous décrivons dans les paragraphes suivants des propositions méthodologiques afin de pallier les limitations des WfMS précédemment identifiées concernant (i) l'efficacité des moteurs de workflows, (ii) le prototypage et la conception de chaînes de traitements intensives, (iii) le déploiement et la diffusion des chaînes de traitements.

Nous décrivons tout d'abord un nouveau modèle de calcul, adapté au calcul intensif dans le contexte des moteurs de workflows des WfMS intégratifs, appelé UDFAS (URI based DataFlow for Asynchronous Services). Ce modèle de calcul permet l'exploitation de multiples niveaux de parallélisme présents dans un DataFlow. Il peut interpréter un langage DataFlow minimal et expressif, capable de représenter les analyses de données à haut-débit.

Dans un deuxième temps, nous présentons une méthodologie de conception de chaînes de traitements, basée sur une approche orientée modèle, qui rend possible le prototypage des workflows et leur transformation semi-automatique en implémentation adaptée au calcul intensif.

Nous abordons ensuite la diffusion des chaînes de traitements et proposons l'intégration au WfMS d'une représentation des dépendances des composants et des plates-formes nommée *Components and Platforms Meta-Model* (CPMM). Elle rend possible la simplification de la configuration ou du déploiement des workflows sur les infrastructures de calcul cibles (cluster, PaaS et CaaS) et favorise l'indépendance des spécifications de workflows vis-à-vis des environnements d'exécution. Nous mettons finalement en avant en quoi certains formalismes que nous avons produits sont particulièrement adaptés à la mise en œuvre de répertoires de workflows communautaires.

3.2 Proposition d'un nouveau modèle de calcul

3.2.1 Spécification du modèle de calcul UDFAS

Nous allons décrire dans cette partie la spécification d'un nouveau modèle de calcul, nommé *URI* [1] *based DataFlow for Asynchronous Services* ou UDFAS, conçu pour l'analyse de données à haut-débit. A la différence des moteurs de workflows actuels des WfMS intégratifs, UDFAS est capable de tirer parti à la fois des parallélismes de tâches et de données pour accélérer l'exécution d'une chaîne de traitements.

Un modèle DataFlow

L'implémentation d'un moteur de workflows est déterminée par la représentation des processus et de leurs interactions, définis dans le modèle de calcul (MoC) sous-jacent. Nous faisons le choix de la conception d'un moteur de workflows basé sur un MoC DataFlow, famille de modèles de calcul que nous avons présentée. Le modèle DataFlow est communément employé dans les WfMS scientifiques et doit permettre l'élaboration d'un langage graphique intuitif, à la fois adapté au calcul intensif et suffisamment expressif pour implémenter la plupart des chaînes de traitements d'analyse de données.

Les langages DataFlow, conçus à l'origine pour le hardware, permettaient de produire massivement du parallélisme. Nous voulons transposer cette aptitude dans le contexte du SOA et l'étendre à l'ensemble des niveaux de parallélisme accessibles dans une spécification de workflows.

1. Une URI ou Uniform Resource Identifier est une chaîne de caractères courte identifiant une ressource sur un réseau. Une URL est une URI.

Nous nous limiterons à l'utilisation exclusive d'acteurs sans état (cf. section 6.1.1). Ainsi, la spécification de notre langage pourra intégrer des caractéristiques inspirées du cadre formel des langages fonctionnels et tirer parti de certains de ses patrons de conception.

Une chaîne de traitements peut être représentée sous la forme d'un DataFlow, un graphe orienté acyclique dont les nœuds représentent les acteurs réalisant les traitements sur les données et dont les arêtes correspondent à des dépendances de données.

Dans le modèle DataFlow, les jetons de données sont produits et consommés sur les arêtes. Les arêtes sont associées à des canaux de communication permettant le transfert des jetons de données de sortie, produit par un acteur, vers l'entrée d'un acteur successeur, consommant ces jetons.

Dans un acteur, chaque arête entrante est associée à un port d'entrée et chaque arête sortante à un port de sortie. Ces ports sont associés à des noms uniques. Ainsi l'expression $Actor['get\_sequence'].port['fasta'] -> Actor['blast'].port['sequence']$
désigne la dépendance de données entre le port de sortie $fasta$ de l'acteur $get\_sequence$ et le port d'entrée *sequence* de l'acteur *blast*.

Nous considérons un modèle de calcul DataFlow capable d'interpréter une spécification de modèle de workflow semi-concret que nous qualifierons de Semi-Concrete DataFlow Model (SCDFM).

Définition 3.1 *Un workflow représenté sous forme de Semi-Concrete DataFlow Model (SCDFM) est une représentation DataFlow qui présente l'information nécessaire à l'exécution du workflow, en incluant l'information permettant de générer les exécutions des tâches dépendantes de ses acteurs mais en omettant les informations spécifiques à la plate-forme d'exécution.*

Les " informations spécifiques" sont des éléments de configuration particuliers, qui peuvent correspondre à des localisations de librairies ou de données et sont souvent représentables par des variables d'environnement. Ainsi, par exemple, l'expression de la dépendance de données
$Actor['get\_sequence'].port['fasta'] -> Actor['blast'].port['sequence']$
associée au patron *blastall* -i $*sequence* -d *protdb* -p *blastp* -o *out.txt* de l'acteur *blast* est une information suffisante pour produire une exécution de ligne de commande fonctionnelle lors de l'exécution du DataFlow, moyennant configuration de l'environnement. Elle constitue les éléments types d'un modèle SCDFM. La configuration de la variable PATH permettant au système de retrouver le binaire *blastall* est exclue du SCDFM puisqu'elle est spécifique de la plate-forme d'exécution.

Dans le cadre de l'architecture dirigée par les modèles (MDA), le SCDFM peut être considéré comme une forme de PIM (Platform Independent Model), modèle indépendant des technologies liées au déploiement.

Un MoC inspiré du SDF

L'exécution d'un acteur est déterminée par la disponibilité des données et nous définissons la règle de déclenchement particulière suivante: un acteur peut déclencher l'exécution de la méthode métier qu'il encapsule dans la mesure où un jeton est disponible sur

l'ensemble de ses ports d'entrée. La méthode métier est alors exécutable avec chacun de ses paramètres instancié avec un jeton d'entrée. Notre modèle possède un ordonnanceur global qui analyse alternativement les acteurs. Celui-ci détermine si les conditions sont remplies pour que les acteurs deviennent éligibles à l'exécution. Cependant l'ordre d'exécution au sein des acteurs éligibles est délégué à la plate-forme d'exécution. Il est donc non prévisible au niveau du MoC. L'intérêt de cette délégation d'une partie de l'ordonnancement des tâches à la plate-forme d'exécution est de permettre d'utiliser notre MoC comme la couche supérieure d'un ordonnanceur hiérarchique gérant l'éligibilité à l'exécution suivant l'analyse des dépendances de données, tout en restant compatible avec les couches d'ordonnancement déjà en place sur les diverses plates-formes d'exécution. Ces outils externes sont dédiés à la gestion de l'ordonnancement des tâches atomiques du workflow et prennent en compte la gestion d'un environnement potentiellement partagé entre utilisateurs.

Pour chaque exécution de la méthode associée à un acteur, nous fixons qu'un seul jeton de données est produit. Cette règle permet d'exploiter une caractéristique du modèle Synchronous Data Flow (SDF) [30] dans lequel les nombres de jetons de données consommés et produits sont connus lors de l'exécution.

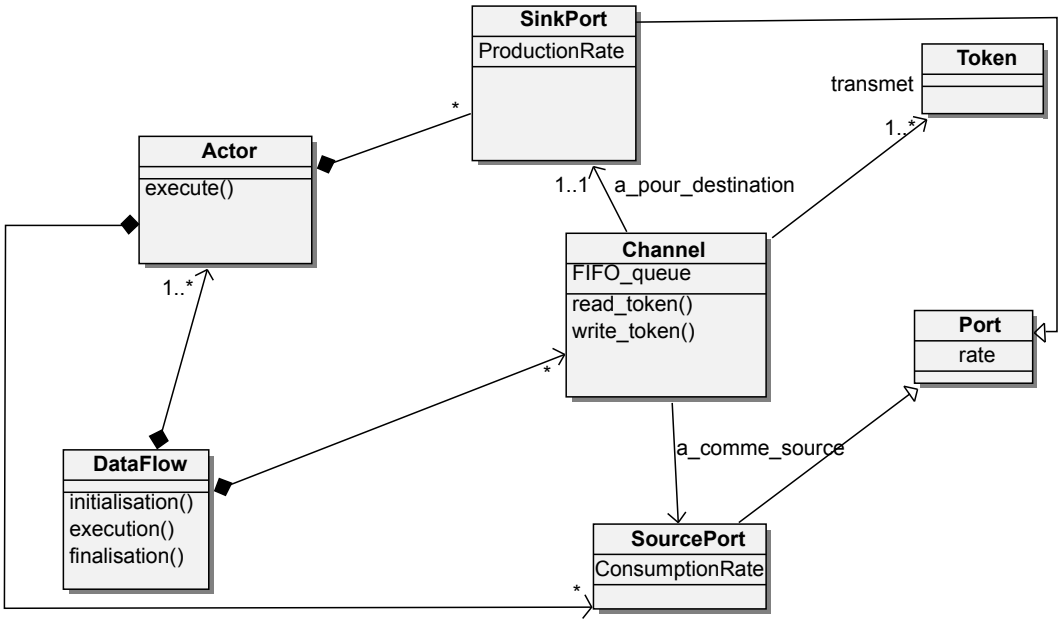


FIGURE 3.1: Modèle de calcul Synchronous Data Flow (SDF): diagramme de classe (UML). Le taux de consommation et de production des jetons est prévisible. Les canaux associés aux arêtes peuvent être implémentés suivant une queue FIFO (first-in, first-out)

Dans le SDF (cf. figure 3.1), la règle du taux constant d'émission et de consommation de jetons permet de rendre possible le *pré-calcul statique de la séquence de déclenchement des acteurs.*

Cependant, nous ne désirons pas comme dans un modèle SDF simple que l'ordre d'exécution des méthodes encapsulées dans les acteurs soit déterminé de façon statique puisque nous désirons: (i) déléguer à la plate-forme d'exécution sous-jacente une partie de l'ordonnancement, (ii) lancer les traitements associés à l'instance d'un acteur dès que ces paramètres sont disponibles afin d'optimiser le temps d'exécution global du workflow.

Nous exploitons néanmoins la capacité de *pré-calcul statique de la séquence de déclenchement des acteurs* permise par notre *règle du jeton unique de sortie.*

En effet, avant orchestration, nous pré-définissons statiquement toutes les exécutions des acteurs sans qu'aucune valeur ne soit attribuée à leurs paramètres. Ainsi chaque instance d'acteur est initialisée et chacun de ces paramètres est déclaré et indexé dans les tableaux des canaux des arêtes entrantes.

Par ailleurs, nous calculons à cette étape le *graphe des dépendances d'exécution* des instances des acteurs du DataFlow. C'est-à-dire que l'on détermine par l'analyse des dépendances de données l'ensemble des exécutions permettant de rendre éligible à l'exécution chaque instance d'acteur. Néanmoins, l'ordonnancement n'est pas statique. Comme nous allons le voir, c'est un ordonnancement dynamique qui exploite le *graphe des dépendances d'exécution.*

Flot de données et flot de contrôle

Nous n'exploitons pas de jeton de contrôle comme dans le Boolean DataFlow (BDF) [32]. Le principe de deux classes de jetons, l'une dédiée aux données et l'autre au contrôle nous semble intéressant pour la phase d'exécution du MoC mais reste complexe à maîtriser par des utilisateurs non experts dans le contexte de la conception graphique rapide de workflows. Au contraire, nous préférons l'utilisation d'une seule classe de jetons. Il s'agit de jetons associés à des jeux de données nommés DSURI (DataSet Uniform Resource Identifiers).

Comme nous le verrons ultérieurement, ces jetons ont plusieurs fonctions. Ils référencent les données et rendent également possible la réalisation d'itérations et de conditions. Le flot de contrôle est donc intégré au flux de données grâce à l'expressivité permise par ces jetons.

La gestion des valeurs NULL est un mécanisme général du modèle UDFAS exploité dans le contrôle du flot d'exécution. En premier lieu, la valeur NULL est associée à un jeton, par exemple, dans le cas où une exécution ne produit pas de sortie sur un port donné. Ce patron de conception correspond en programmation fonctionnelle à la monade *maybe* notée

$$data\ Maybe\ a\ =\ Process\ a \mid Null$$

Lors du traitement, si une exception est générée, le reste du traitement n'est pas réalisé et son résultat est la valeur Null. Si le traitement réussit sans exception, le résultat sera $Process\ x$ pour une entrée x.

Ainsi, lorsque l'un des ports d'entrée d'un acteur reçoit une valeur NULL, l'acteur n'est plus éligible et sa méthode de soumission n'est donc pas déclenchée. Un jeton NULL est produit puis assigné aux ports de sortie (cf. tableau 3.1). La valeur NULL

jeton d'entrée A	jeton d'entrée B	jeton de sortie C
NULL	NULL	NULL
NULL	string/URI B	NULL
string/URI A	string/URI B	string/URI C

TABLE 3.1: Propagation des valeurs NULL dans un acteur comportant deux ports d'entrée A et B et un port de sortie C

correspond donc à un jeu NULL ou à un jeu de données vide. Si une ligne de commande déclenchée par un acteur ne produit pas de fichier de sortie, le jeton DSURI produit correspondra bien à un jeu de données mais celui-ci sera vide. Dans ce cas l'acteur suivant qui reçoit ce jeton n'exécutera pas sa méthode de soumission mais transmettra directement un jeton NULL, propageant ainsi la valeur NULL dans le DataFlow. Chaque valeur NULL arrivant sur un port d'entrée d'un acteur rend donc inéligible une série d'instances d'acteurs définies à partir du *graphe des dépendances d'exécution.*

En deuxième lieu, la monade *maybe* permet la création de structures de contrôle de conditionnement $if\ then\ else$: pour un acteur ayant deux ports de sorties $s1$ et $s2$, la méthode métier peut être implémentée de telle sorte qu'un jeton Null soit produit et transmis au port $s1$ lorsque la condition c est vraie et sur le port $s2$ si elle est fausse. Ainsi, par propagation des valeurs Null, le sous graphe DataFlow dépendant de $s1$ sera exécuté si c est vraie. Dans le cas contraire, c'est le sous-graphe DataFlow dépendant de $s2$ qui sera exécuté. La monade *maybe* nous permet donc d'implémenter un mécanisme de conditionnement $if\ then\ else$ dans un modèle DataFlow sans altérer son orientation fonctionnelle. Ainsi, par l'intégration de ce patron de conception, nous étendons l'expressivité du MoC tout en maintenant son aptitude au parallélisme.

La figure 3.2 montre un diagramme qui illustre le mécanisme d'exécution conditionnelle. Si l'acteur $n1$ produit en sortie un jeu de données ($id = 456$) qui contient un pointeur vers un fichier $A.fastq$, l'acteur $n2$ qui reçoit un jeton contenant une DSURI référençant $A.fastq$ ($id = 456\&filter = A.fastq$) est exécuté. Par contre l'acteur $n3$ qui reçoit un jeton contenant une URI référençant $B.fastq(id = 456\&filter = B.fastq)$ ne sera pas exécuté mais propagera la valeur NULL. L'attribut $filtre$ disponible dans la spécification des DSURI permet une mise en place simple de ce routage qui correspond au if des langages impératifs.

Une classe unique de jetons abstraits

L'émission d'un nombre prédéterminé de jetons de données, ici fixé à un, nécessite l'introduction de la notion de jeton de données particulier. Le type de jeton employé correspond à une classe que nous nommons *classe de jetons abstraits*, pour signifier qu'elle n'est par directement interprétable en données concrètes. Cette unique classe de jetons associe plusieurs notions. Tout d'abord, elle intègre un mécanisme de référence pointant vers un jeu de données structuré en une collection regroupant différentes données atomiques, suivant le patron de la *monade collection*. Ensuite, cette référence peut être

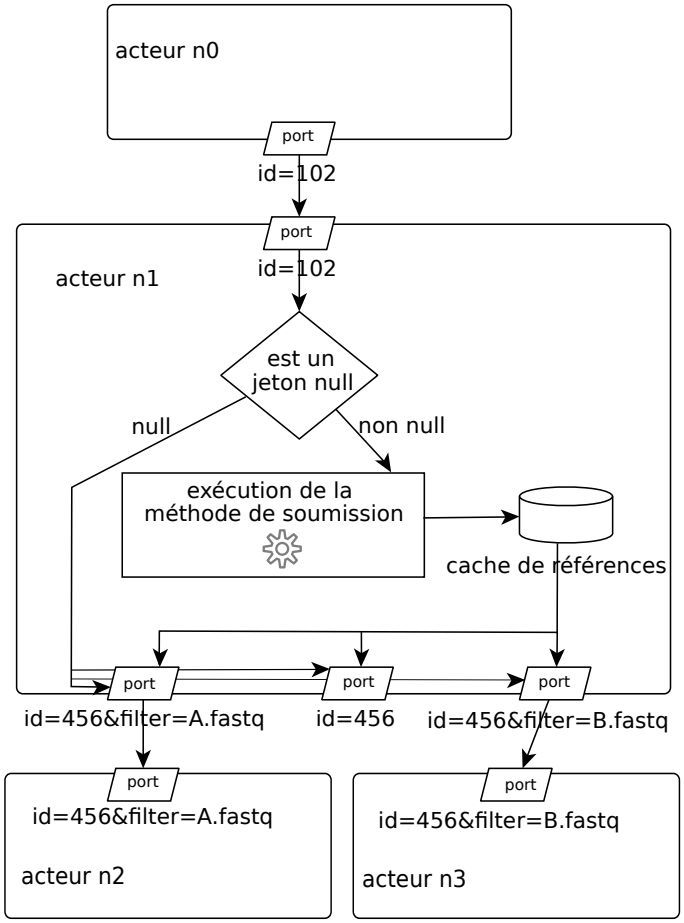


FIGURE 3.2: Mécanisme d'exécution conditionnelle géré par propagation de valeurs NULL.

complétée par une ou des fonctions de filtre qui, à la manière des *listes en compréhension* des langages fonctionnels, définissent une nouvelle liste en filtrant les éléments de la collection.

Les jetons abstraits que le MoC manipule correspondent de fait à des conteneurs encapsulant des références vers les données et à des opérations associées. Dans le contexte d'un langage fonctionnel (cf. section 1.3.3.1), il s'agit d'un patron de conception de type *monade*. Celui-ci permet ici de garantir les propriétés suivantes de la couche d'orchestration du modèle de calcul:

— une syntaxe à la fois *concise* et *expressive*;

— un *taux d'émission de jetons prévisible* tout en maintenant un *ordonnancement dynamique*;

— l'*immutabilité des jetons*.

Cette propriété d'immutabilité des jetons est particulièrement importante pour l'adaptation du MoC au calcul intensif. En effet, en programmation parallèle on peut faire appel à de nombreux threads qui manipulent simultanément les mêmes jetons. Du point de vue de l'implémentation, nous évitons ainsi la lourdeur de la synchronisation lors de l'orchestration.

Dans le cadre du paradigme fonctionnel, les acteurs du modèle UDFAS peuvent être considérés comme des *fonctions d'ordre supérieur* qui prennent comme paramètres des fonctions conteneurs de type *monade* (cf. section 1.3.3.1), représentées ici par les DSURI. Celles-ci sont assimilables à des conteneurs encapsulant des fonctions opérant sur des données. La sélection de sous-ensembles de données mais aussi le mode d'itération et le conditionnement peuvent être gérés par cette formalisation.

Des jetons référençant des jeux de données

La classe unique de jetons abstraits (DSURI) que le MoC UDFAS manipule exploite les patrons de conception fonctionnels de la *monade collection* et de *la liste de compréhension* (cf. section 1.3.3.1). Chaque jeton référence une collection de données et peut être représenté de façon compacte suivant une syntaxe d'URI (Uniform Resource Identifier)(cf. figure 3.3). Chaque fonction *process*() associée à un acteur ne sera déclenchée que pour les références de données *uri* du jeton *dsuri* telle que:

$$[process(uri)\ \ for\ \ uri\ \ in\ \ dsuri\ \ if\ \ filter(uri)]$$

Le mécanisme de référence est adapté à la manipulation à la fois de données distribuées et de grande taille. Il exploite une cache de références [1].

Une URI de jeu de données est un conteneur qui peut être représenté par un identifiant et abstrait une donnée comme un fichier ou un groupe de données associées. Il s'agit, par exemple, de l'ensemble des fichiers générés par une unique exécution ou d'un ensemble des données d'entrée qui doivent être indépendamment traitées. Le MoC UDFAS exploite ce type de jeton de façon exclusive (cf. figure 3.4). Lors de l'orchestration,

1. La spécification complète DSURI est décrite dans le chapitre 4

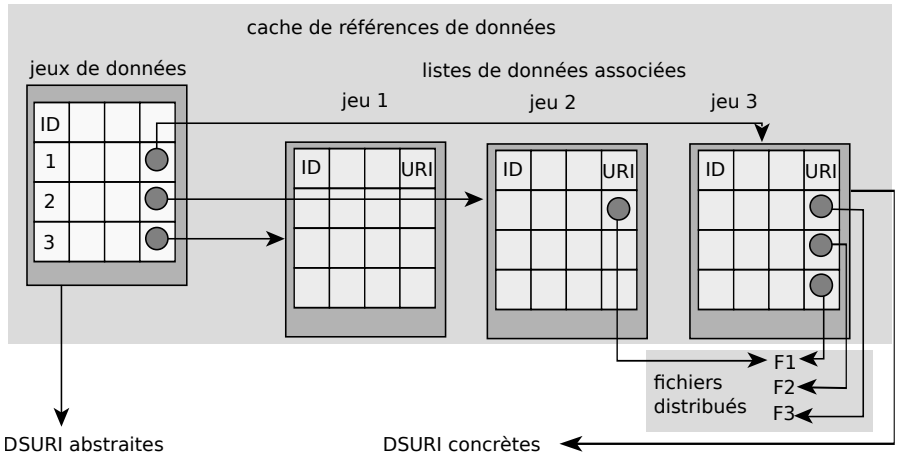


FIGURE 3.3: Le MoC UDFAS consomme et produit des jetons *DSURI* (DataSet Uniform Resource Identifier). Chaque jeton peut être interprété, lors d'un processus de résolution, en une structure de données référençant une collection de ressources distribuée, généralement des fichiers. L'ensemble de ces structures de données correspondant à la cache de références de données.

un acteur n'émettra qu'un seul jeton de sortie DSURI par invocation de job, afin de se conformer à notre *règle du jeton unique de sortie*. En d'autres termes, chaque exécution d'un acteur retourne un jeton d'URI de jeu de données référençant les résultats par commande exécutée. Ce jeton est produit avant la soumission du traitement, permettant, au niveau du MoC, d'obtenir un nombre de jetons prédictible afin de rendre possible le *pré-calcul statique de la séquence de déclenchement des acteurs*.

Les méthodes de type *listes de compréhension* (cf. section 1.3.3.1) qui opèrent un filtre sur les éléments du jeu de données sont intégrées à la syntaxe de l'URI. Chaque acteur gère de façon interne ces opérations sur les jetons DSURI d'entrée et de sortie. Celles-ci étant déclarées directement dans les DSURI, il n'est plus nécessaire d'insérer des acteurs liés aux opérations sur les jeux de données. Finalement, en hiérarchisant ainsi l'information, on facilite la production de vues intelligibles du DataFlow.

Itérations

Dans notre implémentation, le comportement par défaut d'un acteur est l'itération sur tous les éléments référencés par une DSURI d'entrée. Lorsque l'acteur possède plusieurs ports d'entrée, un produit cartésien est calculé puis chaque groupe de paramètres est itéré pour produire un job par groupe. L'utilisation de l'attribut *LoopMode*, associé à chaque port d'entrée, complète ce mécanisme. Il permet d'altérer ce comportement et d'obtenir un job unique pour l'ensemble des élements d'une DSURI (cf. section 3.2.2). Par ailleurs, les itérations sont associées à des propriétés des jetons qui sont redéfinissables. Ainsi, en modifiant la syntaxe des DSURI, le mode d'exploitation des données peu être altéré. Ce changement ne sera visible que par les acteurs qui consommeront cette référence. On peut ainsi sélectionner des données particulières dans la collection

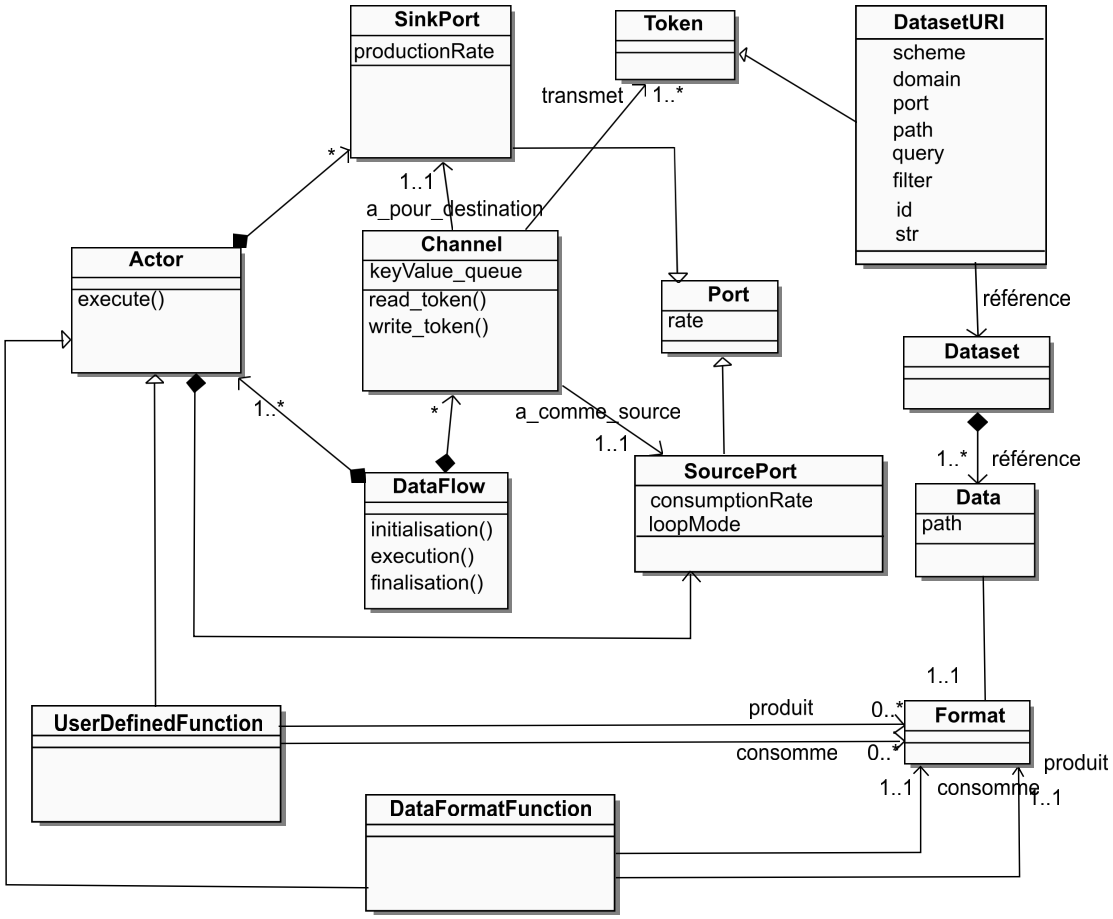


FIGURE 3.4: Modèle de calcul UDFAS (URI based DataFlow for Asynchronous Services): diagramme de classe (UML). Les canaux associés aux arêtes peuvent être implémentés sous la forme de tableaux clé / valeur. La valeur correspond au jeton de données et la clé à un indice d'itération. Les indices permettent l'association des jetons provenant de différents acteurs sources dans un contexte où l'ordre d'exécution des méthodes couplées aux éléments du tableau n'est pas prévisible.

référencée par la DSURI par filtrage sur des valeurs d'attributs [1].

Modèle hiérarchique

Notre Moc est un modèle DataFlow hiérarchique qui permet la réalisation d'*acteurs composites*, représentant des sous-graphes. La hiérarchisation est essentielle car elle simplifie la visualisation d'un workflow complexe en répartissant ces différents acteurs dans différentes vues (cf. section 1.2). On peut obtenir de cette façon une vision globale des fonctionnalités d'un workflow ou, au contraire, se concentrer sur le détail d'une étape particulière. Le mécanisme d'acteur composite permet de convertir un workflow en un acteur et favorise donc la création de composants complexes réutilisables au sein d'un workflow ou entre workflows. Ce mécanisme est donc important à la réalisation d'un de nos objectifs: la mise en place d'un répertoire de chaînes de traitements échangeables et réutilisables. Un *acteur composite* C, correspondant à un sous-workflow imbriqué, ne déclenche pas l'exécution directe d'une méthode UDF mais de l'ensemble d'un "workflow fils" F. Les acteurs d'entrée et de sortie du workflow F sont mappés sur les ports d'entrée et de sortie de l'acteur C. Les acteurs composites sont le pendant des fonctions d'un langage textuel dans le contexte d'un langage graphique DataFlow. Il s'agit d'un patron correspondant à l'implémentation de la notion de composabilité que l'on retrouve au cœur des langages fonctionnels. Un workflow maître M peut appeler le workflow F, vu comme l'acteur composite C, tout comme une fonction peut appeler une autre fonction (cf. section 1.3.3.2).

Les acteurs composites permettent une grande flexibilité dans la conception. Un workflow F qui correspond à une fonctionnalité d'usage multiple pourra être utilisé dans de nombreux workflows maîtres M tout en étant mis à jour de façon centralisée. En effet, nous avons implémenté l'intégration des acteurs composites au workflow M non par copie mais par référence. La mise à jour et la ré-utilisabilité des processus capturés sont ainsi facilitées. Néanmoins, il faut noter que ces avantages sont contrebalancés par un risque d'incohérence lors de la mise à jour du sous-workflow F, notamment si un changement dans les acteurs d'entrée et de sortie de F a lieu. Si ces changements concernent le nombre ou la sémantique de ces acteurs, une procédure de validation de la compatibilité doit être systématiquement lancée. Si le répertoire de workflows possède une structure de donnée (graphe) permettant de retrouver les relations de type $F \xrightarrow{\text{is\_used\_by}} M$, cette validation pourra être effectuée.

Les deux couches du modèle UDFAS

Le Moc délègue aux acteurs des tâches comme la soumission des jobs mais aussi le déplacement des données ou la génération de sous-ensembles de jeux de données. Ainsi, lors de la phase d'orchestration, le MoC déclenche toujours les mêmes méthodes génériques associées aux acteurs (initialize() submit(), finalize()...).Ces méthodes encapsulent la soumission des jobs mais également toute la logique de pré-traitement et

1. La fonction *filter* associée à une DSURI provoque le filtrage des éléments de la collection de données. Son fonctionnement est décrit ultérieurement, dans le chapitre 4.

post-traitement des jetons DSURI par le biais de services du WfMS. Il y a donc deux couches dans le MoC UDFAS:

— la couche DataFlow qui gére l'orchestration à partir du flot de jetons DSURI;

— la couche Acteur de préparation des données et de soumission des jobs, interne à chaque acteur.

Ces deux couches interagissent par le biais de messages contenant à la fois les états de chaque soumission et les jetons d'entrée et sortie DSURI associés.

La couche DataFlow

Si l'on se place au niveau de la couche DataFlow, le MoC fonctionne comme un modèle d'exécution DataFlow générique d'orientation fonctionnelle. Il est par conséquent particulièrement favorable à:

— la validation et l'analyse des spécifications;

— la mise au point de méthodes d'ordonnancement efficaces;

— l'extraction de multiples niveaux de parallélisme.

Chaque arête représente une dépendance de données. Une arête relie un port de sortie d'un acteur source vers un port d'entrée d'un acteur cible et représente un canal de jetons de données.

Trois classes principales d'acteurs sont disponibles, les acteurs d'entrée, de sortie et d'exécution (simples ou composites) (cf. figure 3.5A), Un acteur peut avoir de multiples ports d'entrée et sortie. Chaque entrée du workflow est associée à une liste de jetons.

La couche Acteur

La couche Acteur concerne l'intégration de la gestion de la soumission des tâches par le biais d'appels aux services du WfMS. Elle consomme et produit des jetons DSURI à taux fixe, Quel que soit le nombre de données référencées et, de ce fait, masque une partie de la complexité de la soumission. Chaque acteur peut avoir six différents états (*undefined, initialized, submitted, done, error, finalized*) (cf. figure 3.5B). Les états sont définis pour supporter des appels distants asynchrones, un patron de conception nécessaire pour les soumissions de tâches longues dans le cadre du calcul intensif. A chaque changement d'état, une méthode spécifique est appelée. La transition entre l'état *initialized* et l'état *submitted* correspond à l'appel de la méthode UDF. La transition entre l'état *submitted* et l'état *done* correspond à son exécution jusqu'à son terme.

3.2.2 Extraction du parallélisme dans le Moc UDFAS

Nous nous concentrons sur le parallélisme de données gros grain. Traiter du parallélisme spécifique à l'implémentation d'un outil encapsulé dans un acteur est en dehors de notre sujet (cf. section 2.7.2). Si ces niveaux de parallélisme existent dans l'implémentation de l'outil, ils subsisteront au sein de l'acteur UDF. Par exemple, des implémentations MPI ou GPU peuvent être appelées en utilisant une queue de soumission particulière définie par une propriété de l'acteur.

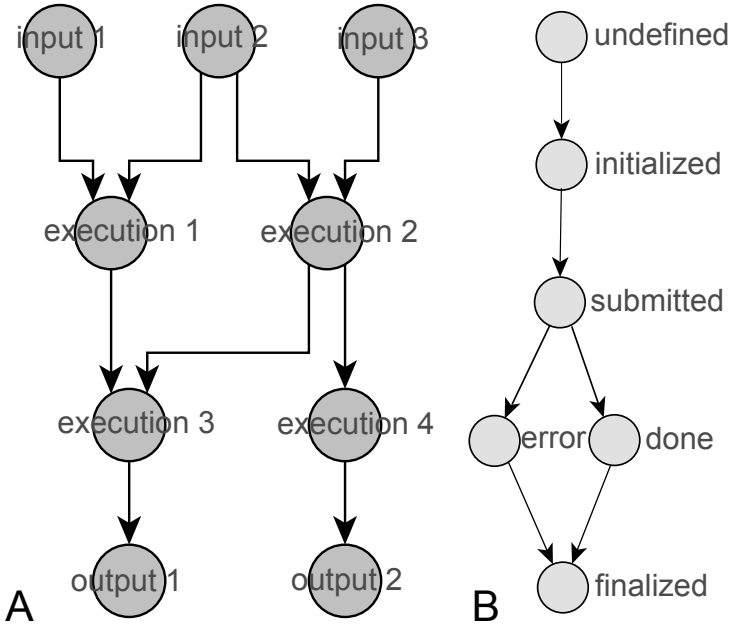


FIGURE 3.5: Le modèle d'exécution correspond à un DAG DataFlow (A) où les acteurs d'exécution encapsulent des fonctions définies par l'utilisateur (UDF pour user-defined functions). Chaque acteur est assigné à différents états (B) pour permettre le contrôle d'appels distants asynchrones, directement durant l'orchestration.

Nous allons maintenant expliquer comment le modèle d'exécution peut cibler spécifiquement différents niveaux de parallélisme complémentaires en utilisant: (i) les itérations, (ii) les dépendances de données, (iii) les fonctions *split* et *merge*.

Iterations

Tous les éléments d'une liste de jetons d'entrée peuvent être soumis en parallèle. Quand un acteur dépend de multiples paramètres, correspondant à des ports, chacun lié à une liste de jetons d'entrée, nous pouvons calculer tous les jeux de paramètres correspondant à toutes les soumissions de tâches indépendantes.

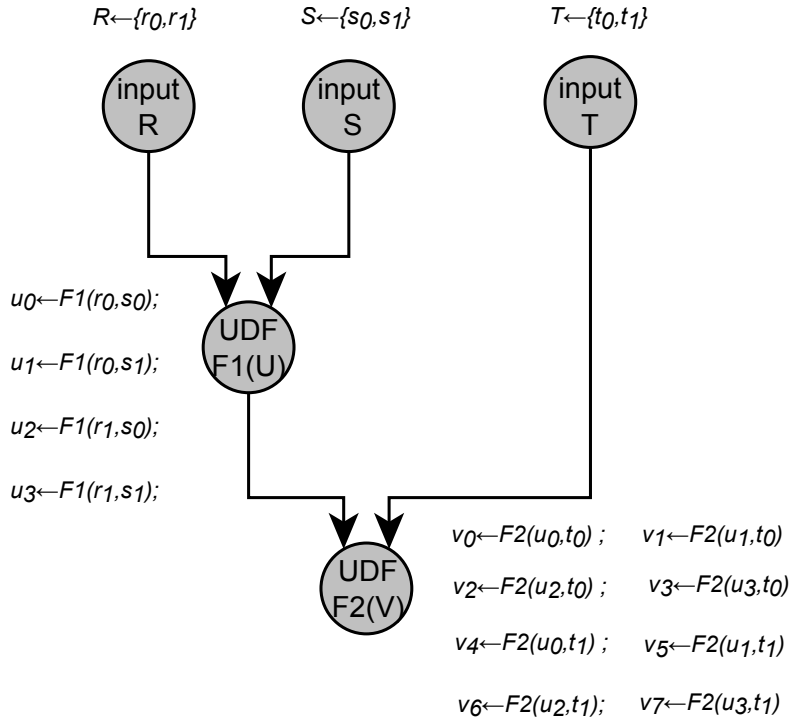


FIGURE 3.6: Calcul statique des jeux de paramètres associés aux tâches assignées aux acteurs UDF.

A titre d'exemple, cela est similaire à l'invocation en parallèle de multiples instances d'un acteur avec différents jeux de paramètres. La séquence de tous les jeux de données U d'un acteur est obtenue par un produit cartésien de toutes les listes de jetons d'entrée associées à ses ports d'entrée (cf. figure 3.6). Dans le cas d'un acteur UDF *F1* avec deux ports d'entrée, respectivement peuplés avec les listes de jetons d'entrée R et S, cela correspond à la génération de toutes les paires possibles formées à partir des éléments des deux listes , $U = R \times S$. Les jobs qui en résultent sont $F1(U)$. Chaque élément de U est un jeu de paramètres qui peut être appliqué en parallèle à la fonction *F1*.

Dépendances de données

L'analyse dynamique des dépendances de données permet d'extraire du parallélisme de tâches, adapté au contexte de l'exécution sur une infrastructure de calcul distribuée comme un cluster.

Algorithm 1 Extraction du parallélisme obtenue à partir d'un ordonnancement dynamique en utilisant les dépendances de données définies par la topologie du DAG

```
D ← setWorkflowGraph() //workflow DAG
T ← getAllJobSubmissionStatus(D)//tous les états des soumissions de job
for each n in D do
  if (n is UDFActor) then
    P ← getPredecessors(n) //tous les prédécesseurs de n
    J ← getJobSubmissions(n) //tableau des
    // soumissions de job réservées
    for each s in J do
      if (s is not submitted) then
        a ← TRUE
        for each m in P do
          l ← getLinkedJobSubmission(m, s)
          //chaque soumission de n est associée à une autre
          //soumission de job de l'acteur prédécesseur de m
          if (l is not finished) then
            a ← FALSE //Le job est défini comme non disponible
        T(s) ←a //La disponibilité du job est définie
        // pour une exécution future
S ← getAllJobSubmissions(D)//toutes les sousmissions de job
for each s in S do
  if (T(s) is TRUE) then
    if (externalSchedulingCondition(s) is TRUE) then
      p ← new thread()
      e ← fire(p, s) //exécuter la soumission de job dans un
      //nouveau thread. Cela résulte en de multiples exécutions
      //de jobs parallèles sur le cluster cible
```

L'exécution en parallèle parmi les acteurs d'un workflow est obtenue en exploitant la topologie du graphe DataFlow. En suivant les arêtes représentant les dépendances de données, chaque acteur peut être déclenché quand tous ses prédécesseurs ont été déclenchés. Chaque acteur n associé à un jeu de paramètres, représente une soumission de tâche s. Les états courants de toutes les soumissions de tâches sont stockés dans une structure de données qui les centralise, appelée table d'états. La table d'états est synchronisée de telle sorte qu'elle accepte des requêtes de lecture ou d'écriture en mode multi-threads. La lecture des états de chaque soumission de tâche de ses prédécesseurs, définie par le graphe de dépendances, permet à l'ordonnanceur d'identifier dynamiquement toutes les tâches qui peuvent être éligibles à la soumission (cf. algorithme 1). C'est à cette étape

qu'intervient *le conditionnement par valeur NULL* (cf. figure 3.2). Chaque jeton NULL atteignant un port d'entrée implique l'inéligibilité des tâches planifiées, appartenant à un sous-graphe du *graphe des dépendances d'exécution*, produit à partir des dépendances de données. Les tâches éligibles peuvent alors être soumises en parallèle, notamment par l'ordonnanceur d'un cluster de calcul, qui suivra d'autres contraintes, définies par sa stratégie d'ordonnancement.

Parallélisme de données gros grain et Patron de conception Split-Map-Merge

Le parallélisme de données gros grain est une méthode de parallélisation qui peut déboucher dans de nombreux cas sur un parallélisme massif sans modifier l'application métier (cf. section 2.7.1). Ce type de parallélisme peut être implémenté suivant un patron conception *Split-Map-Merge* ou *Split-Map-Reduce*. Le Moc UDFAS permet de réaliser ces patrons de conception grâce à un mécanisme de *gestion différentielle des itérations* sur les éléments des collections associées aux jetons DSURI.

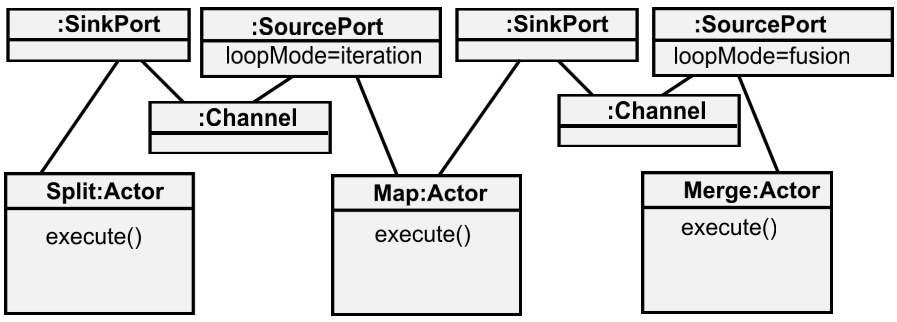


FIGURE 3.7: MoC UDFAS: diagramme d'objets UML d'une implémentation du Patron *Split-Map-Merge*.

C'est l'attribut *LoopMode* des ports d'entrée (cf. figure 3.7) des acteurs impliqués dans le patron *Split-Map-Merge* qui permet la gestion différentielle des itérations sur les éléments des jetons DSURI.

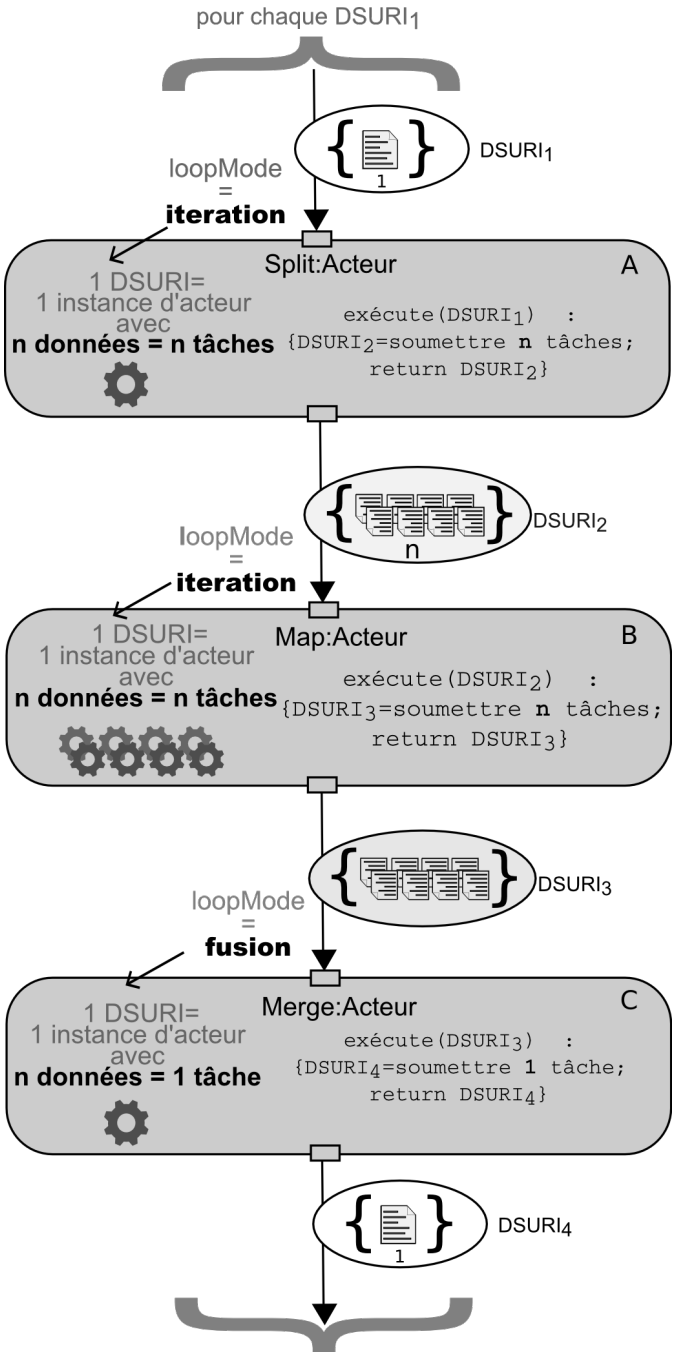


FIGURE 3.8: Gestion differentielle des itérations sur les élements des DSURI dans le MoC UDFAS par l'attribut loopMode dans un patron de conception *Split-Map-Merge*. Les acteurs *Split* et *Map* (A, B) sont en mode *itération* ($LoopMode = iteration$): chaque donnée d'une DSURI d'entrée produit donc une tâche unitaire. L'acteur *Merge* C est en mode *fusion* ($LoopMode = fusion$): l'ensemble des données d'une DSURI d'entrée produit alors une seule tâche unitaire.

En fonction de la valeur de *LoopMode* , le jeton DSURI présent sur une arête entrante d'un acteur provoquera l'exécution d'une tâche pour chaque source de données qu'il référence ou au contraire la création d'une seule tâche qui devra traiter en une fois l'ensemble des sources de données référencées. Ainsi comme le montre la figure 3.8, chaque donnée d'une DSURI produit une tâche en mode *itération* ($LoopMode =' iteration'$) et l'ensemble des données d'une DSURI produit une tâche en mode *fusion* ($LoopMode =' fusion'$). Par ce mécanisme, une méthode *Split* et une méthode $Merge$ peuvent borner un seul acteur UDF mais également différents acteurs UDF. Dans ce dernier cas, tous les fragments de données sont simplement groupés en utilisant une unique DSURI. Ainsi, grâce à la fonction de conteneur et de collection du *type monadique DSURI* (cf. section 3.2.1) et à la *gestion différentielle des itérations* , les références des données associées aux jetons seront transmises de façon transparente dans le flot de données de la couche DataFlow du Moc UDFAS (cf. section 3.2.1), simplifiant la validation, l'analyse du modèle et l'ordonnancement des tâches. En d'autres termes, la complexité du patron *Split-Map-Merge* est prise en charge par la couche Acteur et non accessible à la couche DataFlow.

Nous verrons dans la partie suivante comment une méthodologie de conception de workflows basée sur le MDA et les ontologies permet d'automatiser l'intégration de ces patrons *Split-Map-Merge* et ainsi de faciliter l'extraction du parallélisme de données gros grain.

3.3 Une démarche de conception inspirée de l'architecture dirigée par les modèles

Après avoir décrit un nouveau modèle de calcul conçu pour être efficace dans le contexte des WfMS intégratifs, nous abordons la capture des chaînes de traitements d'analyse intensive de données. Afin de la simplifier, nous proposons une méthodologie de conception de workflows guidée par l'utilisateur que nous avons élaborée en nous inspirant du MDA (cf. section 1.4). Il s'agit d'une démarche itérative, permettant le raffinage du modèle par étapes successives, du prototype jusqu'à l'implémentation. Nous la décrivons ci-dessous. Pour cela, nous utilisons une typologie des acteurs permettant de mettre en avant les composants d'analyse associés au métier. De cette façon, les chaînes de traitements sont épurées de leurs aspects techniques préalablement isolés. L'environnement logiciel se charge ultérieurement de les injecter suivant des règles liées au contexte d'exécution. Dans un premier temps, nous décrivons la phase d'intégration au WfMS de la connaissance du domaine métier, ici une discipline scientifique illustrée par l'exemple de la bioinformatique. Celle-ci s'effectue par le biais de la création d'un méta-modèle partiel constitué par le biais d'ontologies (cf. section 1.5) et accessible dans l'environnement de conception. Nous nous focalisons sur l'exploitation d'une ontologie particulière, EDAM, décrivant notamment les types de données et facilitant la mise en place d'un mécanisme de typage fort dans les spécifications DataFlow. L'exploitation du typage des entrées et des sorties des acteurs rendra possible l'introduction de la gestion automatique du parallélisme de données gros grain dans le système. Ainsi, nous mettons

en lumière que c'est la connaissance du méta-modèle intégré à l'environnement qui fait émerger les fonctionnalités avancées d'aide à la conception de workflows. Nous abordons ensuite l'exploitation de la réécriture de workflows, vu comme une transformation de modèles, capable de produire des implémentations adaptées au haut-débit.

Dans un deuxième temps, nous décrivons l'emploi d'une approche orientée modèle pour faciliter la diffusion et la ré-utilisation simplifiée des WfMS scientifiques. Pour cela, nous proposons l'intégration au WfMS d'un méta-modèle des composants et des plates-formes. Son utilisation simplifie la configuration et le déploiement des workflows sur des infrastructures PaaS et CaaS et présente un intérêt pour la mise en place d'architectures robustes de répertoires de workflows communautaires.

3.3.1 Méta-Modèle des données, conception rapide et parallélisation semi-automatique

MDA et conception rapide

Nous détaillons ici notre méthodologie de capture d'un chaîne de traitements inspirée du MDA. L'utilisateur commence par spécifier un workflow en utilisant une interface graphique. Il connecte les nœuds représentant les outils d'analyse dans un graphe DataFlow. La Figure 3.9 illustre l'approche. Il doit intégrer ensuite des annotations pour spécifier les types de données circulant entre les nœuds du graphe (tâches d'analyses bioinformatiques). A partir de ce graphe, un "graphe d'exécution" correspondant à une spécification DataFlow interprétable par un moteur de workflows, est généré en utilisant la transformation de modèles. C'est à partir de ce nouveau graphe que du parallélisme peut être automatiquement extrait.

Pour débuter le processus de design, nous définissons un "modèle de conception" DataFlow simplifié adapté à la capture graphique d'un workflow. Ce modèle est un graphe DataFlow où les processus sont des nœuds et les dépendances de données les arêtes d'un graphe direct acyclique (DAG pour direct acyclic graph). Chaque nœud du DAG est un acteur dont les trois classes majeures sont les entrées, les sorties et les exécutions. Un acteur d'exécution encapsule un script ou un outil en ligne de commande, qui, sans sémantique additionnelle, sera vu par le système comme une boîte noire correspondant à une *fonction définie par l'utilisateur* ou (cf. section 2.4). Chaque arête représente une dépendance de données. Une arête relie un port de sortie d'un acteur source vers le port d'entrée d'un acteur cible et représente un canal de jetons de données. Les acteurs peuvent avoir de multiples ports d'entrée et de sortie. L'ordonnancement des tâches est implicite. La conception de la chaîne de traitements s'effectue via une interface graphique incluant un éditeur de graphes mais la spécification DataFlow peut également être écrite dans un fichier en suivant une syntaxe dédiée. Pendant la capture du prototype, seules les étapes majeures du traitement doivent être représentées en tant qu'acteurs UDF. Les outils utilitaires qui réalisent des opérations en rapport avec la validation, la transformation et la conversion des données et qui n'agrégent ou ne génèrent pas de connaissances additionnelles dans les données sont appelées ici Fonction de Format de Données (DFF pour data Format Function). Les acteurs DFF doivent être omis durant la capture du

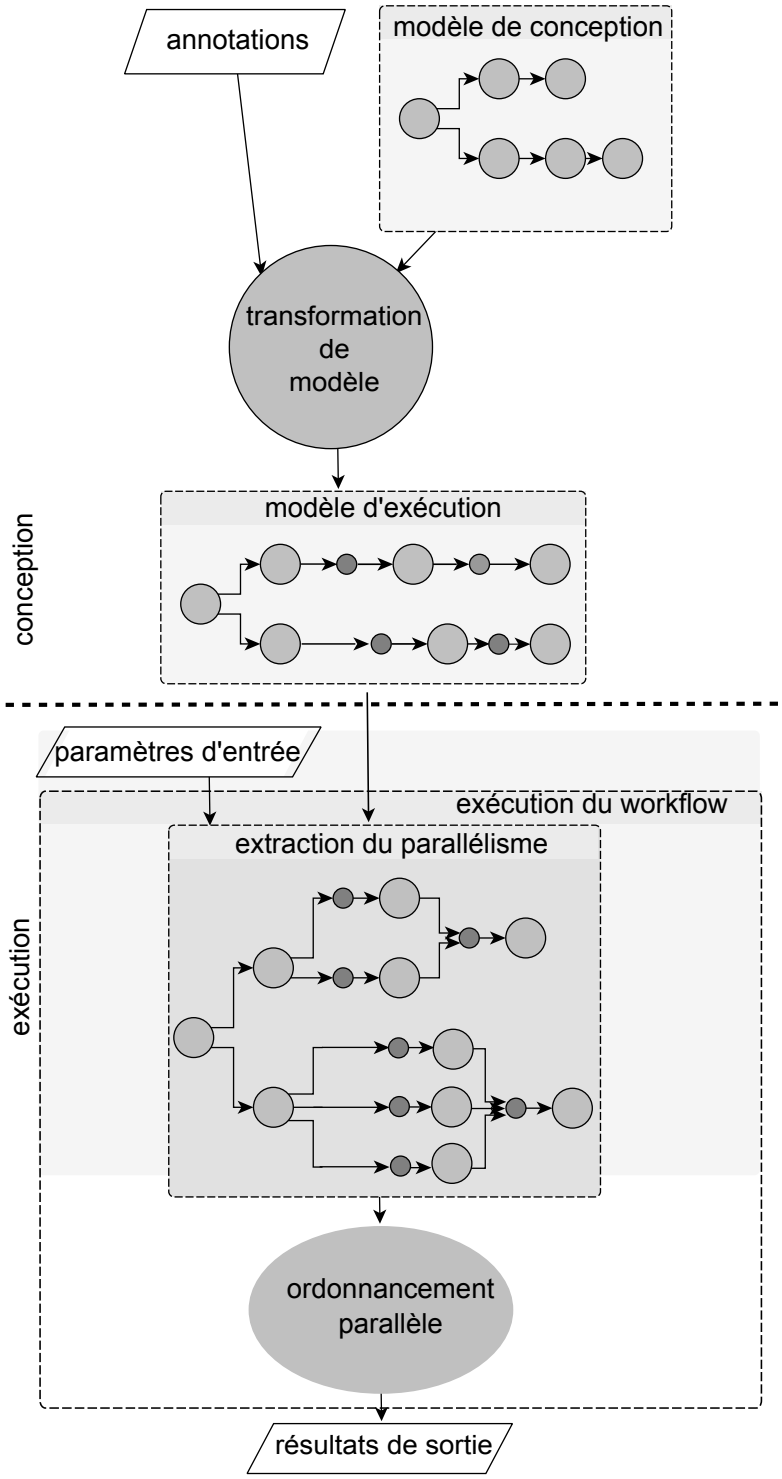


FIGURE 3.9: Vue d'ensemble de la conception et de l'exécution d'un workflow.

prototype. Les outils préexistants et les scripts qui respectent ces conditions sont encapsulés au sein des acteurs UDF.

Exploitation du Méta-Modèle des données pour la parallélisation semi-automatique

Après cette première étape de capture d'un prototype, nous allons présenter comment nous intégrons au système un méta-modèle permettant à l'utilisateur de bénéficier d'un mécanisme de typage fort pouvant déboucher sur de la parallélisation massive par les données.

Pour commencer, les ports d'entrée et de sortie des acteurs UDF doivent être partiellement ou complètement annotés par le concepteur en utilisant une hiérarchie des formats de données fournis par le système (cf. figure 3.10). Cette partie du Méta-Modèle des données représente les formats et leurs méthodes associées. L'annotation des paramètres des UDF constitue une sémantique additionnelle interprétable comme un typage fort des données.

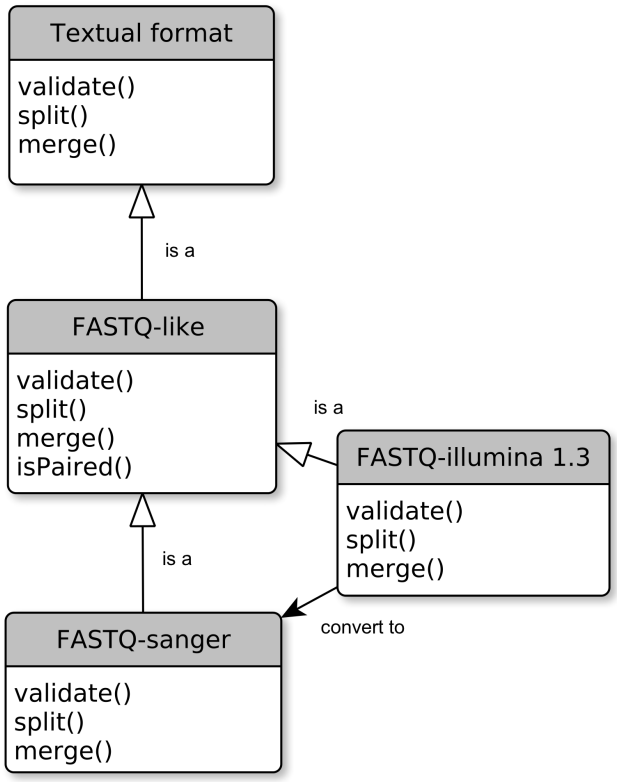


FIGURE 3.10: Nous utilisons une hiérarchie des formats de données dérivée de l'ontologie EDAM. Les méthodes *Split* et *Merge* sont implémentées pour chaque format de données d'intérêt.

En fonction du typage choisi, les acteurs DFF pré-définis sont ensuite proposés au concepteur en tant que méthodes de pré-traitement et de post-traitement applicables aux ports d'un nœud acteur.

Cette intégration correspond à une séquence de méthodes appliquées sur les arêtes du DAG DataFlow. La séparation résultante des tâches du workflow entre UDF et DFF permet d'afficher une vue claire du workflow (cf. figure 3.12). Cette "vue de l'analyse" ou "vue de conception" ne représente que les entrées, les sorties et les acteurs UDF, en rapport avec les tâches du domaine (ici mapping, assembly, snp calling...). Les acteurs DFF (validation, conversion et autres utilitaires) restent masqués. Cela produit une vue d'ensemble plus intelligible facilitant une compréhension plus directe du workflow. C'est également un moyen de limiter la prolifération d'acteurs techniques visibles ou d'adaptateurs de données, parfois appelée "shim problem" [86]. Nous obtenons un modèle de workflow complet semi-concret qui sera ultérieurement transformé en un modèle directement interprétable par un moteur de workflows.

Pour produire et organiser nos DFF, nous avons généré une hiérarchie de formats de données, centrée sur le domaine d'application de la bioinformatique et dérivée de l'ontologie EDAM [98]. EDAM (EMBRACE Data And Methods) est une ontologie des opérations, des types de données et des formats en bioinformatique (cf. figure 3.11). Le vocabulaire des termes et relations fourni a déjà été utilisé pour classifier des outils et pour la composition de workflows [120].

L'intérêt d'intégrer à l'avance des DFF au système est d'autant plus fort que les formats de données sont exploités par de nombreux composants.

Transformation de modèles

Après avoir décrit le méta-modèle partiel qui permet le typage des acteurs, nous présentons maintenant comment l'exploitation du typage rend possible l'expression d'un parallélisme de données massif.

La transformation de modèles est utilisée dans les architectures dirigées par les modèles (Model-Driven Architecture) pour la génération de code en incluant la parallélisation automatique [121]. Elle est employée couramment dans la capture graphique des processus et a déjà été appliquée à la conversion de formats dans des chaînes de traitements [122]. A notre connaissance, elle n'a pas encore été appliquée pour générer du parallélisme de données dans des workflows.

Notre approche dépend de la définition initiale d'une librairie de méthodes efficaces de découpage et de fusion (Split, Merge) des données en rapport avec les formats utilisés les plus communément dans notre domaine d'application. Pour organiser de façon efficace ces jeux d'utilitaires, la hiérarchie des formats introduite préalablement est employée (cf. figure 3.10). Ces utilitaires sont manuellement créés et associés à leur format de référence au sein d'une hiérarchie de formats. Pour chaque format et ses variants, un jeu personnalisé de fonctions est défini, incluant les implémentations des méthodes "Split" et "Merge", ainsi que les options de paramétrage appropriées. Ce sont des DFF (Data Format Function).

Pour un port d'acteur marqué avec un format de données, toutes les méthodes asso-

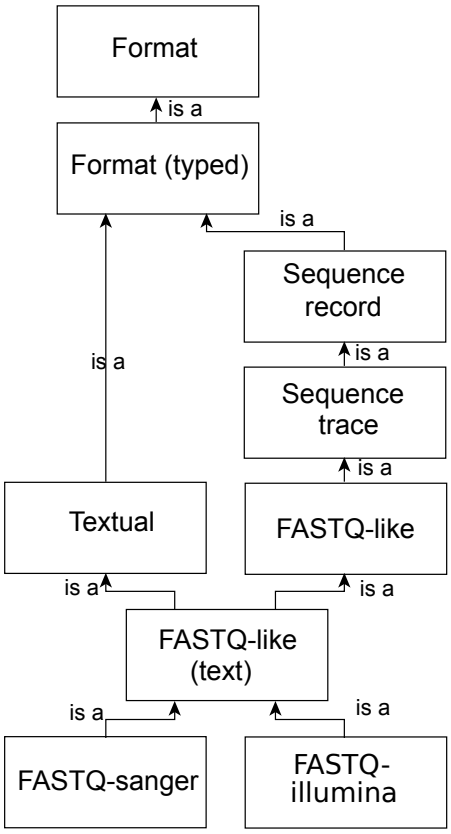


FIGURE 3.11: Exemple de la hiérarchie associée au format de données FASTQ-sanger dans EDAM (EMBRACE Data And Methods), une ontologie des opérations, type et formats en bioinformatique.

ciées avec ce format et ses prédécesseurs dans la hiérarchie peuvent alors être employées pour annoter sémantiquement les arêtes passant par ce port dans le graphe du "modèle de conception" (cf. figure 3.12A). Ce processus d'annotation est défini par l'utilisateur.

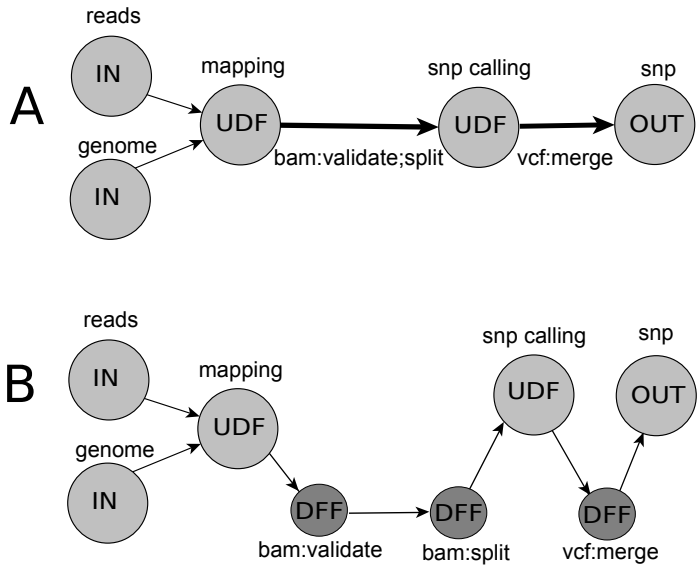


FIGURE 3.12: La définition des acteurs UDF et DFF permet la création de deux vues différentes montrant des modèles DataFlow dédiés à la conception (A) ou à l'exécution et au contrôle de celle-ci (B).

Un mécanisme de transformation basé sur des patrons de graphes, convertit le "modèle de conception" annoté en un "modèle d'exécution", après validation des contraintes. Nous précisons que dans le cadre de l'implémentation de ce mécanisme, le modèle d'exécution correspond à une spécification de workflows compatible avec le Moc UDFAS (cf. section 3.2.1).

A chaque modèle correspond une vue, la "vue de conception" (cf. figure 3.12A) et la "vue d'exécution" (cf. figure 3.12B). La "vue d'exécution" est une vue technique où tous les acteurs sont représentés. Les acteurs UDF et DFF sont tous représentés comme des nœuds. Cette vue est plus proche de l'implémentation et est utile pour le *monitoring* des exécutions. Parce que tous les DFF sont générés comme de nouveaux acteurs dans le modèle exécution, les méthodes *Split* et *Merge* deviennent représentées sous forme d'acteurs. Cela signifie que lorsque des formats de données ont été spécifiés sur les ports d'entrée et de sortie via les annotations de l'utilisateur, un patron *Split-Map-Merge* est généré, en utilisant les méthodes associées à chaque format. Dans de nombreux cas d'utilisation, un simple patron *Split-Map-Merge* conduit à l'extraction d'un parallélisme massif de données gros grain, comme nous le décrirons en détail dans le chapitre 5 (). Comme nous l'avons vu, les méthodes *Split* et *Merge* sont des Fonctions de Format de Données ou DFF. Au sein du modèle de calcul exploité par le moteur de workflows,

les acteurs DFF ne différent pas des acteurs UDF. Ainsi, après la transformation de modèles qui exploite les annotations définies par l'utilisateur, un patron de conception *Split-Map-Merge* est automatiquement inséré dans le modèle DataFlow. Par exemple, le nœud correspondant à l'acteur $UDF(F1)$ est remplacé par un sous-graphe:

$$split \rightarrow UDF(F1) \rightarrow merge$$

Dans le cas de l'utilisation du MoC UDFAS, chaque port d'acteur sera généré avec un attribut *LoopMode* initialisé de telle sorte que le moteur de workflows puisse gérer différentiellement les itérations sur les éléments des jetons DSURI (cf. section 3.2.2).

Finalement, l'interprétation du modèle résultant par un moteur dédié correspond à l'exécution d'une implémentation parallèle du workflow capturé.

3.3.2 Méta-Modèle des composants et des plates-formes et adaptation à de multiples environnements d'exécution

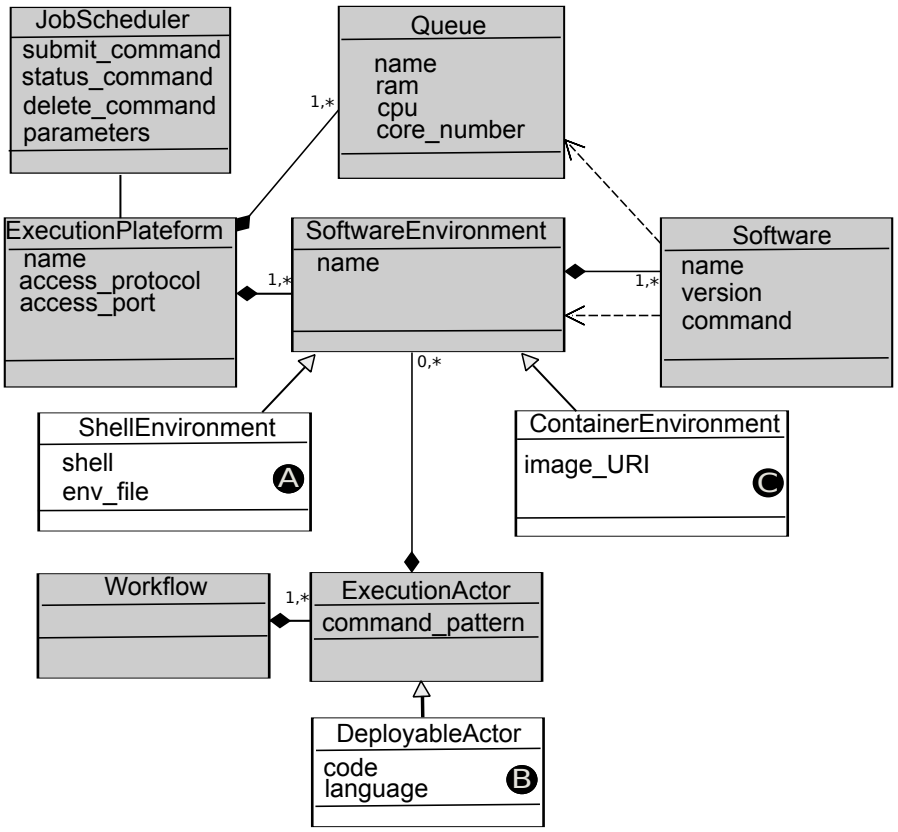


FIGURE 3.13: Méta-modèle des composants et des plates-formes (UML). La partie grise inclut les entités du cœur du modèle. Les entités en blanc (A, B, C) correspondent à des raffinages du Méta-Modèle que nous exploitons pour favoriser l'indépendance des spécifications de workflows vis-à-vis des infrastructures d'exécution.

Dans la partie précédente nous avons présenté un modèle de calcul DataFlow adapté aux analyses de données intensives puis une méthode de conception rapide orientée modèle. Une fois capturé, le passage de la spécification à l'exécution intensive a été envisagé par la transformation de modèles qui automatise l'extraction du parallélisme de données grain. Après avoir traité de la phase de capture des spécifications, nous poursuivons ici nos propositions méthodologiques d'aide à la conception en abordant la phase de déploiement.

D'une part, il s'agit d'une étape critique qui limite actuellement la ré-utilisation des workflows sur une infrastructure non prédéfinie. De ce fait, la capacité des scientifiques à tirer parti des répertoires de workflows communautaires est réduite. D'autre part, à l'heure actuelle, dans les WfMS intégratifs, le degré de personnalisation des acteurs UDF est directement déterminé par la capacité à simplifier le déploiement des dépendances des acteurs sur une infrastructure cible (cf. section 2.3.1). L'intégration de mécanismes de simplification du déploiement de nouveaux composants au WfMS peut donc bénéficier à la fois aux scientifiques consommateurs de services et concepteurs services (cf. section 2.2.2). Nous allons donc proposer d'étendre les WfMS de telle sorte qu'ils permettent un processus de conception et d'exécution de workflows totalement dirigé par l'utilisateur par la mise en place de méthodes de mise à disposition rapide des chaînes de traitements. Afin de simplifier la configuration et/ou le déploiement de l'ensemble des dépendances nécessaires à l'exécution d'un workflow nous allons décrire une approche orientée modèle qui consiste à associer à un modèle générique de workflow, un ensemble de spécifications décrivant les composants de façon indépendante des plates-formes. Ces spécifications doivent pouvoir être capturées par le concepteur de workflows et permettre d'automatiser le déploiement.

Nous ciblons ici différents types d'infrastructures de calcul: des clusters de calcul déjà peuplés d'outils, et plus généralement les environnements PaaS et CaaS (cf. section 2.6.1). Nous travaillons dans un contexte de conception dirigée par l'utilisateur et cherchons à minimiser le rôle d'autres intervenants médiateurs (administrateurs système...). Pour cela, nous proposons d'intégrer au WfMS un méta-modèle des composants et des plates-formes, permettant de définir un processus générique de déploiement des dépendances des workflows.

Spécification d'un méta-modèle des composants et des plates-formes

Notre objectif est d'étendre le méta-modèle exploité dans un système de gestion de workflows intégratif afin de favoriser l'indépendance des chaînes de traitements aux environnements d'exécution. Pour cela nous proposons l'ajout au méta-modèle d'une partie décrivant les composants et les plates-formes en mettant l'accent sur les dépendances des acteurs que nous nommons le *méta-modèle des composants et des plates-formes* (cf. figure 3.13), abrégé CPMM (Components and Platforms Meta-Model).

Définition 3.2 *Le méta-modèle des composants et des plates-formes (CPMM pour Components and Platforms Meta-Model) est le méta-modèle qui décrit les plates-formes d'exécution et les composants logiciels associés à leurs dépendances, indépendamment des workflows.*

La mise en relation des modèles de workflow SCDFM et du CPMM a pour but la génération d'implémentations fonctionnelles sur chaque plate-forme d'exécution. Dans le cadre d'une démarche orientée modèle, ces implémentations sont des PSM (Platform Specific Model) et la spécification SCDFM est assimilable à une forme de PIM (Platform Independent Model) (cf. section 1.4.1).

Au sein du*méta-modèle des composants et des plates-formes* nous cherchons à formaliser les dépendances des acteurs UDF et des plates-formes afin de permettre un processus automatique et générique de déploiement des chaînes de traitements. L'ajout du *méta-modèle des composants et des plates-formes* au WfMS se traduit par la mise en relation des entités des modèles de workflow (*Workflow*, *ExecutionActor*) avec de nouvelles entités représentant les dépendances des acteurs (*SoftwareEnvironment*, *Software*) en rapport avec les plates-formes(*ExecutionPlateform*, *JobScheduler*, *Queue*). Le CPMM comporte donc:

— la classe *Software* désigne un outil ou composant logiciel utilisable par un *ExecutionActor* d'un *Workflow*;

— *SoftwareEnvironment* représente une configuration de la plate-forme permettant l'exécution d'une liste de composants logiciels;

— *ExecutionPlateform* correspond à une plate-forme d'exécution comme un cluster de calcul sur laquelle des workflows peuvent être déployés;

— la classe *JobSchedulter* désigne l'ordonnanceur qui répartit les tâches sur les nœuds de calcul de la plate-forme (*ExecutionPlateform*);

— *Queue* est la classe d'entités relative aux files de soumission d'une plate-forme et qui rend possible l'exécution sur un sous-ensemble de nœuds de calcul homogènes.

Après cette présentation, nous introduisons trois raffinements du méta-modèle CPMM (cf. figure 3.13) qui accroissent l'adaptation d'un WfMS au prototypage et facilite le nomadisme des châines de traitements. Les mécanismes de gestion des dépendances d'acteurs qui les exploitent et que nous allons décrire ici rendent possible soit l'automatisation de la configuration des dépendances d'acteurs soit leur déploiement, dans un processus guidé par l'utilisateur.

Adaptation au changement d'environnement d'exécution par le mécanisme commun des variables d'environnement

La capture de l'ensemble des dépendances nécessaires à l'exécution d'une méthode UDF, ici une ligne de commande, encapsulée dans un acteur, doit rendre possible son exécution mais également faciliter le changement de plate-forme. Les dépendances logicielles nécessaires à l'exécution d'un programme en ligne de commande peuvent être configurées par le mécanisme commun des variables d'environnement. Ainsi le simple fait d'ajouter l'appel à un script, qui configure des variables d'environnement avant l'appel à une méthode UDF, est suffisant pour permettre la configuration d'un outil d'analyse sur un nœud de calcul, dans la mesure où les dépendances logicielles sont déjà présentes sur le système. Chaque infrastructure, représentée par une instance de la classe *ExecutionPlateform*, peut donc être caractérisée dans le CPMM par un ensemble de configurations

logicielles associées à des scripts de configuration (*SoftwareEnvironment*). Le concepteur de composants n'aura plus qu'à établir l'ensemble des liens entre Acteur UDF et configuration pour permettre l'exécution sur l'infrastructure de son choix. La séparation dans le modèle entre patron de commande d'un acteur (attribut *commandPattern*) et commande de configuration rend possible l'adaptation rapide d'un workflow à un contexte d'exécution en redéfinissant simplement les configurations logicielles associées aux acteurs du workflow. Ce mode d'exploitation du méta-modèle des composants et des plates-formes est adapté à des environnements d'exécution déjà peuplés ou bénéficiant d'un médiateur (administrateur système). Nous les assimilerons à des cas du modèle PaaS (cf. section 2.6.1). Lorsque le médiateur installe un outil, il crée un fichier de configuration des variables d'environnement permettant de l'exécuter. La tâche supplémentaire pour le médiateur est alors de référencer cette nouvelle configuration pour une plate-forme donnée dans l'instance du CPMM intégrée au WfMS. Lorsqu'un concepteur créera un composant dans le WfMS, il devra aussi procéder à une tâche additionnelle. Il associera le composant à une configuration pour chaque plate-forme où il veut rendre possible l'exécution.

Le catalogue de configurations d'applications résultant gère des scripts de configuration spécifique de chaque composant métier et pour chaque plate-forme d'exécution référencée (cf. figure 3.14). Cette exploitation du méta-modèle des composants et des plates-formes favorise le prototypage de façon marginale. Elle permet de simplifier le basculement entre un environnement de test et un environnement de production par exemple. Cependant, elle accroît l'indépendance des spécifications des workflows vis-à-vis des environnements d'exécutions et favorise donc le nomadisme, même si elle est limitée à des environnements comme des clusters déjà peuplés ou au PaaS. Par ailleurs son utilisation reste coûteuse du fait de la nécessité de médiateurs.

Dépendances de composant générique et plates-formes agnostiques

Nous cherchons à faciliter le déploiement des workflows sur les infrastructures de calcul. Nous avons vu qu'un système de configuration dynamique basé sur le méta-modèle CPMM permettait la configuration sur des plates-formes existantes déjà peuplées d'outils métiers, clusters ou PaaS. Néanmoins, la configuration des différents environnements visés reste laborieuse et l'utilisateur reste peu autonome dans le cas du déploiement de nouveaux composants. Afin de mieux adapter les systèmes de gestion de workflows au prototypage d'applications scientifiques d'analyse de données, nous allons maintenant décrire des usages du CPMM rendant possible le déploiement de composants, soit en cours d'élaboration et de ce fait, non connus du système, soit préexistants mais non encore déployés sur le système. Il s'agit de deux méthodes de capture des dépendances des acteurs exploitant des raffinements du CPMM, les acteurs UDF à code déployable (cf. figure 3.13B) et la gestion des environnements logiciels à base de containers (cf. figure 3.13C).

Acteur UDF à code déployable Afin de donner la possibilité au concepteur de chaînes de traitements d'éditer directement le code de ses composants à partir du GUI

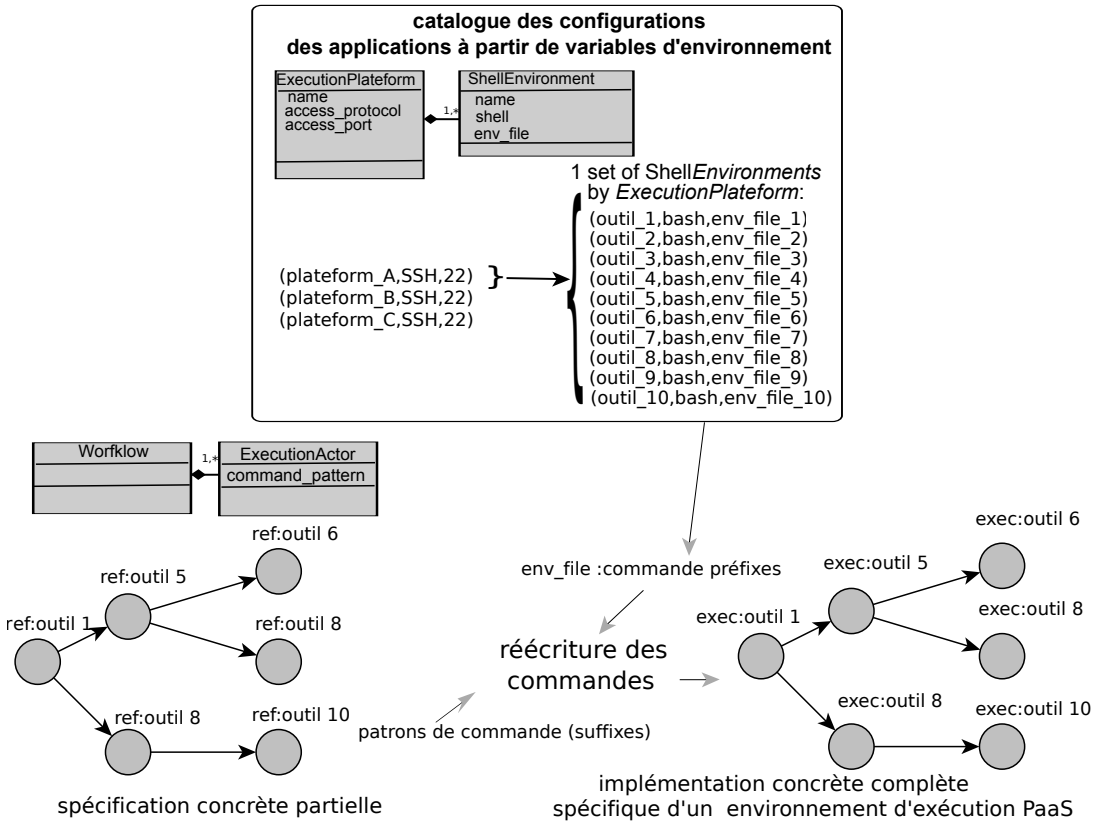


FIGURE 3.14: Spécification concrète partielle ou SCDFM (Semi-Concrete DataFlow Model) associée à des catalogues de configurations d'applications. L'adaptation à différents contextes d'exécution est permise par l'association du modèle de workflow, sous la forme d'une spécification concrète partielle, associée à des catalogues de configurations d'applications basés sur un méta-modèle des composants et des plates-formes.

du WfMS, nous définissons un type d'acteur UDF particulier, associé à du code éditable ou acteurs UDF à code déployable (*DeployableActor*). Dans le cadre du prototypage, les acteurs UDF à code déployable sont conçus pour permettre l'édition en ligne de code interprétable par un langage de script. Chaque exécution d'un workflow incluant ce type d'acteur doit inclure une phase d'analyse de dépendance des acteurs en rapport avec l'état de la plate-forme d'exécution cible. En, effet, chaque modification du script doit impliquer un redéploiement du script.

Environnements logiciels à base de containers Classiquement les dépendances d'une application en ligne de commande sont caractérisées par un ensemble de fichiers (librairies, packages, binaires...). Les méthodes communes de déploiement exploitent la notion de package et de script de déploiement. Elles impliquent encore des opérations nécessitant des médiateurs et une formalisation minimum pour la production de packages. Ce travail de packaging est assez laborieux et éloigné des préoccupations d'un scientifique, concepteur de composants. Afin de simplifier la définition rapide des dépendances d'acteur UDF, nous proposons une approche de gestion des dépendances basée sur des containers. La technologie des containers Linux apporte des propriétés d'isolation des processus et d'encapsulation des dépendances logicielles adaptées à la fois au prototypage et aux environnements multi-utilisateurs. Elle facilite le déploiement des dépendances logicielles nécessaires à l'exécution des UDF en fonction des besoins, lors d'une phase d'analyse des dépendances du workflow ou même au moment de l'exécution. Dans ce dernier cas, l'exécution de l'UDF est précédée d'une vérification de la disponibilité du container sur le nœud de calcul. Celui-ci sera téléchargé intégralement ou mis à jour si nécessaire puis la commande sera exécutée dans l'environnement logiciel du container. Ce mode opératoire est permis par la technologie de gestion de container Docker qui gère le téléchargement d'images de containers à partir de son identifiant, qui intégre le nom de la machine hôte. Les identifiants de containers sont donc assimilables à des URI. Nous formalisons ce mécanisme de gestion des dépendances en introduisant dans le CPMM la classe *ContainerEnvironment* qui possède un attribut *image_uri* (cf. figure 3.13C) destiné à contenir les valeurs des différentes URI des images des containers.

Pour rendre possible le déploiement d'un nouvel acteur UDF, le concepteur du composant doit l'asscocier à un environnement (*ContainerEnvironment*) possédant un attribut *image_uri* référençant un container conçu par lui-même ou par un tiers.

L'intégration de ce type de méthode de déploiement de workflows (cf. figure 3.15) au WfMS nécessite de coupler le répertoire de modèles de conception de workflows à un répertoire de containers, référençant les dépendances et outils d'analyse du domaine. Nous pensons que la gestion communautaire de cette collection de containers permettrait une couverture fonctionnelle large du domaine d'application et amplifierait notablement les capacités de nomadisme des chaînes de traitements.

Le modèle de déploiement basé sur les containers présente des avantages par rapport aux modèles communément exploités aujourd'hui puisqu'il rend possible l'automatisation de l'ensemble des processus de déploiement, directement avant exécution du workflow, quelque soit l'infrastructure dans la mesure où celle-ci présente un gestionnaire

de containers (Docker) et de leur cycle de vie (modèle CaaS). Les containers peuvent être créés directement par le concepteur de composants et déposés dans un répertoire communautaire de containers. Il s'agit donc bien ici d'un modèle adapté à la conception et au prototypage.

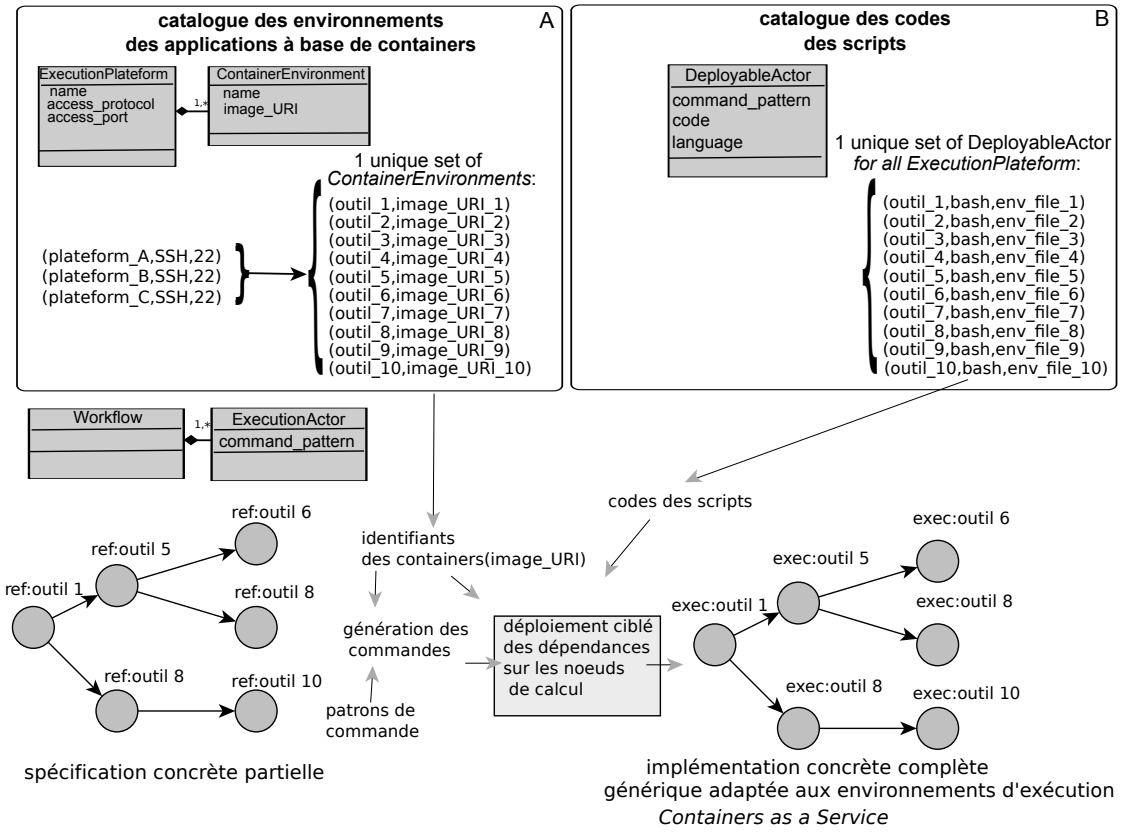


FIGURE 3.15: Spécification concrète partielle de workflows associée à des environnements logiciels à base de containers (A) et un catalogue de codes (B).

Spécification DataFlow autonome pour l'échange et la reproductibilité (A-SCDFM) Pour développer la capacité au nomadisme des chaînes de traitements et la généraliser mais aussi faciliter le prototypage de composants dans les WfMS, nous avons proposé un méta-modèle des composants et des plates-formes (CPMM). Celui-ci peut structurer les informations nécessaires au déploiement d'un workflow sur une infrastructure d'exécution agnostique de type CaaS. Il permet au concepteur de capturer les dépendances essentielles à l'exécution des composants, indépendamment des spécifications des chaînes de traitements. Pour cela, nous avons étendu le CPMM (cf. section 3.2) pour qu'il facilite la création d'acteurs à code déployable et d'acteurs à base de containers.

Nous allons maintenant formaliser la définition d'une spécification de workflows issue du modèle SCDFM et tirant parti du CPMM pour faciliter l'échange et la ré-utilisation.

Définition 3.3 *C'est la mise en relation des dépendances d'acteurs formalisées par le CPMM avec une spécification de workflows SCDFM (voir définition 3.1) qui rend envisageable la définition d'une spécification de workflows autonome (cf. figure 3.16), nommée A-SCDFM (Autonomous Semi-Concrete DataFlow Model). Nous considérons la spécification de workflows comme autonome lorsqu'elle contient l'information minimale suffisante pour permettre l'exécution, indépendamment de l'infrastructure de calcul.*

La notion de spécification de modèle de workflow semi-concret autonome A-SCDFM permet le référencement des dépendances des composants dans le modèle. Il s'agit d'une spécification DataFlow générique permettant le déploiement sur toute plate-forme de *cloud computing* CaaS (Container As A Service).

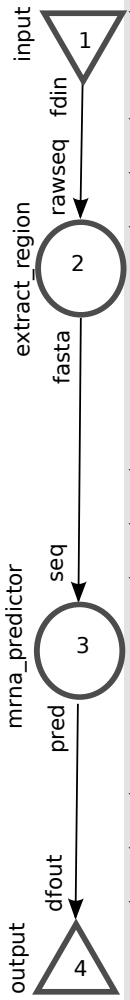
Avec cette approche, un répertoire de workflows peut être basé sur des *modèles de conception générique* puis raffiné en des *spécifications autonomes*, grâce aux liens entre acteurs, code et containers. Il s'agit d'une solution concrète apportée au problème de réutilisation des workflows diffusés [115]. Nous proposons l'élaboration de répertoires de workflows à partir de spécifications A-SCDFM associées à un répertoire de dépendances de composants à base de containers et de scripts (CPMM). Ces spécifications sont destinées à être exploitées par des moteurs de workflows exploitant des infrastructures CaaS (Containers as a Service).

3.4 Résumé

Nous avons décrit dans ce chapitre un ensemble de propositions méthodologiques permettant l'adaptation des WfMS scientifiques intégratifs à la conception de chaînes de traitements d'analyse intensive de données et à leur diffusion. Nous suggérons d'introduire dans les WfMS de formalismes concernant le prototypage, l'adaptation au calcul intensif, et le déploiement sur les infrastructures de calcul. Nous résumons nos propositions ci-dessous.

— Tout d'abord, nous avons défini un nouveau modèle de calcul DataFlow, nommé UDFAS, dédié à l'exploitation du parallélisme par les moteurs de workflows. UDFAS est conçu pour être performant et adapté aux architectures SOA des WfMS intégratifs. Il est compatible avec un modèle de workflow semi-concret (SCDFM). Cette spécification de workflows rend possible une représentation expressive de l'analyse et prend en compte les multiples niveaux de parallélisme extractibles d'une représentation DataFlow d'une analyse de données.

— En outre, nous avons élaboré une méthodologie de conception orientée modèle particulièrement adaptée à la capture et au prototypage des chaînes de traitements et à leur transformation en implémentation capable de traiter les données à haut-débit; Cette approche exploite un typage fort des données provenant d'un méta-modèle partiel relatif au domaine disciplinaire et construit à partir d'une ontologie des types de données et de formats de la bioinformatique (EDAM).




```
<?xml version="1.0" encoding="UTF-8" ?>
<dataflow>

<actor name="input" nodeId="1" actor_type="input"  >
  <outPort name="dfin" >
   <data ds_uri="scp://data.genouest.org/1245" >
  </outPort>
</actor>

<channel sourceId="1" targetId="2" sourcePort="fdin" targetPort="rawseq" />

<actor  name="extract_region" nodeId="2"  actor_type="exec">
  <commandPattern
      image_uri="http://docker-registry.genouest.org/bioperl/latest"
      script_uri="http://data.genouest.org/7895?filter=extractRegion" >
    perl extractRegion.pl -i $rawseq -o seqout.fasta -l extReg.log
  </commandPattern>
  <inPorts name="rawseq" />
  <outPort name="fasta" filter="*fasta" />
  <outPort name="log" filter="*log"  />
</actor>

<channel sourceId="2" targetId="3" sourcePort="fasta" targetPort="seq" />

<actor name="mrna_predictor" nodeId="3" actor_type="exec" >
  <commandPattern
          image_uri="http://docker-registry.genouest.org/RNApred/V1.2" >
    RNApredictor -s 120 -t m -i $seq -o ${seq.basename()}.pred.txt
  </commandPattern>
  <inPort  name="seq" loopmode="iteration" />
  <outPort name="pred" filter="*_pred.txt" loopmode="iteration" />
  <outPort name="log" filter="*.log" loopmode="iteration" />
</actor>

<channel sourceId="3" targetId="4" sourcePort="pred" targetPort="dfout" />

<actor nodeId="4" name="output" actor_type="output">
  <inPort name="dfout" />
 </actor>

</dataflow>
```

FIGURE 3.16: Spécification DataFlow Autonome: Représentation XML d'un workflow A-SCDFM (Autonomous Semi-Concrete DataFlow Model) correspondant à une spécification autonome. L'ajout d'URI de jeux de données permet de relier la spécification du workflow aux codes des scripts (attribut script_uri) déployables et à des jeux de données d'entrée (attribut ds_uri). Les URI d'image de containers (attribut image_uri) permettent de rapatrier les dépendances des acteurs d'exécution sur les nœuds de calcul. L'ensemble des informations contenues dans cette spécification est suffisant pour permettre l'exécution du workflow sur une infrastructure de type CaaS quelconque.

— Nous avons également formalisé l'association de la spécification de workflows SCDFM avec un méta-modèle des composants et des plates-formes (CPMM). Celui-ci facilite la génération, à partir d'un seul modèle, de multiples implémentations, chacune compatible avec une plate-forme PaaS particulière, dans la mesure où ces infrastructures sont pré-peuplées avec les composants nécessaires.

— L'usage du CPMM a ensuite été étendu à la création d'une spécification de workflows *autonome* (A-SCDFM). Celle-ci est suffisante pour automatiser le déploiement de l'ensemble des dépendances des acteurs d'un workflow sur une infrastructure de calcul CaaS. Ainsi, une unique capture d'une chaîne de traitements en A-SCDFM rend possible son exécution sur toute plate-forme agnostique CaaS, indépendamment de l'hétérogénéité et de l'évolution des environnements logiciels. A partir de ce constat, nous proposons de baser l'architecture des répertoires de workflows communautaires sur le formalisme A-SCDFM. Les spécifiations de workflows doivent dans ce cas être associées avec un répertoire dédié d'images de containers encapsulant les composants de façon normalisée. La mise en œuvre de cette architecture pourrait fortement simplifier la diffusion et la ré-utilisation des chaînes de traitements d'analyse de données.

Chapitre 4

Implémentations

Nous abordons dans ce chapitre la description des artefacts logiciels que nous avons implémentés pour démontrer l'intérêt de nos propositions méthodologiques. Nous allons tout d'abord présenter l'architecture générale d'un nouveau WfMS intégratif orienté service élaboré pour la conception rapide et l'exécution des chaînes de traitements d'analyse intensive de données. Celui-ci agrège l'ensemble des fonctionnalités que nous avons implémentées. Nous exposerons les principales. Nous détaillerons la réalisation d'un moteur de workflows capable d'orchestrer les acteurs puis nous décrirons la couche de services utilisée au sein des composants. Nous présenterons alors les fonctionnalités rendant possible l'édition d'un DataFlow et l'encapsulation des lignes de commande. Les étapes de typage des données permettant d'exploiter le parallélisme de données seront également décrites ainsi que la génération des implémentations ciblant divers contextes d'exécution. Enfin, nous aborderons la mise en œuvre de l'association des spécifications de workflows au méta-modèle des composants et des plates-formes (CPMM) afin de faciliter la réutilisation et l'indépendance vis-à-vis des infrastructures de calcul.

4.1 Un nouveau WfMS intégratif

Dans cette première partie, nous allons présenter la nouvelle plate-forme logicielle dans laquelle nous avons intégré l'ensemble de nos développements. Ce WfMS intégratif constitue le cadre de la mise en œuvre de nos propositions méthodologiques.

4.1.1 Prototypage à partir de Kepler/Ptolemy

Dans un premier temps, nous avons intégré nos développements dans le WfMS Kepler [65]. Celui-ci exploite le moteur de workflows Ptolemy [123] qui permet le choix entre de multiples modèles de calcul (SDF, PN, BDF...). Nous avons opté pour le modèle SDF(cf. section 1.3.1) qui nous a permis de réaliser nos premiers tests d'orchestration. L'intégration d'acteurs dédiés rend possible l'exécution de workflows sur un cluster de calcul (cf. figure 4.1). Cependant le modèle SDF présente certaines limitations importantes. D'une part, Il n'offre qu'un degré minimal d'interactions avec nos acteurs. D'autre part,

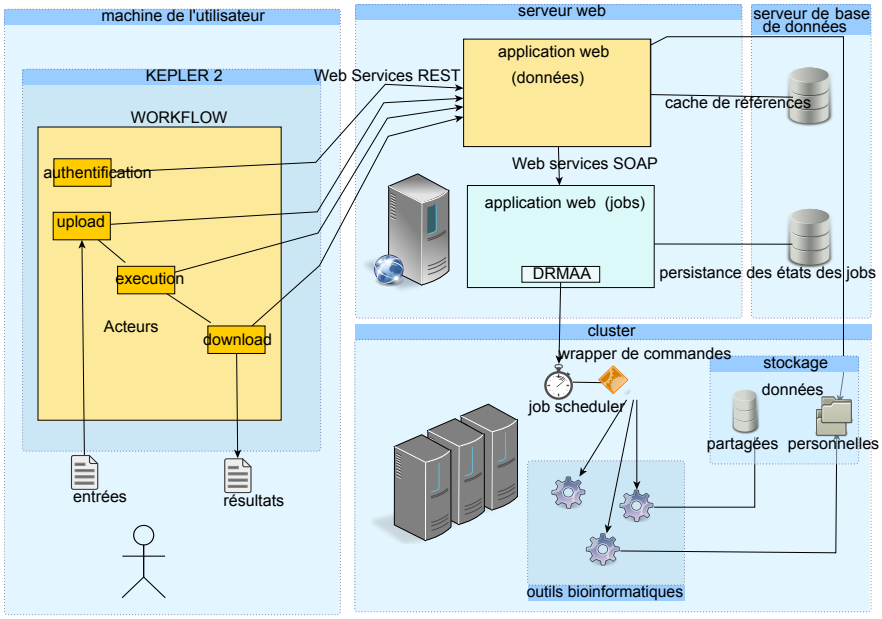


FIGURE 4.1: Architecture de la première version prototype de notre WfMS basée sur le client Kepler et le moteur Ptolemy et intégrant notre couche de services dédiée aux acteurs dénommée SLICEE.

tous les niveaux de parallélisme exprimables dans un DataFlow ne sont pas exploités. Par ailleurs, la modification de l'interface de conception de Kepler nécessite des modifications du code importantes qui sont particulièrement laborieuses. Finalement, nous constatons que, pour poursuivre l'implémentation telle que nous l'avons prévue, il est plus efficace pour nous, de coder à la fois un moteur de workflows dédié et un client de conception plutôt que de modifier en profondeur la plate-forme Kepler/Ptolemy.

4.1.2 Une architecture orientée service

Nous avons choisi de développer un nouveau WfMS intégratif. Son évolutivité doit être facilitée par le respect de certains principes de conception issus des Architectures Orientées Service. Nous avons tout d'abord divisé notre plate-forme logicielle en composants assurant chacun une fonctionnalité (cf. figure 4.2). Leur faible couplage est mis en œuvre par une communication inter-processus basée sur des services REST (cf. section 4.2.3) Pour cela, des connexions par HTTP sont gérées par une couche applicative qui fait intervenir des pools de connexions configurables et de la reprise d'interruptions afin de limiter les risques de perte de connexions et de faciliter la montée en charge. En effet, une des faiblesses potentielles de ce type d'architecture est son emploi massif de connexions réseau. Les messages échangés par HTTP sont des représentations d'instances de classes Java, sérialisées sous forme JSON ou XML. Cette sérialisation est effectuée de façon automatique par adjonction aux classes d'annotations java. De cette façon, le

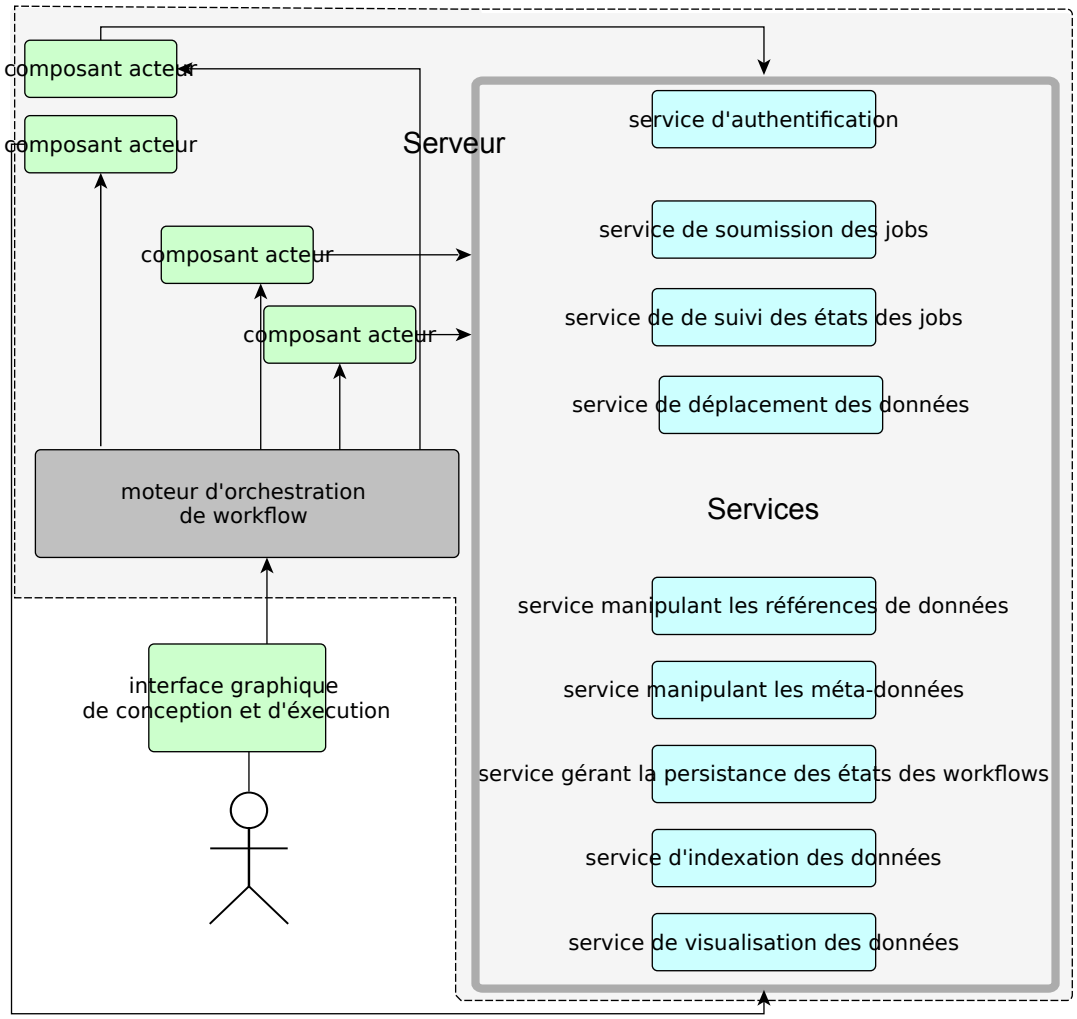


FIGURE 4.2: Architecture de notre plate-forme logicielle. Celle-ci est segmentée en grandes unités découplées, afin de garantir la modularité et l'évolutivité de notre WfMS suivant les principes des architectures orientées service.

coût de développement de nouvelles méthodes est largement allégé. L'API exposé par la partie serveur est exploitée par le biais d'un client, ici interne au WfMS mais qui peut potentiellement faire partie d'outils externes.

La modularité touche de multiples aspects de l'architecture du WfMS.

En premier lieu, l'orchestration est totalement découplée de l'implémentation des acteurs. De cette façon, le moteur peut s'adapter à tout acteur respectant une spécification générique. Réciproquement, il est possible de changer la logique d'orchestration en gardant les mêmes acteurs. Nous avons ainsi pu alternativement intégrer le moteur SDF de Ptolemy puis notre propre moteur basé sur le modèle UDFAS, comme nous le verrons ultérieurement.

En deuxième lieu, il y a également découplage entre les interfaces graphiques assurant le monitoring et la soumission des exécutions et le moteur de workflows. De cette façon, différents clients utilisant le même moteur peuvent être plus facilement développés. Ce choix d'architecture a facilité la migration du client lourd Kepler vers notre propre client web.

4.1.3 WfMS intégratif en mode SaaS

Nous développons un nouveau WfMS incluant l'interface, le moteur de workflows et des fonctionnalités annexes.

Tout d'abord, nous avons implémenté un outil en ligne de commande gérant l'orchestration des workflows par notre nouveau moteur. A ce stade du développement, il est alors envisageable de concevoir une nouvelle interface graphique offrant plus de liberté d'expérimentation. Il s'agit d'un client web [1] capable de contrôler l'exécution du moteur, et de rendre compte du suivi des exécutions. Nous intégrons également un éditeur graphique avancé permettant la conception d'un workflow et l'édition des propriétés des acteurs. Ainsi, un ensemble d'outils permet de faciliter la capture des lignes de commande des UDF ou la définition des ports et l'ajout de DFF. La création de formulaires de soumission de workflows ainsi que la plupart des opérations liées à la conception et à l'exécution sont configurables en ligne. L'usage d'un terminal n'est nécessaire que ponctuellement lors du débogage durant la création d'un composant. Des interfaces web sont également développées pour la visualisation des résultats et la recherche des données ainsi que l'édition de la configuration et des méta-données.

L'ensemble de la persistance dont la cache de références(cf. figure 3.3) et des états d'exécution est géré par une base de données [2]. Le mécanisme de persistance garantit nativement la provenance et facilite la reproductibilité. On peut ainsi ré-exécuter un

1. L'interface de ce client graphique léger est codée en HTML 5 et Javascript. La partie serveur est développée en Java en utilisant le framework MVC STRUTS sur un serveur Tomcat.

2. Un ORM (Object Relational Mapper) est utilisé. Il automatise le mapping entre les instances des classes JAVA manipulées et leur représentation en base de données. L'ORM est ici Hibernate, une librairie JAVA qui est capable de générer le SQL pour manipuler les jeux de données, les jobs, ou l'exécution des workflows. Différents Systèmes de Gestion de bases de données peuvent être utilisés.

workflow ou exploiter la base de données pour en extraire des statistiques d'utilisation. La gestion des méta-données est assurée par un outil d'indexation et de recherche(cf. section 4.2.2) afin de faciliter l'organisation et la sélection des données et des workflows. Nous obtenons à ce stade un environnement de conception et d'exécution fortement configurable associant gestion des workflows et accès aux données des utilisateurs.

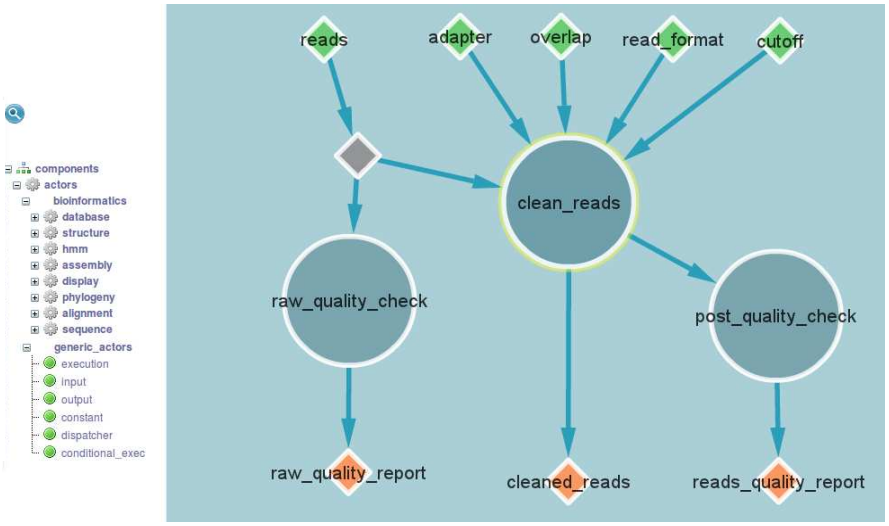


FIGURE 4.3: Interface Web de conception de workflows. Le DAG correspond à un workflow bioinformatique intensif capturé en utilisant l'éditeur graphique embarqué. Les nœuds supérieurs représentent les acteurs d'entrée, les nœuds inférieurs, les sorties. Les nœuds circulaires correspondent à l'exécution d'acteurs UDF qui encapsulent des lignes de commande. Chaque flèche relie un port de sortie vers un port d'entrée.

Une spécification DataFlow peut ainsi être entièrement construite à partir d'une interface web de conception (cf. figure 4.3). Une implémentation interprétable par le moteur de workflows peut ensuite être générée puis exécutée et monitorée (cf. figure 4.4). Finalement, toutes les fonctionnalités d'édition, de configuration, et d'exécution implémentées sont associées pour constituer un nouveau WfMS intégratif dédié à l'analyse de données intensive [124]. Nous le configurons en y intégrant des composants, des formats de données et des DFF associés au domaine de la bioinformatique.

4.2 Un moteur de workflows basé sur le Modèle de calcul UDFAS

Nous allons présenter dans cette partie le nouveau moteur de workflows adapté aux architectures orientées service qui est intégré à notre WfMS. Nous détaillons ici certaines caractéristiques de son implémentation, conforme à notre modèle de calcul UDFAS (URI based DataFlow for Asynchronous Services) décrit précédemment (cf. section 3.2.1).

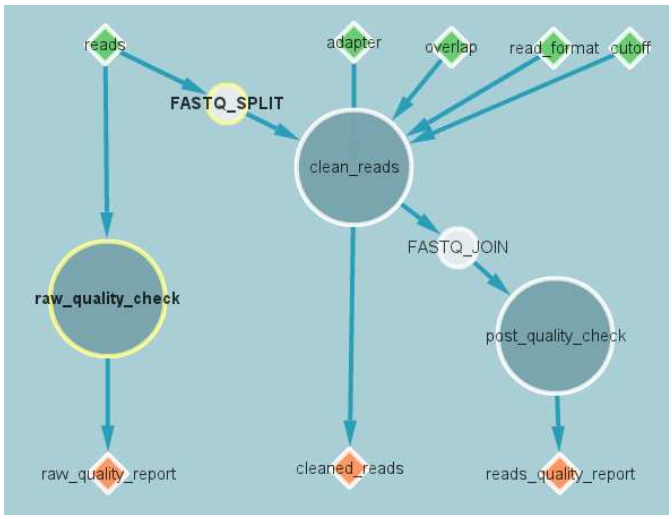


FIGURE 4.4: Interface web d'exécution de workflows. Le graphe correspond à l'exécution d'une instance d'un workflow bioinformatique intensif généré à partir d'un modèle de conception utilisant la transformation de graphe. Dans cet exemple, un patron *Split-Map-Merge* a été généré en utilisant les annotations fournies par l'utilisateur. L'état d'avancement des jobs est directement visualisable par la coloration du contour des nœuds. En cliquant sur un nœud, on accède au détail des exécutions associées.

4.2.1 La spécification DSURI

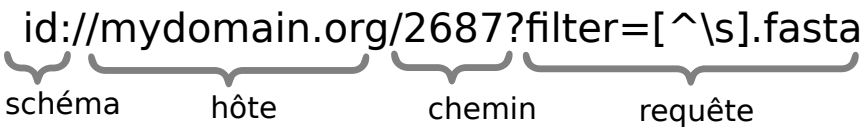


FIGURE 4.5: Jeton DSURI (DataSet Uniform Resource Identifier). Le MoC UDFAS consomme et produit des jetons *DSURI* qui référencent des groupes de fichiers ou de ressources à travers une URI. La syntaxe URI utilisée supporte différents schémas, qui spécifient le protocole d'accès aux données et l'hôte. La partie chemin de l'URI représente l'identifiant du jeu de données. La partie requête de l'URI permet l'extraction de sous-ensembles du jeu.

Nous avons défini les spécifications de la classe de jetons DSURI (DataSet Uniform Resource Identifier). Il s'agit de l'unique type de jetons consommés par notre modèle de calcul (cf. section 3.2.1). La forme abstraite du jeton DSURI est une référence vers une collection de données (cf. figure 4.5). Elle est relative à un *espace de nommage* défini par le domaine déclaré dans la partie *hôte* de l'URI. L'élément *schéma* détermine le mode d'interprétation de l'URI (abstrait, concret) et les propriétés associées à ses éléments syntaxiques *chemin* et *requête*(cf. figure 4.6).

— Le schéma *id* désigne des URIs exploitant le mécanisme générique des références. Un service dédié du WfMS permet la conversion des DSURI références ayant le

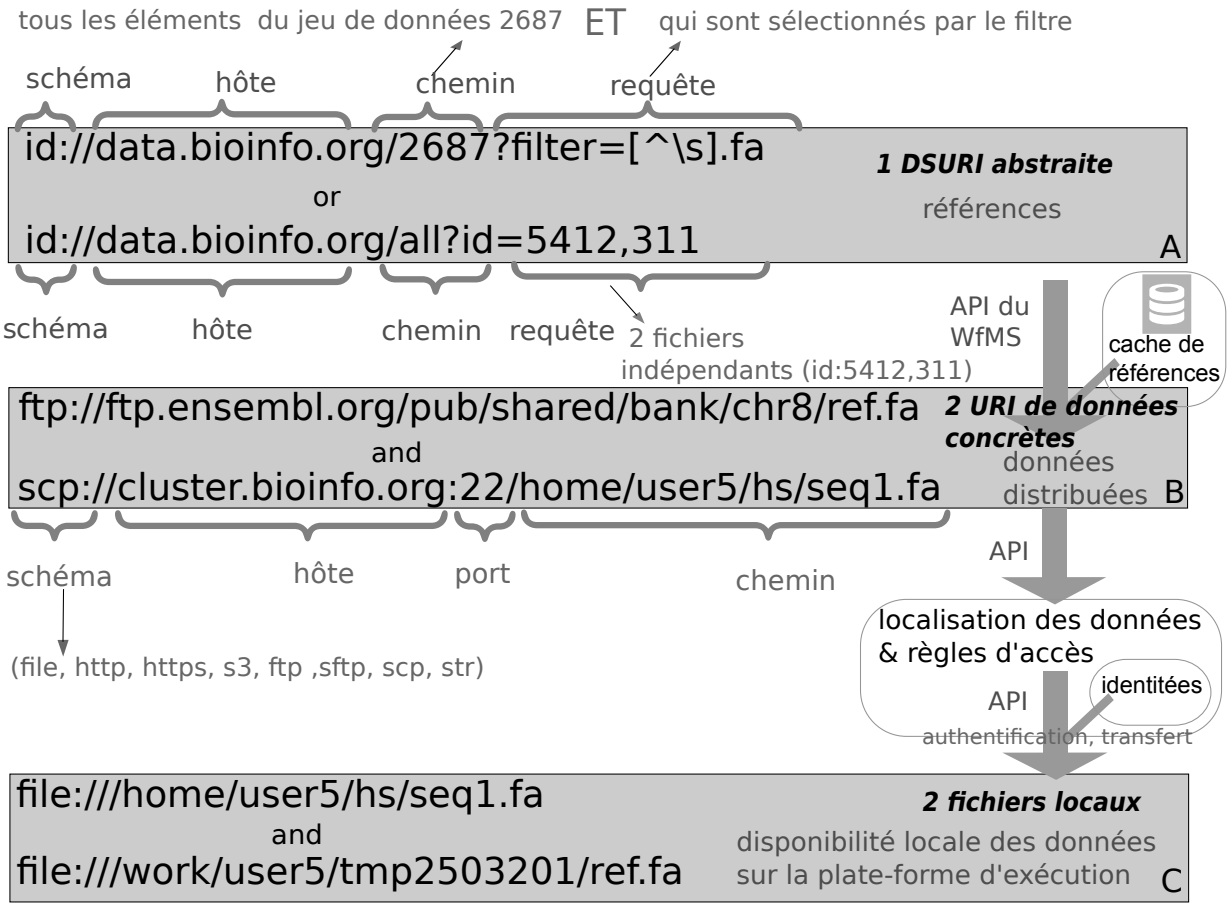


FIGURE 4.6: Détail de la syntaxe du jeton DSURI (DataSet Uniform Resource Identifier). Les différentes représentations (A, B, C) URI d'un même jeu de données sont utilisées lors des diverses étapes du cycle de vie du jeton. Elles sont ainsi exploitées par le GUI du WfMS manipulé par l'utilisateur (A, B), par le moteur d'exécution de workflows basé sur le MoC UDFAS (A) et par la plate-forme d'exécution (C).

schéma *id* en liste de DSURI concrètes qui exploitent une *cache de références* persistée. Ce mécanisme est destiné à faciliter la manipulation des jeux de données volumineux.

— Les schémas *http*, *https*, *s3*, *ftp*, *sftp*, *scp* désignent des URI concrètes qui décrivent explicitement le mode ou protocole d'accès aux données.

— Le schéma *file* désigne un mode d'accès local, utile notamment pour la gestion interne des traitements sur la plate-forme d'exécution.

— L'introduction du schéma *str* dans la spécification DSURI donne la possibilité de passage par valeur pour permettre le routage efficace des valeurs non volumineuses représentables sous la forme d'une courte chaîne de caractères, que l'on trouve communément associée à certains paramètres d'exécution de méthodes métiers.

Une DSURI NULL est représenté par la chaîne *null* ou par une DSURI d'un schéma quelconque n'ayant pas de données associées.

Par ailleurs, l'ajout du paramètre *filter* dans la partie *requête* de d'une DSURI (cf. figure 4.6) provoque le filtrage des éléments de la collection de données associées (cf. section 3.2.1). Ce principe est notamment utilisé lors de la conception des acteurs pour rediriger un seul fichier dans un port de sortie particulier. Pour cela, une expression régulière permettant de sélectionner les noms de fichiers est associée au paramètre *filter* de la DSURI qui transitera par ce port. Ceux sont les acteurs consommant ce jeton qui auront la charge de résoudre la nouvelle DSURI pour sélectionner le fichier.

4.2.2 La gestion distribuée des données et des exécutions

Nous présentons ici l'implémentation de la gestion distribuée des données et des exécutions dans le moteur. Dans le WfMS, les DSURI sont utilisées tout au long du cycle de vie de la donnée. Ainsi, la localisation des données est basée sur l'exploitation de cette spécification (cf. figure 4.5). L'accès aux fichiers de données est déterminé par des identités liées à l'utilisateur courant. Elles doivent être renseignées préalablement pour chaque hôte et protocole. Pour cela, l'utilisateur doit configurer son pool d'identités (cf. figure F.6 en annexe). Une fois cette opération effectuée, le système peut prendre en charge le rapatriement des données, juste avant l'exécution d'une commande rattachée à un acteur. Le fonctionnement du système de localisation des données suit plusieurs étapes. Tout d'abord, le serveur détermine la localisation de l'exécution à partir de la configuration choisie qui est définissable par l'utilisateur. Elle inclut l'adresse du frontal d'une plate-forme d'exécution. Des paramètres plus fins peuvent également être précisés comme le nom d'une queue de soumission, en fonction de l'outil d'ordonnancement associé à la plate-forme. Nous remarquerons que la configuration des acteurs exploite ici le méta-modèle CPMM (cf. section 3.3.2). Ensuite, lorsque c'est nécessaire, le serveur résoud les DSURI abstraites et les convertit en DSURI concrètes. L'usage des DSURI facilite la gestion des données distribuées. En effet, chacune des DSURI abstraites correspond à une liste de DSURI concrètes(cf. figure 4.6) précisant chacune la machine qui héberge les données, le protocole d'accès, le chemin et le nom du fichier (cf. figure 4.7).

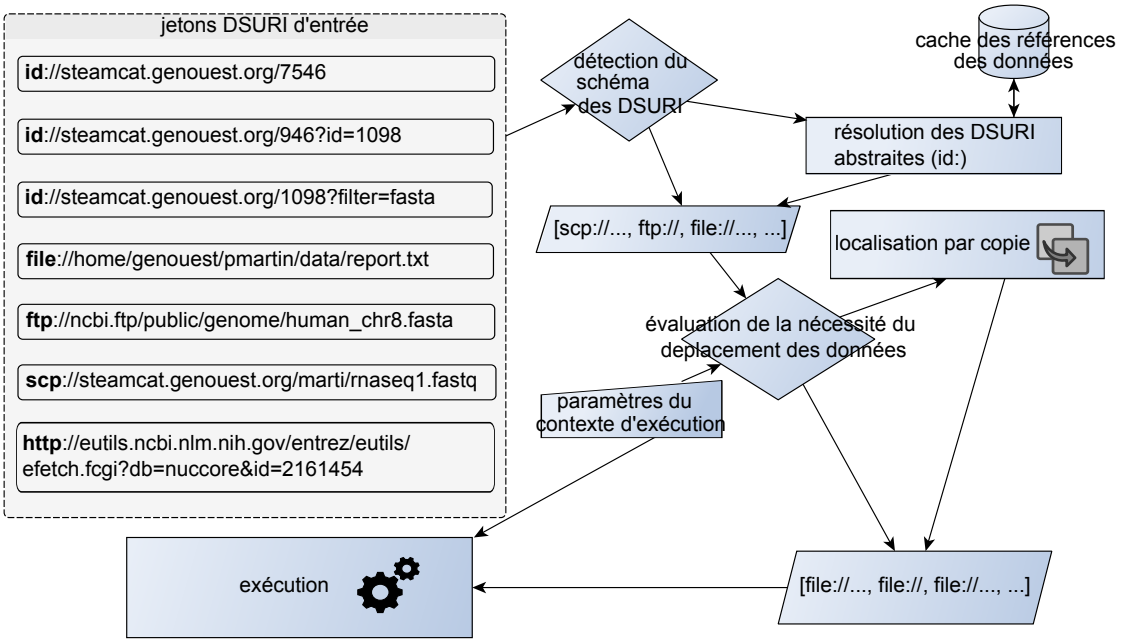


FIGURE 4.7: Diagramme présentant les étapes de gestion des DSURI permettant de préparer l'exécution d'une commande. Ce mécanisme inclut la résolution des DSURI aboutissant à leur conversion en listes de chemins fichiers. Lorsque les fichiers ne sont pas initialement disponibles sur la machine d'exécution, le système gère leur localisation. Dans ce cas, des paramètres d'authentification adaptés doivent être préalablement configurés dans le pool d'identités de l'utilisateur.

Le système vérifie alors que l'utilisateur courant possède une identité valide pour la machine qui héberge les données et pour le protocole d'accès précisé dans l'URI [1]. La même opération est effectuée pour déterminer l'existence d'une identité valide permettant d'accéder à la plate-forme d'exécution. L'accès depuis le serveur web vers les machines d'exécution est uniquement en scp et ssh.

Si l'ensemble de ces conditions n'est pas rempli, une exception est levée. Dans le cas où les données d'entrée sont distantes, le serveur les rapatrie sur la machine de calcul, dans un répertoire d'exécution dédié. Ce transfert peut faire intervenir la copie des données sur un hôte intermédiaire en fonction des contraintes d'authentification. Les copies inutiles sont évitées en exploitant un système de table d'équivalence de noms de machines. Le serveur génère ensuite la ligne de commande à partir des données d'entrée et d'un patron, puis soumet les jobs, suivant la configuration de l'acteur, directement ou via un outil d'ordonnancement. Le moteur de workflows permet donc la distribution des jobs entre différentes infrastructures distantes, chaque acteur ayant sa propre configuration. Cette flexibilité ouvre la voie à la gestion de nombreux cas particuliers.

Après le signal de fin d'exécution d'un workflow, le système réalise une série de post-traitements. Il met à jour les résultats dans la cache de références (cf. figure F.2 en annexe). Celle-ci est constituée de tables d'une base de données. Elle est utilisée pour retrouver les fichiers associés aux jeux de données lors de la résolution des DSURI abstraites. Ensuite, les jeux de données produits ou utilisés durant l'exécution sont classifiés en différents groupes (entrées, données provisoires, sorties d'acteurs, résultats des workflows). Ainsi les données considérées comme temporaires peuvent être supprimées afin de limiter les besoins en stockage et les résultats du workflow identifiés.

Finalement, les résultats du workflow sont associés automatiquement à certaines méta-données afin d'assurer leur traçabilité puis indexés dans un moteur de recherche interne [2]. Ainsi l'utilisateur pourra par la suite sélectionner des données suivant de multiples critères (format, extension, nom de fichier, workflow d'origine...). A la suite de ces étapes, les données produites deviennent disponibles. L'utilisateur peut alors les manipuler par le biais des interfaces du WfMS [3].

Nous avons donc vu que le système implémente de façon transparente un mécanisme de localisation de données hétérogènes et qui rend également possible l'usage de multiples localisations d'exécution, simultanément au sein d'un même workflow. On peut donc exécuter des traitements distribués sur différentes plates-formes. Ce mécanisme reste néanmoins minimal et ne permet pas la répartition automatique de la distribution, pour gérer dynamiquement la montée en charge ou la reprise sur erreur. Cependant, il peut

1. En l'état de l'implémentation actuelle, les données peuvent être lues ou écrites suivant plusieurs protocoles (copie local, scp, sftp, ftp, http)

2. Nous utilisons pour indexer les données et effectuer des requêtes de recherche l'application web SOLR qui expose des services REST dédié à a recherche *full text* exploitant le moteur de recherche Lucene.

3. Le WfMS possède des interfaces web dédiées permettant notamment la visualisation, le téléchargement ou l'annotation par ajout de méta-données.

être mis en pratique pour adapter une spécification de workflows à un environnement d'exécution particulier.

4.2.3 L'architecture des acteurs

Les services internes aux acteurs

Dans notre moteur de workflows, les acteurs jouent un rôle prépondérant. En effet, nous avons conçu des composants génériques qui assurent un ensemble de fonctionnalités essentielles:

— l'authentification;

— le routage des données;

— la soumission des jobs;

— la persistance;

— l'interception des erreurs.

Chaque instance d'acteur possède un état pouvant prendre des valeurs prédéfinies. Chaque transition entre deux états est associée à l'appel d'une méthode de signature générique (initialize() submit(), finalize()...). Ainsi dans le cas d'un acteur d'exécution, la transition entre l'état *submitted* et *done* déclenche la méthode submit() qui correspond à la soumission d'un job.

Par ailleurs, les acteurs utilisent un client interne communiquant avec un serveur en utilisant une API REST *Ad hoc* [1]. Le serveur dispose d'une base de données assurant notamment la persistance de l'ensemble des informations relatives aux exécutions, états des traitements [2] et relations entre données et jobs (cf. figure F.2 en annexe).

En fonction des tâches qu'ils ont accomplies, les acteurs mettent à jour leur état personnel. Cette mise à jour correspond au seul type d'événement qui est détectable par le moteur de workflows. Il y a donc une stricte séparation entre la couche acteur et la couche DataFlow. Le moteur peut alors réagir de façon adéquate et ordonner l'exécution de la méthode associée à la transition suivante ou lever une exception.

Nous avons défini quatre classes principales d'acteurs: entrée, sortie, exécution et composite.

— Tout d'abord, l'acteur d'entrée gère la réception des jetons de données et assure leur normalisation et leur conversion en références abstraites de type *DSURI*(cf. section 3.2.1). Grâce à cette étape, durant l'exécution du DataFlow, le cycle de consommation et de production des jetons est simplifié.

— L'acteur de sortie, quant à lui, se charge d'identifier les résultats du workflow. Lorsque tous les acteurs de sortie ont atteint l'état "finalized", le moteur informe

1. Pour définir notre API, nous utilisons Jersey, une implémentation de la spécification JAX-RS dédiée à la création de Services Web REST (Representational State Transfer). La communication est basée sur HTTP et HTTPS.

2. Les états d'exécution des commandes (run, done, error...) (cf. section 3.2.1) déterminent indirectement les états des acteurs d'exécution

A

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
 <commandConfig>
     <cmdPattern>
     python computeEntropy.py -i $fasta \
     -o ${fa.basename()}_stat.txt
     </cmdPattern>
     <datasetEntrys>fasta</datasetEntrys>
     <dsIds>{id://steamcat.genouest.org/2616}</dsIds>
     <dsIds>{id://steamcat.genouest.org/2617}</dsIds>
     <dsIds>{id://steamcat.genouest.org/2618}</dsIds>
     <loopModeMap>
         <entry>
             <key>fasta</key>
             <value>0</value>
         </entry>
     </loopModeMap>
     <queuId>sgi144.q</queuId>
     <serviceURL>ssh://genocluster2.genouest.org</serviceURL>
 </commandConfig>
```

B

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<asyncJobResult>
    <error></error>
    <finished>false</finished>
    <message>all jobs not finished</message>
    <outputDataSetID>2564</outputDataSetID>
    <size_done>2</size_done>
    <size_total>3</size_total>
    <status>RUN</status>
</asyncJobResult>
```

FIGURE 4.8: API REST d'exécution de jobs

Représentation XML d'un message de demande d'exécution de jobs soumis au serveur par le client interne à un acteur d'exécution (A) et d'un message d'état d'avancement retourné par le serveur périodiquement durant l'attente de la fin des traitements (B).

le serveur de la fin de l'exécution du workflow. Celui-ci peut alors effectuer la mise à jour de la base de données et l'exécution de post-traitements [1].

— L'acteur d'exécution gère la génération des lignes de commande construites à partir d'un patron et des données associées aux jetons d'entrée (cf. figure 4.8 - A). Il transmet ensuite des ordres de soumission au serveur, qui communique directement avec l'infrastructure de calcul suivant un patron asynchrone (cf. figure 2.3). La gestion de l'avancement des traitements est également déléguée au serveur (cf. figure 4.8 - B). En fonction de ces informations, l'acteur met à jour régulièrement son propre état dans la table centrale des états du workflow pour permettre au moteur de réagir à cet événement.

— Enfin, l'acteur composite gère la soumission d'un workflow imbriqué dans un autre workflow. Il est implémenté de telle sorte à reproduire le comportement d'un acteur d'exécution standard vis-à-vis du moteur de workflows. Nous le détaillerons ultérieurement.

La couche de services dédiée aux acteurs est appelée SLICEE (Service Layer for Intensive Computation Execution Environment) [125]. Elle forme un middleware indépendant, compatible avec notre spécification DSURI (cf. figure 4.5). SLICEE gère le monitoring des jobs, l'authentification des utilisateurs, le déplacement des données et la soumission des jobs en permettant d'exploiter différents types d'ordonnanceur (SGE[126], TORQUE[127], SLURM, OAR ...). Les acteurs sont codés en Java tout comme la partie serveur, gérée par TOMCAT.

4.3 Design rapide de workflows

4.3.1 De la ligne de commande à l'acteur

Dans cette partie, nous allons exposer en détail l'implémentation du processus de capture d'une ligne de commande et de son encapsulation dans un acteur. La génération de la ligne de commande a nécessité plusieurs versions.

La première implémentation utilise un système de parser/lexer. Elle s'est avérée complexe à la mise en œuvre et difficile à faire évoluer lors de nos tests. Nous avons donc opté pour une nouvelle version plus extensible qui utilise un langage de template, Velocity. (cf. figure 4.9). Velocity permet d'expliciter une ligne de commande sous forme de patron possédant des variables associées à un contexte d'évaluation. Celles-ci correspondent ici en particulier aux DSURI des ports d'entrée de l'acteur. Nous avons implémenté ces variables sous la forme d'instances de classe Java possédant des méthodes utilitaires dédiées à l'intégration. Leur utilisation peut concerner, par exemple, la définition des noms de fichiers de sortie, la localisation des données temporaires ou la manipulation des méta-données. Le concepteur peut ainsi simplifier l'intégration en limitant le recours à des wrappers(cf. figure F.7 en annexe).

1. Les post-traitements sont répartis entre post-traitements système (nettoyage des données, indexation par ajout de méta-données) et post-traitements définis par l'utilisateur (envoi d'email, déclenchement d'un autre workflow, déplacement de données...).

A

```
### fastq reads cleaner
$debug token:{adapter:$adapter,format:$format,
$debug reads:$reads,cutoff:$cutoff,overlap:$overlap}
$debug
#if($cutoff==’’)
   #set($cutoff=’20’)
#end
flexbar.sh  --adapters $adapter --format $format \
    --source $reads \
    --target ${reads.basename()}_cleaned --cut-off $cutoff \
    --min-overlap $overlap --trim-end any   ${STDERR.grep("Inc")}
```

B

```
####  token:{adapter:/work/adapt.fa,format:sanger,
####         reads:/data/set1.fastq,cutoff:,overlap:4}
####
 flexbar.sh  --adapters /work/adapt.fa --format sanger \
     --source /data/set1.fastq \
     --target set1_cleaned --cut-off 20 \
     --min-overlap 4 --trim-end any   \
     2> xtmpErr.txt ;cat xtmpErr.txt | \
     egrep -i ’Inc’  >&2 ;rm xtmpErr.txt
```

FIGURE 4.9: Patron de lignes de commande

Patron de lignes de commande (A) et exemple de commandes générées (B). La syntaxe du patron est basée sur le langage Velocity. Les variables sont des instances de classes possédant des méthodes utilitaires conçues pour limiter l’usage des wrappers. Les lignes de commande générées sont interprétables par le Bash.

Ces objets sont ajoutés au contexte du template [1] puis, après évaluation du patron, la chaîne de caractères correspondant aux commandes est générée. Ce mécanisme, associé à la prise en charge native des conditions, des itérateurs et des collections par Velocity, rend possible la gestion de nombreux cas particuliers complexes. L'ensemble de la logique de génération des commandes à partir des jetons de données peut ainsi être défini par l'écriture d'un patron dans une syntaxe unifiée et extensible. La procédure d'intégration d'une ligne de commande simplifie donc la conception de nouveaux acteurs UDF. Un mécanisme similaire est mis en œuvre dans plusieurs WfMS étudiés comme Galaxy(cf. section 2.1.2) mais notre implémentation est enrichie par la mise à disposition des méthodes utilitaires simplifiant l'intégration et la possibilité pour l'utilisateur final d'accéder à ces mécanismes via l'interface.

4.3.2 Génération du parallélisme de données

Après avoir traité de la conception rapide d'acteurs d'exécution, nous abordons la description des outils permettant la génération du parallélisme de données. Nous avons dans un premier temps extrait une hiérarchie des formats de données à partir de l'ontologie EDAM puis ceux-ci ont été stockés en base de données [2]. Pour chacun des principaux formats, un jeu de méthodes DFF a été prédéfini dans le système (cf. figure F.4 en annexe). Les différentes méthodes DFF sont classées suivant des rôles techniques comme la validation, la conversion de format ou le découpage et la fusion des paquets de données. Chaque définition de DFF inclut un patron de commande référençant un outil en ligne de commande qui doit être déployé préalablement sur les machines de calcul. En exploitant la hiérarchie des formats mis à disposition dans l'environnement, l'utilisateur peut effectuer un typage partiel ou total des ports des acteurs. Une fois les ports typés, il peut leur associer des séquences de méthodes DFF. La liste des DFF disponibles pour un format donné intègre les méthodes des formats hérités. Ainsi la liste disponible pour le format Fasta comprend les méthodes de ce format mais aussi des fichiers texte générique. L'ensemble de ce processus d'édition, appelé ici *annotation*, est réalisé dans l'interface web par des formulaires contextuels spécifiques. L'architecture choisie simplifie la configuration. Elle est extensible et rend possible l'adaptation du WfMS à tout format de données, grâce à son méta-modèle.

Lors de la phase de génération des implémentations, les méthodes DFF sont injectées dans le graphe du DataFlow d'après la logique suivante. Tout d'abord, les commandes relatives aux DFF sont générées à partir du patron de commande stocké en base et de son paramétrage. Puis, pour chaque DFF, une instance d'acteur d'exécution est peuplée avec le patron de commande correspondant. Le modèle est alors validé et le système procède alors à l'injection des nouveaux acteurs DFF lors d'un processus de transformation de modèles débouchant sur la réécriture du graphe DataFlow. C'est lors de cette étape que

1. Un tableau clef/valeur.
2. Les données d'EDAM sont réparties dans une base de données relationnelle et dans une base NOSQL dédiées aux graphes, NEO4J, afin de faciliter la navigation dans l'arborescence des termes.

sont générés les patrons de conception *Split-Map-Merge* capables d'exprimer du parallélisme de données massif (cf. section 3.2.2). Le nouveau graphe est ensuite stocké mais n'est pas directement accessible à l'utilisateur: il constitue la spécification interprétable par le moteur de workflows.

4.4 Transformation de modèles et adaptation aux contextes d'exécution

Nous abordons maintenant la description d'artefacts visant l'adaptation d'une spécification de workflow à différents contextes d'exécution. Ces implémentations mettent en évidence la capacité de notre approche orientée modèle à garantir l'indépendance entre spécification et implémentation.

4.4.1 Une spécification, plusieurs implémentations

Tout d'abord, nous avons implémenté un processus de transformation de modèles configurable permettant, à partir d'un unique modèle de workflow, la production automatique de différents types d'implémentations. Ainsi, à partir d'une spécification SCDFM nous sommes capables de générer des implémentations non seulement *haut-débit* ou *bas-débit* environnées, mais aussi des scripts de chaînes de traitements, indépendants de l'environnement de conception.

La spécification haut-débit correspond à l'implémentation intégrant les DFF et est interprétable par notre moteur de workflows. Elle est adaptée à l'analyse des données massives.

La spécification bas-débit est simplement obtenue en éliminant les DFF lors de la transformation de modèles et en lui associant une plate-forme d'exécution minimale [1]. Elle est interprétable par le moteur de workflows et peut être utile au test ou au traitement des petits jeux de données, voire à un usage pédagogique.

La spécification indépendante est obtenue par génération de code. Nous produisons ainsi une implémentation dans un langage *general purpose* permettant de garantir la portabilité et la capacité à raffiner la spécification en dehors du WfMS. Nous avons choisi pour nos tests les langages PERL et JAVA. Le code généré comporte un "main" utilisant des fonctions d'ordonnancement prédéfinies, et des scripts associés dans le cas des acteurs déployables.

La figure 4.10 décrit les étapes des différents processus de transformation de modèles.

4.4.2 CPMM et personnalisation pour le PaaS

Implémentation du modèle des Composants et des Plates-formes

Nous voulons limiter la dépendance du concepteur vis-à-vis des médiateurs et finalement lui permettre d'être le seul intervenant, du prototypage jusqu'à la mise en production. Pour cela, nous avons intégré la gestion de la configuration et du déploiement des

1. L'hôte associé à chaque acteur est dans ce cas, une machine unique sans ordonnanceur

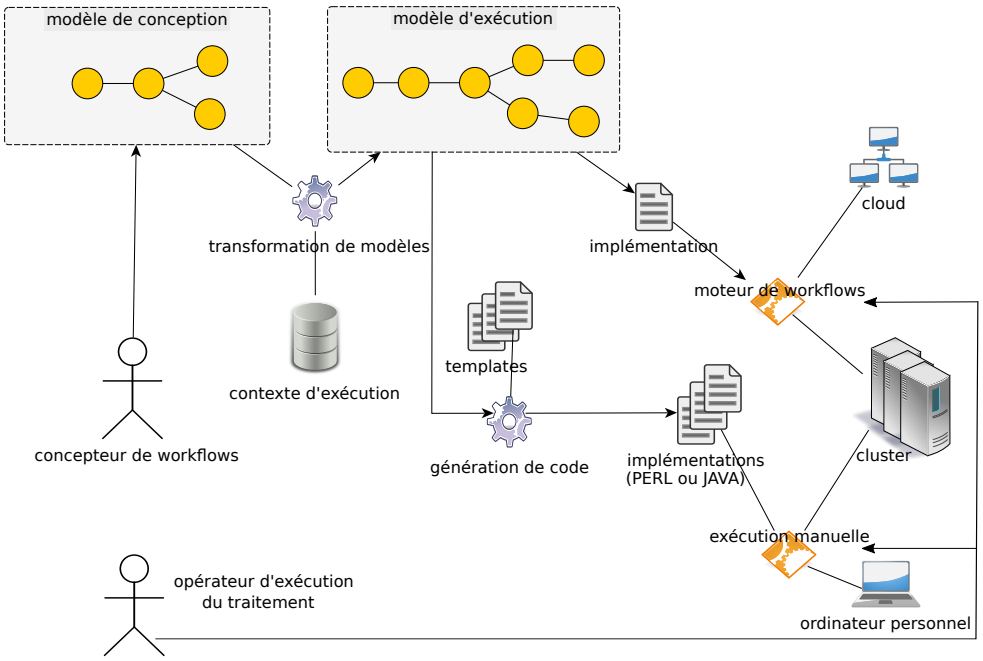


FIGURE 4.10: Etapes de la génération d'implémentations multiples suivant les plates-formes cibles à partir d'une seule spécification de modèle de workflow semi-concret capturées par le concepteur.

composants d'exécution au système. Il s'agit d'éléments des spécifications techniques des plates-formes d'exécution qui sont regroupés dans le CPMM (cf. section 3.3.2) et stockés en base de données. Les spécifications de workflows SCDFM sont implémentées indépendamment du CPMM. C'est lors du processus de transformation de modèles que sont intégrées les caractéristiques de l'environnement d'exécution afin de produire une implémentation dédiée à un contexte particulier.

Configuration des environnments à partir du SHELL

Nous commençons par présenter une exploitation simple du CPMM en employant un mécanisme de gestion basé sur les variables d'environnement. Il s'agit d'une méthode de gestion des configurations et non du déploiement. Chaque plate-forme est décrite par un ensemble d'informations stockées en base de données, suivant un schéma générique (cf. figure F.3 en annexe). Les propriétés de chaque cluster cible doit y figurer ainsi que l'ensemble des scripts de configuration des outils métiers.

A

```
### fastq validator
$debug host:$host
$ENV('fastqutils')
validatefastq.sh -f $format $reads
```

B

```
####  host:snp.genotoul.inra.fr
PATH=$PATH:/usr/bin/fastq_utils/
validatefastq.sh -f sanger  /data/readset2.fastq
```

C

```
####  host:genocluster2.genouest.org
source /local/env/envfastq_utils.sh
validatefastq.sh -f sanger  /data/readset2.fastq
```

FIGURE 4.11: Usage du CPMM pour la configuration multi-plate-formes des lignes de commande A l'aide d'une interface, le concepteur de composants associe tout d'abord le code de l'acteur UDF à un environnement (A). Le système injecte avant l'exécution le code correspondant à la configuration du composant, spécifiquement à chaque plate-forme d'exécution (B, C).

Ensuite, le concepteur préfixe le patron de commande de l'acteur d'exécution par une directive *ENV* (cf. figure 4.11, A) qui précise le nom d'un environnement. Celui-ci référence un script, différent pour chaque plate-forme d'exécution, qui définit les variables d'environnement nécessaires aux appels des commandes (cf. figure 4.11, B et C). Un pré-traitement du patron de commande gère cette fonctionnalité. Nous obtenons ainsi le découplage entre spécification de workflows SCDFM et plates-formes d'exécution décrites dans le CPMM.

Intégration du déploiement de scripts au contexte d'exécution

Nous avons intégré dans l'éditeur de composants la capacité de définir directement le code d'un script dans différents langages interprétables. Le code est directement associé à un patron de commande auquel a été ajoutée une directive d'appel du script. Ces informations sont stockées en base de données en tant qu'élément du CPMM (cf. figure F.3 en annexe). Avant l'exécution d'un job, lorsque l'actualisation du code est nécessaire, un pré-traitement prend en charge le déploiement du script sur le nœud de calcul. Ce mécanisme d'*acteurs déployables* est conçu pour accélérer le prototypage des composants UDF. L'autonomie du concepteur est ainsi accrue lorsqu'il exploite une infrastructure contenant déjà des langages de script et librairies associées (cluster, PaaS). Néanmoins, le déploiement des binaires et l'installation des librairies restent ici une tâche manuelle nécessitant un médiateur.

4.4.3 Exploitation du modèle Containers as a Service

Intégration d'un gestionnaire de containers

Afin d'augmenter la capacité de personnalisation des traitements et l'autonomie du concepteur, nous avons implémenté un autre mode d'exploitation du CPMM visant l'automatisation complète du déploiement. Le concepteur de workflows acquièrt une plus grande autonomie grâce l'intégration d'un gestionnaire de containers dans le WfMS. Ainsi, les commandes des UDF peuvent être exécutées au sein d'un container Docker choisi par le concepteur. De cette façon, chaque acteur UDF peut bénéficier d'un contexte d'exécution logiciel personnalisé et isolé, contenant des librairies et binaires *Ad hoc*. Ici, les containers remplacent des packages de déploiement linux avec un niveau de généricité supérieur. Ils simplifient la personnalisation et la diffusion de nouvelles implémentations. En effet l'absence d'un médiateur permet la création d'environnements logiciels à façon, directement par l'utilisateur du WfMS et sans risque pour l'infrastructure grâce au principe d'isolation.

BioShaDock

Dans un premier temps, nous avons initié la création d'un répertoire d'images Docker ou *registry* dédié à la communauté disciplinaire (cf. figure F.8 en annexe) [1]. Ce répertoire nommé BioShaDock est consultable par une interface web. Les utilisateurs peuvent

1. BioShaDock a été élaboré avec l'aide d'Olivier Sallou de Genouest et la contribution de Marie Gros jean d'IFB Core

rechercher et télécharger des images préexistantes puis les utiliser et les modifier. Ils peuvent également déposer leurs propres images. Du point de vue de l'architecture, il s'agit d'un répertoire de composants applicatifs non couplés exploitables en tant que service dans de multiples contextes applicatifs. Il est possible d'utiliser les composants encapsulés dans différents environnements comme des WfMS ou même par ligne de commande. Nous utiliserons ce répertoire de containers pour expérimenter la mise en œuvre du modèle de workflow A-SCDFM.

Spécification A-SCDFM

Dans un deuxième temps, nous avons intégré dans notre WfMS un mécanisme de couplage entre la définition du workflow (SCDFM) et le contexte d'exécution (CPMM) à base de containers. Celui-ci se traduit par l'ajout de la directive $IMAGE\_URI$ aux acteurs. Cet attribut contient l'URI d'une image Docker contenant le contexte logiciel complet et portable nécessaire à l'exécution du composant. L'URI est suffisante pour identifier de façon unique l'image et permet son téléchargement. Juste avant l'exécution, la directive $IMAGE\_URI$ est analysée et les scripts sont configurés pour initialiser le container et exécuter la commande dans celui-ci. L'appel au gestionnaire de container (Docker) est effectué de telle sorte que les données soient sur l'hôte. Ainsi les librairies et exécutables nécessaires seront disponibles le temps de l'exécution et pour ce seul processus, garantissant une capacité de personnalisation totale sur toute infrastructure de type CaaS (cf. section 2.6.1). Un modèle de workflow devient ainsi une spécification complète autonome A-SCDFM [1](cf. figure 3.16). Elle associe définition du flot de données et des acteurs comme dans SCDFM mais inclut aussi leurs dépendances logicielles portables. La chaîne de traitements est ainsi complètement déployable quelle que soit l'infrastructure d'exécution CaaS. D'une part, un concepteur capable de capturer un DataFlow et de produire des containers devient donc autonome dans la phase de déploiement. D'autre part, il peut directement contribuer à un répertoire de workflows partagés, si celui-ci exploite une spécification comme A-SCFDM.

4.5 Résumé

Dans ce chapitre, nous avons exposé les implémentations que nous avons conçues pour évaluer et mettre en œuvre nos propositions méthodologiques.

Tout d'abord, nous avons décrit comment a été élaboré un nouveau WfMS intégratif dédié aux chaînes de traitements d'analyse intensive de données. C'est dans celui-ci que sont intégrés tous les artefacts que nous avons produits. Cet environnement logiciel est conçu suivant une architecture orientée service, particulièrement évolutive que nous avons présentée.

Nous avons ensuite décrit la conception d'un moteur de workflows basé sur le modèle de calcul UDFAS. Sa capacité au parallélisme et son architecture modulable en font un outil d'orchestration DataFlow efficace, adapté aux chaînes de traitements d'analyse

1. Le modèle A-SCDFM peut être converti en représentation XML. Dans ce cas, IMAGE_URI devient un attribut de commandPattern.

intensive de données. Nous abordons alors en détail la description de la spécification DSURI et son emploi par le moteur pour la gestion des données. Puis nous exposons comment est conçue la couche de services interne aux acteurs, qui masque la complexité de la soumission des jobs, de l'authentification distribuée et du routage des données au niveau du DataFlow. En effet, ce développement à été nécessaire car les middlewares actuels qui gèrent la soumission aux ordonnanceurs comme Opal2 [79] ne disposent pas de fonctionnalités d'authentification et de localisation des données.

Les outils permettant la capture de scripts et de lignes de commande ont ensuite été présentés. L'ensemble des fonctionnalités permettant la conversion d'un prototype en workflow a été produit mais nous avons choisi de ne présenter l'implémentation que des plus représentatifs de notre approche. Il s'agit des mécanismes (i) d'annotation des acteurs UDF, (ii) d'injection des DFF débouchant sur l'expression du parallélisme de données, (iii) de la génération d'implémentations compatibles avec notre moteur de workflow. Nous avons également implémenté la génération de code et nous pensons que cette fonctionnalité pourrait intéresser le public des utilisateurs techniques que nous visons ou des usagers ne disposant pas d'un accès à un WfMS. Nous pourrions ainsi élargir les usages potentiels des spécifications de workflows. La génération de code est déjà intégré au WfMS Wings [81, 69], mais nous ajoutons ici la possibilité de produire différents langages *general purpose*.

D'autre part, nous avons présenté l'intégration du couplage entre la spécification de workflows SCDFM et le modèle CPMM. Elle permet de mettre en œuvre la simplification du déploiement et la ré-utilisation des workflows pour différents types d'infrastructures: clusters, PaaS et CaaS. Notre implémentation du recours aux environnements Shell est une solution technique concrète pour la gestion de la configuration des workflows afin de permettre une exécution multi-plates-formes pour les clusters et le PaaS. Cependant les mécanismes de déploiement adapté aux architectures CaaS que nous décrivons sont bien plus prometteurs. Ils rendent opérationnelles la personnalisation et la gestion communautaire de la connaissance des processus d'analyse de données du domaine applicatif, caractéristique d'une approche Open Science. Des exemples concrets de la mise en œuvre de notre nouvel environnement logiciel en bioinformatique feront l'objet du chapitre suivant.

Chapitre 5

Résultats

Dans ce chapitre nous allons montrer l'intérêt de la mise en œuvre de nos propositions méthodologiques. Cette évaluation est rendue possible par la réalisation du nouveau WfMS présenté dans le chapitre précédent. Nous allons décrire en quoi l'usage de cet environnement logiciel simplifie la conception, le déploiement et l'exécution des workflows d'analyse de données. Nous débuterons par l'évaluation des performances des implémentations des chaînes de traitements produites. Nous exposerons ensuite l'exploitation de notre WfMS dans des cas concrets d'utilisation issus du domaine de la bioinformatique. Ces exemples illustreront les capacités de l'environnement que nous avons conçu dans l'intégration, l'aide à la conception et le traitement des grandes masses de données. Le gain d'autonomie pour le concepteur dans les phases de développement, de déploiement et de mise à disposition sera également mis en évidence.

5.1 Evaluation des performances

Evalutation des performances du moteur de workflows du WfMS

Dans cette section, nous nous concentrons ici uniquement sur l'évaluation des performances de notre moteur de workflows. Les WfMS permettent de bénéficier de processus d'analyse de données ayant une maintenabilité et une ré-utilisabilité accrues mais qui ont également des désavantages. En effet, notre moteur est conçu suivant une architecture orientée service et utilise pour cela de multiples couches logicielles qui entraînent potentiellement un surcoût. Nous allons évaluer l'impact de cette architecture sur les performances. C'est un critère important dans le choix d'utiliser un WfMS mais nous discuterons de la prise en compte d'autres paramètres comme le gain de vitesse de conception et de facilité de déploiement.

Pour commencer, nous choisissons un workflow de test permettant de réaliser notre benchmark. Celui-ci est dérivé d'un cas d'utilisation réel. Il correspond à une chaîne de traitements de nettoyage des lectures de séquences NGS usuelle. Celle-ci est constituée de 3 étapes. La qualité des séquences est tout d'abord déterminée en utilisant le logiciel FASTQC [128] puis les séquences des adapteurs sont détectées et supprimées avec l'outil Flexbar [129]. Enfin, la qualité des séquences résultant du traitement est évaluée

(FASTQC). Chacune de ces étapes appelle des lignes de commande correspondant aux outils métiers. Nous définissons 2 conditions de conception:

— dans la condition manuelle (M) la chaîne de traitements est codée manuellement sous la forme de scripts;

— dans la condition d'utilisation du WfMS (W), une implémentation issue d'un modèle capturé et annoté au sein de notre environnement est exécutée.

Dans ces deux conditions, la même logique d'ordonnancement des tâches est implémentée. Nous créons 4 workflows fonctionnels. Dans chaque condition M et W, nous concevons 2 versions du workflow, avec (P) ou sans (N) patron de parallélisme de données. Les deux implémentations de la même condition, respectivement [PM, PW] et [NM, NW], produisent le même nombre de jobs. Les deux jeux de données d'entrée tests sont constitués de fichiers volumineux issus de séquenceurs Roche 454 et Illumina Hi-Seq, dans le format FASTQ (cf. figure I.2 en annexe). A partir du jeu de données original, nous générons des sous-ensembles de différentes tailles que nous utilisons comme entrées de notre workflow de test.

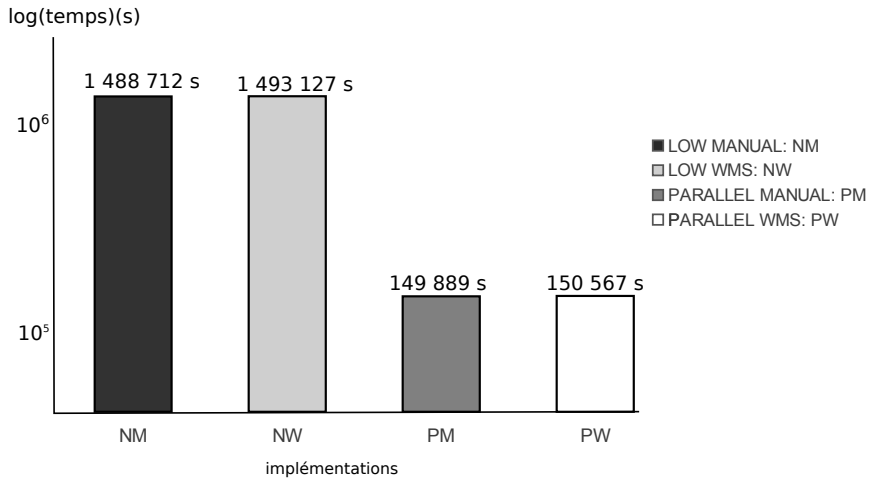


FIGURE 5.1: Comparaison des temps d'exécution entre implémentations parallèles (PW, PM) et non parallèles (NW, NM) d'un même modèle de workflow, avec un jeu de données d'entrée de 10 Go, réalisé sur 10 cœurs répartis sur deux machines (CPU: 8 x 2.73 GHz, RAM:64 Go, OS: Linux - Centos 6), via l'ordonnanceur Sun Grid Engine. Toutes les implémentations soumettent des jobs en utilisant le même ordonnanceur sur la même queue du cluster, correspondant à des nœuds identiques. Sur les nœuds du cluster est monté un répertoire de données partagées.

Nous comparons le temps d'exécution de l'ensemble du workflow. Les deux implémentations parallèles (PM, PW) produisent des performances comparables. Comme attendu, elles délivrent des résultats plus rapidement que les non parallèles (cf. figure 5.1). Ce résultat démontre l'intérêt de notre mécanisme d'aide à l'intégration du parallélisme de données dans les workflows.

taille du fichier d'entrée	PW(s)	PM(s)	PW/PM
250K	146	52	2,81
1000K	171	113	1,51
10M	300	234	1,28
3G	46714	43385	1,08
10G	150567	149889	1,005

TABLE 5.1: Performances des implémentations parallèles produites par le WfMS Comparaison du temps d'exécution entre deux implémentations parallèles, l'une générée et exécutée par le WfMS (PW) et l'autre codée manuellement (PM).

Par ailleurs, nous remarquons qu'avec un jeu de données correspondant à la plus petite taille testée (250Ko), nous obtenons, en utilisant le moteur de workflow, un temps d'exécution qui correspond à 281% du temps obtenu pour la version parallèle codée manuellement (cf. tableau 5.1). Avec un jeu de données correspondant à la plus grande taille testée (10 Go), nous obtenons un temps d'exécution presque équivalent (100.5%) à celui de la version codée manuellement. La différence de performances entre le WfMS (PW) et l'implémentation codée manuellement (PM) s'atténue quand la taille du jeu d'entrée augmente. Ainsi, comme attendu, une instance d'un modèle de workflow interprétée par notre moteur de workflows s'exécute plus lentement qu'une version codée manuellement, probablement à cause du coût des multiples couches applicatives et fonctionnalités additionnelles du WfMS [1]. Nous remarquons que le surcoût lié à l'utilisation du moteur de workflows est très limité dès lors que les jeux de données d'entrée sont de taille importante. Ce résultat confirme que l'implémentation de notre moteur est adaptée à un traitement efficace des grands jeux de données. De façon générale, nous pensons que le coût d'exécution d'un processus au sein d'un environnement associant conception et exécution doit être analysé au regard des gains globaux de productivité obtenus. Par ailleurs, notre WfMS (cf. section 4.4.1) permet à l'utilisateur de choisir entre l'exécution par le moteur de workflows ou la génération de code produisant des scripts indépendants [2].

1. Nous rappelons qu'en dehors de la soumission des jobs proprement dite, le WfMS effectue de nombreuses autres tâches techniques durant l'orchestration d'un DataFlow. Il s'agit, par exemple, des pré-traitements et post-traitements des données, du monitoring, de la persistance.

2. La fonctionnalité de génération de code produit des implémentations se rapprochant du codage manuel et donc de ses performances. Cependant, il faut noter que, même si la génération d'une chaîne de traitements sous forme de scripts garantit une exécution plus rapide pour les petits jeux de données, elle ne permet pas de bénéficier de l'ensemble des fonctionnalités implémentées dans le WfMS, comme le nettoyage et l'indexation des données ou la reprise sur erreur.

5.2 Illustrations

Nous décrivons ici deux cas d'utilisation de notre WfMS dans le cadre de projets scientifiques nécessitant la mise en œuvre et l'exploitation de chaînes de traitements d'analyse de données.
Ces illustrations, PrimerFinder et Orthocis, proviennent de projets du domaine de la bioinformatique et ont en commun de mettre en œuvre des traitements exploitant de grandes masses de données génomiques et nécessitant d'importantes ressources de calcul. PrimerFinder nous donnera l'occasion de montrer l'intérêt de notre méthode de conception de chaînes de traitements. Nous détaillerons notamment la méthode de capture à partir de composants hétérogènes et de transformation d'un modèle prototype en une implémentation intensive opérationnelle basée sur le MoC UDFAS, en incluant l'extraction semi-automatique du parallélisme de données gros grain. L'implémentation d'Orthocis nous permettra de montrer comment notre méthodologie de capture rapide des chaînes de traitements s'inscrit dans une démarche globale de simplification de la conception des applications scientifiques, du prototypage à la mise à disposition. Nous verrons également que les workflows gérés par un WfMS peuvent être utilisés de multiples façons, incluant la conception d'une application SaaS *Ad hoc* d'analyse de données. Enfin, nous mettrons en lumière les avantages tirés de l'association entre modèle de workflow et modèle des composants et des plates-formes (CPMM). Ainsi, nous décrirons les capacités de simplification et d'automatisation du déploiement, obtenues sur des infrastructures de type cluster, PaaS et CaaS.

5.2.1 PrimerFinder

Contexte Scientifique

Le séquençage d'ADN est le processus consistant à déterminer l'ordre précis de nucléotides dans une molécule d'ADN. Il inclut les procédés et techniques qui sont utilisés pour déterminer l'ordre des quatre bases, adénine, guanine, cytosine, thymine dans un brin d'ADN. La mise au point des méthodes de séquençage rapide de l'ADN a accéléré la recherche en biologie. Les premières séquences d'ADN ont été obtenues dans les années 1970 en utilisant des méthodes longues et laborieuses basées sur la chromatographie à deux dimensions. Des technologies performantes de séquençage (NGS pour Next Generation Sequencing) ont émergé à partir de 2005 pour ensuite se diffuser très largement aujourd'hui grâce à la rapidité et au coût réduit du séquençage.

Le contexte de recherche [1] de PrimerFinder concerne l'utilisation de la génomique dans l'étude de la diversité biologique des sols, appelée métagénomiques des sols. La diversité génétique peut être estimée par des études métagénomiques exploitant le séquençage de l'ADN ribosomique. La proportion des espèces est estimée par les quantités respectives de leurs amplicons. Ces amplicons sont des fragments de séquences d'ADN

1. L'outil PrimerFinder a été réalisé dans le cadre d'un travail mené en collaboration avec l'équipe DYLISS de l'IRISA et Frédéric Mahé de l'équipe Diversité Microbienne de l'université de Kaiserslautern. Cette équipe de recherche étudie la diversité microbienne eucaryote dans le sol des forêts tropicales d'Amérique du Sud et d'Amérique centrale.

ribosomaux 18S (cf. figure G.1 en annexe), spécifiques d'une espèce ou sous-espèce répertoriée et sont positionnés entre deux séquences de primers [1]. Nous disposons de deux jeux de données NGS composés de lectures d'ADN, l'un issu d'un séquenceur Roche 454 (10 échantillons de 40 000 lectures) et l'autre d'un séquenceur Illumina Hi-Seq (10 échantillons de 450 000 lectures). C'est dans ces jeux de données que doivent être détectées les amorces génériques des ARNr 18S (cf. figure G.1 en annexe). L'objectif de l'analyse de données est de retrouver l'ensemble des primers pour ensuite être capable d'extraire tous les amplicons associés. En effet pour les biologistes étudiant la biodiversité des sols, chaque amplicon est important puisqu'il est potentiellement associé à une espèce.

Définition des besoins

Les bioinformaticiens veulent concevoir une chaîne de traitements, destinée aux biologistes, capable d'identifier toutes les séquences qui sont des amorces génériques contenues dans des lectures d'ADN correspondant aux ARNr 18S extraits d'échantillons de sols. La méthode classique par recherche d'expression régulière (egrep) utilisée à Kaiserslautern ne donne pas de résultat suffisamment précis du fait de l'expressivité insuffisante des expressions régulières. De ce fait, l'utilisation d'un logiciel exploitant une grammaire riche et dédiée aux séquences est pertinente. Pour cela nous pouvons utiliser l'outil de *pattern matching* Logol [130]. Ce logiciel peut rechercher de façon exhaustive des séquences cibles, définies par une grammaire, dans une banque de séquences. Cependant Logol n'a pas été conçu pour traiter des grands jeux de données. Chaque traitement Logol nécessiterait plusieurs journées de calcul sur un nœud puissant d'un cluster de calcul, ce qui gênerait la mise au point de la chaîne de traitements et pourrait décourager les utilisateurs finaux d'utiliser cette méthode d'analyse prometteuse.

Pour accélérer les traitements, une méthode simple de parallélisation est évoquée. On découperait préalablement les séquences au format FASTA(cf. annexe I) par un prétraitement puis de multiples instances de Logol seraient lancées simultanément sur un cluster de calcul et analyseraient les fragments de séquences, préalablement identifiés par leur position dans les séquences d'origine (cf. figure 5.2). Cependant l'outil Logol n'a pas été conçu pour le très haut-débit et nous ne pouvons pas modifier son code dans des délais raisonnables pour y intégrer des méthodes de parallélisation de données adaptées à nos besoins. De ce fait, la génération du parallélisme telle qu'elle est gérée par notre WfMS apparaît particulièrement adaptée (cf. section 4.3.2). Ainsi, c'est l'intégration rapide de l'outil Logol pour l'analyse de données à haut-débit qui motive tout d'abord l'usage de notre WfMS.

Conception de la chaîne de traitements

La conception du workflow d'analyse, nommé PrimerFinder (cf. figure G.2 en annexe), est réalisée par un collaborateur bioinformaticien [2]. Celui-ci travaille de façon

1. Les séquences de primers sont retrouvées systématiquement pour toutes les espèces étudiées. On peut donc les qualifier de séquences universelles.

2. Ce bioinformaticien, Aymeric Antoine Lorquin, travaille sur l'exploitation de Logol dans le cadre de sa thèse. Il fait lui même le choix d'utiliser notre WfMS, suite à une présentation.

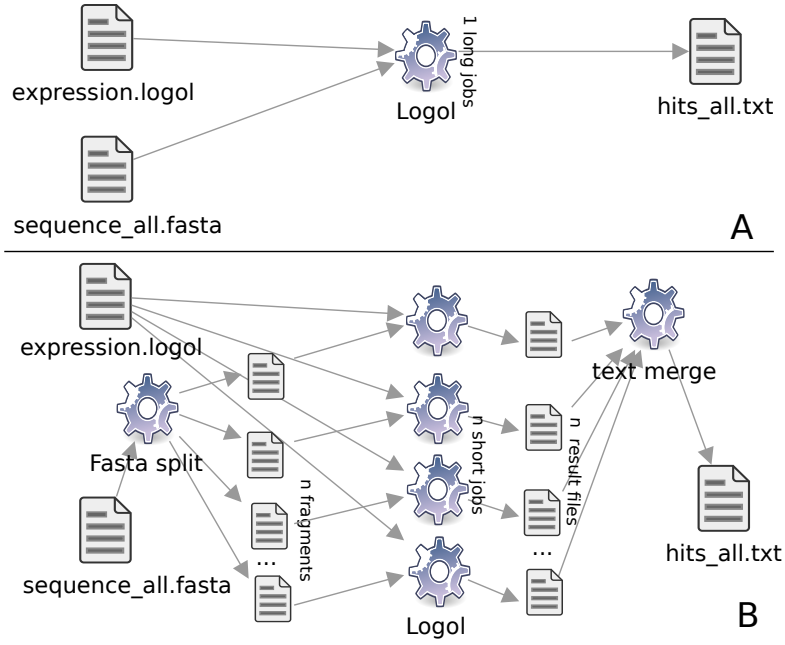


FIGURE 5.2: Extrait d'un DataFlow appelant l'outil Logol dans PrimerFinder. Le prototype (A) est converti en implémentation intégrant un patron de conception de parallélisme de données gros grain (B). Ce patron *Split/Map/Merge* peut être implémenté de différentes manières. Il est soit codé ou intégré manuellement, soit généré par transformation de modèles tel que nous le décrivons dans le chapitre méthodologie.

autonome sur la conception du workflow et de ses composants. Cet intervenant, que nous qualifierons d'expert fonctionnel, exploite l'environnement du WfMS de la façon suivante: il conçoit des lignes de commande mais aussi des scripts (wrapper, parsers, outils d'analyse...) dont les premières versions sont testées dans la console d'un terminal. Les lignes de commande sont ensuite directement converties en patron de commande et encapsulées dans des acteurs d'exécution (cf. section 4.3.1). Par ailleurs, les codes des scripts écrits par le bioinformaticien sont directement intégrés dans le WfMS en tant que composants déployables (cf. figure 5.3) via un interface (cf. section 4.4.2) [1]. Le WfMS pourra ainsi déployer des versions à jour de ces scripts sur les nœuds de calculs (cf. section 4.4.2). Une phase de test des nouveaux composants est ensuite réalisée où il subsiste ponctuellement l'usage du terminal, afin de faciliter le débogage. Le reste du travail consiste à qualifier les acteurs (ports, description...) puis à les connecter dans le graphe DataFlow de l'éditeur de workflows (cf. figure 4.3).

Le préalable à cette étape de composition est une brève formation facilitant la compréhension de la sémantique de notre modèle DataFlow [2]. La phase finale de conception consiste à configurer l'interface de soumission par défaut en utilisant le GUI du générateur d'interface du WfMS (cf. figures G.4 et G.3 de l'annexe G). La conception de PrimerFinder confirme que l'expressivité de notre langage graphique DataFlow est suffisante pour permettre l'implémentation d'une chaîne de traitements de données à haut-débit en bioinformatique. L'expert fonctionnel exploite pour cela un vaste ensemble de fonctionnalités disponibles dans le WfMS. Comme nous l'avons détaillé dans les chapitres 3 et 4, les aspects techniques concernant en particulier la génération du parallélisme, l'ordonnancement des tâches et le déplacement des données sont délégués à l'environnement qui a pour vocation la simplification de la conception et de la mise en œuvre de l'exécution des traitements.

Après finalisation du design, le bioinformaticien exécute des instances du workflow pour traiter les jeux de données NGS. Le workflow élaboré est fonctionnel et donne des résultats exploitables. Les méthodes "Logol" et "expression régulière" trouvent respectivement 100 et 90% des primers connus pour le jeu de données Roche 454 et 90, et 80 % pour le jeu de données Illumina Hi-Seq. Le bioanalyste a pu mettre en évidence l'intérêt de la méthode Logol par rapport aux expressions régulières par la réalisation rapide d'un workflow de benchmarking d'outils d'analyse de données à haut-débit. Durant le traitement des jeux de données, une unique instance du workflow PrimerFinder a produit jusqu'a 10000 jobs Logol en parallèle [3]. Cette montée en charge a été supportée par le moteur dont l'implémentation prend en compte ces contraintes, notamment au niveau

1. Le répertoire d'acteurs de l'intervenant est alors peuplé de ces nouveaux acteurs.

2. Cette formation d'une heure environ a concerné en particulier la gestion du flot de données (cf. section 3.2.1) (l'enchaînement des tâches par les dépendances de données, les itérations implicites, les conditions) et la manipulation des acteurs

3. Le temps d'exécution constaté est d'approximativement une semaine sur des nœuds de 32 et 64 Go de RAM (CPU: 8 x 2.73 GHz, OS: Linux - Centos 6)

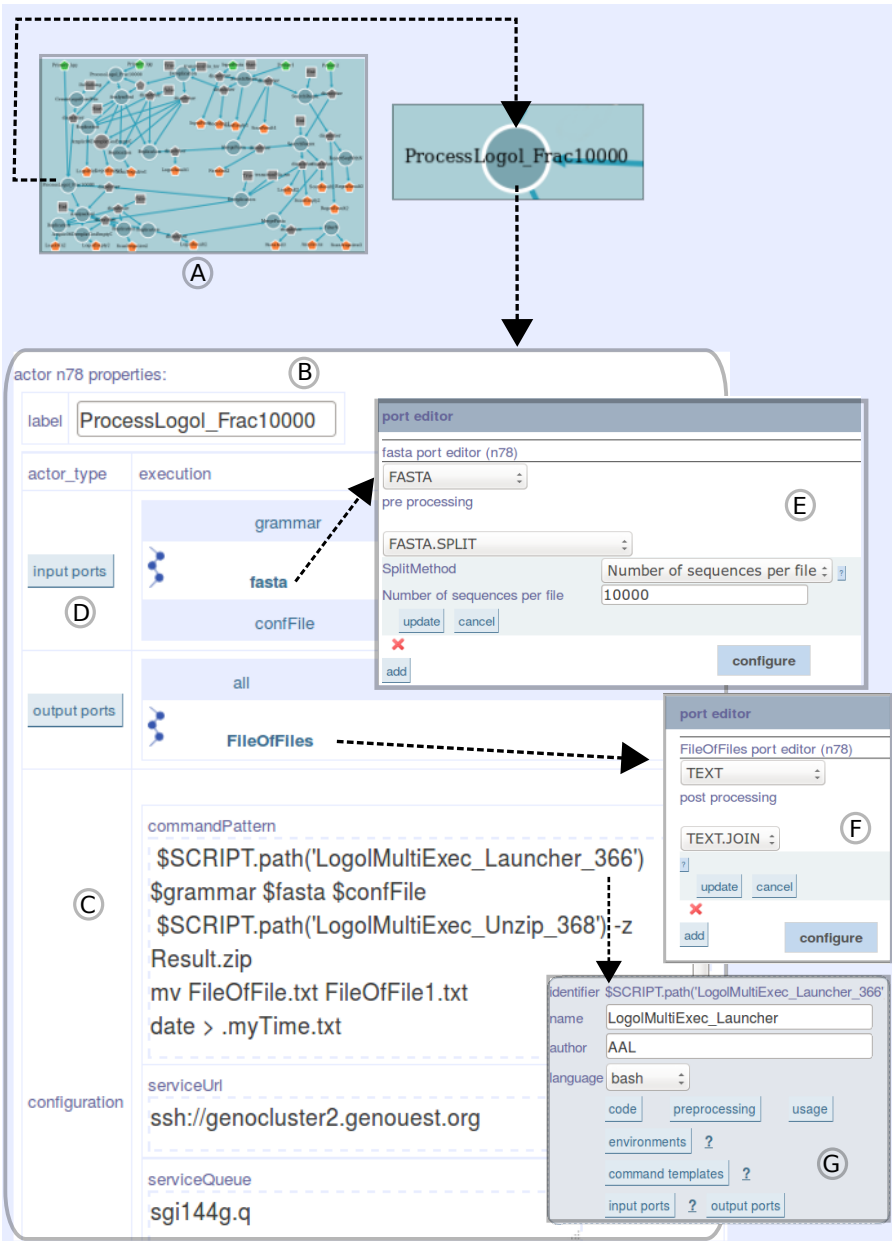


FIGURE 5.3: L'acteur LOGOL dans PrimerFinder. L'éditeur de workflows du WfMS (A) donne accès à l'interface de configuration de l'acteur LOGOL (B). Cette interface permet la définition de la ligne de commande (C) et des ports de l'acteur (D). Ces ports peuvent ensuite être typés et associés à des méthodes DFF. L'interface d'édition d'un port d'entrée (E) de l'acteur Logol permet le typage et l'appel de la méthode DFF FASTA.SPLIT(m,n). L'interface d'édition d'un port de sortie (F) appelle ici la méthode DFF TEXT.JOIN(). Dans ce cas précis, trois acteurs seront générés avant l'exécution par le moteur du WfMS (FASTA.SPLIT, LOGOL, TEXT.JOIN). L'acteur métier LOGOL sera donc exécuté suivant un patron *Split-Map-Merge*. Si l'on se réfère au paradigme MapReduce, l'appel à LOGOL constitue la fonction Map . Dans la ligne de commande, l'expression *$SCRIPT.path* référence des scripts déployables à l'exécution par le moteur du WfMS . Ils sont éditables via un gestionnaire de sources intégré (G). Une vue d'ensemble du workflow est disponible en annexe.

du mécanisme de persistance et de la gestion des connexions, éléments sensibles dans un contexte de fortes sollicitations (cf. section 4.1.2).

Evaluation du WfMS par l'utilisateur

Nous réalisons alors un entretien avec l'expert fonctionnel. Celui-ci met en évidence les éléments qui l'ont motivé à concevoir PrimerFinder dans notre WfMS plutôt qu'à implémenter une chaîne de traitements à partir de scripts. Le bioinformaticien considère que les éléments qui ont motivé son choix sont, par ordre d'importance:

1. l'automatisation de l'intégration du parallélisme de données pour l'analyse à haut-débit;
2. la gestion par l'environnement de la soumission des jobs et de leur enchaînement;
3. la rapidité d'obtention d'un prototype fonctionnel;
4. les facilités dans la gestion des données et la traçabilité des résultats.

Dans le cas de PrimerFinder, la capacité à intégrer rapidement le parallélisme de données est l'élément le plus déterminant. Le concepteur a fait le choix d'utiliser notre WfMS afin de pouvoir évaluer les potentialités de l'outil Logol pour l'analyse d'un grand jeu de données (1) dans un temps minimum (3). Par ailleurs, l'expert fonctionnel considère qu'il n'aurait pas pu explorer la piste de Logol en l'absence d'un outil d'aide à la conception de chaînes de traitements adapté au prototypage comme un WfMS intégratif orienté conception.

Apports du cas d'utilisation

Même si nous ne disposons pas ici de mesures quantitatives permettant de généraliser, nous constatons ici une prise en main rapide du WfMS et une productivité importante dans la phase de conception. La parallélisation semi-automatique (cf. figure 5.3E, F) constitue dans ce cas d'utilisation un avantage indéniable: le bioinformaticien a pu tester rapidement l'emploi de LOGOL sur un grand jeu de données et sa pertinence par rapport aux expressions régulières. Une implémentation manuelle aurait nécessité un temps de développement nettement plus important, ce qui, d'après le concepteur, aurait pu l'inciter a ne pas réaliser ces expérimentations. La rapidité de conception prime ici sur la performance brute.

Le cas d'utilisation PrimerFinder correspond plus généralement au prototypage d'une application scientifique d'analyse de données à partir de composants hétérogènes. Dans ce contexte, nous considérons que cet exemple met en évidence l'intérêt de la méthodologie de conception rapide de workflows et de l'intégration des fonctionnalités d'aide à la génération du parallélisme que nous avons mises en œuvre. Nous mettons également en évidence l'autonomisation de l'expert fonctionnel, un concepteur d'analyse de données proche du domaine métier applicatif. Ce bioinformaticien ayant des compétences de scripting aura travaillé de façon totalement autonome pour la conception de PrimerFinder mais aussi lors de la phase de production des résultats.

5.2.2 Orthocis

Contexte Scientifique

Notre second cas d'utilisation correspond à l'implémentation d'Orthocis, un nouvel outil d'annotation de listes de gènes qui associe recherche de motifs et exploitation de l'orthologie entre espèces [1]. En bioinformatique, un motif ou Pattern est défini par une expression qui représente les caractéristiques communes d'un ensemble de séquences. Par ailleurs, l'orthologie est un type de relation évolutive entre gènes d'espèces différentes. Deux gènes sont des orthologues si leurs séquences proviennent de la même séquence du plus récent ancêtre commun aux deux espèces [2]. Le but de l'outil Orthocis est de permettre l'exploration des séquences de régulation des zones en amont des gènes (CIS) dans les génomes eucaryotes [3]. Orthocis se focalise sur l'étude des cibles des facteurs de transcription. Ce sont des protéines nécessaires à la régulation de la transcription des gènes. Orthocis vise à trouver des gènes cibles putatifs, spécifiques d'un facteur de transcription particulier, amplificateur, co-activateur ou inhibiteur de leur expression. Pour chaque facteur étudié, un motif nucléique de signature d'appariement sur les génomes doit être utilisé.

Elaboration de la source de données

Dans Orthocis, la chaîne de traitement de recherche de patterns prend en entrée deux formats de motifs nucléiques. Il s'agit soit d'une matrice point position (PWM), soit d'une grammaire spécifique qui reflète la séquence consensus du pattern d'intérêt et qui sera recherchée dans les séquences, jusqu'à 10 kb en amont des gènes, pour l'ensemble des génomes traités.

Les hits trouvés dans les génomes sont caractérisés par leur position et leur p-value. Pour réduire le taux de faux positifs, Orthocis filtre les hits suivant la p-value et l'information d'orthologie des gènes.

Les données des génomes de 65 espèces eucaryotes, composées des séquences des chromosomes, des positions des gènes et de leurs relations d'orthologie, sont exploitées dans Orthocis. L'ensemble de ces données provient de la banque de données de référence Ensembl [131, 132].

A partir des orthologues, une source de données comportant *des groupes d'orthologie* est construite en plusieurs étapes. Tout d'abord, un graphe est construit où les nœuds sont les gènes et les arêtes les relations d'orthologie. Ensuite, les composantes connexes du graphe sont calculées puis on procède au calcul des cliques qui représentent des *groupes d'orthologie* [4]. Nous obtenons une source de données composées de:

1. L'application Orthocis est développée dans le cadre du projet ANR FATINTEGER qui concerne la recherche des régulateurs impliqués dans la plasticité lipidique chez les animaux d'élevage.
2. Les gènes orthologues peuvent être classés en 2 catégories principales nommées *one2one* et *one2many*, exprimant des relations de cardinalité entre gènes.
3. C'est-à-dire chez les organismes dont les cellules présentent un noyau et des mitochondries.
4. Un groupe d'orthologie contient un gène par espèce, orthologue avec tous les autres membres du groupe.

— 65 espèces sous la forme de séquences au format FASTA (cf. figure I.1 en annexe) et de coordonnées des gènes;

— 21 000 000 relations d'orthologie entre gènes;

— 98 000 cliques équivalents à des *groupes d'orthologie*.

L'information des *groupes d'orthologie* et des hits génomiques est exploitée par les workflows exécutables depuis l'interface d'Orthocis.

Prototypage sous forme d'une collection de workflows

Nous débutons le développement d'une application d'analyse de données *Ad hoc* en créant une série de workflows intégrée au WfMS. Les biologistes et bioinformaticiens participant au projet valident ensuite ce prototype.

fonctionnalité	workflow	GUI
WF1: Hits de motif filtrés	oui	dédié
WF2: Intégration d'un nouveau motif	oui	générique
WF3: Scan à partir d'un motif	oui	générique
WF4: Configuration du GUI WF1	oui	générique
API1: Authentification	non	générique

TABLE 5.2: Fonctionnalités de l'outil Orthocis
Les fonctionnalités WF1, WF2, WF3 et WF4 sont des workflows gérés par le WfMS. Les interfaces de WF2, WF3 et WF4 sont générés par l'environnement tandis que WF1 présente une interface dédiée créée par un développeur.

Les fonctionnalités principales d'Orthocis sont listées dans le tableau 5.2. Les résultats préliminaires d'analyse exploitant les hits des motifs et les groupes d'orthologie semblent bruités. La précision reste insuffisante. Le bioinformaticien [1] expérimente donc l'ajout de filtres additionnels des hits génomiques de l'application élaborée exploitant les différences de valeurs de p-value et l'homologie de séquences. Ces filtres sont créés tout d'abord sous forme de scripts indépendants, qui sont testés puis intégrés, soit en tant que nouveaux acteurs du workflow de filtrage des hits (WF1), soit en modifiant le code d'acteurs déployables. L'exécution de WF1 après ajout de ces filtres apporte des résultats *in silico* pertinents [2].

Conception rapide d'applications SaaS dédiées à partir d'un répertoire de workflows

Dans un deuxième temps, les résultats obtenus motivent l'élaboration d'une application web permettant de diffuser l'usage de l'outil auprès d'un large public. Nous spécifions

1. Aymeric Antoine Lorquin

2. Certains gènes qui présentent en amonts de leur séquence génomique des motifs de régulation du facteur de transcription $LXR\alpha$ ont des gènes orthologues qui présentent également ce motif chez de multiples espèces. Par ailleurs l'ensemble de ce jeu de hits comporte des séquences fortement homologues entre elles. Cela laisse penser à l'identification de nouvelles cibles de $LXR\alpha$.

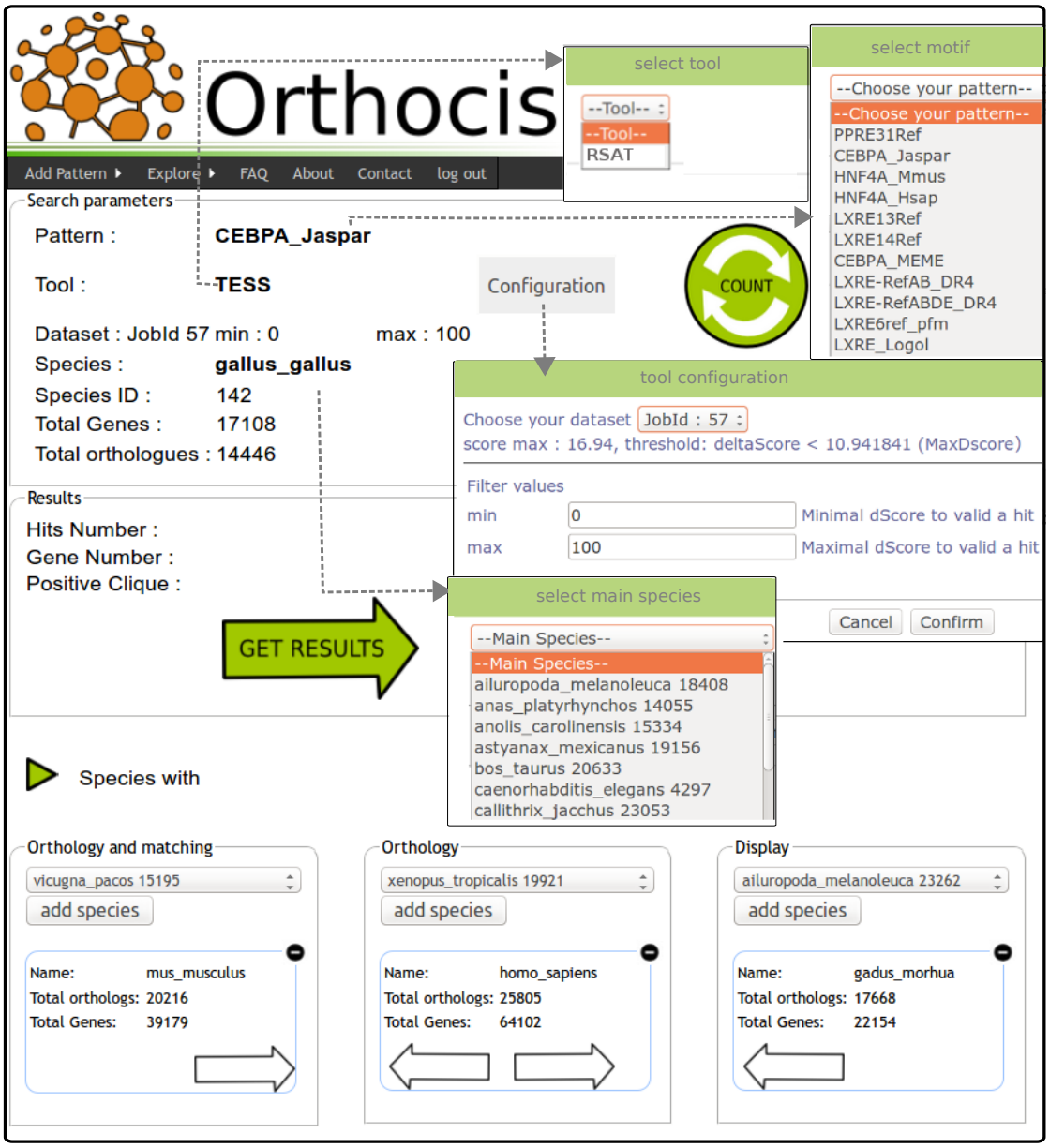


FIGURE 5.4: Interface web principale de l'outil SAAS Orthocis qui concerne la fonctionnalité de recherche de motifs avec filtrage par orthologie. Le workflow correspondant $WF1$ est éxécuté par le WfMS. L'interface facilite la recherche d'un pattern dans toutes les séquences en amont des gènes de plusieurs espèces et la sélection des gènes pour qui le pattern est également trouvé dans le voisinage de leurs orthologues. Pour explorer les résultats, l'utilisateur détermine dans un premier temps l'espèce principale P dans laquelle il considère que le pattern étudié doit être présent. Dans un deuxième temps, différentes autres espèces sont sélectionnées dans lesquelles les gènes orthologues des gènes de P doivent également comporter le pattern.

alors avec les biologistes une interface spécialisée, orientée vers l'exploration de données. Pour l'élaborer, nous utilisons une architecture orientée service permettant d'exploiter directement la collection de workflows élaborée précédemment (cf. figure 5.2). Pour intégrer nos workflows à la nouvelle interface, nous pouvons exploiter les formulaires configurables et les services Rest du WfMS. Nous utilisons ici la complémentarité de ces deux modes de mise à disposition pour concilier rapidité de développement et besoin de personnalisation. L'interface d'Orthocis est composée de formulaires permettant de paramétrer les outils et de les exécuter (cf. figure 5.4). Elle offre également l'accès aux états des jobs et aux résultats (cf. figures H.1, H.2 et H.3 de l'annexe H). L'interface graphique principale qui concerne la fonctionnalité de recherche de motifs (WF1) est développée manuellement. Dans ce cas, les demandes de traitements paramétrés par l'utilisateur sont envoyées aux services Rest qui constituent l'API du WfMS (cf. section 4.1.3). Dans d'autres cas (WF2, WF3), les interfaces personnalisables générées par le WfMS se sont avérées suffisantes et ont été directement intégrées à Orthocis.

La réalisation d'Orthocis a donc permis de mettre en évidence la complémentarité des différentes formes de mise à disposition des services du WfMS, API Rest et interface web. Finalement, nous remarquons que l'architecture employée dans Orthocis met en œuvre un usage innovant des workflows (cf. section 2.1.2). Ceux-ci sont exploités en tant que services et agrégés pour produire une application *Ad hoc* d'analyse de données.

Rôle des intervenants

fonctionnalité	composants-acteurs	orchestration-workflow	GUI soumission	GUI acces résultats
WF1:hits de motif filtrés	EF	EF	**DI**	**DI**
WF2: intégration d'un nouveau motif	EF	EF	**DI**	**DI**
WF3: scan à partir d'un motif	EF	EF	**DI**	**DI**
WF4:configuration du GUI WF1	**DI**	**DI**	**DI**	**DI**
API1:authentification	**DI**	**DI**	**DI**	-

TABLE 5.3: Orthocis: rôle des intervenants dans un processus de conception classique
Dans un processus de conception classique supposé, les intervenants, expert fonctionnel (EF) et développeur / intégrateur (DI) ont des rôles pré-déterminés. L'application client / serveur est directement codée sous forme de scripts et de pages web. La soumission des jobs sur le cluster de calcul est effectuée par des appels à des fonctions provenant d'API comme DRMAA. L'intervention du développeur est importante.

Nous comparons les rôles d'un bioinformaticien expert fonctionnel et d'un développeur dans deux cas de figure. Les tableaux 5.3 et 5.4 rendent compte respectivement du rôle des intervenants dans le processus de conception de l'application suivant une architecture classique client/serveur supposée ou suivant l'architecture implémentée, qui exploite les API et interfaces du WfMS. Il apparaît que les items nécessitant le recours aux compétences du développeur sont nettement plus nombreux dans l'approche

fonctionnalité	composants-acteurs	orchestration-workflow	GUI soumission	GUI accès aux résultats
WF1: Hits de motif filtrés	EF	EF	**DI**	WfMS
WF2: Intégration d'un nouveau motif	EF	EF	WfMS(EF)	WfMS
WF3: Scan à partir d'un motif	EF	EF	WfMS(EF)	WfMS
WF4:Configuration GUI WF1	EF	EF/**DI**	WfMS	WfMS
API1:Authentification	-	-	WfMS	-

TABLE 5.4: Orthocis: rôle des intervenants dans le processus de conception basé sur le nouveau WfMS
Rôle des intervenants, respectivement expert fonctionnel (EF) et développeur / intégrateur (DI), dans le processus de conception mis en œuvre pour l'application web Orthocis. Ces intervenants exploitent l'environnement du WfMS (GUI et/ou API). C'est le WfMS qui gère la soumission des jobs sur le cluster de calcul. L'intervention du développeur est ici limité. Dans l'ensemble de ces fonctionnalités, seule la partie cliente exploitant les workflows WF1 et WF4 a été conçue en dehors du WfMS et a nécessité l'implication d'un développeur. Les interfaces graphiques liées à d'autres fonctionnalités (WF2, WF3...) sont directement générées par le WfMS, ainsi que l'ensemble des tâches liées aux traitements d'analyse de données.

classique. Même si cet exemple est insuffisant pour en tirer des conclusions générales, nous pouvons supposer qu'avec l'aide du WfMS, la phase de conception de l'application est plus rapide et nécessite moins de main-d'œuvre et de compétences techniques. Par ailleurs, le rôle des intervenants est clairement défini. Le développeur n'est sollicité que pour les aspects d'interfaçage particuliers à l'application (fonctionnalités WF1 et WF4) et qui ne sont pas gérés de façon automatique par le générateur d'interfaces du WfMS (cf. section 4.1.3).

D'autre part, l'architecture logicielle orientée service permet une séparation claire entre les couches applicatives. Le WfMS est le cœur applicatif qui gère la conception et l'exécution des composants d'analyse de données. L'interface est complètement découplée de la couche de traitements. Il en découle des rôles clairement définis pour les intervenants (cf. section 2.2.2) dans l'organisation de la conception de l'application. L'expert fonctionnel se concentre sur l'intégration et le développement des composants métiers. Le développeur a un rôle restreint (cf. figure H.4 en annexe). Il se charge de la création des interfaces dédiées dans les cas complexes où la personnalisation des vues générées par le WfMS est insuffisante (cf. figure 5.4). Il en résulte une productivité accrue dans le développement de l'outil, de la conception des composants métiers jusqu'à l'interfaçage, tout en maintenant la capacité de personnalisation et d'intégration dans des environnements tiers.

Par ailleurs, l'application basée sur une exploitation interne du WfMS bénéficie d'une maintenabilité et d'une évolutivité importante. Après l'intervention du développeur, le bioinformaticien a pu concevoir de nouvelles fonctionnalités d'Orthocis de façon autonome, en intégrant à l'interface un nouveau workflow. Il modifie également seul les

modèles des workflows déjà exploités dans Orthocis [1].

Changements d'infrastructure de calcul et Déploiement rapide

Nous avons testé la capacité des workflows conçus à partir de notre WfMS à être déployés sur deux types d'infrastructure, cluster de calcul classique et cluster virtuel [2] mis à disposition suivant le modèle *Containers as a Service* (cf. section 2.6.1). Dans ces deux cas, nous utilisons la collection de workflow constituant le cœur fonctionnel d'Orthocis, gérée par une instance de notre WfMS.

Déploiement rapide sur un cluster Dans un premier temps, nous évoquons le cas du cluster de calcul qui est, dans le cadre de cette étude, comparable au PaaS. L'ensemble des dépendances des composants des workflows sont déployés, puis, pour simplifier la configuration et le déploiement, nous exploitons la configuration des outils par les variables d'environnement et les acteurs déployables (cf. section 3.3.2 et les figures 3.14 et 3.15-B). Nous allons décrire l'usage de ces mécanismes de couplage entre modèle de workflow et CPMM. Tout d'abord, le concepteur configure les acteurs par une interface du WfMS. Il peut éditer le code des acteurs déployables (cf. figure 5.5-A) et/ou leur associer un environnement générique qui sera adapté spécifiquement à la plate-forme d'exécution ciblée (cf. figure 5.5-B). Ainsi, le moteur de workflows peut exploiter le CPMM. Durant l'orchestration, lors de la génération des commandes, le déploiement des scripts aura lieu (cf. figure 5.6-A, B) puis les configurations des commandes seront injectées (cf. figure 5.6-C, D). Cette dernière étape du processus de réécriture des commandes est spécifique à l'hôte [3].

Déploiement rapide pour le modèle CaaS Dans un deuxième temps, les workflows d'Orthocis ont été déployés puis exécutés en mode CaaS. Dans ce cas, les dépendances des acteurs formalisées dans le CPMM ont été encapsulées dans des containers Docker. Ceux-ci ont été mis à disposition dans un répertoire communautaire dédié aux containers d'outils bioinformatiques, BioShaDock, dont nous avons initié la création [133].

Nous utilisons d'abord un environnement cloud sur lequel est installé Docker (http://genocloud.genouest.org) puis nous exploitons un cluster en mode CaaS sur lequel est installé un middleware supplémentaire chargé de la gestion du cycle de vie des containers [134]. Sur les nœuds de calculs, si c'est nécessaire, le système télécharge et met à jour les images des containers, juste avant la soumission des jobs. Les commandes sont ensuite générées par le moteur de workflows puis exécutées (cf. figure 5.7). Des résultats sont obtenus dans des temps comparables à ceux d'un cluster classique.

Nous montrons donc l'intérêt de l'association entre spécification de modèle de workflow semi-concret (SCDFM) et CPMM pour simplifier le déploiement. Nous obtenons

1. La modification du DataFlow et/ou des acteurs est permise par l'interface du WfMS et impacte directement l'application Orthocis cliente. L'aide d'un développeur devient nécessaire si les acteurs d'entrée/sortie sont modifiés.

2. Pour nos expérimentations de *cloud computing*, nous utilisons l'infrastructure Genocloud de Genouest qui exploite le gestionnaire de cloud OpenNebula [104]

3. L'instance d'Orthocis mise en production avec accès public exploite cette implémentation.

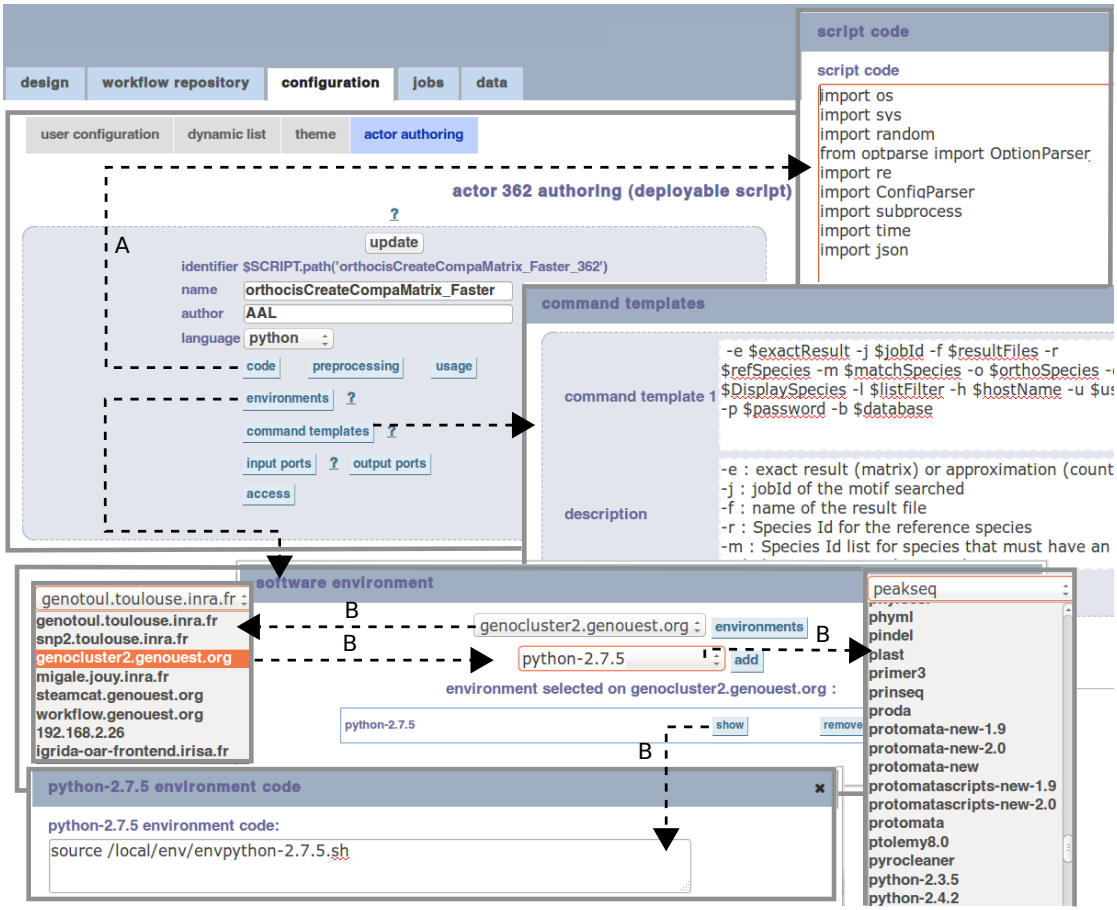


FIGURE 5.5: Interface de configuration des acteurs du WfMS. Dans le cas du composant d'Orthocis *orthocisCreaCompaMatrix_Faster*, les mécanismes d'acteur déployable (A) et de configuration des commandes par variables d'environnement (B) sont utilisés. Ces méthodes constituent une forme d'implémentation du couplage entre modèle de workflow semi-concret SCDFM et méta-modèle des composants et des plates-formes (CPMM)

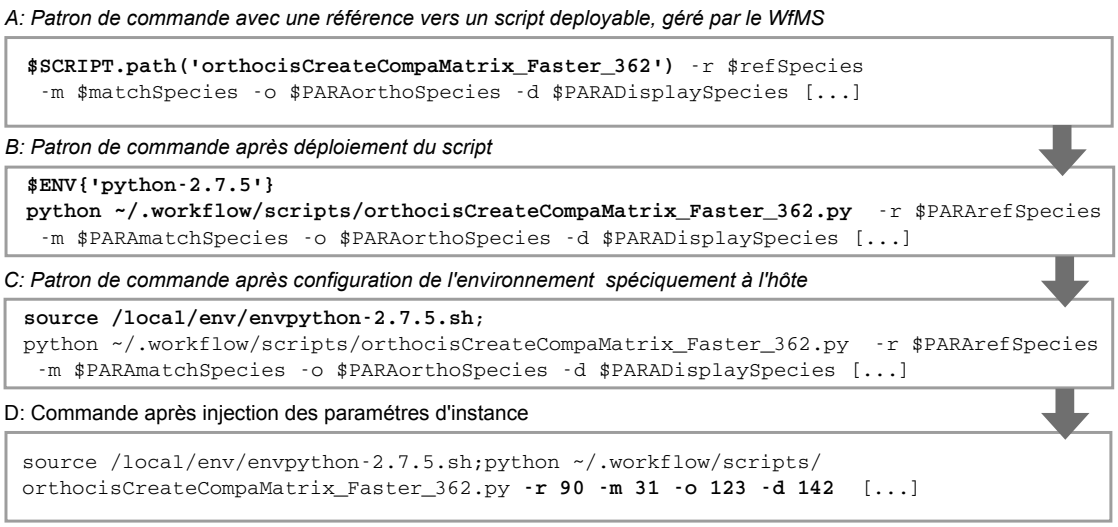


FIGURE 5.6: Exemple d'un acteur d'exécution d'un workflow d'Orthocis pour lequel le passage du PIM (Platform Independent Model) au PSM (Platform Specific Model) est géré par le mécanisme de scripts déployables (A) et l'injection de configurations spécifiques à l'environnement d'exécution par le mécanisme des variables d'environnement (B).

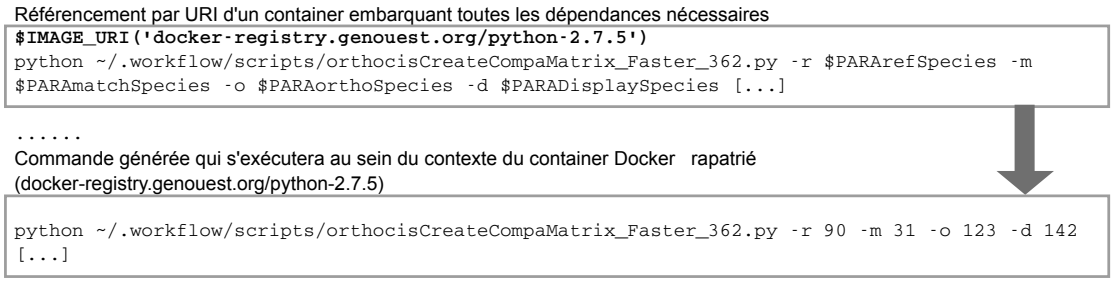


FIGURE 5.7: Exemple d'un acteur UDF d'un workflow d'Orthocis pour lequel le passage du PIM (Platform Independent Model) au PSM (Platform Specific Model) est géré par l'exécution de la ligne de commande dans un environnement logiciel isolé dans un container. Ce mécanisme est rendu possible par l'intégration dans la spécification A-SCDFM de l'adresse du container (image_uri). Il s'applique à toute plate-forme CaaS (Containers as a Service).

notamment, par ce couplage, une spécification autonome (A-SCDFM) qui rend possible le déploiement automatiquement quelle que soit l'infrastructure CaaS. Il s'agit d'un moyen efficace pour favoriser le nomadisme des chaînes de traitements.

5.3 Résumé

Les deux cas d'utilisation exposés, PrimerFinder et Orthocis, nous ont permis de confirmer un ensemble de suppositions à la base de nos propositions méthodologiques.

Dans un premier temps nous avons pu montrer qu'un moteur de workflows orienté service basé sur le modèle de calcul UDFAS était adapté à l'analyse de données intensive dans des contextes à la fois de conception et de mise en production. Ainsi nous avons pu établir que notre moteur pouvait traiter de grands jeux de données de façon efficace. Par ailleurs, l'expressivité du langage DataFlow s'est révélée suffisante pour la mise en œuvre des outils PrimerFinder et Orthocis. Néanmoins d'autres cas plus complexes, faisant par exemple un usage important des conditions et des acteurs composites seraient nécessaires pour une meilleure évaluation. L'intérêt de notre démarche de conception a également été illustrée dans un contexte de prototypage rapide d'applications bioinformatiques. Ainsi, la capture des spécifications à partir de lignes de commande est simple et appropriée aux pratiques existantes. Nous avons aussi montré, que la prise en charge d'une grande partie des aspects techniques de la conception par WfMS intégratif permet au concepteur de chaînes de traitements de gagner en autonomie. En outre, PrimerFinder a tout particulièrement mis en évidence que l'intégration semi-automatique du parallélisme gros grain présente un intérêt direct pour le concepteur de workflows intensifs. En effet la conversion d'un prototype en version intensive peut être extrêmement rapide et simplifiée avec notre WfMS dans la mesure ou les types de données exploitées et leur DFF sont présents dans l'environnement.

Dans un deuxième temps, Orthocis nous a permis d'expérimenter le couplage entre modèle de workflow et méta-modèle des composants et des plates-formes afin de faciliter le déploiement sur différents types d'infrastructures de calcul (clusters, PaaS, CaaS). Dans le cas du CaaS, ce lien entre modèles se traduit par l'association entre répertoire de modèles de workflow et répertoire d'images de containers. La spécification de workflows A-SCDFM est une extension de SCDFM qui capture ces relations. En se basant uniquement sur cette spécification, nous avons montré que l'on peut automatiser le déploiement de l'ensemble des dépendances référencées par les acteurs d'un DataFlow. Nous pouvons ainsi cibler indifféremment toutes infrastructures CaaS. D'une part, la mise en œuvre réussie de cette spécification de workflows autonome constitue un résultat important permettant d'augmenter l'autonomie du concepteur pour le déploiement. Finalement, nous pouvons envisager l'usage de A-SCDFM pour faciliter l'élaboration de répertoires communautaires de modèles de workflow intrinsèquement déployables, dans lesquels les concepteurs de composants jouent un rôle actif.

Chapitre 6

Discussion et Conclusion

6.1 Bilan

Dans cette étude, nous nous sommes intéressés à l'adaptation des WfMS scientifiques à la conception, à l'exécution et à la diffusion des chaînes de traitements d'analyse intensive de données. Nous passerons en revue et commenterons nos contributions dans les paragraphes suivants.

6.1.1 Un moteur de workflows performant

En premier lieu, nous avons défini un langage graphique DataFlow, minimal et expressif, adapté à la conception rapide. Le workflow capturé de cette manière peut ensuite être exécuté avec un moteur basé sur un nouveau modèle de calcul, nommé UDFAS (URI based DataFlow for Asynchronous Services).

De par son formalisme associant modèle DataFlow et automates finis, UDFAS se rapproche de certains modèles de calcul DataFlow dynamiques (cf. section 1.3.1) utilisés dans le traitement du signal ou des données multimédias [39] mais il est adapté aux architectures orientées service des WfMS.

A la différence des moteurs actuels (cf. section 2.7.2), notre moteur exploite efficacement l'ensemble des niveaux de parallélisme que l'on peut extraire de façon générique des chaînes de traitements en bioinformatique. Le formalisme DataFlow simplifie l'expression du parallélisme de tâches. L'exploitation du parallélisme de données gros grain est, quant à elle, facilitée par l'usage de jetons de données abstraits (DataSet URI ou DSURI) représentant des collections de données, suivant un patron fonctionnel. Le moteur de workflows est ainsi doté de capacités habituellement trouvées dans les environnements DISCS (Data-Intensive Scalable Computing Systems) .

Nos tests montrent de bonnes performances pour le traitement de grands jeux de données et cela, pour un temps de conception de workflows attendu particulièrement court. Notre moteur permet donc d'exécuter des chaînes de traitements d'analyse intensive de données aussi rapidement qu'en implémentant manuellement les patrons de parallélisme, ce qui nécessite bien plus d'expertise technique.

6.1.2 Une méthode de conception rapide

En deuxième lieu, l'approche de la conception des workflows scientifiques orientée modèle que nous avons élaborée constitue un apport important de cette étude.

Nous la mettons en œuvre pour extraire de façon simplifiée le parallélisme de données gros grain. Pour cela, nous avons exploité l'ontologie EDAM (EMBRACE Data and Methods) qui décrit les formats de données exploités en bioinformatique. Après avoir typé les données, l'utilisateur est guidé et peut définir une séquence de méthodes DFF associée à chaque entrée ou sortie d'un acteur. Ensuite, le graphe DataFlow est réécrit et de nouveaux acteurs sont injectés. Ceux-ci encapsulent les implémentations des DFF, mises à disposition par l'environnement logiciel. Cette méthode guidée d'extraction du parallélisme de données rend ainsi possible la production de patrons de conception *Split-Map-Merge*, similaires à ceux de MapReduce [11].

On peut, de ce fait, obtenir une spécification DataFlow capable d'exprimer, en plus du parallélisme de tâches, le parallélisme de données. C'est cette combinaison qui, dans de nombreux cas, rend possible la génération rapide d'un parallélisme massif approprié aux clusters de calcul. En conséquence, l'usage de cette approche dans un WfMS induit une diminution conjointe des temps de conception et des temps d'exécution. Il s'agit donc d'une solution efficace pour la création simplifiée des chaînes de traitements d'analyse intensive de données. A la différence des WfMS actuels orientés calcul intensif [63, 68], le parallélisme de données est ici contrôlable finement et de façon générique par le concepteur de workflows (cf. section 4.3.2) et peut être étendu simplement à tout format de données (cf. section 3.3.1).

Nous noterons que notre emploi d'EDAM s'inscrit dans une démarche générale d'intégration des ontologies de domaine dans un processus de conception orienté modèle (cf. section 1.5.2), comme celle qui est proposée par Aßman [52]. D'autre part, même si l'usage de la transformation de modèles pour la génération automatique de code parallèle est documenté [121], nous ne sommes pas au courant d'études préalables utilisant cette méthode pour la génération du parallélisme de données appliquée aux workflows.

6.1.3 L'automatisation du déploiement

Nous utilisons également l'approche orientée modèle pour introduire la représentation des plates-formes et des dépendances logicielles dans les workflows. Nous avons ainsi exposé comment, à partir d'un unique modèle de workflow nommé SCDFM (Semi-Concrete DataFlow Model), il est possible de générer différentes implémentations adaptées spécifiquement à certains environnements.

Pour cela, les caractéristiques des plates-formes d'exécution, préalablement capturées dans le *méta-modèle des composants et des plates-formes* (CPMM) sont injectées lors de la réécriture du DataFlow pour produire un PSM (Platform Specific Model). La modélisation fine des plates-formes d'exécution pour la génération d'implémentations est déjà étudiée [57] (cf. section 1.5.2). Nous l'avons adaptée aux contextes d'exécution des chaînes de traitements d'analyse intensive de données (clusters, PaaS).

Par ailleurs, nous avons approfondi l'étude de l'exploitation du mode de *cloud computing Containers as a Service* (CaaS) pour le déploiement des workflows. Les composants diffusés dans le répertoire de Galaxy utilisent ponctuellement des containers mais aucun formalisme adapté n'est pour l'instant proposé (cf. section 2.6.2). Nous avons donc élaboré pour le CaaS une spécification de workflows originale, nommée A-SCDFM (Autonomous Semi-Concrete DataFlow Model), qui référence un environnement portable pour chaque composant. De cette façon, on peut garantir un déploiement automatique sur toute plate-forme compatible sans que les outils métiers y soient initialement déployés. Il s'agit d'un des apports essentiels de cette étude puisque on peut ainsi notablement simplifier le déploiement des workflows. On obtient de cette façon des spécifications directement interprétables et réutilisables. La diffusion et l'échange des workflows sont ainsi grandement simplifiés. Les spécifications A-SCDFM peuvent être agrégées dans un répertoire de workflows associé à un répertoire de containers comme BioShadock (cf. section 4.4.3), qui inclut les dépendances logicielles.

Cette proposition contribue largement à lever le verrou technique de la réutilisation des workflows. Son adoption accélérerait la diffusion et la mise à disposition des analyses de données *in silico*. Elle est adaptée à des domaines comme la bioinformatique où l'on pratique largement le développement communautaire.

6.1.4 Un nouveau WfMS intégratif

Enfin les artefacts issus de nos propositions ont été implémentés et regroupés pour produire un nouveau WfMS orienté conception. Nous l'avons qualifié d'*intégratif* car il exploite une partie de la connaissance du domaine d'application par l'usage de métamodèles. Par rapport aux outils existants [62, 63, 82], cet environnement logiciel simplifie grandement la création rapide d'acteurs à partir de lignes de commande (cf. section 2.3.1) et leur orchestration. Il est capable, à partir d'une même spécification, de produire différents types d'implémentations et modes de mise à disposition. Ainsi de multiples usages pour plusieurs publics sont conciliables. De ce fait, nous pensons que ce nouveau type de WfMS, conçu à la fois pour les producteurs et les consommateurs de services, contribue à favoriser une conception plus efficace des workflows d'analyse de données, notamment intensive, tout en garantissant leur capacité à être diffusés et utilisés simplement. D'autre part, il est le seul WfMS à notre connaissance à proposer l'intégration guidée de l'extraction du parallélisme de données pour le traitement des grandes masses de données.

6.2 Perspectives

Nous proposons ici des pistes d'amélioration des résultats de nos travaux sur les WfMS et évaluons leurs perspectives d'application.

Tout d'abord, la recherche de la production d'implémentations de workflows performantes est une des préoccupations qui ont motivé ce travail. Pour cela, nous avons conçu un modèle de calcul capable d'exploiter efficacement les niveaux de parallélisme les plus

communément extractibles. Cependant, nous n'avons pas abordé le contrôle de l'usage du parallélisme interne aux composants. En effet, en bioinformatique, la nature hétérogène des composants, souvent conçus par de multiples intervenants est un obstacle à la généralisation de cette approche. Celle-ci nécessiterait, soit la mobilisation des concepteurs, soit un laborieux travail de modifications des codes a posteriori. Néanmoins, dans de futurs travaux, nous pourrions nous pencher sur ce problème et, par exemple, évaluer comment normaliser l'usage par les composants de librairies de gestion du parallélisme comme MPI [135] pour pouvoir contrôler leur paramétrage par un mécanisme générique du moteur de workflows.

Par ailleurs, la rapidité de l'exécution de notre modèle UDFAS peut probablement encore être améliorée. En effet, des pistes d'optimisation existent. Nous pensons en particulier à l'intégration de la technique de fusion des nœuds de Bic [48] (cf. section 1.3.3.2). L'intégration dans notre moteur de workflows d'une telle méthode de génération des jobs basée sur la décomposition du DataFlow en sous-graphes permettrait probablement d'obtenir un gain de performance. Celui-ci serait maximum pour des workflows essentiellement composés de séquences linéaires d'exécutions, fortement représentés en bioinformatique. Cette méthode, intégrée comme un raffinement d'UDFAS, pourrait contribuer à limiter le coût de l'architecture SOA et de l'ordonnancement (cf. section 5.1).

De plus, l'apport de cette optimisation devrait également être étudié dans différents contextes d'exécution:

— Dans le cas d'un cluster partagé entre de nombreux utilisateurs, les instances de workflows qui produisent de nombreux jobs, parce qu'elles sont composées de nombreux acteurs, peuvent provoquer une perte de priorité de l'utilisateur. Dans ce cas, la fusion de plusieurs étapes du DataFlow dans une même tâche est efficace.

— Dans le cas d'un déploiement sur une infrastructure de *cloud computing* commerciale, le modèle est différent. En effet, dans ce cas, la quantité de ressources allouables n'est pas limitante. Cependant un équilibre doit être trouvé entre le coût de l'infrastructure mise à disposition, la performance recherchée et la performance potentielle dépendante du parallélisme intrinsèque à l'application [136]. Ces deux contextes d'exécution différents illustrent l'intérêt d'étudier comment générer différentiellement le parallélisme exprimable dans le DataFlow.

D'autre part, nous pouvons poursuivre l'élaboration du langage DataFlow sur lequel nous avons travaillé. Son expressivité est suffisante pour de multiples cas d'utilisation. Néanmoins des améliorations peuvent encore être apportées:

— Tout d'abord, nous gérons les conditions par une méthode de passage des valeurs *null* (cf. section 3.2.1) mais il existe depuis longtemps de nombreuses représentations graphiques des conditions facilement intelligibles, comme celle des diagrammes flowcharts [137]. Ainsi, l'introduction de représentations graphiques

correspondant à des structures syntaxiques familières comme le *if-then-else* de nature impérative, pourrait être étudiée. En effet, ces représentations ne sont pas contradictoires avec le maintien du mécanisme actuel.

— En outre, nous n'utilisons actuellement que le passage par référence pour intégrer dans un workflow maître des acteurs composites représentant des sous-workflows (cf. section 3.2.1). Nous pourrions définir différents modes d'intégration de ces composants dans le DataFlow. Ainsi, le choix entre intégration par copie ou par référence serait laissé à l'utilisateur. L'usage de références a l'avantage de faciliter la gestion des composants et notamment les mises à jour. L'intégration par copie offre, quant à elle, la possibilité de personnaliser l'acteur composite dans chaque workflow maître. Peut être qu'un mécanisme de spécialisation basé sur l'héritage et associé à un système de versioning pourrait combiner l'intérêt des deux méthodes.

— Un autre point d'amélioration potentiel est la gestion des constantes. En effet, actuellement, lorsqu'il est nécessaire de paramétrer plusieurs acteurs d'un workflow avec la même valeur constante, le mécanisme des ports que nous avons mis en œuvre peut rendre inutilement complexe la vue du DataFlow. Cela limite la compréhension du traitement. Une solution envisageable est utilisée dans Taverna [64]. Il s'agit du simple masquage des ports indésirables. L'implémentation du modèle de calcul ne change pas et seule la vue est altérée. Une autre solution possible pour l'implémentation de constante pourrait consister en l'ajout d'un tableau associatif clefs-valeurs correspondant à l'ensemble des constantes disponibles pour tous les acteurs d'une instance d'un workflow. Nous notons que la notion de constante est implémentée dans certains langages d'orientation fonctionnelle, les langages *fonctionnels impératifs* comme LISP, dans lesquels il est permis d'effectuer des affectations et que la théorie du lambda calcul n'exclut pas l'utilisation de constante [138]. Nous pourrions ainsi introduire cette notion sans altérer les propriétés formelles de notre langage afin de garantir la possibilité de valider, analyser, voire de réécrire nos modèles (cf. section 6.1.1).

Nous avons également montré dans cette étude l'intérêt d'une approche orientée modèle dans la conception de workflows.

D'une part, c'est par la transformation de notre modèle de workflow UDFAS, considéré comme un PIM (Platform Independant Model), que l'on peut produire des implémentations de workflows PSM (Platform Specific Model), chacune dédiée à une plateforme d'exécution particulière. Pour l'instant, nous avons ciblé des machines isolées ou des clusters de calcul avec différents ordonnanceurs. Cependant, nous aimerions investiguer comment formaliser le mécanisme de transformation du PIM pour qu'il puisse produire des PSM dédiés à des environnements d'analyse de type DISCS (Data-Intensive Scalable Computing Systems) comme Hadoop [11] ou Spark [17]. En effet les DISC présentent des caractéristiques interessantes pour le traitement des grandes masses de données. Plus précisément, nous pensons à la production de PSM par injection de méthodes inhérentes à ces environnements et assimilables à des DFF. Ainsi, pour certains types de données NGS, des librairies incluant ces méthodes sont déjà disponibles pour

Hadoop [139, 12, 140] et pour Spark [18]. Néanmoins, pour chaque type d'environnement d'exécution visé, la structuration d'un corpus de méthode DFF plus large est souhaitable. Des efforts doivent encore être réalisés pour que notre approche d'intégration du parallélisme de données dans les workflows puisse être généralisable et couvrir l'ensemble du domaine d'application.

D'autre part, l'usage de l'approche orientée modèle présente des potentialités non encore explorées pour la production d'implémentations performantes. Ainsi, l'automatisation totale de la génération du parallélisme en fonction du contexte d'exécution constitue une perspective particulièrement intéressante. Pour cela, une solution consisterait à représenter les propriétés liées aux capacités des plates-formes d'exécution (CPU, RAM, nombre de cœurs...) et des besoins en ressources des outils constitutifs des acteurs. Pour un outil donné, on pourrait par exemple prendre en compte la relation entre quantité de données d'entrée, RAM, CPU et espace de stockage utilisé, même si nous remarquons que cette relation n'est prévisible que dans certains cas. Le *Méta-Modèle des Composants et des Plates-formes* (CPMM) (cf. figure 3.13) que nous avons proposé serait alors enrichi avec ces informations. Le système mettrait ensuite à profit le CPMM pour paramétrer automatiquement les méthodes Split DFF (nombre de chucks de données...) et le nombre de nœuds ou cœurs maximums exploités pour le parallélisme de tâches par le moteur de workflows. En conséquence, gràce à cette possibilité d'automatisation, il deviendrait envisageable d'exploiter dans un DataFlow le parallélisme de données gros grain de façon *implicite*, au même titre que le parallélisme de tâches. Il en découlerait un gain notable dans la facilité de conception des analyses de données massives à haut-débit.

D'autres possibilités sont également offertes par cet enrichissement du modèle CPMM:

— Ce méta-modèle pourrait être utilisé pour définir finement les caractéristiques de l'environnement d'exécution qui sont nécessaires et suffisantes pour le bon déroulement d'un workflow particulier. On pourrait ainsi déterminer automatiquement l'attribution des ressources sur un cluster particulier. Dans le cadre du *cloud computing*, cette approche pourrait faciliter la génération d'environnements d'exécution dimensionnés de façon optimale et donc avec un coût d'exploitation réduit.

— Comme nous l'avons vu, l'approche orientée modèle, nous a aussi permis d'expérimenter des fonctionnalités de génération de code (cf. section 5.1) à partir d'un modèle SCDFM. On peut de cette manière produire des scripts portables, exécutables indépendamment de l'environnement du WfMS. Ainsi, le concepteur de workflows ne serait plus prisonnier d'une unique solution logicielle. Il pourrait s'affranchir de l'environnement du WfMS et garder la possibilité de manipuler une implémentation sous forme de code. On peut supposer que l'intégration de la génération de code dans les WfMS favoriserait l'acceptation de l'approche workflow auprès d'une audience technique. En effet, la possibilité de "*roll-back*" vers un environnement classique faciliterait l'adhésion des scientifiques et des développeurs chargés de l'implémentation des méthodes d'analyse. Les concepteurs de composants pourraient ainsi passer de la ligne de commande aux modèles de workflow et

réciproquement. Pour cela, il pourrait être nécessaire d'intégrer dans l'environnement de conception des mécanismes de transformation bi-directionnelle [141].

Une des applications principales de nos travaux concernent la mise en place de répertoires communautaires de workflows facilement échangeables.

Il est probable que plus la couverture fonctionnelle d'un répertoire de workflows partagés est importante plus le nombre de contributions potentielles est élevé. A partir d'une certaine couverture, un mécanisme de cercle vertueux communément appelé *effet réseau* doit pouvoir s'enclencher. Ce mécanisme est évoqué par Goble concernant le répertoire myExperiment [5] et a déjà été étudié dans le cas des plates-formes PaaS par Scholten [142]. Cet auteur cite l'exemple de la plate-forme Salesforce [143] qui facilite le passage des intervenants du rôle de client, consommateur de service, à celui de producteur de service. De la même manière, si l'environnement du WfMS, offre à chaque intervenant la possibilité de contribuer lui-même à la capture des workflows ou des composants, celui-ci peut passer progressivement du rôle de consommateur à celui de producteur de services. Il peut alors participer à l'extension de la couverture fonctionnelle du répertoire qui bénéficie ainsi d'un effet réseau accru. De ce fait, un WfMS orienté à la fois conception et exécution et qui cherche à fédérer l'audience technique et les utilisateurs finaux est probablement un outil de choix pour faciliter l'édification d'un répertoire communautaire de workflows.

En outre, nous avons introduit dans les représentations SCDFM et A-SCDFM des caractéristiques de généricité, d'aptitude à la transformation, d'adaptation à de multiples plates-formes d'exécution et de simplification du déploiement pour cela. Gràce à ces propriétés, ces spécifications sont particulièrement favorables à la capitalisation communautaire de la connaissance des processus d'un domaine applicatif. L'élaboration d'un répertoire de workflows réellement ré-exploitables est de ce fait facilitée. La spécification A-SCDFM favorise tout particulièrement l'ubiquité des chaînes de traitements. Elle inclut la représentation des dépendances et garantit de cette façon la diffusion de processus d'analyse *in silico* directement ré-instanciables.

Les perspectives de son exploitation sont multiples:

— Tout d'abord, un répertoire communautaire de workflows basé sur ce type de spécification favoriserait la capacité à échanger et à reproduire les analyses de données d'autant plus que le modèle de workflow peut s'accompagner des données (entrées types et résultats attendus). Cette association entre modèle de workflow déployable et références de données pourrait déboucher sur la création de *benchmarks actualisés et automatisés*. Le besoin existe en bioinformatique, comme le montre l'existence d'initiatives d'évaluation de différents types de méthodes de traitement de données NGS. Elles concernent aujourd'hui les logiciels d'assemblage [144] mais également l'analyse méta-génomique [145]. A chaque fois qu'un nouvel outil est intégré au benchmark, toutes les applications à comparer, disponibles dans des containers, sont exécutées simplement, sur le même jeu de données d'entrée. Les auteurs n'utilisent pas pour l'instant de WfMS et se contentent de

définir un protocole uniforme d'encapsulation et d'appel des logiciels. Cette procédure est appliquée par les équipes de développement des différents outils à évaluer. L'intégration des outils dans un répertoire de containers comme BioShadock puis une formalisation du benchmark suivant un modèle de type A-SCDFM simplifierait probablement la mise en œuvre. Finalement le benchmark pourrait être directement produit à partir d'un répertoire de workflows.

— Par ailleurs, nous pourrions proposer l'intégration de certaines caractéristiques du modèle A-SCDFM (DSURI, typologie DFF/UDF, URI de container) dans des spécifications génériques et à vocation de standard comme celles issues du récent groupe de travail *Common Workflow Language* [146]. L'adoption de futures spécifications ouvertes inspirées de A-SCDFM par les contributeurs de répertoire comme le Toolshed de Galaxy [114], pourrait constituer un pas décisif permettant l'interopérabilité des WfMS et la généralisation de l'usage de l'approche workflow en bioinformatique, mais aussi, à terme, dans d'autres disciplines scientifiques.

La diffusion de l'usage de ce type de formalisme dans la communauté disciplinaire permettrait de capturer plus largement les processus métiers et de gérer plus efficacement cette connaissance mise en commun. Par la suite, nous pourrions également mieux tirer parti des outils d'analyse sémantique, orientés aide à la composition [147, 120] ou travailler sur la découverte de processus métiers [88] à une échelle suffisamment large pour être pertinente.

Annexe A

Publications

Conférences internationales avec comité de lecture et proceedings

— F. Moreews and D. Lavenier, "Seamless Coarse Grained Parallelism Integration in Intensive Bioinformatics Workflows," in *Proceedings of the 20th European MPI Users' Group Meeting*, EuroMPI '13, (New York, NY, United States), pp. 277–282, ACM, 2013

— J. Piat, F. Moreews, O. Collin, D. Lavenier, and A. Cornu, "SLICEE: A Service oriented middleware for intensive scientific computation," in *SERVICES 2011*, July 2011

Conférences internationales avec comité de lecture

— F. Moreews, O. Sallou, Y. Le Bras, G. Marie, C. Monjeaud, T. A. Darde, O. Collin, and C. Blanchet, "A curated Domain centric shared Docker registry linked to the Galaxy toolshed," in *Galaxy Community Conference 2015*, (Norwich, United Kingdom), July 2015

— F. Moreews, J. Piat, and O. Sallou, "OBIWEE : an open source bioinformatics cloud environment," in *BOSC 2011 - 12th Annual Bioinformatics Open Source Conference*, (Vienne, Austria), July 2011

Autre

— F. Moreews, Y. Le Bras, O. Dameron, C. Monjeaud, and O. Collin, "Integrating GALAXY workflows in a metadata management environment," in *Galaxy Community Conference*, (Baltimore, United States), July 2014 (poster)

Seamless coarse grained parallelism integration in intensive bioinformatics workflows

Francois Moreews
GENSCALE
IRISA/INRA
Rennes, France
fmoreews@irisa.fr

Dominique Lavenier
GENSCALE
IRISA/INRIA
Rennes, France
lavenier@irisa.fr

ABSTRACT

To be easily constructed, shared and maintained, complex in silico bioinformatics analysis are structured as workflows. Furthermore, the growth of computational power and storage demand from this domain, requires workflows to be efficiently executed. However, workflow performances usually rely on the ability of the designer to extract potential parallelism. But atomic bioinformatics tasks do not often exhibit direct parallelism which may appears later in the workflow design process.

In this paper, we propose a Model-Driven Architecture approach for capturing the complete design process of bioinformatics workflows. More precisely, two workflow models are specified: the first one, called design model, graphically captures a low throughput prototype. The second one, called execution model, specifies multiple levels of coarse grained parallelism. The execution model is automatically generated from the design model using annotation derived from the EDAM ontology. These annotations describe the data types connecting differents elementary tasks. The execution model can then be interpreted by a workflow engine and executed on hardware having intensive computation facility.

Categories and Subject Descriptors

J.3 [**LIFE AND MEDICAL SCIENCES**]: Biology and genetics; D.2.2 [**Design Tools and Techniques**]: Flow charts; D.1.3 [**Concurrent Programming**]: Parallel programming; C.4 [**PERFORMANCE OF SYSTEMS**]: Modeling techniques

General Terms

Design,Performance

Keywords

Scientific workflow, dataflow, model transformation,parallelism, map-reduce, ontology, model-driven, bioinformatics

PBio 2013: International Workshop on Parallelism in Bioinformatics
September 15-18, 2013 - Madrid, Spain

1. INTRODUCTION

Bioinformatics applications challenge today's computation resources by raising the amount of data to process and computation requirement to a new level. As an illustration, many algorithms used for Next Generation Sequencing (NGS) data processing, like genome assembly or polymorphism discovery, are computationally intensive and have to deal with a huge amount of data.

The common answer to such challenge is to use large storage facilities associated with classical computer clusters that combine the processing power of multiples machines. A job scheduler is then in charge of dispatching the processing demand onto the available processing resources. Such architecture takes advantage of coarse grain parallelism to speed-up computation. This parallelism can either be used by running multiple applications in parallel or by designing the application in such way that it can be divided into smaller grain tasks. Even if more and more tools like mpiBlast [5] use distributed computational resources like cluster-nodes or CPU-cores, the data parallelism pattern often need to be manually coded. For this purpose, APIs that eases the implementation of coarse grain data parallelism patterns have been developped [4].

Bioinformatics analysis usually consists in writing a script calling heterogeneous specialized softwares provided by the research community. Such "pipelines" are usually described as a sequence of operations represented as a dataflow. In that case, applications are modeled as a network of operations connected through their data dependencies. Such representations exploit the available parallelism by analyzing data dependencies between operations.

Workflow management systems (WMS) used in bioinformatics were often limited to educational or low throughput usage. With the development of middlewares designed to integrate the power of intensive computation infrastructure in client softwares, the orchestration of bioinformatics services deployed on a computation intensive production environment became realistic [15]. Among these middlewares, the DRMAA API [2] standardizes the access to job shedulers and the Opal toolkit [11] wraps command lines and manages job posting through web services. Thus, nowadays, WMS that enable cluster or cloud job submissions have emerged as an interesting solution to face the high throughput sequencing challenge.

Taverna [18] is a service oriented graphical workflow authoring and execution tool, able to orchestrate remote web services or local components. Associated with appropriate OPAL or PBS middleware clients embedded in actors, Tav-

erna, but also Kepler [17], can be run on clusters. But many actors must be laboriously defined by the designer for a single intensive computation job execution (upload , submission, wait, get result, download). The resulting graph representation contains many technical actor nodes that do not clearly display the main analysis steps.

In contrast, Galaxy [8] has been successfully adopted by the scientific community as a bioinformatics production environment. Galaxy can integrate intensive computation tools and support a large collection of predefined bioinformatics components. It is used more as a tools repository where each task is independently manually launched than for its orchestration capabilities. Galaxy has proved to be a solution for workflow prototyping using, for example, a conversion of the user activities as a "pipeline". Galaxy workflow module, like many dataflow based scientific WMS, proposes to ease the workflow specifications through a graph editing GUI and can schedule and run workflows, addressing clusters through job scheduler.

All these tools authorize the creation of atomic tasks, that internally manage parallelism. Thus, when available, coarse grain data parallelism is based on the implementation of each components. But these environments do not propose a way to integrate coarse grain data parallelism without hard coding or complex manipulations. They do not offer a general mechanism for data parallelism extraction.

In contrast, some generic grid computing oriented environments like P-GRADE [7] offer a unified access to multiple complementary middlewares, each one dedicted to a level of parallelism extraction. The configuration of these tools is user-defined. Building highly intensive bioinformatics workflow still relies on the technical ability of the designer to take advantage of the available data parallelism.

The purpose of this work is to greatly simplify the design process of bioinformatics workflows. Integrating a general mechanism of data parallelism extraction from a captured workflow prototype remains challenging. The goal is to hide to the designers (bioanalyst people) time consumming and error prone tasks dedicated to technical aspect of parallel implementation. This seamless parallelism integration could not only speed up the treatments but also facilitate the design process and disseminate the usage of parallel workflows.

To achieve this objective, we propose an approach based on a Model-Driven Architecture allowing the designer to easily capture bioinformatics workflows using a high level model from which a parallel execution model is semi automatically derived.

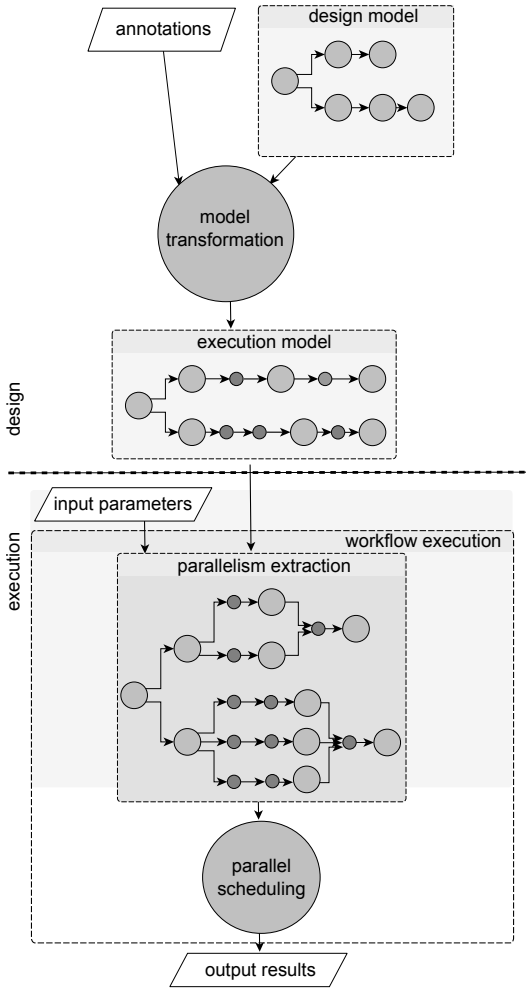


Figure 1: Overview of the workflow design and parallel execution steps

Figure 1 illustrates the approach: the user first specifies a workflow using a graphical interface. He connects bioinformatics tools as a dataflow graph. The user is asked to integrate annotations for specifying data types flowing between nodes (bioinformatics tasks) of the graph. From this graph, and using a model transformation, an execution graph is generated from which parallelism can be automatically extracted.

The rest of the paper is organized as follows: section 2 and 3 respectively present the workflow design model and the way the model is transformed. Section 4 describes the execution model. Section 5 discusses the integration of this Model-Driven approach in a new WMS dedicated to bioinformatics intensive data treatment and gives directions for future works.

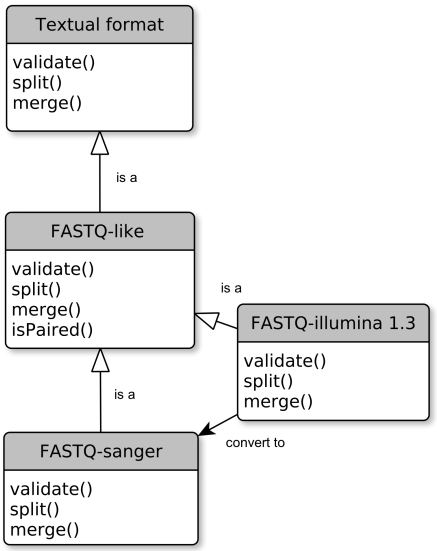


Figure 2: We use a data format hierarchy derived from EDAM ontology. Split and merge methods are implemented for each relevant data format.

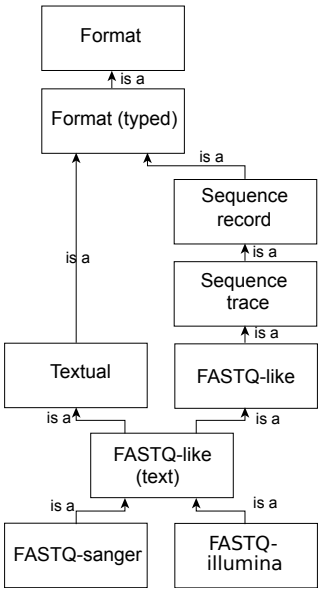


Figure 3: Example of the FASTQ-sanger data format hierarchy in EDAM (EMBRACE Data And Methods), an ontology of bioinformatics operations, types of data and formats.

2. THE DESIGN MODEL

To initiate the design process, we defined a "design model" that eases the capture of a worfklow, omitting technical tasks such as paralellization.

The workflow design model is a simple dataflow graph where processes are the nodes and the data dependencies the edges of a direct acyclic graph (DAG). Each node in the DAG is named actor. We use 3 majors classes of actors: input, execution, output. An execution actor wraps a script or a command line tool, that, without any additional semantic, will be seen as a black box, called here a user-defined function (UDF). Each edge represents a data dependency. An edge links a source actor output port to a target actor input port and represents a channel of data tokens. Actors can have multiple input and output ports. The scheduling aspects are implicit.

2.1 Prototype capture

During the prototype capture, only major processing steps are represented as UDF actors. Utilitary tools which perform operations related to data validation, transformation and conversion methods and which do not aggregate or generate additional knowledge are called here Data Format Function (DFF). DFF actors must be omitted during the prototype capture. Pre-existing tools and scripts that fit theses conditions are embedded within UDF actors following a template syntax.

2.2 Annotations

The input and output ports of the UDF actors need to be partially or fully annotated by the designer using a data format hierarchy provided by the framework (Figure 2). This additional semantic defines a strong data typing that enables the integration of DFF actors.

Predefined DFF actors are proposed to the designer as pre-processing and post-processing methods applied to the ports of an actor node. It is similar to a sequence of methods applied on the edges of the dataflow DAG. The resulting separation of workflow tasks between UDF and DFF permits to display a clear workflow "design" view (Figure 4). This "analysis" or "design" view represents only the input, output and UDF actors, related to the domain tasks (here mapping, assembly, snp calling...). DFF actors (validation, conversion and other utilities) remains masked. This results in a better overview and semantic analysis of the workflow. It is also a way to limit the proliferation of visible technical actors or data adaptors, sometimes called the "shim problem" [16]. We obtain a complete semi concrete workflow model that will subsequently be transformed in a fully executable model. We have generated the domain-specific data format hierarchy, derived from the EDAM ontology [9]. EDAM (EMBRACE Data And Methods) is an ontology of bioinformatics operations, types of data and formats (Figure 3). The vocabulary of terms and relations provided have already been used to classify tools and also for workflow composition purpose [13].

3. MODEL TRANSFORMATION

Model transformation is used in Model-Driven Architecture for code generation including automatic parallelisation [12]. It is commonly used in graphical capture of processes and has already been applied to workflow formats conversion [6]. We are not aware of any previous use of this method for a concrete coarse grain parallelisation of workflows.

Basically our approach depends on the prior definition of a library of efficient split and merge methods, related to most common data format types used in the application domain. To efficiently organise this set of utilities, the format hierarchy previously introduced is used (Figure 2). This set of utilities is manually created and associated to related formats within the format hierarchy.

For each format and its variants, a custom set of functions is defined including the implementations of the split and merge methods, with appropriated parameters. Split and merge methods are DFF. For an actor port tagged with a data format, all methods related to the data format and their predecessors in the hierarchy can then be used to semantically annotate the related edges of the design model (Figure 4-A). This annotation process is user-defined.

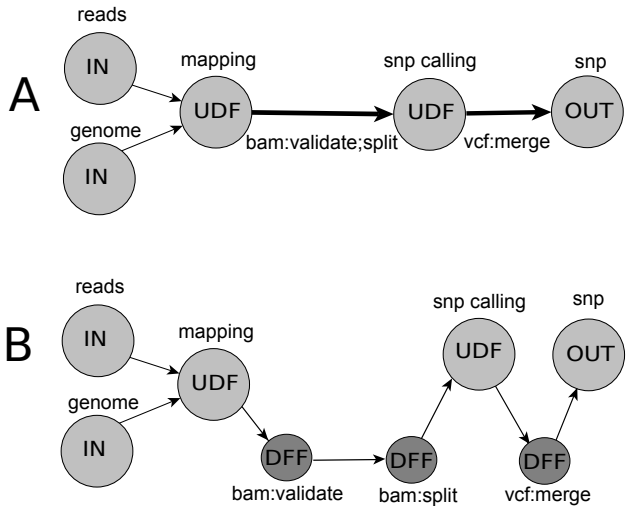


Figure 4: The definition of UDF and DFF actors enables the creation of two different dataflow views dedicated to design (A) or execution and monitoring (B).

A graph transformation mecanism, based on graph patterns, converts the annotated design model to an execution model, after checking the constraints. To each model corresponds a view, the "design view" (Figure 4-A) and the "execution view" (Figure 4-B). The "execution view" is a technical view where all actors are represented. UDF and DFF actors are all represented as nodes. This view is closer to the implementation and useful for monitoring execution. Because all DFF are generated as new actors in the execution model, the split and merge actors methods become represented as actors. This means that when input and output port data formats have been specified by user-defined annotations, a map reduce pattern is consequently applied using split and merge methods related to each data format (Figure 2).

Finally, the execution of the resulting model by a dedicated engine corresponds to the execution of a parallelised implementation of the captured workflow.

4. THE EXECUTION MODEL

The execution model enables the parallelism extraction and a high level of task scheduling. The execution model is also a dataflow with input, execution, and output actors (Figure 5-A), but each actor has 6 different states (undefined, initialized, submitted, done, error, finalized) (Figure 5-B). The states are defined to support asynchronous remote calls, a required pattern for long running job submissions. At the transition between states, a specific method is launched. The transition between the state "submitted" and the state "done" corresponds to the UDF, here a bioinformatics tool. Each edge represents a data dependency. An edge links an output port of a source actor to an input port of a target actor and represents a channel of data tokens. Actors can have multiple input and output ports. Each workflow input is represented as a list of tokens. Conditional statements like "if else" structures can be represented using the propagation of a null token.

A dataflow model consumes and emits data tokens [14]. Two data token classes have been defined, a simple string for short values and a dataset Uniform Resource Identifier (URI). A dataset URI is an identifier that abstracts a file or a group of related files, like all files generated by an execution. This unified data container can seamlessly be used by the orchestration layer that manipulates abstract data tokens or by the services called internally by actors for data movement and the generation of data subsets.

An actor can emit one job output URI per invocation. In other words, each execution of an actor returns one URI for one command execution result set. At the scheduler level, this means that the number of tokens is predictable, allowing static scheduling and compliance with the Synchronous Data Flow model (SDF) [14].

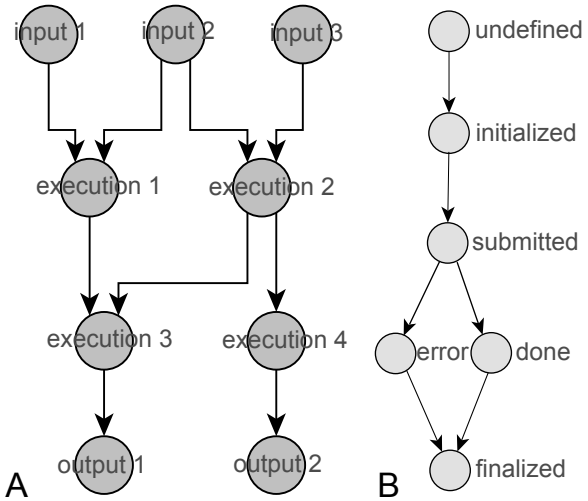


Figure 5: The execution model corresponds to a dataflow DAG (A) where execution actors wrap user-defined functions (UDF). Each actor has different states (B) to enable the control of asynchronous remote calls, directly during the orchestration.

4.1 Parallelism extraction

We focused on coarse grained parallelism. Defining implementation specific parallelism is out of the scope of this work but existing ones can be wrapped as UDF actors. For example, MPI or GPU implementation can be called using a particular job scheduler queue defined by an actor property.

We now explains how the execution model specifically targets different complementary levels of parallelism using:
(i) iterations (ii) data dependencies (iii) map-reduce

4.1.1 Iterations

All elements of a list of input tokens can be submitted in parallel. When an actor depends on multiple parameters (ports), each one linked to a list of input tokens, we can compute all parameter sets corresponding to all independent job submissions. As an example, it is similar to multiple parallel workflow instances execution with different parameter sets.

The sequence of all parameter sets U of an actor is obtained by computing a cartesian product of all lists of input tokens connected to its input ports (Figure 6). In the case of an UDF actor *F1* with two input ports, respectively populated with the lists of input tokens R and S, it corresponds to the generation of all possible pairs formed from the two lists, $U = R \times S$. The resulting jobs are $F1(U)$. Each element of U is a set of parameters which can be applied to the function *F1* in parallel.

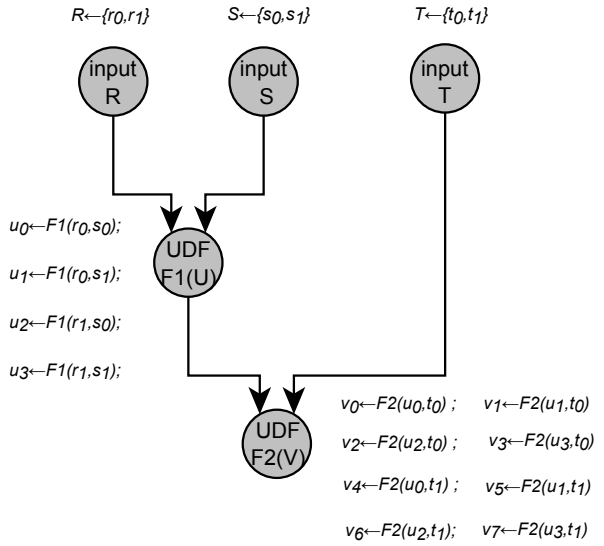


Figure 6: Static computation of UDF actors job parameter sets.

4.1.2 Data dependencies

Parallel executions among workflow actors is obtained by exploiting the graph topology. Following the dependency graph, any actor can be fired when all its predecessors have been fired. Each actor n associated with a set of parameters represents a job submission s. The current states of all job submissions are stored in a synchronized data structure. Reading the data structure holding the state of each defined job submission and the states of the related predecessors allows the scheduler to dynamically identify all job submissions that can be illegible for execution (algorithm 1). Then jobs can be submitted in parallel to a cluster job scheduler, following other implemented constraints, defined by the sheduling strategy.

Algorithm 1 parallelism extraction obtains from dynamic scheduling using the data dependencies defined by the DAG topology.

```
D ← setWorklowGraph() //workflow DAG
A ← getAllJobSubmissions(D)//all job submissions
for each n in D do
  if (n is UDFActor ) then
    P ← getPredecessors(n) //get all predecessors of n
    J ← getJobSubmissions(n) //get reserved job
    // submission array
    for each s in J do
      if (s is not submitted ) then
        for each m in P do
          t ← getLinkedJobSubmission(m, s)
          //each submission of n is related to another
          //job submission of the predecessor actor m
          if (t is finished ) then
            A(s) ← TRUE //Job is defined as available
            // for future launch
for each s in A do
  if (A(s) is TRUE ) then
    if (externalSchedulingCondition(s) is TRUE ) then
      p ← new thread()
      e ← fire(p, s) //execute the job submission in a
      //new thread. Its results in multiple parallel
      //job executions on the target cluster
```

4.1.3 Map-Reduce patterns

In many application cases, a simple Map-Reduce pattern leads to the extraction of a massive coarse grain data parallelism from the workflow prototype. As previously defined, split and merge methods are Data Format Function (DFF). Within the execution model, DFF actors do not differ from UDF actors. It means that after model transformation, according to the user-defined annotations, a map reduce pattern is automatically inserted in the dataflow. For example, the actor node $UDF(F1)$ is replaced by a sub graph :

$$split \rightarrow UDF(F1) \rightarrow merge$$

A split method and a merge method can be applied on the same UDF actor on different UDF actors. In this last case, all data chucks are simply grouped using a shared dataset URI, and, thanks to this abstract data token, transparently routed in the dataflow graph.

5. CONCLUSION

We have implemented a workflow engine based on our models. This engine can be integrated in a graphical WMS dedicated to intensive computation [1]. It generates an execution model from the design model, previously created with a web workflow design GUI (Figure 7), and subsequently executes the workflow [3].

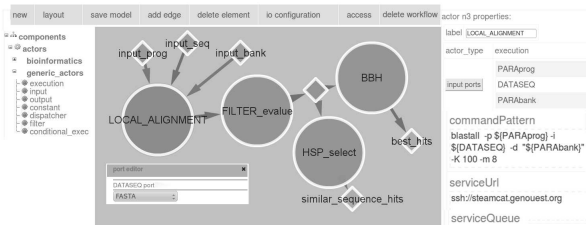


Figure 7: Web workflow design GUI. The DAG corresponds to an intensive bioinformatics workflow captured using a graphical editor. For execution, the underlying workflow engine integrates a dedicated middleware, SLICEE (Service Layer for Intensive Computation Execution Environment), embedded in actors

In this paper, we have proposed a modeling process of intensive bioinformatics workflows, closely related to the actual observed development process. The transformation of an annotated design model to an execution model enables the extraction of different levels of coarse grain parallelism with minimum human intervention. By this way, the conversion of a low throughput prototype to a workflow adapted to intensive computation can be more easily achieved. It could also result in a larger dissemation of coarse grain parallelism usage in bioinformatics treatments, especially within popular graphical WMS dedicated to non-technical users. In addition, we have defined a minimal classification of the workflow actors in two classes (UDF, DFF) that clearly highlights what can be automated or must stay user-defined.

However, the effectiveness of our approach depends on the structuration of a large corpus of utilitary methods related to bioinformatics types and formats. Major efforts still need to be made in those directions, especially, to study if the annotation process of the design model, actually user-defined, could be automated. It would also be worth to investigate how model transformations could be used to seamlessly target a larger scope of execution environments [10].

6. REFERENCES

[1] Bioinformatics Intensive Workflow Portal. *http://workflow.genouest.org*.

[2] Drmaa working group. *http://www.drmaa.org/*.

[3] Slicee - Service Layer for Intensive Computation. *http://vapor.gforge.inria.fr/*.

[4] P. Bui, L. Yu, and D. Thain. Weaver: integrating distributed computing abstractions into scientific workflows using python. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, pages 636–643, New York, NY, USA, 2010. ACM.

[5] A. Darling, L. Carey, and W.-c. Feng. The design, implementation, and evaluation of mpiBLAST. In *Proceedings of ClusterWorld Conference*, pages 13–15, San Jose, California, June 2003.

[6] J. Eder, W. Gruber, and H. Pichler. Transforming workflow graphs. In *Proceedings of the first international conference on interoperability of enterprise software and applications INTEROP-ESA 2005 ,Geneva, Switzerland*, pages 203–215, 2005.

[7] Z. Farkas and P. Kacsuk. P-grade portal: A generic workflow system to support user communities. *Future Generation Computer Systems*, 27(5):454 – 465, 2011.

[8] J. Goecks, A. Nekrutenko, J. Taylor, and T. Team. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biol*, 11(8):86, 2010.

[9] J. Ison, M. Kalas, I. Jonassen, D. Bolser, M. Uludag, H. McWilliam, J. Malone, R. Lopez, S. Pettifer, and P. Rice. EDAM: an ontology of bioinformatics operations, types of data and identifiers, topics and formats. *Bioinformatics*, 29(10):1325–32, May 2013.

[10] L. Jourdren, M. Bernard, M.-A. Dillies, and S. Le Crom. Eoulsan: a cloud computing-based framework facilitating high throughput sequencing analyses. *Bioinformatics*, 28(11):1542–3, June 2012.

[11] S. Krishnan, B. Stearn, K. Bhatia, K. Baldridge, W. Li, and P. Arzberger. Opal: Simpleweb services wrappers for scientific applications. In *Proceedings of ICWS '06. International Conference on Web Services, 2006.*, pages 823–832, 2006.

[12] O. Labbani, J. luc Dekeyser, P. Boulet, and E. Rutten. Introducing control in the gaspard2 data-parallel metamodel: Synchronous approach. In *Proceedings of the 1st International Workshop on Modeling and Analysis of Real-Time and Embedded Systems*, 2005.

[13] A.-L. Lamprecht, S. Naujokat, B. Steffen, and T. Margaria. Constraint-guided workflow composition based on the edam ontology. In *Proceedings of the Workshop on Semantic Web Applications and Tools for Life Sciences*, 2010.

[14] E. Lee and D. Messerschmitt. Static scheduling of synchronous data flow programs for digital signal processing. *IEEE Transactions on Computers*, 36(1), Jan. 1987.

[15] W. Li, S. Krishnan, K. Mueller, K. Ichikawa, S. Date, S. Dallakyan, M. Sanner, C. Misleh, Z. Ding, X. Wei, O. Tatebe, and P. Arzberger. Building cyberinfrastructure for bioinformatics using service oriented architecture. In *Proceedings of the sixth IEEE International Symposium on Cluster Computing and the Grid, 2006. CCGRID 06.*, volume 2, pages 8 pp.–39, 2006.

[16] C. Lin, S. Lu, X. Fei, D. Pai, and J. Hua. A task abstraction and mapping approach to the shimming problem in scientific workflows. In *Proceedings of SCC '09. IEEE International Conference on Services Computing, 2009.*, pages 284–291, 2009.

[17] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao. Scientific workflow management and the kepler system. *Concurrency and Computation: Practice and Experience*, 18(10):1039–1065, Aug. 2006.

[18] T. Oinn, M. Greenwood, M. Addis, M. N. Alpdemir, J. Ferris, K. Glover, C. Goble, A. Goderis, D. Hull, D. Marvin, P. Li, P. Lord, M. R. Pocock, M. Senger, R. Stevens, A. Wipat, and C. Wroe. Taverna: lessons in creating a workflow environment for the life sciences. *Concurrency and Computation: Practice and Experience*, 18(10):1067–1100, Aug. 2006.

Annexe B

Éléments de théorie des graphes

Nous reprenons ici des notions élémentaires de théorie des graphes abordée tout au long de notre travail.

Graphe

Un graphe est un ensemble de sommets ou nœuds qui peuvent être reliés par des arêtes. Les graphes constituent un formalisme précis pour modéliser des entités reliées par des relations.

Graphe orienté

Un graphe est orienté si les nœuds sont reliés par des liens orientés, nommés arcs.

Multigraphe

Un multigraphe ou pseudographe est un a graphe qui permet que deux sommets puissent être connectés par plus d'un arc.

Graphe biparti

Un graphe biparti est un graphe dont les nœuds appartiennent à un ensemble U ou un ensemble V tel que chaque arête ayant une extrémité associée à un nœud de U ait une extrémité associée à un nœud de V.

Chemin

Dans un graphe, un *chemin* de longueur $k \geq 0$ d'un sommet s à un sommet t est une séquence $(v_0, v_1, \ldots, v_k)$ de sommets telle que $v_0 = s$, $v_k = t$ et pour chaque i dans $1 \leq i \leq k$ le graphe contienne un arc (v_{i-1}, v_i).

Cycle

Un *cycle* de longueur $k \geq 1$ est un chemin de longueur ≥ 1 vers le sommet lui-même.

DAG

Un graphe qui ne contient pas de cycles est appelé *graphe acyclique*; S'il est orienté, il est appelé *Graphe Orienté Acyclique* (ou *DAG* pour Directed Acyclic Graph).

Complexité Cyclomatique

La complexité cyclomatique ou complexité de McCabe est une métrique indépendante du langage qui permet de mesurer la complexité d'un composant d'un programme. Elle mesure le nombre de chemins indépendants dans le composant modélisé sous la forme d'un graphe.

Composante connexe

Une composante connexe d'un graphe non orienté est le sous-graphe maximal tel que quels que soient les sommets s et t , il existe toujours un chemin entre s vers t.

Clique

Une clique d'un graphe non orienté est le sous-graphe maximal tel que quels que soient les sommets s et t , il existe toujours un arc entre s vers t.

Homomorphisme de graphe

un homomorphisme f d'un graphe $G = (V1, E1)$ vers un graphe H=(V2,E2), noté $f : G \to H$, est une fonction, des sommets de G vers les sommets $f : V \to V'$ de H telle que l'image de toute arête de G est une arête de H:

$\{u, v\} \in E1$ implique que $\{f(u), f(v)\} \in E2$.

L'homomorphisme associe à chaque sommet de H un sommet de G tel que que si u et v sont deux sommets voisins de H, alors leurs images sont voisines dans G. on peut dire également que G se projette dans H.

Isomorphisme de graphe

Soit $G = (E_1, V_1)$ et $H = (E_2, V_2)$ deux graphes.
Un isomorphisme entre deux graphes G et H est une bijection Φ entre les sommets V1 et V2 qui préserve les arêtes E1:

telle que deux sommets u et v de G sont adjacents dans G seulement et seulement si $\Phi(u)$ and $\Phi(v)$ sont adjacents dans H.

$$(u, v) \in V_1 \iff (\Phi(u), \Phi(v)) \in E_2$$

Transformation de graphes

La transformation de graphes désigne l'application d'une séquence de règles de transformation sur un graphe donné menant à la modification du graphe.

Une *transformation de graphe* depuis un état initial G à un état final H, noté par $G \overset{p(o)}{\Longrightarrow} H$, est donnée par un homomorphisme de graphe o: $L \cup R \to G \cup H$, appelé occurrence, tel que

— $o(L) \subseteq G$ et $o(R) \subseteq H$
, c'est-à-dire que le coté gauche de la règle est inclus dans l'état initial G et le coté droit dans l'état final H, et

— $o(L \backslash R) = G \backslash H$ et $o(R \backslash L) = H \backslash G$, c'est-à-dire, que la partie de G qui est supprimée correspond aux éléments de L n'appartenant pas à R et, symétriquement, la partie de H qui est ajoutée correspond aux éléments de nouveau dans R.

Annexe C

Les langages Fonctionnels: constructions syntaxiques

Nous décrivons ici des propriétés syntaxiques des langages orientés fonctionnels.

Fonction d'ordre supérieur Un fonction d'ordre supérieur est une fonction qui prend une ou plusieurs fonctions comme entrée et/ou renvoie une fonction.

Fonctions anonymes Les fonctions anonymes qui sont la base du Lambda-calcul qui constitue la base théorique de la programmation fonctionnelle. Une fonction anonyme est constituée de code réutilisable comme une fonction mais sans nom. Les fonctions anonymes sont passées en argument des fonction d'ordre supérieur pour réaliser un traitement.

Fermeture les fermetures sont des fonctions qui sont définies au sein du corps d'une autre fonction et qui partage des variables avec celle-ci. Elles sont utilisées comme paramètres pour adapter un traitement générique à un contexte particulier.

Listes en compréhension Les listes en compréhension sont des constructions syntaxiques qui définissent une nouvelle liste à partir d'un liste existante sur les éléments de laquelle on applique un filtre.

Elles sont utilisées pour définir par une syntaxe très compacte des sous ensembles d'éléments.

Pattern Matching Dans le contexte de la programmation fonctionnelle le *Pattern Matching* est une technique syntaxique permettant de différencier les traitements en fonction des propriétés des données ou de leur type.

Variable immuable Une variable est immuable si son état ne peut pas être modifié après sa création Dans de nombreux langages fonctionnels, les variables sont immuables

par défaut. Cela permet notamment l'exécution des fonctions sans effet de bord. Cela rend possible la parallélisation de données sans verrou. Dans certain cas, le compilateur peut analyser le code , déterminer les fonctions a priori les plus lentes puis automatiquement définir une exécution parallèle. Cela est beaucoup plus complexe dans un langage impératif ou chaque fonction peut modifier l'état associé à sa portée extérieure qui pourrait alors influencer le comportement d'autres fonctions.

Monades Les monades sont des patrons de conception de haut niveau qui permettent d'encapsuler des traits impératifs pour respecter la contrainte d'absence d'état des langages purement fonctionnels. Ces traits peuvent être par exemple les effets de bord ou les exceptions. On peut par exemple utiliser un patron de conception de type monade pour encapsuler un objet de type prédéfini dans un objet conteneur ayant d'autres attributs afin de gérer des opérations initialement non permises dans le langage de l'implémentation.

L'usage des monades présente plusieurs intérêts dont:

— L'analyse simplifiée du comportement d'un programme;

— La possibilité d'optimisations dont la parallélisation.

Annexe D

Taxonomie et usage des WfMS scientifiques

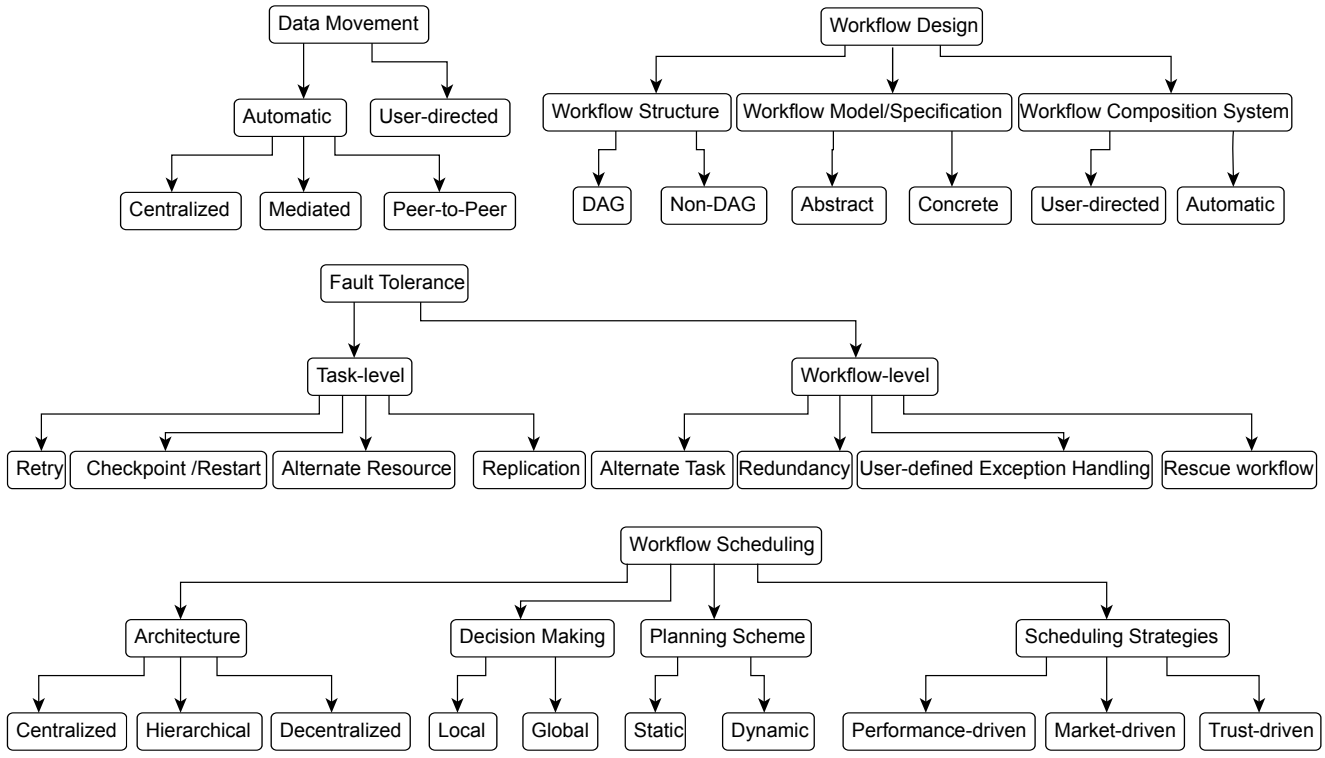


FIGURE D.1: Taxonomie des WfMS scientifiques pour la grille de calcul d'après Yu et Buyya [60]

TABLE D.1: Résumé des observations des méthodes de mise en œuvre des chaînes de traitements dans différents laboratoires et plates-formes de services dédiées à l'analyse de données bioinformatiques

organisation & tutelle	activité	groupe représentatif	discipline	usage (U) ou mise à disposition (M) de WfMS	conception d'applications en ligne de commande
LPGP, INRA	recherche	biologistes, bio-analystes	biologie dont génomique	U	rare
IGEPP, INRA	recherche	biologistes, bio-analystes	biologie dont génomique	U	rare
Dyliss, INRIA	recherche	chercheurs, algorithmiciens	bioinformatique, réseaux et séquences	-	oui
Genscale, INRIA	recherche	chercheurs, algorithmiciens	bioinformatique, NGS	-	oui
Genotoul, INRA	service	ingénieurs	bioinformatique	U-M	oui
Genouest, INRIA	service	ingénieurs et développeurs	bioinformatique	M	oui
Sigenae, INRA	service	ingénieurs et développeurs	bioinformatique	M	oui
Migale, INRA	service	ingénieurs	bioinformatique	M	oui
Bird, INSERM	service	ingénieurs	bioinformatique	M	oui
Abyms, CNRS & UPMC	service	ingénieurs	bioinformatique	U-M	

Annexe E

Notions de parallélisme

	instruction unique	**instructions multiples**	**programme unique**	**programmes multiples**
donnée unique	SISD	MISD		
données multiples	SIMD	MIMD	SPMD	MPMD

TABLE E.1: Types de parallélisme matériel d'après la taxonomie de Flynn

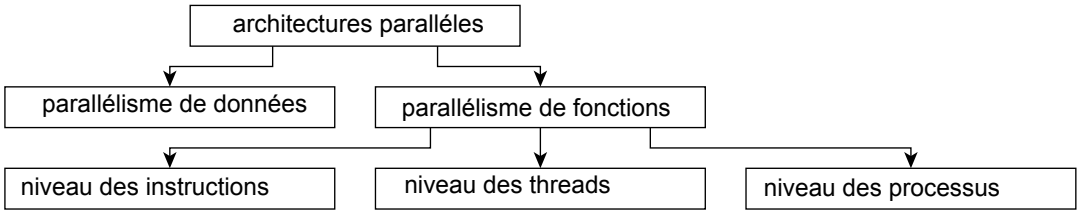


FIGURE E.1: Taxonomie moderne des architectures parallèles d'après Sima

Annexe F

Implémentation

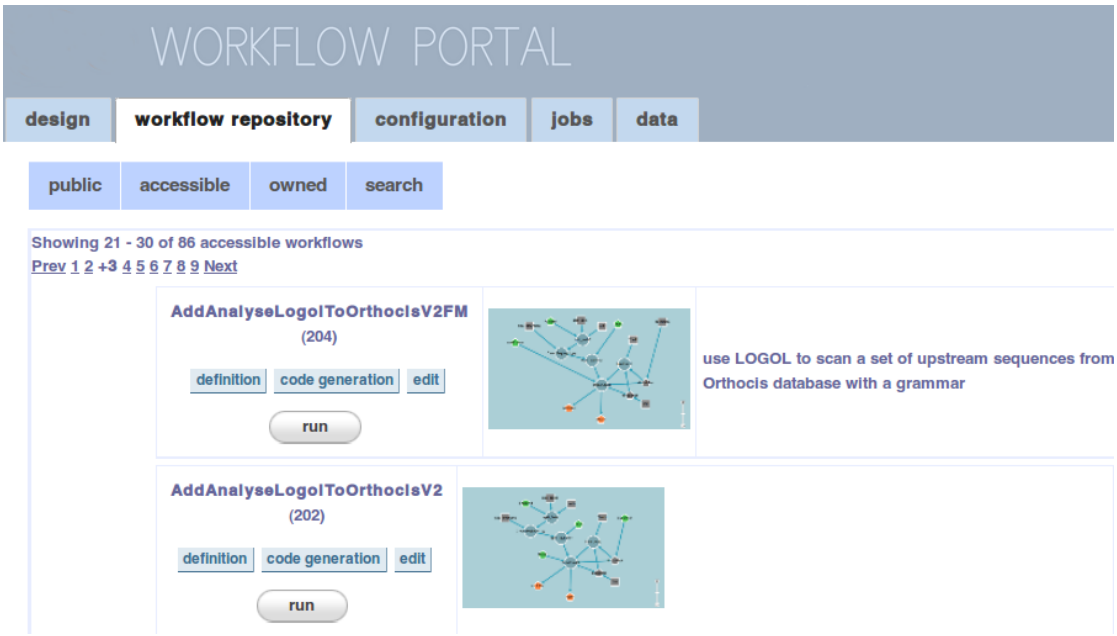


FIGURE F.1: Interface du WfMS développé, accessible sur http://workflow.genouest.org. Vue du Répertoire de workflows

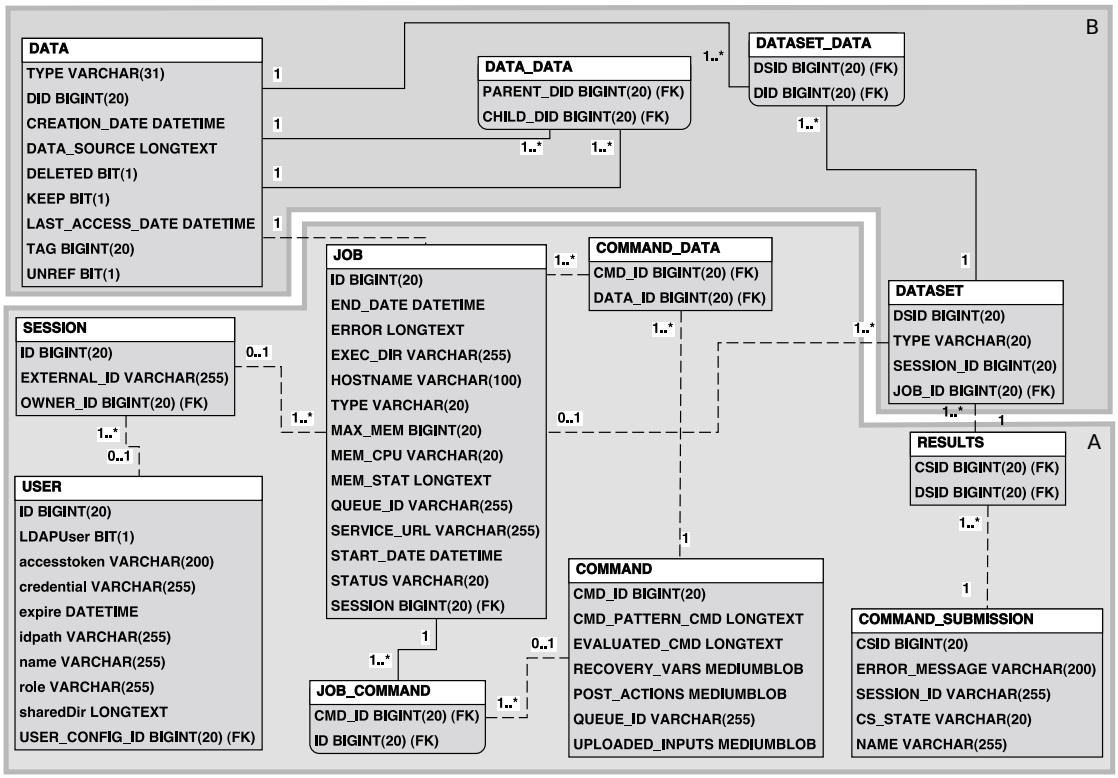


FIGURE F.2: Schéma de base de données assurant la persistance des jobs (A) et des références de données (B). Le stockage de ces informations permet notamment le suivi des états des traitements, la résolution des DSURI et des analyses de provenance.

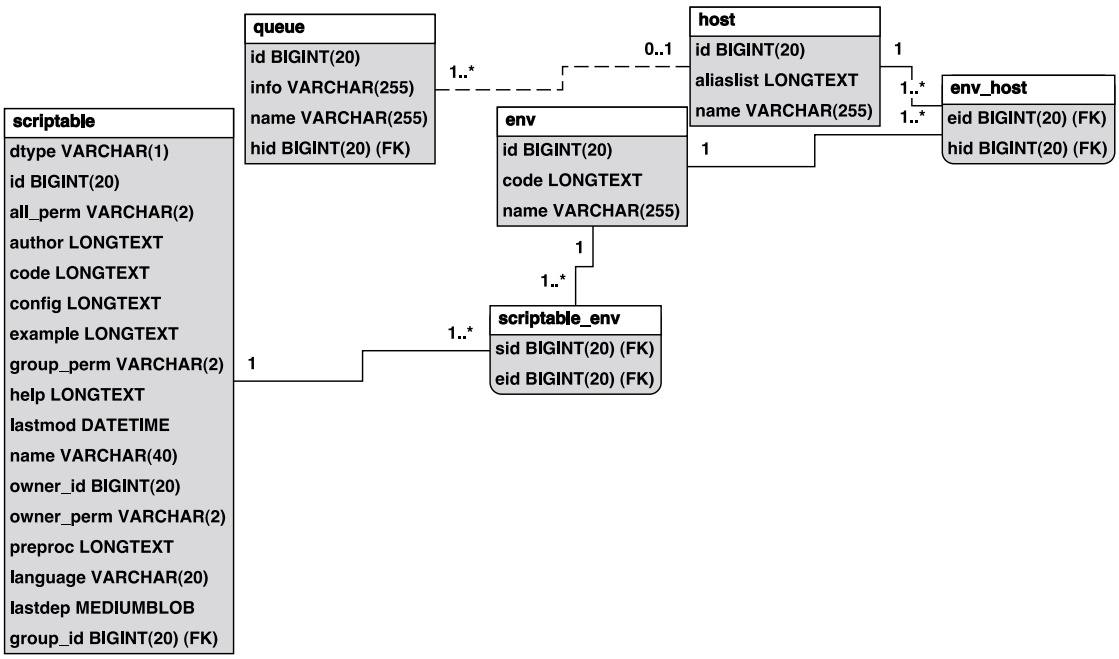


FIGURE F.3: Schéma de base de données représentant le CPMM et ses différents raffinements. Les entités de la classe *Scriptable* correspondent aux acteurs déployables.

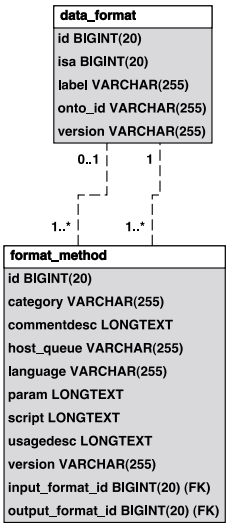


FIGURE F.4: Schéma de base de données représentant le méta-modèle partiel du domaine métier, incluant les types de données et les méthodes DFF associées.

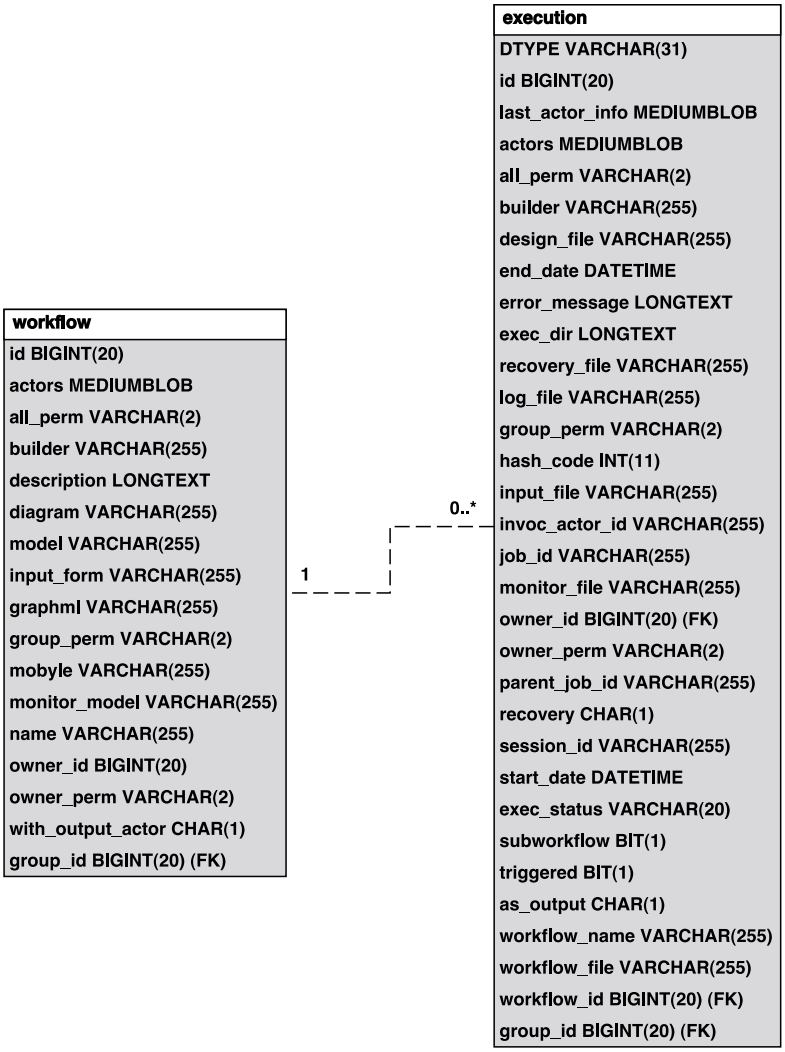


FIGURE F.5: Schéma de base de données des spécifications de workflow et leurs exécutions.

```
<user>
    <id>1</id>
    <identitys>
        <entry>
            <key>genotoul.inra.fr</key>
            <value enc="1">
                <id>fmoreews</id>
                <pass>74g35ea856621j1dq1f9hkf8mbfa0548</pass>
            </value>
        </entry>
        <entry>
            <key>steamcat.genouest.org</key>
            <value enc="1">
                <id>fmoreews</id>
                <pass>39d01ui805487b1da5f2egh9dbaf0357</pass>
            </value>
        </entry>
        <entry>
            <key>genocluster2.genouest.org</key>
            <value enc="1">
                <id>fmoreews</id>
                <pass>39d01ui805487b1da5f2egh9dbaf0357</pass>
            </value>
        </entry>
    </identitys>
    <LDAPUser>false</LDAPUser>
    <name>fmoreews</name>
    <sharedDir>default=~/.workflow,
               igrida=/temp_dd/igrida-fs1/{LOGIN}/SCRATCH/,
               genotoul=/work/{LOGIN}/.workflow,
               192.168=/omaha-beach/wtest
    </sharedDir>
</user>
```

FIGURE F.6: Représentation XML de la gestion d'un pool d'identités.

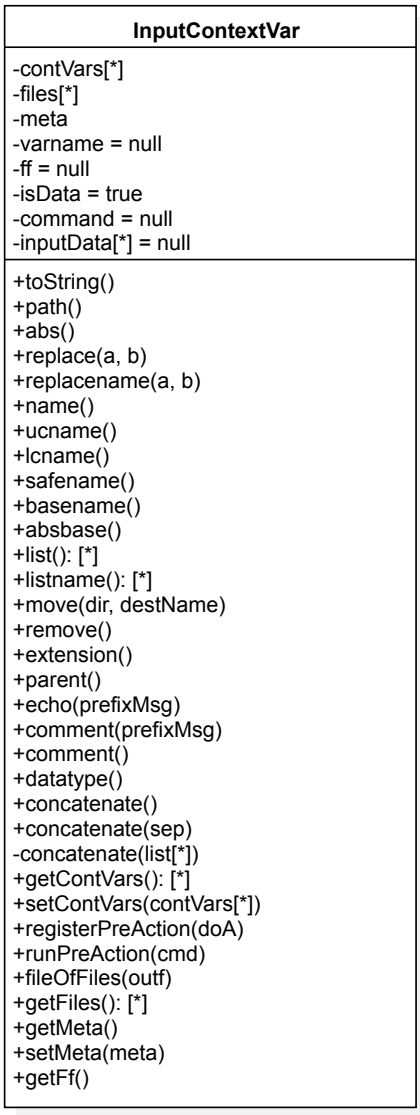


FIGURE F.7: Classe associée aux données d'entrée lors de la génération d'un ligne de commande par *Velocity*. Ses méthodes sont des utilitaires simplifiant le génération de commandes complexes à partir d'un patron encapsulé dans un acteur d'exécution. Elles intègrent, par exemple, la conversion des jetons DSURI sous forme de chemins de fichiers, absolus ou relatifs, le formatage des noms de fichiers, l'extraction de préfixes ou d'extensions à partir d'un nom de fichier, la conversion de types de variables, l'exploitation des méta-données associées aux jetons, la détection des exceptions par filtrage de la sortie d'erreur. L'utilisation additionnelle de la syntaxe SHELL est possible. En effet, le patron de commande est interprété par *Velocity* puis par *Bash*.

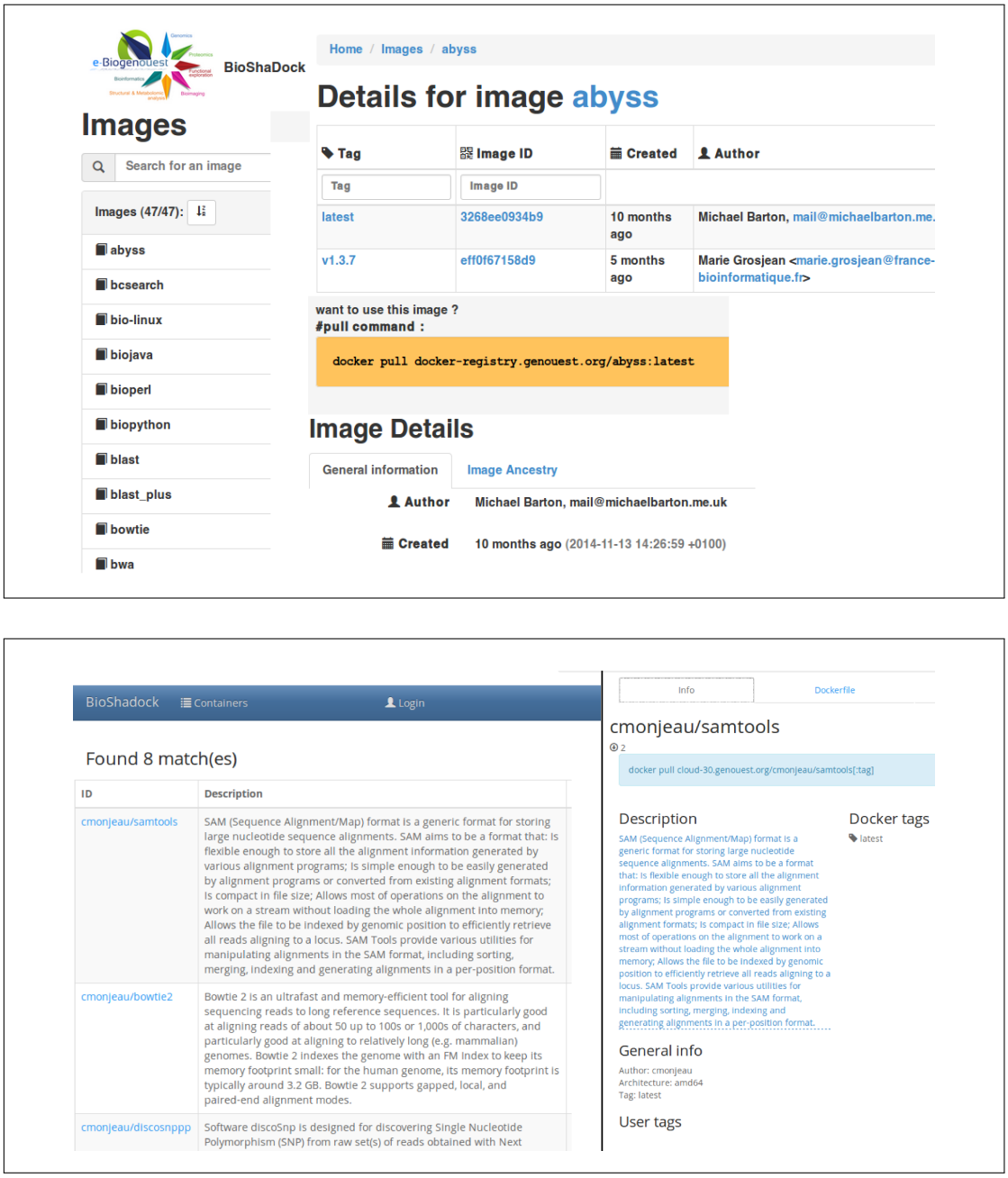


FIGURE F.8: Le répertoire BioShaDock

Le répertoire de containers Docker BioShaDock (Bioinformatics Shared Docker Registry). Les images déposées sont directement accessibles par un mécanisme d'URI.

Annexe G

Le workflow PrimerFinder

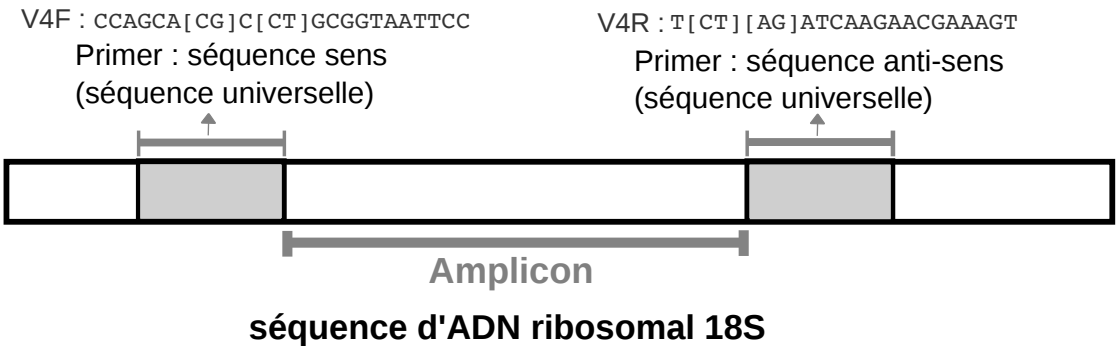


FIGURE G.1: Amplicon d'ADN 18S. L'ADN ribosomique est l'ADN qui code pour l'ARN constituant des ribosomes (ARNr). Les ribosomes ont pour fonction de traduire les ARN messagers en permettant l'assemblage des acides aminés en protéines. Chez les eucaryotes, le ribosome est composé de différentes sous-unités dont les 28S, 18S et 5S. Dans un jeu de séquences d'ADN, la diversité des séquences flanquantes de ces amorces permet d'estimer la diversité biologique.

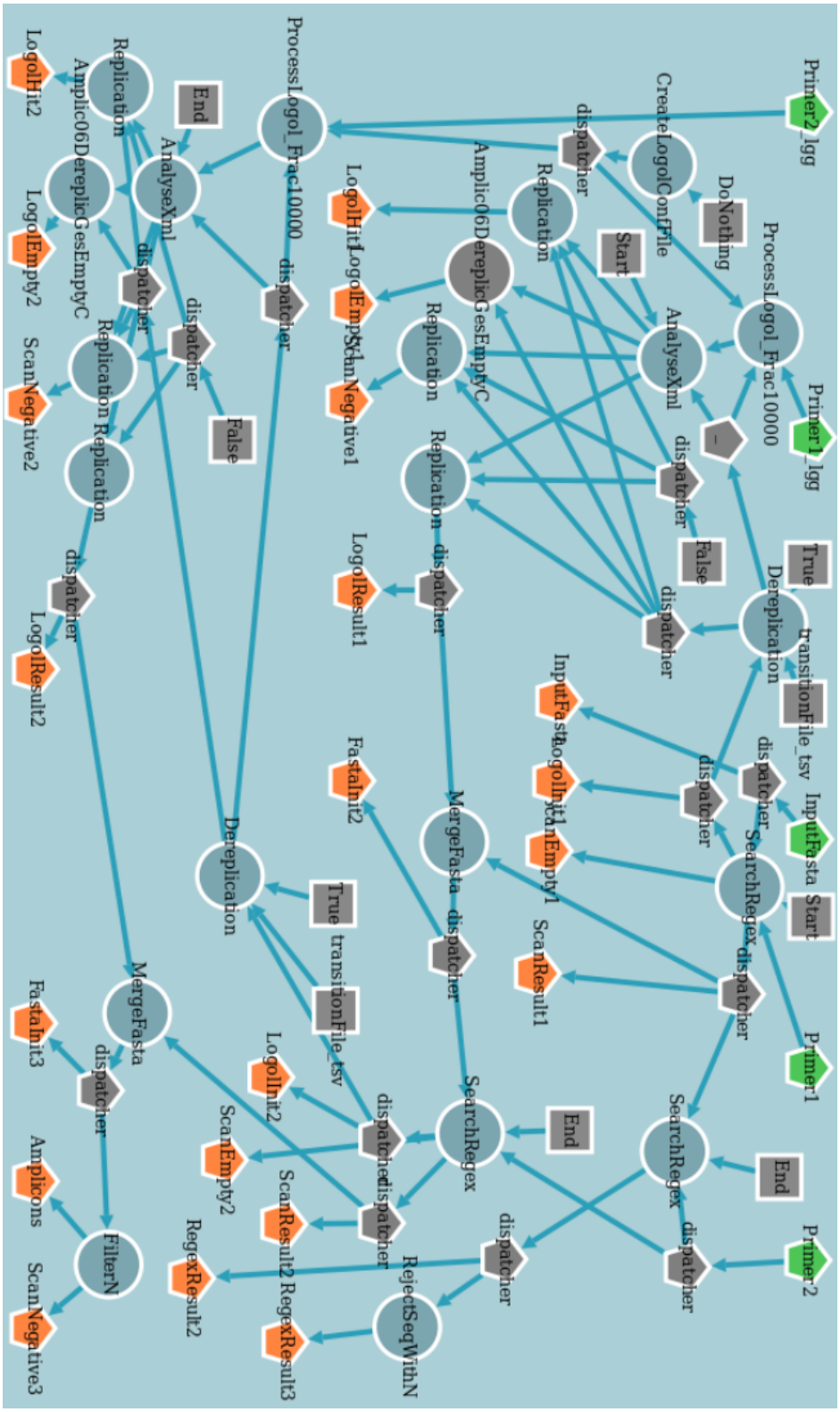


FIGURE G.2: PrimerFinder: vue du graphe DataFlow depuis l'éditeur de workflow du WfMS

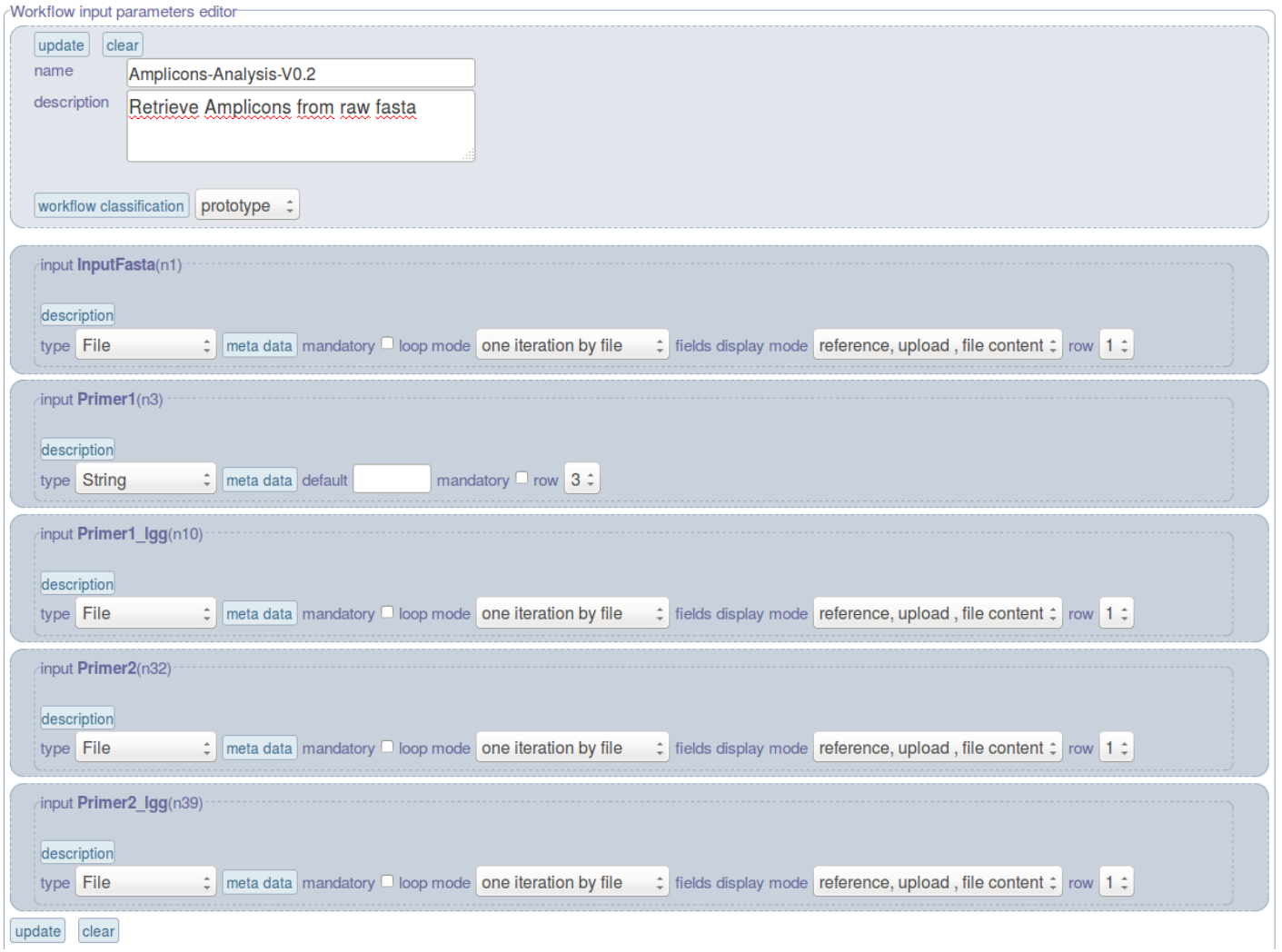


FIGURE G.3: Formulaire de génération d'interfaces de soumission de workflows (WYSIWYG)

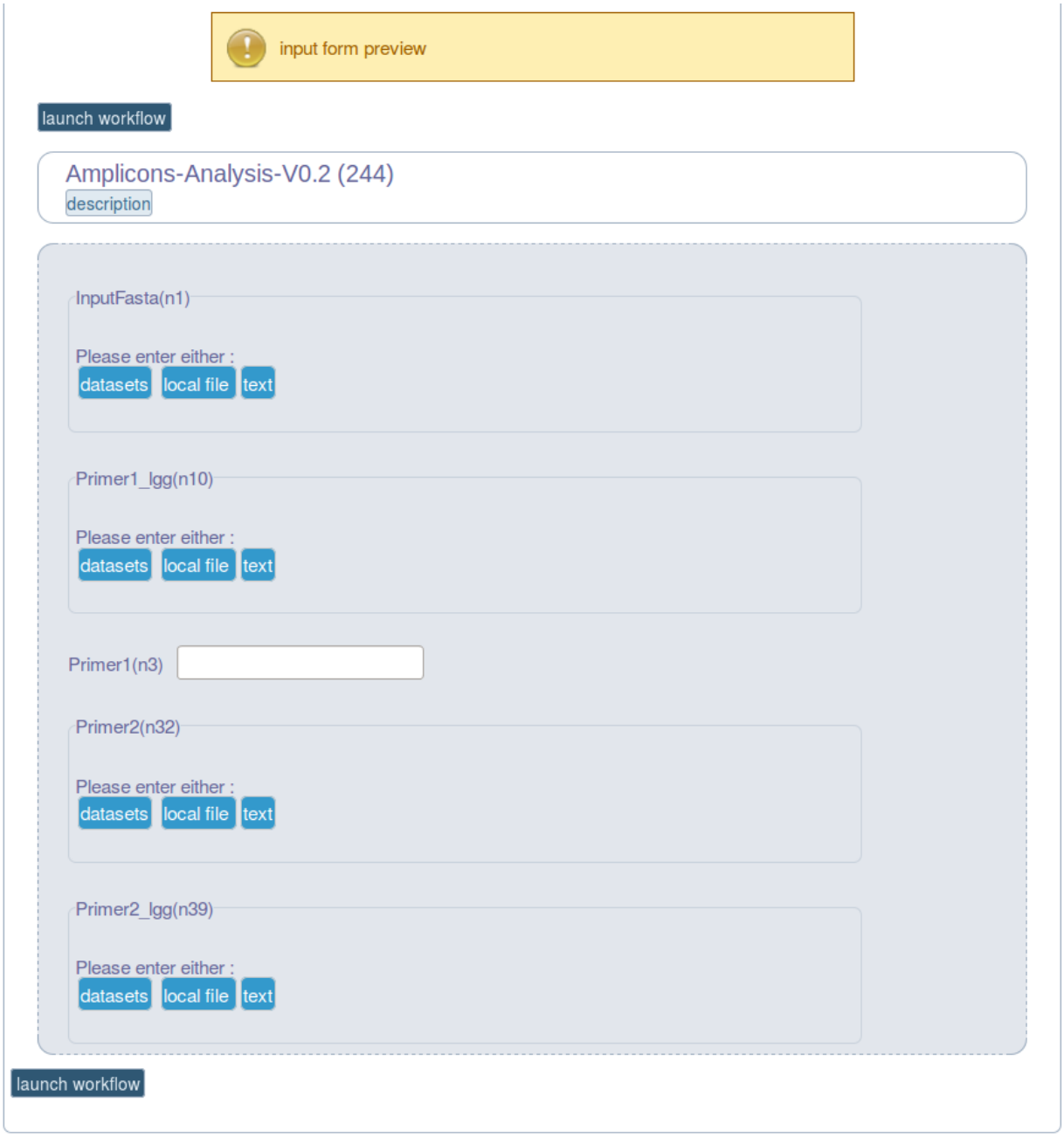


FIGURE G.4: Formulaire de soumission de workflows produit par le générateur d'interfaces WYSIWYG du WfMS.

Annexe H

L'outil SaaS d'analyse Orthocis

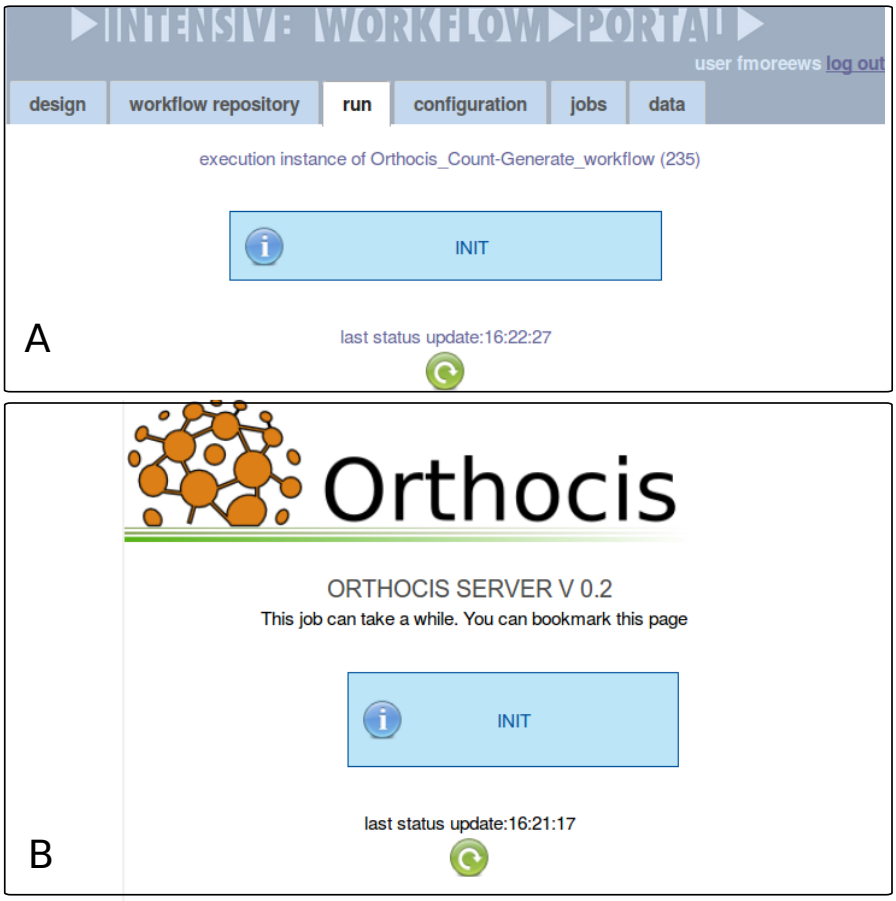


FIGURE H.1: Vue d'état d'avancement de l'exécution d'une instance d'un workflow $WF1$ dans l'environnement normatif du WfMSA et configuré pour Orthocis B.

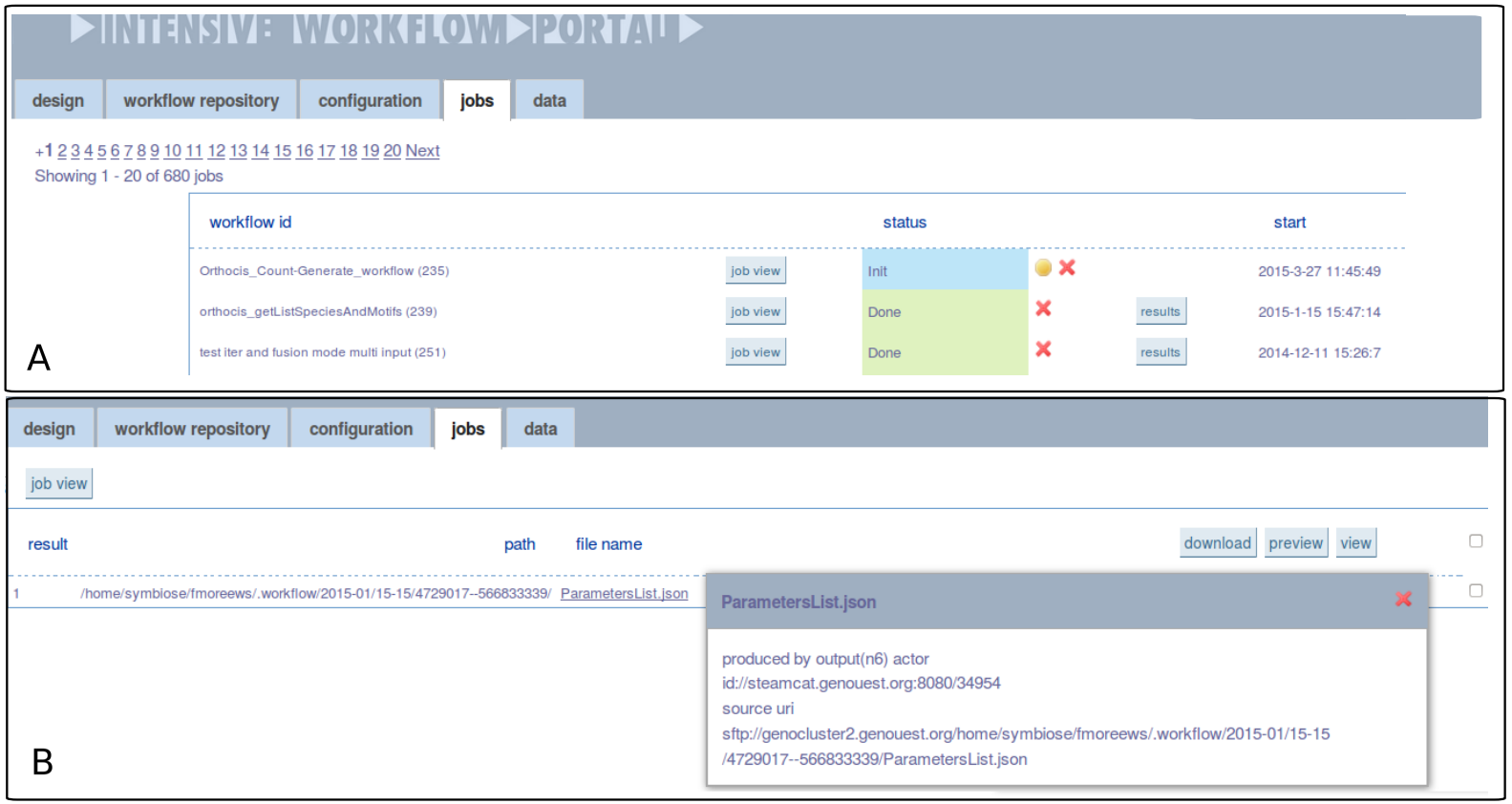


FIGURE H.2: Vue du WfMS permettant au concepteur le suivi de l'avancement de l'ensemble des exécutions des workflows d'Orthocis et l'accès aux résultats.

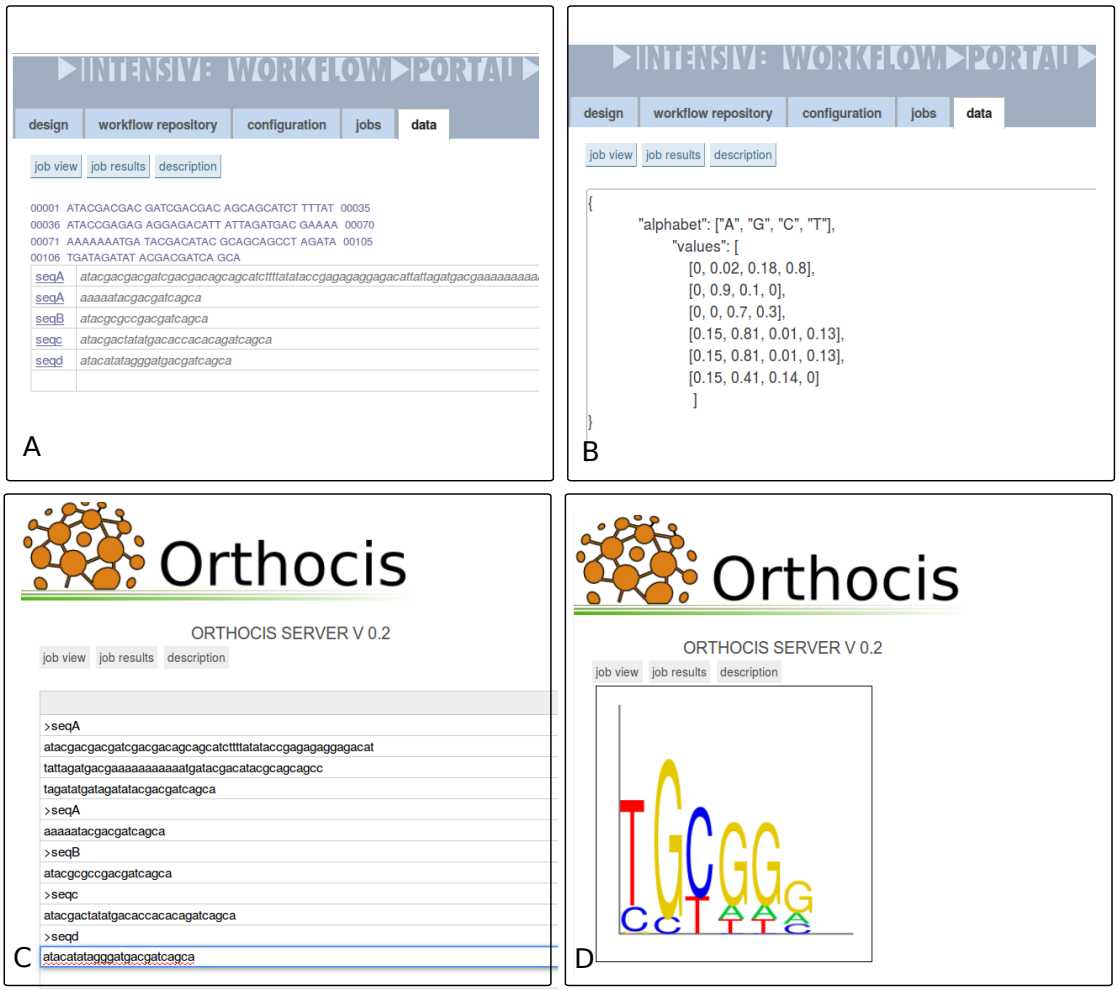


FIGURE H.3: Les visualisateurs de données embarqués dans le WfMS et leur intégration directe dans l'interface d'Orthocis par paramétrage d'URL.

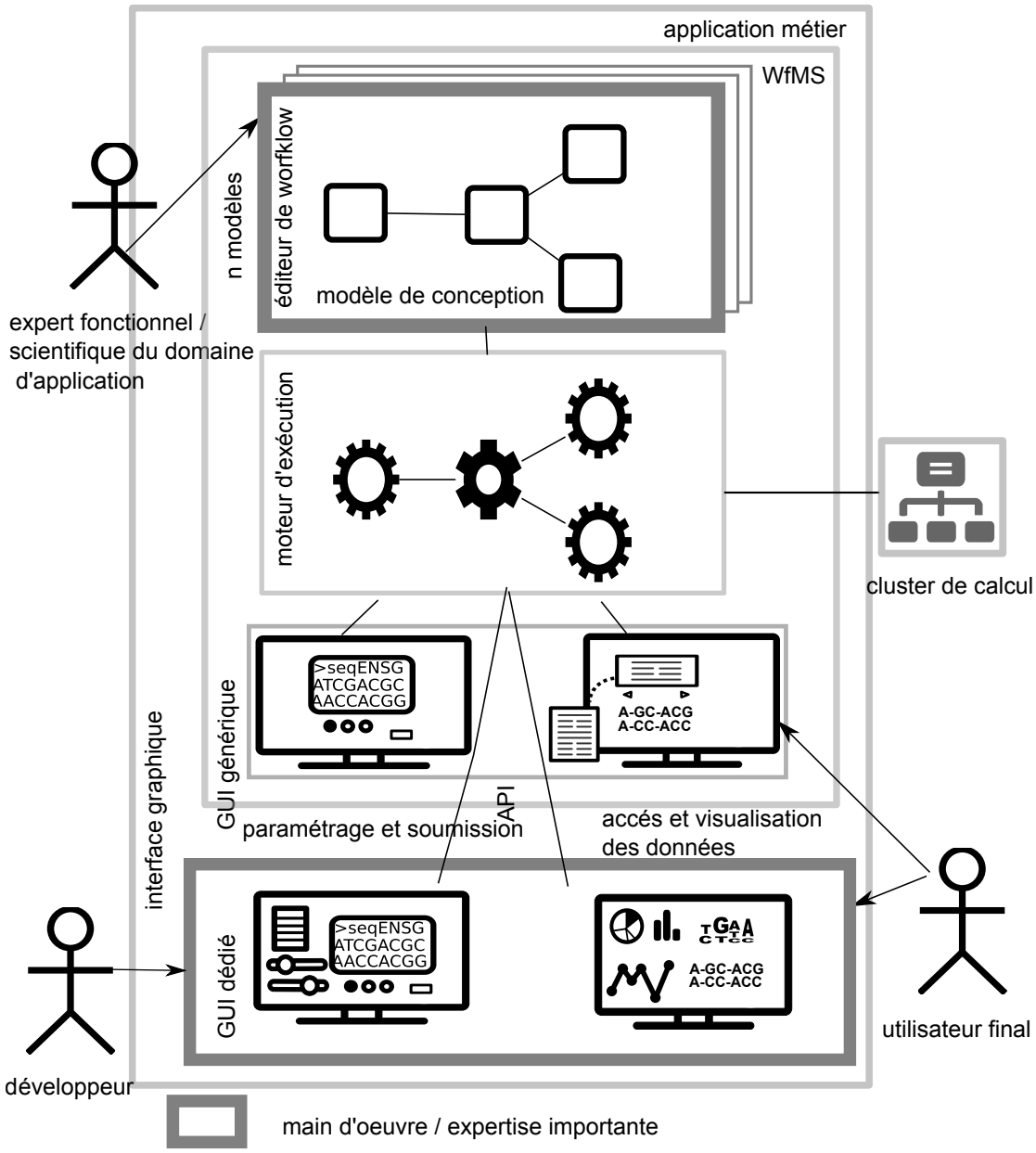


FIGURE H.4: Orthocis: schéma de l'architecture basée sur le WfMS. Le WfMS est utilisé en back-end pour la conception et l'exécution des chaînes de traitements. La soumission des jobs s'effectuent par des formulaires génériques générés par défaut ou modifiés rapidement par un éditeur WYSIWYG. La conception d'interfaces dédiées reste possible comme pour le workflow WF1 (cf. figure précédente). Pour cela, un développeur doit intervenir et exploiter les API du WfMS (HTML/Javascript). Ces deux cas sont présents dans l'application Orthocis, conçue à partir de l'aggrégation de différents workflows. L'accés aux données est directement géré par le WfMS. Des composants de visualisation générique utilisés dans Orthocis ont été directement intégré dans le WfMS (logo, motifs, séquences).

Annexe I

Formats de fichiers en bioinformatique

```
>4 dna:chromosome chromosome:GRCm38:4:43027690:43027790:1
TGCTTTTGTAAACCCTAGTTTATTAATAGCTTAATAGTCTACACCCTCTGCCCTTTAAGA
CCGCCATACCATAAGGGCAAGCTGTCCTTTCCCGTGTGTCCCAGTGA
```

FIGURE I.1: Le format FASTA

Exemple de séquence d'ADN au format FASTA. La première ligne commence par le symbole > et décrit la séquence de façon non standardisée. Elle peut contenir identifiants et annotations.

```
@HWI-EAS107_0012_EC803HG:4:48:3891:21172#ACTAAG/1
TTAATTGGTAAATAAATCTCCTAATAGCTTAGATNTTACCTTNNNNNNNNNNTAGTTTCTTGAGATTTGTTGG
+HWI-EAS107_0012_EC803HG:4:48:3891:21172#ACTAAG/1
ffcfcffccfeeeffcfffddddff‘fded]‘]_Ba_^__[YYBBBBBBBRRTTT\]][]ddddddd^ddda
```

FIGURE I.2: Le format FASTQ

Exemple de données de séquençage au format FASTQ. Ce format contient des chaînes de caractères correspondants à des fragments courts de séquence d'ADN ou *lectures* (reads). Une lecture est représentée par 4 lignes en ASCII, respectivement l'identifiant, la séquence, l'identifiant et la qualité par base. Dans la séquence, chaque lettre, A, T, C et G pour Adénine, Thymine, Cytosine et Guanine, représente une paire de bases. chaque paire de bases est associée à un score numérique permettant d'évaluer sa fiabilité et, par analyse, de déduire la qualité globale ou locale du séquençage. Un fichier fastq peut contenir plusieurs millions de lectures.

Glossaire

A-SCDFM: Autonomous Semi-Concrete DataFlow Model
API: Interface de programmation (Application Programming Interface)
CPMM: Components and Platforms Meta-Model
DAG: Direct Acyclique Graph
DBMS: Database Management System , équivalent à SGBD
DISCS: Data-Intensive Scalable Computing Systems
DSURI: DataSet Uniform Resource Identifier
MoC: Model of Computation
PIM: Platform Independent Model
PSM: Platform Specific Model
SCDFM: Semi-Concrete DataFlow Model
SGBD: Système de gestion de base de données, équivalent à DBMS
SOA: Service Oriented Archirecture
UDFAS: URI based DataFlow for Asynchronous Services
URI: Uniform Resource Identifier
WfMS: Workflow Management System
WYSIWYG: What You See Is What You Get

Bibliographie

[1] van den Beek Marius and A. Christophe, "Opening Galaxy to script execution by everyone," in *Galaxy Community Conference 2015*, (Norwich, United Kingdom), July 2015.

[2] R. Eric, G. Bjorn, C. John, and B. Dannon, "Galaxy Interactive Environments – a new way to interact with your data," in *Galaxy Community Conference 2015*, (Norwich, United Kingdom), July 2015.

[3] M. Airaj, C. Blanchet, S. Kenny, and C. Loomis, "Appliance management for federated cloud environments," in *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*, vol. 1, pp. 232–239, Dec 2013.

[4] G. B. and R. E. C. J. B. D., "Galaxy flavours – shipped by a whale," in *Galaxy Community Conference 2015*, (Norwich, United Kingdom), July 2015.

[5] C. Goble and D. D. Roure, "myExperiment: social networking for workflow-using e-scientists," *Proceedings of the 2nd workshop on ...*, pp. 1–2, 2007.

[6] S. Edwards, L. Lavagno, E. A. Lee, and A. Sangiovanni-Vincentelli, "Design of embedded systems: formal models, validation, and synthesis," *Proceedings of the IEEE*, vol. 85, pp. 366–390, Mar. 1997.

[7] F. Puhlmann and M. Weske, "Using the π-calculus for formalizing workflow patterns," *Business Process Management*, 2005.

[8] E. Ogasawara, J. Dias, and D. Oliveira, "An algebraic approach for data-centric scientific workflows," *Proceedings of the VLDB Endowment (PVLDB)*, vol. 4, no. 12, pp. 1328–1339, 2011.

[9] E. Ogasawara, D. Jonas, V. Silva, C. Fernando, O. Daniel De, F. Porto, M. Mattoso, and P. Valduriez, "Chiron: A Parallel Engine for Algebraic Scientific Workflows," *Concurrency and Computation: Practice and Experience*, vol. 25, no. 16, pp. 2327–2341, 2013.

[10] C. Olston, B. Reed, and U. Srivastava, "Pig latin: a not-so-foreign language for data processing," in *Proceedings of SIGMOD*, pp. 1099–1110, 2008.

[11] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, pp. 1–13, 2008.

[12] A. Schumacher, L. Pireddu, M. Niemenmaa, A. Kallio, E. Korpelainen, G. Zanetti, and K. Heljanko, "SeqPig: simple and scalable scripting for large sequencing data sets in Hadoop," *Bioinformatics*, 2013.

[13] L. Lin, V. Lychagina, W. Liu, Y. Kwon, S. Mittal, and M. Wong, "Tenzing a sql implementation on the mapreduce framework," in *Proceedings of VLDB*, pp. 1318–1327, 2011.

[14] R. S. Xin, J. Rosen, M. Zaharia, M. J. Franklin, S. Shenker, and I. Stoica, "Shark: SQL and rich analytics at scale," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, SIGMOD '13, (New York, NY, USA), pp. 13–24, ACM, 2013.

[15] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, S. Antony, H. Liu, and R. Murthy, "Hive - a petabyte scale data warehouse using Hadoop," *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*, pp. 996–1005, 2010.

[16] M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi, and M. Zaharia, "Spark SQL: Relational data processing in Spark," in *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD '15)*, 2015.

[17] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: cluster computing with working sets," in *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, HotCloud'10, (Berkeley, CA, USA), p. 10, 2010.

[18] M. Wiewiórka and A. Messina, "SparkSeq: fast, scalable, cloud-ready tool for the interactive genomic data analysis with nucleotide precision," *Bioinformatics (Oxford, England)*, pp. 1–3, 2014.

[19] W. van der Aalst, "The application of Petri nets to workflow management," *Journal of circuits, systems, and computers*, 1998.

[20] W. V. der Aalst and A. T. Hofstede, "YAWL: yet another workflow language," *Information systems*, 2005.

[21] G. Kahn, "The semantics of a simple language for parallel programming," in *Information processing* (J. L. Rosenfeld, ed.), (Stockholm, Sweden), pp. 471–475, North Holland, Amsterdam, 1974.

[22] S. Ha and H. Oh, "Decidable dataflow models for signal processing: Synchronous dataflow and its extensions," in *Handbook of Signal Processing Systems*, pp. 1083–1109, 2013.

[23] W. M. P. Van Der Aalst, A. H. M. Ter Hofstede, B. Kiepuszewski, and A. P. Barros, "Workflow patterns," *Distrib. Parallel Databases*, vol. 14, pp. 5–51, July 2003.

[24] B. Kiepusewski, *Expressiveness and suitability of languages for control flow modelling in workflows*. PhD thesis, 2003.

[25] J. Bertin, *Semiology of graphics*. University of Wisconsin Press, 1983.

[26] S. K. Card and J. Mackinlay, "The structure of the information visualization design space," in *Proceedings of the IEEE Symposium on Information Visualization*, pp. 92–99, 1997.

[27] A. MacLean, K. Carter, L. Lövstrand, and T. Moran, "User-tailorable Systems: Pressing the Issues with Buttons," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '90, (New York, NY, USA), pp. 175–182, ACM, 1990.

[28] H. Lieberman, F. Paternò, M. Klann, and V. Wulf, "End-user development: An emerging paradigm," *End User Development*, pp. 1–8, 2006.

[29] M. L. Dertouzos, "Creating the people's computer," *Technology Review*, vol. 100, pp. 20–28, Apr. 1997.

[30] E. Lee and D. Messerschmitt, "Static scheduling of synchronous data flow programs for digital signal processing," *IEEE Transactions on Computers*, vol. 36, Jan. 1987.

[31] G. Bilsen, M. Engels, R. Lauwereins, and J. A. Peperstraete, "Cyclo-static data flow," in *Acoustics, Speech, and Signal Processing, 1995. ICASSP-95., 1995 International Conference on*, vol. 5, pp. 3255–3258 vol.5, 1995.

[32] J. Buck and E. Lee, *The Token Flow Model.* Advanced Topics in Dataflow Computing and Multithreading, Wiley IEEE Computer Society, 1995.

[33] J. Piat and R. M. , Bhattacharyya S. S., "Interface-based hierarchy for synchronous data-flow graphs," in *IEEE Workshop on Signal Processing Systems, SiPS: Design and Implementation*, vol. 1, pp. 145–150, 2009.

[34] S. S. Bhattacharyya, E. F. Deprettere, and B. D. Theelen, *dynamic dataflow graphs.* New York, NY: Springer New York, 2013.

[35] G. Gao, R. Govindarajan, and P. Panangaden, "Well-behaved dataflow programs for dsp computation," in *Acoustics, Speech, and Signal Processing, 1992. ICASSP-92., 1992 IEEE International Conference on*, vol. 5, pp. 561–564 vol.5, Mar 1992.

[36] S. Bhattacharyya, E. Deprettere, and J. Keinert, "Dynamic and multidimensional dataflow graphs," in *Handbook of Signal Processing Systems* (S. S. Bhattacharyya, E. F. Deprettere, R. Leupers, and J. Takala, eds.), pp. 899–930, Springer US, 2010.

[37] A. Girault, B. Lee, and E. Lee, "Hierarchical finite state machines with multiple concurrency models," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 18, pp. 742–760, Jun 1999.

[38] K. Strehl, L. Thiele, M. Gries, D. Ziegenbein, R. Ernst, and J. Teich, "Funstate-an internal design representation for codesign," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 9, pp. 524–544, Aug 2001.

[39] J. Boutellier, C. Lucarz, S. Lafond, V. M. Gomez, and M. Mattavelli, "Quasi-static scheduling of CAL actor networks for reconfigurable video coding," *Journal of Signal Processing Systems*, vol. 63, no. 2, pp. 191–202, 2011.

[40] W. M. Johnston, J. R. P. Hanna, and R. J. Millar, "Advances in dataflow programming languages," *ACM Computing Surveys*, vol. 36, pp. 1–34, Mar. 2004.

[41] M. Auguston and A. Delgado, "Iterative constructs in the visual data flow language," in *Visual Languages, 1997. Proceedings. 1997 IEEE Symposium on*, pp. 152–159, Sep 1997.

[42] H. Andrade and S. Kovner, "Software synthesis from dataflow models for G and LabVIEW/sup TM," in *Proceedings of the IEEE Conference Record of the 32nd Asilomar Conference on Signals, Systems and Computers*, vol. 2, pp. 1706–1709, 1998.

[43] E. Baroth and C. Hartsough, "Visual programming in the real world," in *Visual Object-Oriented Programming* (M. M. Burnett, A. Goldberg, and T. G. Lewis, eds.), pp. 21–42, Manning Publishing Co., 1995.

[44] H. Reekie, *Realtime signal processing : dataflow, visual, and functional programming*. PhD thesis, 1995.

[45] a. Bondavalli and L. Simoncini, "Functional paradigm for designing dependable large-scale parallel computing systems," *Proceedings ISAD 93: International Symposium on Autonomous Decentralized Systems*, pp. 108–114, 1993.

[46] R. Ebner and A. Pfaffinger, "Transformation of functional programs into data flow graphs implemented with PVM," in *Parallel Virtual Machine - EuroPVM'96, Third European PVM Conference*, (München, Germany), pp. 251–258, 1996.

[47] H. Vandierendonck, "A unified scheduler for recursive and task dataflow parallelism," in *International Conference on Parallel Architectures and Compilation Techniques*, pp. 1–11, 2011.

[48] L. Bic, "A process-oriented model for efficient execution of dataflow programs," 1990.

[49] F. Yazdanpanah, *Hardware design of task superscalar architecture*. PhD thesis, Universitat Politecnica de Catalunya, 2014.

[50] O. M. G. (OMG), "Omg meta-object facility (mof), version 2.4.1," specifications, Object Management Group (OMG), 2013.

[51] T. Gruber, "A translation approach to portable ontology specifications," *Knowledge acquisition*, no. April, 1993.

[52] U. Aß mann, S. Zschaler, and G. Wagner, "Ontologies, meta-models, and the model-driven paradigm," in *Ontologies For Software Engineering And Software Technology*, pp. 249–272, 2006.

[53] Object Management Group, "Ontology definition metamodel (omg) version 1.0," Tech. Rep. formal/2009-05-01, Object Management Group, 5 2009.

[54] A. Lamprecht, T. Margaria, and B. Steffen, "Bio-jETI: a framework for semantics-based service composition," *BMC bioinformatics*, vol. 10, p. S8, Jan. 2009.

[55] Y. Gil, "Mapping Semantic Workflows to Alternative Workflow Execution Engines," *2013 IEEE Seventh International Conference on Semantic Computing*, vol. 7, pp. 377–382, Sept. 2013.

[56] S. Zivković, M. Murzek, and H. Kühn, "Bringing Ontology Awareness into Model Driven Engineering Platforms1," *CEUR Workshop Proceedings*, vol. 395, no. section 3, pp. 47–54, 2008.

[57] D. Wagelaar and R. Van Der Straeten, "Platform ontologies for the model-driven architecture," *European Journal of Information Systems*, vol. 16, no. 4, pp. 362–373, 2007.

[58] D. Hollingsworth, "the workflow reference model," Tech. Rep. 1, Workflow management coalition, Hampshire, UK, 1995.

[59] V. Curcin and M. Ghanem, "Scientific workflow systems - can one size fit all?," *2008 Cairo International Biomedical Engineering Conference*, pp. 1–9, Dec. 2008.

[60] J. Yu and R. Buyya, "A taxonomy of workflow management systems for grid computing," *Journal of Grid Computing*, vol. 3, no. 3-4, p. 29, 2005.

[61] V. Korkhov, D. Krefting, J. Montagnat, T. Truong Huu, T. Kukla, G. Terstyanszky, D. Manset, M. Caan, and S. Olabarriaga, "SHIWA workflow interoperability solutions for neuroimaging data analysis.," *Stud Health Technol Inform*, vol. 175, 2012.

[62] B. Néron, H. Ménager, C. Maufrais, N. Joly, J. Maupetit, S. Letort, S. Carrère, P. Tufféry, and C. Letondal, "Mobyle: a new full web bioinformatics framework.," *Bioinformatics*, vol. 25, no. 22, pp. 3005–3011, 2009.

[63] J. Goecks, A. Nekrutenko, J. Taylor, and T. Team, "Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences," *Genome Biol*, vol. 11, no. 8, p. 86, 2010.

[64] T. Oinn, M. Greenwood, M. Addis, M. N. Alpdemir, J. Ferris, K. Glover, C. Goble, A. Goderis, D. Hull, D. Marvin, P. Li, P. Lord, M. R. Pocock, M. Senger, R. Stevens, A. Wipat, and C. Wroe, "Taverna: lessons in creating a workflow environment for the life sciences," *Concurrency and Computation: Practice and Experience*, vol. 18, pp. 1067–1100, Aug. 2006.

[65] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao, "Scientific workflow management and the kepler system," *Concurrency and Computation: Practice and Experience*, vol. 18, pp. 1039–1065, Aug. 2006.

[66] E. Deelman, G. Singh, M. Su, J. Blythe, Y. Gil, C. Kesselman, J. Kim, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. C. Jacob, and D. S. Katz, "Pegasus: A framework for mapping complex scientific workflows onto distributed systems," *Scientific Programming*, vol. 13, no. 3, 2005.

[67] Y. Zhao, M. Hategan, B. Clifford, I. Foster, G. Von Laszewski, V. Nefedova, I. Raicu, T. Stef-Praun, and M. Wilde, "Swift: Fast, reliable, loosely coupled parallel computation," in *Services, 2007 IEEE Congress on*, pp. 199–206, IEEE, 2007.

[68] M. R. Berthold, N. Cebron, F. Dill, G. D. Fatta, T. R. Gabriel, F. Georg, T. Meinl, P. Ohl, C. Sieb, and B. Wiswedel, "Knime: The konstanz information miner," in *in proceedings of the workshop on multi-agent systems and simulation mass, 4th annual industrial simulation conference*, pp. 58–61, 2006.

[69] Y. Gil, "Intelligent workflow systems and provenance-aware software," in *Proceedings of the Seventh International Congress on Environmental Modeling and Software*, (San Diego, CA), 2014.

[70] Y. Gil, V. Ratnakar, E. Deelman, G. Mehta, and J. Kim, "Wings for pegasus: Creating large-scale scientific applications using semantic representations of computational workflows," in *Proceedings of the Twenty-Second National Conference on Artificial Intelligence (AAAI)*, (Vancouver, British Columbia, Canada), pp. 1767–1774, 2007.

[71] T. Glatard, J. Montagnat, D. Lingrand, and X. Pennec, "Flexible and efficient workflow deployement of data-intensive applications on grids with MOTEUR," *International Journal of High Performance Computing Applications*, vol. 22, pp. 347–360, Aug. 2008.

[72] J. Liu, E. Pacitti, P. Valduriez, and M. Mattoso, "A survey of data-intensive scientific workflow management," *Journal of Grid Computing*, pp. 1–37, 2015.

[73] M. Bux and U. Leser, "Parallelization in scientific workflow management systems," *CoRR*, vol. abs/1303.7195, 2013.

[74] D. Hull, K. Wolstencroft, R. Stevens, C. Goble, M. R. Pocock, P. Li, and T. Oinn, "Taverna: a tool for building and running workflows of services," *Nucleic Acids Research*, vol. 34, pp. W729–W732, Jul 2006.

[75] E. Kawas, M. Senger, and M. Wilkinson, "BioMoby extensions to the Taverna workflow management and enactment software," *BMC bioinformatics*, 2006.

[76] A. Shah, Z. Singhera, and S. Ahsan, "Web Services for Bioinformatics," in *Bioinformatics: Concepts, Methodologies, Tools, and Applications* (Information Resources Management Association (USA), ed.), ch. 19, pp. 345–363, 2011.

[77] L. Cabral, J. Domingue, and E. Motta, "Approaches to semantic web services: an overview and comparisons," in *1st European Semantic Web Symposium, Heraklion, Greece*, pp. 225–239, 2004.

[78] M. Wilkinson, H. Schoof, R. Ernst, and D. Haase, "BioMOBY successfully integrates distributed heterogeneous bioinformatics Web Services. The PlaNet exemplar case.," *Plant physiology*, vol. 138, pp. 5–17, May 2005.

[79] S. Krishnan, B. Stearn, K. Bhatia, K. Baldridge, W. Li, and P. Arzberger, "Opal: Simpleweb services wrappers for scientific applications," in *Proceedings of ICWS '06. International Conference on Web Services, 2006.*, pp. 823–832, 2006.

[80] I. Taylor, M. Shields, I. Wang, and A. Harrison, "The triana workflow environment: Architecture and applications," *Workflows for e-Science*, pp. 320–339, 2007.

[81] " Wings. The Workflow System.," *http://www.wings-worfklows.org.* visited on 2015-09-24.

[82] S. Wernicke and K. Brubaker, "The IGOR Cloud Platform: Collaborative, Scalable, and Peer-Reviewed NGS Data Analysis," *Journal of Biomolecular Techniques: JBT*, vol. 24, no. May, p. 2013, 2013.

[83] C. Sloggett, N. Goonasekera, and E. Afgan, "BioBlend: automating pipeline analyses within Galaxy and CloudMan," *Bioinformatics*, vol. 29, pp. 1685–1686, Jul 2013.

[84] M. Kalas, P. l. Puntervoll, A. Joseph, E. Bartaseviciute, A. Töpfer, P. Venkataraman, S. Pettifer, J. C. Bryne, J. Ison, C. Blanchet, K. Rapacki, and I. Jonassen, "BioXSD: the common data-exchange format for everyday bioinformatics web services.," *Bioinformatics (Oxford, England)*, vol. 26, pp. i540–6, Sept. 2010.

[85] A. Kashlev, S. Lu, and A. Chebotko, "Typetheoretic Approach to the Shimming Problem in Scientific Workflows," *IEEE Transactions on Services Computing*, vol. 1374, no. c, pp. 1–1, 2014.

[86] C. Lin, S. Lu, X. Fei, D. Pai, and J. Hua, "A task abstraction and mapping approach to the shimming problem in scientific workflows," in *Proceedings of SCC '09. IEEE International Conference on Services Computing, 2009.*, pp. 284–291, 2009.

[87] G. Mentzas and C. Halaris, "Workflow on the Web: Integrating E-Commerce And Business Process Management," *International Journal of E-Business Strategy Management*, vol. 1, no. 2, pp. 147–157, 1999.

[88] W. V. der Aalst, *Process mining: discovery, conformance and enhancement of business processes*. 2011.

[89] J. Estefan and K. Laskey, "Reference architecture foundation for service oriented architecture version 1.0," Tech. Rep. December, OASIS Committee Specification, 2012.

[90] D. Bovenzi, G. Canfora, and A. Fasolino, "Enabling legacy system accessibility by web heterogeneous clients," in *Conference on Software Maintenance and Reengineering, 2003. Proceedings. Seventh European*, pp. 73–81, 2003.

[91] E. Rasche, B. Gruning, J. Chilton, and D. Baker, "Galaxy Interactive Environments – a new way to interact with your data," in *Galaxy Community Conference 2015*, (Norwich, United Kingdom), July 2015.

[92] F. Pérez and B. E. Granger, "IPython: a system for interactive scientific computing," *Computing in Science and Engineering*, vol. 9, pp. 21–29, May 2007.

[93] T. Erl, *SOA Principles of Service Design (The Prentice Hall Service-Oriented Computing Series from Thomas Erl)*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2007.

[94] M. Maynard and G. Dimitoglou, "A Service Oriented Architecture Complexity Metric, Based on Statistical Hypothesis Testing," in *Proceedings of the 2008 International Conference on Software Engineering Research {&} Practice, {SERP} 2008, July 14-17, 2008, Las Vegas Nevada, USA, 2 Volumes*, pp. 214–220, 2008.

[95] F. Hueske, M. Peters, M. J. Sax, R. Bergmann, A. Krettek, and K. Tzoumas, "Opening the Black Boxes in Data Flow Optimization," in *Proceedings of the VLDB Endowment (PVLDB)*, vol. 5, pp. 1256–1267, 2012.

[96] K. Eilbeck, S. E. Lewis, C. J. Mungall, M. Yandell, L. Stein, R. Durbin, and M. Ashburner, "The sequence ontology: a tool for the unification of genome annotations," *Genome Biology*, vol. 6, no. 5, p. R44, 2005.

[97] M. Ashburner, C. A. Ball, J. A. Blake, D. Botstein, H. Butler, J. M. Cherry, A. P. Davis, K. Dolinski, S. S. Dwight, J. T. Eppig, M. A. Harris, D. P. Hill, L. Issel-Tarver, A. Kasarskis, S. Lewis, J. C. Matese, J. E. Richardson, M. Ringwald, and G. M. R. . G. Sherlock, "Gene ontology: tool for the unification of biology," *Nature Genetics*, vol. 25, pp. 25–29, May 2000.

[98] J. Ison, M. Kalas, I. Jonassen, D. Bolser, M. Uludag, H. McWilliam, J. Malone, R. Lopez, S. Pettifer, and P. Rice, "EDAM: an ontology of bioinformatics operations, types of data and identifiers, topics and formats.," *Bioinformatics*, vol. 29, pp. 1325–32, May 2013.

[99] J. K. Bonfield and M. V. Mahoney, "Compression of FASTQ and SAM format sequencing data.," *PloS one*, vol. 8, p. e59190, Jan. 2013.

[100] J. T. Dudley and A. J. Butte, "In silico research in the era of cloud computing.," *Nature biotechnology*, vol. 28, pp. 1181–5, Nov. 2010.

[101] J. Riley, "StarCluster - NumPy/SciPy Computing on Amazon's Elastic Compute Cloud (EC2)," in *SciPY2010 : Python for Scientific Computing Conference*, (Austin Texas), 2010.

[102] E. Afgan, D. Baker, N. Coraor, B. Chapman, A. Nekrutenko, and J. Taylor, "Galaxy cloudman: delivering cloud compute clusters," *BMC Bioinformatics*, vol. 11, no. Suppl 12, p. S4, 2010.

[103] Amazon Elastic Compute Cloud, "http://aws.amazon.com/ec2/," *http://aws.amazon.com/ec2/*, 2008. visited on 2015-09-24.

[104] D. Milojicic, I. M. Llorente, and R. S. Montero, "OpenNebula: A cloud management tool," *IEEE Internet Computing*, vol. 15, no. 2, pp. 11–14, 2011.

[105] OpenStack - Open Source Software for Building Private and Public Clouds, "http://www.openstack.org," *http://www.openstack.org*, 2013. visited on 2015-09-24.

[106] E. Afgan, B. Chapman, M. Jadan, V. Franke, and J. Taylor, "Using cloud computing infrastructure with cloudbiolinux, cloudman, and galaxy," *Current Protocols in Bioinformatics*, pp. 11–9, 2012.

[107] "IFB, the French Institute of Bioinformatics.," *http://www.france-bioinformatique.fr*. visited on 2015-09-24.

[108] H. J. La, J. S. Her, and S. D. Kim, "Framework for evaluating reusability of component-as-a-service (caas)," in *Principles of Engineering Service-Oriented Systems (PESOS), 2013 ICSE Workshop on*, pp. 41–44, May 2013.

[109] "Docker." `https://www.docker.io/`, 2013. visited on 2015-09-24.

[110] " Docker containers on Google Cloud Platform, powered by Kubernetes.," *https://cloud.google.com/container-engine/*. visited on 2015-09-24.

[111] M. Mohamed, D. Belaid, and S. Tata, "Self-managed micro-containers for service-based applications in the cloud," in *Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2013 IEEE 22nd International Workshop on*, pp. 140–145, June 2013.

[112] M. E. Aranguren, "Merging OpenLifeData with SADI services using Galaxy and Docker (DOI 10.1101/013615)," *BioRxiv, Cold Spring Harbor Labs*, 2015.

[113] P. Mell and T. Grance, "The NIST definition of cloud computing," 2011.

[114] D. Blankenberg, G. Von Kuster, E. Bouvier, D. Baker, E. Afgan, N. Stoler, J. Taylor, and A. Nekrutenko, "Dissemination of scientific software with Galaxy ToolShed.," *Genome biology*, vol. 15, no. 2, p. 403, 2014.

[115] B. Ludäscher, I. Altintas, S. Bowers, J. Cummings, T. Critchlow, E. Deelman, D. Roure, J. Freire, C. Goble, M. Jones, *et al.*, "Scientific process automation and workflow management," *Scientific Data Management: Challenges, Existing Technology, and Deployment, Computational Science Series*, pp. 476–508, 2009.

[116] M. J. Flynn, "Some Computer Organizations and Their Effectiveness," *IEEE Transactions on Computers*, vol. C-21, pp. 948–960, Sept. 1972.

[117] D. Sima, T. Fountain, and P. Kacsuk, *Advanced computer architectures - a design space approach.* a design space approach, Addison Wesley, 1997.

[118] K. Maheshwari, A. Rodriguez, D. Kelly, R. Madduri, J. M. Wozniak, M. Wilde, and I. T. Foster, "Enabling Multi-task computation on Galaxy-based Gateways using Swift," in *Science Gateway Institute Workshop, IEEE Cluster 2013*, (Indianpolis, IN), 2013.

[119] C. Sieb, T. Meinl, and M. R. Berthold, "Parallel and Distributed Data Pipelining with KNIME," *Mediterranean Journal of Computers and Networks ,Special Issue on Data Maining Applications on Supercomputing and Grid Environnments*, vol. 3, no. 2007, pp. 43–51, 2007.

[120] A.-L. Lamprecht, S. Naujokat, B. Steffen, and T. Margaria, "Constraint-guided workflow composition based on the edam ontology," in *Proceedings of the Workshop on Semantic Web Applications and Tools for Life Sciences*, 2010.

[121] O. Labbani, J. luc Dekeyser, P. Boulet, and E. Rutten, "Introducing control in the gaspard2 data-parallel metamodel: Synchronous approach," in *Proceedings of the 1st International Workshop on Modeling and Analysis of Real-Time and Embedded Systems*, 2005.

[122] J. Eder, W. Gruber, and H. Pichler, "Transforming workflow graphs," in *Proceedings of the first international conference on interoperability of enterprise software and applications INTEROP-ESA 2005 ,Geneva, Switzerland*, pp. 203–215, 2005.

[123] J. Buck, S. Ha, E. Lee, and D. Messerschmitt, "Ptolemy: A framework for simulating and prototyping heterogeneous systems," 1994.

[124] "Bioinformatics Intensive Workflow Portal," *http://workflow.genouest.org.* visited on 2015-09-24.

[125] "Slicee - Service Layer for Intensive Computation," *http://vapor.gforge.inria.fr/.* visited on 2015-09-24.

[126] G. Borges, M. David, J. Gomes, C. Fernandez, J. Lopez Cacheiro, P. Rey Mayo, A. Simon Garcia, D. Kant, and K. Sephton, "Sun Grid Engine, a new scheduler

for EGEE middleware," in *IBERGRID - Iberian Grid Infrastructure Conference*, April 2007.

[127] "TORQUE Resource Manager," *http://www.adaptivecomputing.com/products/open-source/torque/*. visited on 2015-09-24.

[128] "FASTQC, A quality control tool for high throughput sequence data," *http://www.bioinformatics.babraham.ac.uk/projects/fastqc*. visited on 2015-09-24.

[129] M. Dodt, J. T. Roehr, R. Ahmed, and C. Dieterich, "Flexbar, flexible barcode and adapter processing for next-generation sequencing platforms," *Biology*, vol. 1, no. 3, pp. 895–905, 2012.

[130] C. Belleannée, O. Sallou, and J. Nicolas, "Logol: Expressive Pattern Matching in sequences. Application to Ribosomal Frameshift Modeling," in *PRIB2014 - Pattern Recognition in Bioinformatics, 9th IAPR International Conference* (M. Comin, L. Kall, E. Marchiori, A. Ngom, and J. Rajapakse, eds.), vol. 8626, (Stockholm, Sweden), pp. 34–47, Lukas KALL, Springer International Publishing, Aug. 2014.

[131] M. Clamp, D. Andrews, D. Barker, P. Bevan, G. Cameron, Y. Chen, L. Clark, T. Cox, J. Cuff, V. Curwen, T. Down, R. Durbin, E. Eyras, J. Gilbert, M. Hammond, T. Hubbard, a. Kasprzyk, D. Keefe, H. Lehvaslaiho, V. Iyer, C. Melsopp, E. Mongin, R. Pettett, S. Potter, a. Rust, E. Schmidt, S. Searle, G. Slater, J. Smith, W. Spooner, a. Stabenau, J. Stalker, E. Stupka, a. Ureta-Vidal, I. Vastrik, and E. Birney, "Ensembl 2002: Accommodating comparative genomics," *Nucleic Acids Research*, vol. 31, no. 1, pp. 38–42, 2003.

[132] F. Cunningham, M. R. Amode, D. Barrell, K. Beal, K. Billis, S. Brent, D. Carvalho-Silva, P. Clapham, G. Coates, S. Fitzgerald, *et al.*, "Ensembl 2015," *Nucleic acids research*, vol. 43, no. D1, pp. D662–D669, 2015.

[133] "BioShaDock, a Bioinformatics Shared Docker registry," *http://docker-ui.genouest.org*. visited on 2015-09-24.

[134] "Go-Docker, a cluster management tool with Dockers.," *http://godocker.genouest.org*. visited on 2015-09-24.

[135] R. Graham, G. Shipman, B. Barrett, R. Castain, G. Bosilca, and A. Lumsdaine, "Open mpi: A high-performance, heterogeneous mpi," in *Cluster Computing, 2006 IEEE International Conference on*, pp. 1–9, Sept 2006.

[136] T. Truong Huu, *Workflow-based applications performance and execution cost optimization on cloud infrastructures*. Theses, Université Nice Sophia Antipolis, Dec. 2010.

[137] I. O. for Standardization, "ISO. Information Processing—Documentation Symbols for Data, Program and System Flowcharts, Program Network, Network Charts and System Resource Charts ," 1985.

[138] Y. Bertot, "lambda-calcul et types." 2006.

[139] M. Niemenmaa, A. Kallio, A. Schumacher, P. Klemelä, E. Korpelainen, and K. Heljanko, "Hadoop-BAM: directly manipulating next generation sequencing data in the cloud," *Bioinformatics*, vol. 28, no. 6, pp. 876–877, 2012.

[140] L. Jourdren, M. Bernard, M.-A. Dillies, and S. Le Crom, "Eoulsan: a cloud computing-based framework facilitating high throughput sequencing analyses.," *Bioinformatics*, vol. 28, pp. 1542–3, June 2012.

[141] P. Stevens, "A landscape of bidirectional model transformations," in *Generative and transformational techniques in software engineering II*, pp. 408–424, Springer, 2008.

[142] U. Scholten, R. Fischer, and C. Zirpins, "The dynamic network notation," *Proceedings of the Fourth Annual Workshop on Simplifying Complex Networks for Practitioners - SIMPLEX '12*, p. 25, 2012.

[143] U. Scholten, *The dynamic network notation.* PhD thesis, Karlsruher Institut für Technologie, 2013.

[144] "continuous, objective and reproducible evaluation of genome assemblers using docker containers ," *http://nucleotid.es.* visited on 2015-09-24.

[145] "CAMI, Critical Assessment of Metagenomic Interpretation," *http://cami-challenge.org.* visited on 2015-09-24.

[146] A. Peter, T. Nebojsa, S.-R. Stian, K. John, S. Luka, P. Tim, C. John, M. Maxim, L. Samuel, M. Hervé, F. Scott, S. M. Venkat, and C. M. R., "Beyond Galaxy: portable workflows and tool definitions with the CWL," in *Galaxy Community Conference 2015*, (Norwich, United Kingdom), July 2015.

[147] a. Lamprecht and S. Naujokat, "Synthesis-based loose programming," *Quality of Information . . .*, pp. 262–267, Sept. 2010.

[148] F. Moreews and D. Lavenier, "Seamless Coarse Grained Parallelism Integration in Intensive Bioinformatics Workflows," in *Proceedings of the 20th European MPI Users' Group Meeting*, EuroMPI '13, (New York, NY, United States), pp. 277–282, ACM, 2013.

[149] J. Piat, F. Moreews, O. Collin, D. Lavenier, and A. Cornu, "SLICEE: A Service oriented middleware for intensive scientific computation," in *SERVICES 2011*, July 2011.

[150] F. Moreews, O. Sallou, Y. Le Bras, G. Marie, C. Monjeaud, T. A. Darde, O. Collin, and C. Blanchet, "A curated Domain centric shared Docker registry linked to the Galaxy toolshed," in *Galaxy Community Conference 2015*, (Norwich, United Kingdom), July 2015.

[151] F. Moreews, J. Piat, and O. Sallou, "OBIWEE : an open source bioinformatics cloud environment," in *BOSC 2011 - 12th Annual Bioinformatics Open Source Conference*, (Vienne, Austria), July 2011.

[152] F. Moreews, Y. Le Bras, O. Dameron, C. Monjeaud, and O. Collin, "Integrating GALAXY workflows in a metadata management environment," in *Galaxy Community Conference*, (Baltimore, United States), July 2014.

Table des figures

Liste des tableaux

Résumé

Concevoir et partager des workflows d'analyse de données. Application aux traitements intensifs en bioinformatique

Dans le cadre d'une démarche d'Open science, nous nous intéressons aux systèmes de gestion de workflows (WfMS) scientifiques et à leurs applications pour l'analyse de données intensive en bioinformatique. Nous partons de l'hypothèse que les WfMS peuvent évoluer pour devenir des plates-formes pivots capables d'accélérer la mise au point et la diffusion de méthodes d'analyses innovantes. Elles pourraient capter et fédérer autour d'une thématique disciplinaire non seulement le public actuel des consommateurs de services mais aussi celui des producteurs de services. Pour cela, nous considérons que ces environnements doivent à la fois être adaptés aux pratiques des scientifiques concepteurs de méthodes et fournir un gain de productivité durant la conception et le traitement. Ces contraintes nous amènent à étudier la capture rapide des workflows, la simplification de l'intégration des tâches techniques, comme le parallélisme nécessaire au haut-débit, et la personnalisation du déploiement. Tout d'abord, nous avons défini un langage graphique DataFlow expressif, adapté à la capture rapide des workflows. Celui-ci est interprétable par un moteur de workflows basé sur un nouveau modèle de calcul doté de performances élevées, obtenues par l'exploitation des multiples niveaux de parallélisme. Nous présentons ensuite une approche de conception orientée modèle qui facilite la génération du parallélisme de données et la production d'implémentations adaptées à différents contextes d'exécution. Nous décrivons notamment l'intégration d'un métamodèle des composants et des plates-formes, employé pour automatiser la configuration des dépendances des workflows. Enfin, dans le cas du modèle *Container as a Service* (CaaS), nous avons élaboré une spécification de workflows intrinsèquement diffusable et ré-exécutable. L'adoption de ce type de modèle pourrait déboucher sur une accélération des échanges et de la mise à disposition des chaînes de traitements d'analyse de données.

Abstract

Design and share data analysis workflows. Application to bioinformatics intensive treatments

As part of an Open Science initiative, we are particularly interested in the scientific Workflow Management Systems (WfMS) and their applications for intensive data analysis in bioinformatics. We start from the assumption that WfMS can evolve to become efficient hubs able to speed up the development and the dissemination of innovative analysis methods. These software platforms could rally and unite not only the current stakeholders, who are service consumers, but also the service producers, around a disciplinary theme. We therefore consider that these environments must be both adapted to the practices of the scientists who are method designers and also enhanced with increased productivity during design and treatment. These constraints lead us to study the rapid capture of workflows, the simplification of technical tasks integration, like parallelisation and the deployment customization. First, we define an expressive graphic worfklow language, adapted to the quick capture of workflows. This is interpreted by a workflow engine based on a new model of computation with high performances obtained by the use of multiple levels of parallelism. Then, we present a Model-Driven design approach that facilitates the data parallelism generation and the production of suitable implementations for different execution contexts. We describe in particular the integration of a components and platforms meta-model used to automate the configuration of workflows' dependencies. Finally, in the case of the cloud model *Container as a Service* (CaaS), we develop a workflow specification intrinsically re-executable and readily disseminatable. The adoption of this kind of model could lead to an acceleration of exchanges and a better availability of data analysis workflows.