département
Informatique

Université Claude Bernard Lyon 1

Université Claude Bernard Lyon I

UFR Biosciences

M1 Bio-informatique

Domaine : Sciences et technologies
Mention : Master
Parcours : Bio-informatique
Année d'étude : M1
Année Universitaire : 2023-2024

Conception d'une interface de visualisation web de workflows et de leurs métadonnées

Étudiant
Salma EL AOUDATI
(p2001332)

Maîtres de stage
Fabien DUCHATEAU
(Enseignant-chercheur)
Emmanuel COQUERY
(Enseignant-chercheur)

Tuteur pédagogique
Vincent LACROIX
(Enseignant-chercheur)

Remerciements

Je tiens tout d'abord à exprimer un grand merci à Mr. Fabien Duchateau et Mr. Emmanuelle Coquery pour m'avoir accueilli en tant que stagiaire au sein de leur équipe. Leurs enseignements précieux m'ont permis d'enrichir mon expérience professionnelle et le temps qu'ils ont passé à la correction de mon rapport est inestimables.

Je remercie également, Mouna pour avoir été disponible et patiente avec moi. Merci de m'avoir expliqué des concepts informatiques complexes qui m'étaient jusqu'à présent inconnus.

Je tiens à exprimer ma sincère reconnaissance à Mme. Saida Bouakaz - directrice du département informatique - pour m'avoir accueillie au sein de son établissement.

Je suis également reconnaissante envers mes proches pour leur soutien constant tout au long de l'élaboration de mon projet professionnel. Leur présence quotidienne et leur soutien moral ont été une source de motivation inestimable.

Résumé

J'ai effectué mon stage au département informatique de l'Université Claude Bernard Lyon 1, où j'ai travaillé sur la visualisation de workflows et de leurs métadonnées dans le cadre du projet national ShareFAIR. Ce projet vise à améliorer la réutilisation de workflows scientifiques pour renforcer la fiabilité et la reproductibilité des résultats d'analyse en bioinformatique. Mon travail s'est concentré sur la réalisation d'un prototype qui repose sur deux grandes parties : la modélisation de la base de données et le développement d'une interface web. La première phase de mon stage a porté sur la conception et l'implémentation d'un métamodèle de base de données. Cette partie a été réalisée en collaboration avec Mouna, une étudiante de Master 2 informatique. J'ai ensuite traduit deux workflows issus d'un système de gestion de workflows en triplets RDF, puis j'ai importé ces données dans un système de gestion de bases de données RDF. Lors de la seconde phase, j'ai développé une interface web en JavaScript pour visualiser les workflows et leurs métadonnées. Cette interface offre la possibilité de passer d'une visualisation d'un workflow sous forme de *dataflow* en workflow sous forme de *controlflow*.

Table des figures

Liste des tableaux

Acronymes

CRISPR *Clustered Regularly Interspaced Short Palindromic Repeats*. 12

CWL *Common Workflow Language*. 3–7, 9, 10

DOM *Document Object Model*. 15

DSL *Domain Specific Language*. 4

EDAM *EMBRACE Data and Methods*. 4, 6, 7, 9, 10

ELICO Equipe de recherche de Lyon en sciences de l'Information et de la COmmunication. 1

ERIC Entrepôts, Représentation et Ingénierie des Connaissances. 1

FAIR *Findable, Accessible, Interoperable and Reusable*. 1

GWAS *Genome-Wide Association Study*. 12

LIP Laboratoire de l'Informatique du Parallélism. 1

LIRIS Laboratoire d'InfoRmatique en Image et Systèmes d'information. 1

RDF *Resource Description Framework*. 2, 3, 8, 9, 11–13, 17–19

RDFS *RDF Schema*. 18

SAGA *Simulated Annealing General Algorithm*. 4

SGBD Systèmes de Gestion de Bases de Données. 7, 13, 17

SPARQL *SPARQL Protocol and RDF Query Language*. 13, 15, 23

UCBL Université Claude Bernard Lyon 1. 1, 2, 17

URI *Uniform Resource Identifier*. 24

W3C *World Wide Web Consortium*. 8

WDL *Workflow Description Language*. 4–6, 10

YAWL *Yet Another Workflow Language*. 4

Liste des logiciels

Nom	**Version**	**Référence**
Blazegraph	2.1.6-SNAPSHOT	`https://github.com/blazegraph/database`
Apache Jena Fuseki	5.0.0	`https://jena.apache.org/documentation/fuseki2/`
Docker Desktop	4.30.0.149282	`https://docs.docker.com/desktop/`
VS Code	1.90.0	`https://code.visualstudio.com/`
Viz.js	3.5.0	`https://github.com/mdaines/viz.js/`

Table des matières

1 Introduction

J'ai réalisé mon stage à l'Université Claude Bernard Lyon 1, où j'ai travaillé sur la visualisation de workflows et de leurs métadonnées.

1.1 Présentation de l'organisme d'accueil

Le Département Informatique de l'Université Claude Bernard Lyon 1 est situé sur le campus de la Doua à Villeurbanne. Il accueille environ 800 étudiants de différentes formations pour 50 enseignants-chercheurs. Le département est également composé de quatre laboratoires de recherche (LIRIS, ELICO, LIP, et ERIC). Mon stage s'est déroulé dans les locaux du bâtiment Nautibus du département informatique, sous la supervision de deux maîtres de conférences spécialisés en informatique : Dr. Emmanuel COQUERY, enseignant-chercheur, faisant partie de l'équipe Bases de Données du LIRIS, et Dr. Fabien DUCHATEAU, également maître de conférences à l'UCBL et chercheur au sein de la même équipe. Mes encadrants de stage sont impliqués dans le projet national ShareFAIR [1], qui vise à améliorer la réutilisation de workflows scientifiques. C'est dans le cadre de ce projet qu'ils m'ont proposé un sujet de stage.

1.2 Contexte scientifique

Aujourd'hui, le développement de nouvelles technologies dans divers domaines scientifiques génère un grand volume de données variées mais essentielles pour le progrès de la recherche. Toutefois, l'utilisation de worflows pour l'analyse de ces données implique des processus répétitifs et à grande échelle. En effet, la diversité des outils informatiques et la quantité massive de données à gérer complexifie de plus en plus le choix du workflow [2]. C'est pourquoi, la réutilisation de jeux de données intermédiaires - issus d'étapes de pré-traitement - serait souhaitable pour contribuer à la robustesse et à la reproductibilité des résultats [3]. Cependant, ce type de ressource est encore trop peu disponible.

Une solution serait d'adopter des workflows FAIR (*Findable, Accessible, Interoperable and Reusable*), c'est-à-dire qu'ils soient facilement localisables, accessibles, interopérables (capables de fonctionner avec d'autres systèmes ou outils) et réutilisables [4]. Ces workflows accompagnés de métadonnées permettent de clarifier et documenter l'ensemble du processus et des résultats obtenus à partir des données traitées [5]. Cette standardisation et cette organisation claire clarifiée des workflows, grâce aux principes FAIR, rendent possible l'utilisation d'un langage de requête spécialisé pour améliorer la recherche précise de workflows en exploitant les méta-

données associées [6]. Par exemple, un chercheur souhaitant trouver des workflows utilisant un outil spécifique de bioinformatique, pourra localiser plus efficacement les workflows .

1.3 Objectifs

Le projet national ShareFAIR vise à répondre à cette problématique afin d'améliorer la reproductibilité et la réutilisation des workflows en bio-informatique, qui jouent un rôle important dans la recherche scientifique. En effet, ce projet comporte plusieurs objectifs principaux : l'extraction et la récupération automatique de workflows avec leurs métadonnées, l'interrogation et la visualisation de workflows, la détection de workflows similaires, ainsi que la reconstitution de workflows à partir de textes, notamment ceux des articles scientifiques. Ainsi, ce projet contribuera à renforcer l'efficacité des recherches en bioinformatique et donc également à améliorer l'analyse de données. Il démontrera comment simplifier la gestion complexe des workflows, en augmentant l'accessibilité et l'utilisation de ces workflows.

Mon stage s'inscrit dans le deuxième objectif du projet, l'interrogation et la visualisation de workflows. La première partie concerne la modélisation de la base de données, c'est à dire la conception d'un métamodèle de haut niveau pour représenter un workflow quelconque, puis de son implémentation dans une base de données. La seconde porte sur le développement d'une interface web pour rechercher et visualiser les workflows stockés dans la base. L'organisation du stage s'est déroulée selon le planning projeté sur le diagramme de Gantt en Annexe A.

Dans la suite de mon rapport, je présente en section 2 un état de l'art, notamment sur les systèmes de workflow ainsi que les travaux visant à une représentation standardisée des worflows. La section 3 détaille la conception du modèle de notre base de données, tandis que la section 4 portera sur la visualisation du workflow. Ces deux parties correspondent aux phases principales de mon stage. Je terminerai par donner quelques perspectives et un retour d'expérience.

2 État de l'art

Dans cette partie, nous commençons par définir les notions essentielles à la bonne compréhension du rapport ; puis nous parlerons des gestionnaires de workflows tels que Snakemake et Nextflow. Enfin, nous aborderons la notion de standardisation des workflows avec des ontologies existantes et nous nous positionnerons par rapport à notre projet.

2.1 Prérequis

Un modèle de workflow en bioinformatique est une représentation structuré de processus connectés, décrivant la séquence ordonnée des tâches de traitement des données [7]. Par exemple, la figure 1 suivante montre de manière simplifié l'enchaînement des processus d'un modèle de workflow, où un processus correspond à une opération du workflow.

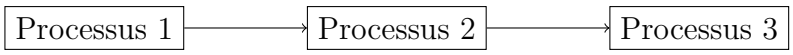

FIGURE 1 – Exemple d'enchaînement des tâches dans un workflow

Les modèles de workflow sont souvent gérés à l'aide de gestionnaires de workflow, des logiciels ou plateformes facilitant la création, l'exécution et la gestion de workflows complexes [8]. Des outils comme Snakemake et Nextflow sont des gestionnaires de workflows couramment utilisés en bioinformatique pour automatiser les pipelines d'analyse génomique.

Pour structurer efficacement un workflow, un métamodèle peut être utilisé. Un métamodèle est une structure qui précise les types de concepts fondamentaux, leurs relations spécifiques, ainsi que les règles et contraintes nécessaires à la création et à la représentation de modèles. Il guide à la construction d'un langage de modélisation en précisant comment interpréter les différents éléments du domaine et comment ils interagissent entre eux [9]. Par exemple, un métamodèle en biologie définit les idées clés comme les plantes, les animaux et les interactions entre eux, comme lorsque les animaux se nourrissent des plantes.

D'autre part, un métamodèle peut être construit à partir de concepts existants appelés ontologies. Une ontologie est une spécification explicite d'un ensemble de concepts et de relations au sein d'un domaine d'intérêt [10].

2.2 Gestionnaires de workflows

Il existe différents systèmes de gestion de workflows, tels que Constellab, SAGA, YAWL, ainsi que les deux très populaires Nextflow et Snakemake. Ce sont ces deux derniers systèmes qui sont étudiés dans le cadre du projet national, et donc sur lesquels j'ai concentré mes travaux.

Les systèmes Snakemake et Nextflow ont pour objectif commun de gérer des workflows, permettant ainsi de structurer et d'automatiser des pipelines d'analyse de données. Ils visent tous deux à assurer la reproductibilité des analyses en spécifiant clairement les dépendances logicielles et les étapes de traitement des données. En revanche, chacun utilise un langage dédié pour décrire les workflows : Snakemake se base sur Python, tandis que Nextflow utilise le DSL de Groovy. De plus, Nextflow adopte un modèle de programmation par flux de données, facilitant l'exécution de workflows parallèles et distribués sur plusieurs machines ou nœuds de calcul [11, 12].

Pour mieux comprendre l'organisation des workflows issus de Snakemake et Nextflow, il est important de considérer les deux manières de représenter un workflow : sous forme de *dataflow* ou de *controlflow*. Le *dataflow* d'un workflow est une représentation du flux de données à travers ses différentes étapes. Chaque étape du *dataflow* peut recevoir des données d'entrée (*input*) et produire des données de sortie (*output*) [13]. En revanche, le *controlflow* correspond à la manière dont un programme s'exécute en suivant des instructions définies par des structures de contrôle telles que les conditionnelles et les boucles [14]. Snakemake et Nextflow sont tous deux utilisés pour représenter le *dataflow* des workflows, Nextflow se distingue par sa capacité à gérer de manière plus explicite le *controlflow*, tandis que Snakemake se concentre principalement sur le *dataflow* avec une gestion du *controlflow* moins prononcé.

2.3 Représentation standardisée des workflows

Vu la quantité de systèmes de workflow, il devient difficile pour les utilisateurs de sélectionner et d'utiliser l'un de ces systèmes sans limiter les possibilités d'exécution et de réutilisation. Des solutions pour standardiser et uniformiser ces représentations de workflow ont donc été proposées afin d'améliorer l'interopérabilité dans ce domaine.

Depuis son introduction en 2014, CWL (*Common Workflow Language*) est devenue

un standard reconnu dans la communauté bioinformatique. Ce langage est largement utilisée par les chercheurs et les institutions pour sa capacité à automatiser des workflows complexes, tout en garantissant la reproductibilité et la portabilité. En effet, les workflows ont la possibilité de fonctionner sur divers types de systèmes informatiques, indépendamment des fournisseurs. Cela signifie que les workflows peuvent être exécutés sur divers environnements informatiques, qu'il s'agisse de clusters de calcul, de *cloud computing* ou de machines locales, sans nécessiter de modifications majeures. CWL définit un workflow comme une série d'opérations organisées en un graphe orienté qui transforme un jeu de données d'entrée en un résultat de sortie. Cette approche graphique permet de modéliser des processus complexes de manière intuitive et reproductible [15]. Les outils existants pour CWL permettent de générer des représentations statiques des workflows tel que les *dataflow*.

La figure 2 illustre les concepts principaux de CWL utilisés dans la création de workflows. Un workflow CWL peut intégrer des outils en ligne de commande (*Command-line tools*), des expressions JavaScript (*Expression tools*) ou des sous-workflows (*Sub-workflows*). Chaque workflow définit ses entrées (*inputs*), ses sorties (*outputs*) et ses étapes (*steps*) dans le document CWL.

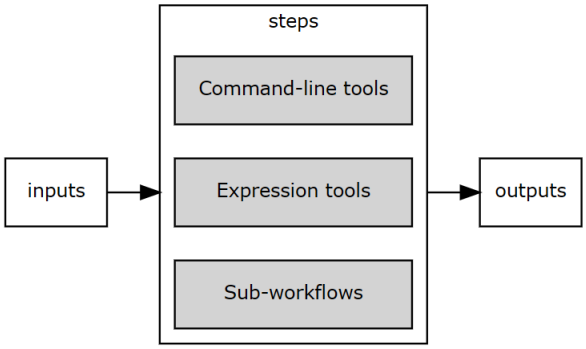

FIGURE 2 – Principaux concepts de CWL

Le *Workflow Description Language* (WDL) est un langage largement utilisé pour standardiser la description des processus de traitement des données et pour exécuter des tâches en parallèle, facilitant ainsi la gestion de *dataflow* complexes. Il offre également la possibilité d'intégrer des structures conditionnelles dans les workflows. Des outils comme Cromwell et Terra sont spécifiquement conçus pour exécuter les workflows écrits en WDL [16]. Cependant, tout comme CWL, le WDL ne prend pas en charge la visualisation interactive des workflows, se limitant généralement à des représentations graphiques statiques des workflows. Le WDL se compose de cinq concepts principaux pour structurer un workflow : le "*workflow*" qui définit le flux global, le "*task*" qui décrit les tâches individuelles, le "*call*" qui fait référence à une tâche spécifique, la section "*command*" contient les instructions spécifiques, et

la section "*output*" spécifie les résultats attendus [17].

D'autres ontologies s'intéressent à un usage plus spécifique des workflows, comme l'ontologie *EMBRACE Data and Methods* (EDAM), qui facilite l'annotation des outils et des workflows utilisés dans l'analyse et la gestion des données bioscientifiques. EDAM se concentre sur la standardisation des descriptions et des métadonnées liées aux données sans pour autant fournir d'outils de visualisation. EDAM organise ses concepts en sections claires telles que "*Topic*", "*Operation*", "*Data*" et "*Format*", assurant une description précise et cohérente [18].

2.4 Positionnement

Bien que CWL facilite la représentation standardisée des workflows, ce standard présente des limites. Il ne permet pas une description détaillée du controlflow, en particulier parce qu'il manque de constructions pour exprimer des structures de contrôle comme les conditionnelles. De plus, CWL ne couvre pas tous les aspects des métadonnées nécessaires pour la découverte d'un workflow, tels que des concepts équivalents aux mots-clés pour la description des workflows.

En revanche, le WDL offre une représentation plus détaillée du *controlflow* en intégrant des structures conditionnelles. Cependant, comme CWL, il ne parvient pas à résoudre toutes les lacunes concernant la gestion approfondie du *controlflow* et la prise en compte complète des métadonnées.

Quant à l'ontologie EDAM, elle se focalise sur l'annotation des outils et des workflows mais ne résout pas non plus ces limitations spécifiques de CWL.

En matière de visualisation, ni CWL, ni WDL, ni EDAM ne permettent la visualisation interactive des workflows. Ils se limitent généralement à la génération d'images statiques, souvent réalisées avec des outils comme Graphviz pour CWL. Cette restriction inclut également la représentation des structures de contrôle, bien que WDL aborde légèrement ce sujet en utilisant des outils comme MiniWDL et d'autres interpréteurs, WDL peut générer des représentations visuelles des structures de contrôle telles que les blocs "*if*", "*scatter*", "*while*", et "*call*". Cependant, ces visualisations restent souvent limitées à des diagrammes statiques.

Pour ces raisons, il est nécessaire de concevoir une nouvelle ontologie qui complète les concepts non couverts par celles existantes.

3 Conception de la base de données

Dans cette partie, je vais vous présenter les besoins spécifiques correspondant à nos critères pour le métamodèle de workflow. Ensuite, je vais décrire la conception du métamodèle, puis je vais donner un exemple d'implémentation de workflow basé sur ce métamodèle. Enfin, je discuterai de l'importation des données dans un SGBD.

3.1 Spécifications des besoins

La conception du métamodèle pour la base de données doit répondre à plusieurs critères pour coïncider avec les objectifs du projet shareFAIR. Une contrainte majeure concerne la réutilisation des concepts, qui peut prendre deux formes : directe et indirecte. La réutilisation directe implique l'intégration de concepts existants dans le métamodèle, en utilisant par exemple des concepts provenant des ontologies comme CWL et EDAM. En revanche, la réutilisation indirecte nécessite l'établissement de mappings, c'est-à-dire qu'un concept défini dans notre métamodèle correspond à un concept d'une ontologie existante (de façon équivalente ou similaire). Le mapping est essentiel lorsque les concepts ont des nuances qui nécessitent une adaptation précise dans le métamodèle. Ils renforcent également l'interopérabilité entre ontologies, permettant par exemple d'interroger nos données avec des concepts externes.

Un autre besoin essentiel est la gestion des métadonnées dans le métamodèle. Il est essentiel de pouvoir récupérer les métadonnées associées aux workflows, telles que les informations spécifiques à chaque étape (par exemple, les logiciels utilisés, les données d'entrée et de sortie). Ces détails sont essentiels pour faciliter la compréhension et optimiser la réutilisation des workflows.

Le métamodèle doit permettre également de modéliser le *controlflow* des workflows. Contrairement au *dataflow*, qui décrit le flux des données à travers les étapes du workflow, le *controlflow* définit la logique de séquencement et d'exécution des processus. Il est important de pouvoir modéliser le *controlflow* des workflows car cela permet de garantir que les processus sont exécutés dans l'ordre correct et selon la logique définie.

En intégrant ces aspects, le métamodèle pourra répondre parfaitement aux besoins spécifiques du projet ShareFAIR. Il permettra de réutiliser facilement les concepts, adaptant ainsi différents workflows sans nécessiter de changements importants. Cela simplifie la gestion des métadonnées et garantit une modélisation précise du *controlflow* des workflows.

3.2 Modélisation du métamodèle

La modélisation de la base de données repose sur l'utilisation du langage RDF. Ce dernier est standardisé par le W3C depuis 1999 pour la modélisation des données. Il permet de représenter les informations sous forme de triplets (sujet, prédicat, objet). En RDF, le sujet est une entité dont nous voulons décrire les propriétés ou les relations. Le prédicat indique la nature de la relation entre le sujet et l'objet, tandis que l'objet est la valeur ou l'entité associée au sujet par le prédicat. Cette structure en triplets simplifie le partage des données sur Internet en fournissant une manière claire et standardisée de décrire les informations [19].

```
ex:p2001332 rdf:type ex:Etudiante .
ex:p2001332 ex:has_name "Salma" .
ex:p2001332 ex:stage ex:Stage1 .
ex:Stage1 rdf:type ex:Stage .
ex:Stage1 ex:has_supervisor ex:ens1234 .
ex:Stage1 ex:at_lab ex:LIRIS .
ex:Stage1 ex:start_date "2024-04-22" .
ex:Stage1 ex:end_date "2024-06-21" .
ex:ens1234 rdf:type ex:EnseignantChercheur .
ex:ens1234 ex:has_name "Emmanuel" .
ex:LIRIS rdf:type ex:Laboratoire .
```

Listing 1: Exemple de triplets RDF

Les données formées de triplets RDF peuvent être visualisées sous forme de graphes RDF, composés de noeuds et d'arêtes. Chaque nœud représente une ressource (entité) et chaque arête représente une relation entre deux ressources. Ainsi l'exemple RDF précédent donne le graphe illustré en figure 3, réalisé avec rdf-grapher [20]. Il démontre visuellement les connexions entre les différentes entités représentées dans les données RDF fournies, mettant en évidence les relations entre moi-même (Salma), mon stage, mon tuteur (Emmanuel) et le laboratoire LIRIS.

Nous avons choisi le modèle RDF pour la modélisation de notre métamodèle pour plusieurs raisons. Tout d'abord, RDF permet de modéliser des données de manière très flexible, sans nécessiter de schéma fixe. Cela facilite l'ajout de nouvelles propriétés ou relations aux données existantes sans devoir modifier une structure de schéma rigide. De plus, étant un standard du W3C, RDF est compatible avec d'autres systèmes et applications, facilitant ainsi l'interopérabilité et l'intégration des données avec des systèmes externes. Il est également possible d'enrichir les données avec des vocabulaires et des ontologies, ajoutant ainsi une dimension sémantique, ce qui permet une meilleure compréhension et utilisation des données. Enfin, RDF permet d'appliquer des règles logiques et de faire des inférences sur les

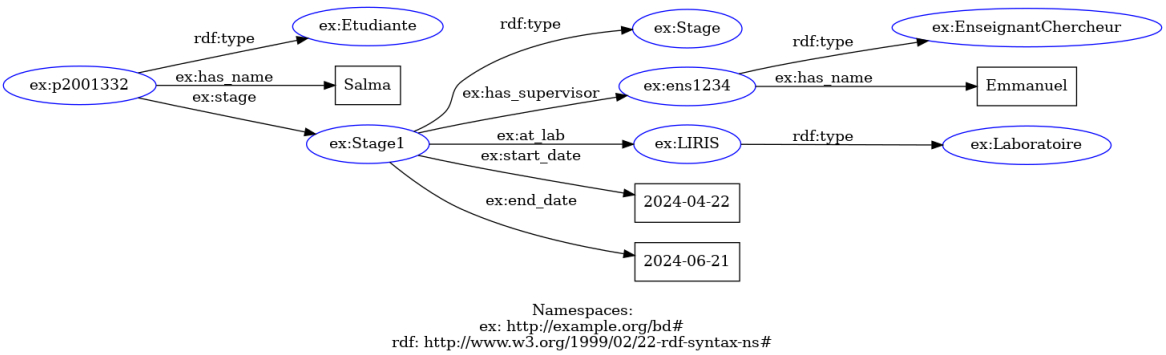

FIGURE 3 – Exemple de graphe RDF généré par rdf-grapher

données, ce qui ouvre de nouvelles perspectives pour dériver automatiquement des informations supplémentaires et mieux comprendre les relations entre les données. Par exemple, étant donné que mon stage (celui de Salma) se déroule au laboratoire LIRIS sous la supervision d'Emmanuel, on pourrait déduire, si l'on dispose de la règle appropriée, qu'Emmanuel travaille au laboratoire LIRIS.

Pour modéliser notre base de données, nous avons opté pour la réutilisation de deux ontologies standard, CWL et EDAM, en complément d'une ontologie personnalisée appelée "bd". Cette approche nous permet de décrire de manière exhaustive les workflows et les métadonnées associées.

Le tableau 1 indique le nombre de classes et de prédicats utilisés dans notre métamodèle, ainsi que leur provenance. Ainsi nous utilisons deux classes de l'ontologie CWL, à savoir Workflow et DockerRequirement, ainsi qu'une classe de l'ontologie EDAM (Data) pour représenter les données. Notre ontologie "bd" vient compléter ces classes en introduisant des entités supplémentaires spécifiques telles que les métadonnées, les opérations ou encore le thème du workflow. Au total, nous avons 14 classes et 47 prédicats.

Très peu de concepts de EDAM et CWL sont utilisés dans notre métamodèle. Cette particularité s'explique par notre choix d'utiliser parfois nos propres noms de concepts pour une meilleure clarté. Par exemple, après discussion avec l'équipe, nous avons préféré utiliser le concept "Step" plutôt que "Operation" proposé par l'ontologie CWL, notamment parce que le terme "Operation" a une signification particulière dans Nextflow. De plus, nous avons parfois choisi de ne pas utiliser le même nom de concept que celui de l'ontologie, car les prédicats associés ne correspondaient pas à nos attentes (par exemple, la classe "topic" dans EDAM ne dispose pas du prédicat standard "rdfs :label").

Tableau 1 – Résumé du nombre de classes et de prédicats par ontologie

Ontologie	Nombre de classes	Nombre de prédicats
cwl	2	1
edam	1	1
bd	11	45
Total	14	47

À défaut de réutiliser directement la majorité des concepts des ontologies existantes, nous avons la possibilité d'établir des correspondances entre nos concepts (de "bd") et ceux de CWL, EDAM et WDL. Ainsi, nous avons élaboré plusieurs mappings comme illustré dans le tableau 2. Nous nous sommes limités à des relations d'équivalence (sémantique) entre les concepts. Au total, nous avons 7 mappings de classe et 22 mappings entre les prédicats. Définir ces mappings avec des ontologies connues augmente les possibilités d'utiliser ou d'interroger notre métamodèle, ce qui est cohérent avec les principes FAIR.

Tableau 2 – Équivalences entre notre ontologie bd et les ontologies existantes CWL, WDL et EDAM

Ontologie	Nombre de mappings de classes	Nombre de mappings de prédicats
CWL	3	7
WDL	2	6
EDAM	2	9

Les différentes ontologies utilisées ont permis la réalisation du métamodèle. Ce dernier, visible entièrement en annexe Annexe B, peut se diviser en deux grandes parties : les métadonnées du workflow et les métadonnées liées aux opérations effectuées dans le workflow. Pour la première partie, comme illustré dans la figure 4. Cette partie de la modélisation permet de décrire les concepts liés au workflow, tels que les contributeurs, les droits d'auteur, le système de gestion, etc. Elle est cruciale pour retrouver un workflow de façon précise, par exemple selon des critères définis par un utilisateur.

Pour la seconde partie du workflow, chaque opération (`bd:Step`) à un identifiant, un nom, un exécuteur et un *label*. Chaque opération peut inclure des données d'entrée, de sortie et des paramètres, ainsi que des outils pour son exécution, tels que des scripts ou des lignes de commande. Pour garantir le bon fonctionnement du workflow, les opérations sont également liées à la classe`bd:ControlOperator`.

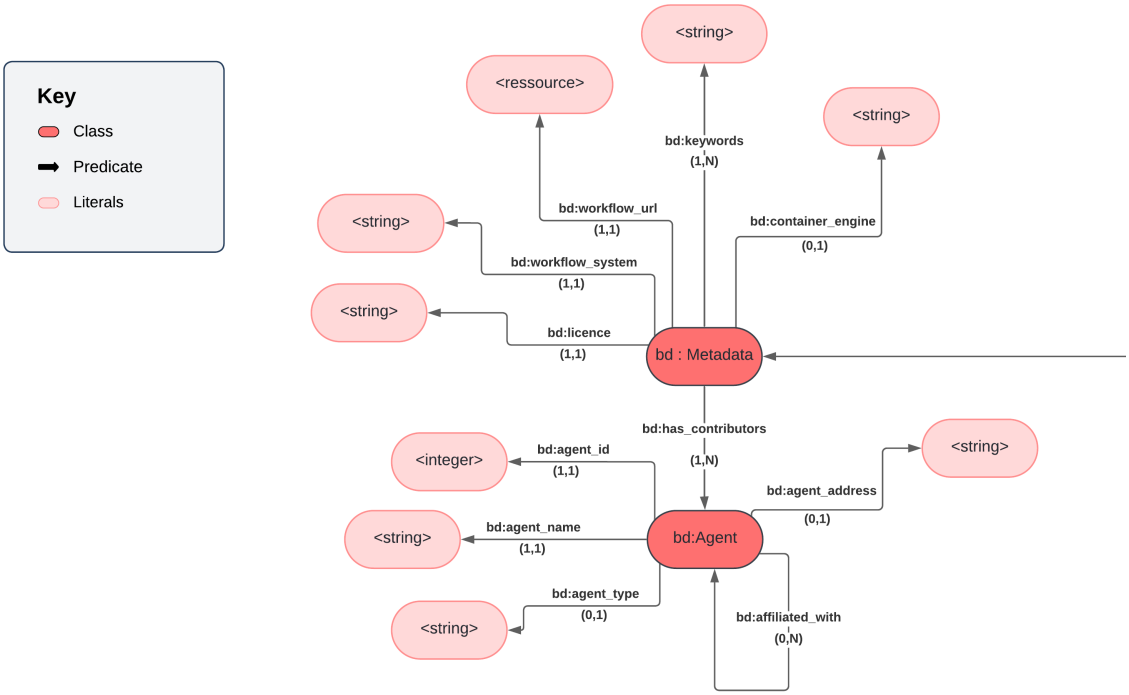

FIGURE 4 – Schéma des métadonnées du workflow

Ce travail de modélisation a nécessité de nombreuses discussions avec mes encadrants et de recherche dans les documentations afin de bien comprendre la définition de chaque concept (représentation d'une classe ou d'un prédicat, types de valeur attendus, etc.). Une stagiaire de M2 Informatique, dont le travail consiste à écrire un langage d'interrogation sur le métamodèle, m'a aidé à vérifier et améliorer cette étape cruciale de modélisation. Enfin, ce sont les tests de représentation de workflows issus de publications scientifiques en RDF qui ont permis d'affiner le métamodèle.

Une fois le métamodèle conforme aux exigences du projet shareFAIR, nous avons procédé à la création des triplets RDF pour les workflows sélectionnés.

3.3 Traduction des workflows en triplet rdf

Pour évaluer notre métamodèle, deux workflows provenant du système Snakemake ont été sélectionnés en raison de leur taille et de leur application distincte. L'objectif consiste à représenter ces deux workflows en RDF selon notre métamodèle afin de s'assurer d'une couverture complète (i.e., toutes les informations du workflow peuvent être représentées) et d'une absence de redondance et d'ambiguïté (i.e., une information ne peut pas correspondre à plusieurs concepts du métamodèle). Il est

à noter que cette traduction en RDF a été réalisée à la main, mais qu'à terme, le projet ShareFAIR devrait permettre d'extraire automatiquement les informations d'un workflow et de générer les triplets RDF basés sur le métamodèle.

Commençons par présenter les workflows choisis. Le premier, intitulé "snakemake-crispr-guides", génère de petits guides ARN pour les applications CRISPR. CRISPR est une technologie de modification génétique qui permet de modifier l'ADN de manière très précise. Cette technologie peut être utilisée, par exemple, pour corriger des gènes défectueux. L'objectif principal de ce workflow est de concevoir simultanément des milliers de guides pour créer des bibliothèques CRISPR [21].

Le workflow "kGWASflow" effectue une étude d'association pangénomique (GWAS) basée sur les k-mers selon la méthode développée par Voichek et al. (2020) [22]. Les k-mers sont de courtes séquences de nucléotides de longueurs fixes, qui sont utilisées pour représenter des régions du génome. Ils permettent une analyse plus rapide et efficace des données génomiques en réduisant la complexité computationnelle par rapport à l'analyse de l'ensemble du génome. Une étude d'association pangénomique est utilisée pour identifier les associations entre les variations à l'échelle du génome entier et des caractéristiques phénotypiques, telles que les maladies. Cette association permet d'identifier efficacement les variations génétiques associées à des caractéristiques phénotypiques en utilisant les k-mers comme marqueurs génétiques pour une analyse précise et rapide [23].

À partir des informations extraites manuellement des articles scientifiques et du dépôt GitHub, le workflow "snakemake-crispr-guides" a été entièrement transformé en triplets RDF, tandis que le workflow "kGWASflow" a été partiellement transformé. Le tableau suivant (tableau 3) renseigne sur le nombre de triplets rdf transformés pour chaque workflow.

Tableau 3 – Nombre de triplets générés pour chaque workflow

Workflow	**Nombre de triplets**
snakemake-crispr-guides	336
kgwasflow	229

Lors de la transformation du workflow "snakemake-crispr-guides" en triplets RDF, certains concepts du métamodèle n'ont pas été utilisés : `bd:data_source`, `bd:has_subworkflow`, `bd:operation_label`, `bd:require` et `bd:operator_type`. Certains de ces prédicats ne sont pas nécessaires pour comprendre le fonctionnement du workflow. C'est pourquoi, dans l'Annexe B ces prédicats ont pour cardinalité 0,1 ou 0,N. Une cardinalitée indique combien d'éléments peuvent être associés à un autre élément dans une relation donnée. Par exemple, une cardinalité de 0,1 signifie qu'une entité du métamodèle peut être associée à aucun élément (0) ou à un seul

élément (1).

En ce qui concerne le prédicat `bd:operator_type`, il n'est pas applicable à ce workflow car le concept de *controlflow* n'est pas explicitement représenté dans Snakemake. Les opérations sont simplement dépendantes les unes des autres, ce qui crée une succession d'opérations en *controlflow*, mais sans une typologie spécifique d'opération `bd:operator_type` définie dans ce contexte.

Maintenant que nous avons extrait les workflows en triplets RDF, nous pouvons les importer dans un Systèmes de Gestion de Bases de Données (SGBD).

3.4 Importation des données

Pour charger les données dans la base de données RDF, une approche basée sur des requêtes SPARQL a été mise en œuvre, utilisant deux SGBD : Blazegraph et Jena. SPARQL, correspondant à l'acronyme "*SPARQL Protocol and RDF Query Language*", est un langage de requête standardisé utilisé pour interroger et manipuler des données stockées dans des bases de données RDF. En d'autres termes, SPARQL permet aux utilisateurs d'écrire des requêtes pour récupérer des données spécifiques stockées dans la base de données RDF.

Blazegraph a été utilisé initialement pour tester et valider les requêtes SPARQL. Ce choix s'est justifié par ma familiarité avec ce SGBD, ce qui a facilité l'utilisation et le test des requêtes. Jena a été sélectionné comme le SGBD principal pour l'importation, le stockage permanent et la gestion des données RDF. Ce système a été choisi en raison de sa nature *open source*.

Un environnement Docker a été configuré pour déployer Jena (Apache Jena Fuseki). Docker est un outil qui crée des espaces isolés pour exécuter des applications, chacun contenant tout ce dont une application a besoin pour fonctionner correctement. Ensuite, pour automatiser le processus d'initialisation et d'importation des données RDF dans Jena, un script nommé "init-data.sh" a été développé par un membre de l'équipe. Pour lancer l'importation des données RDF, il suffit de démarrer Docker Desktop et d'exécuter la commande "./init-data" dans le terminal. Cela lance automatiquement le processus d'initialisation et d'importation des données RDF dans Jena.

Cette importation permet de charger les données RDF sur les workflows dans un SGBD. Il est alors possible d'interroger ce SGBD pour récupérer et afficher les informations sur les workflows, et surtout de les visualiser.

4 Développement d'une interface web

Pour exploiter la base de données de façon plus conviviale (i.e., sans que les utilisateurs n'aient besoin de connaître le langage SPARQL), j'ai construit une interface web permettant à l'utilisateur d'accéder aux données.

4.1 Spécifications des besoins

Pour répondre efficacement à la problématique de visualisation des workflows et de leurs métadonnées, l'interface utilisateur doit inclure plusieurs éléments essentiels. La mise en page de l'interface étant libre, certains éléments doivent néanmoins être intégrés pour satisfaire les besoins spécifiques du projet.

La page d'accueil doit afficher une liste des workflows disponibles présents dans la base de données, permettant ainsi à l'utilisateur de naviguer facilement entre les différents workflows. Ensuite, l'utilisateur doit pouvoir accéder à une page de visualisation détaillée pour chaque workflow. Cette représentation graphique doit inclure les nœuds et les arêtes qui composent le workflow, avec une légende explicative pour clarifier la signification des différents nœuds et arêtes, facilitant ainsi la compréhension du *dataflow*.

Il est également essentiel d'afficher les métadonnées du workflow ainsi que celles des nœuds du graphe, incluant les données et les opérations. Ces métadonnées doivent être facilement accessibles et compréhensibles afin d'optimiser l'analyse et la réutilisation des workflows.

Enfin, l'interface doit également permettre d'afficher le workflow sous une vue correspondant au *controlflow*. Cette vue permettra de comprendre la mise en place des opérateurs de contrôle du workflow pour vérifier la bonne exécution de celui-ci.

En intégrant ces fonctionnalités, l'interface web sera parfaitement adaptée pour visualiser les workflows et leurs métadonnées, répondant ainsi pleinement aux exigences du projet.

4.2 Architecture globale

Le projet est structuré autour de plusieurs fichiers : un fichier JavaScript principal, deux fichiers HTML et un fichier CSS. Le fichier JavaScript communique avec la base de données locale et gère les opérations de manière asynchrone, ce qui est un aspect critique de JavaScript. Cette asynchronicité est gérée à l'aide de promesses, un concept central dans la programmation JavaScript. Au lieu de bloquer l'exécution en attendant les données, JavaScript utilise des promesses pour continuer l'exécution

d'autres parties du programme. Une promesse garantit que les données seront disponibles dans le futur, permettant ainsi au reste du code de s'exécuter normalement jusqu'à ce que la promesse soit résolue [24].

Par exemple, la fonction `run_multiple_queries` du fichier JavaScript du projet et illustré dans le listing 2, exécute plusieurs requêtes SPARQL simultanément en utilisant des promesses. Elle prend en paramètres l'URL du point de terminaison SPARQL (endpoint) et un tableau (queries) contenant les requêtes à exécuter. Cette fonction crée un tableau de promesses, où chaque promesse correspond à une requête SPARQL. Ensuite, elle utilise 'Promise.all' pour attendre que toutes les requêtes se terminent. Le résultat de la requête est présenté dans Annexe C, montrant deux résultats distincts. Chaque résultat est représenté par un objet contenant des propriétés spécifiques telles que 'id', 'name' et 'keywords'. Un des résultats correspond au workflow "snakemake-crispr-guides" et l'autre au workflow "kGWASflow".

```
function run_multiple_queries(endpoint, queries) {
  const p = queries.map((q) => run_sparql_query(endpoint, q));
  console.log('Result :',p);
  return Promise.all(p);
}
```

Listing 2: Code permettant l'exécution de requêtes en parallèle

Pour améliorer la mise en page de l'interface j'ai utilisé des styles CSS dans le code JavaScript, ce qui était difficile à comprendre au début. En effet, JavaScript peut accéder aux propriétés des éléments HTML à travers le DOM (le modèle d'une page web), et je devais donc gérer des événements (comme un clic sur un élément du métamodèle qui déclenche l'affichage des métadonnées correspondantes). Par exemple, dans la figure 3, un extrait de code CSS utilisé pour modifier la mise en forme du noeud sélectionné. En Annexe D, une visualisation met en évidence ce nœud sélectionné.

L'architecture HTML est simple, composée de deux pages : une pour la liste des workflows et une pour la visualisation détaillée de chaque workflow.

La visualisation du workflow "snakemake-crispr-guides" et notamment de son *dataflow* est disponible en annexe Annexe E. En Annexe F, le *dataflow* extrait du workflow à partir de l'article scientifique [21] et du dépôt GitHub du workflow [25]. Les étapes principales du workflow sont les suivantes :

— Obtention de la base de données du génome au format FASTA et GFF
— Création de la séquence de référence

```
svgDataFlow.addEventListener('click', function(event) {
    const node = event.target.closest('.node');
    if (node) {
        document.querySelectorAll('.node').forEach(node => {
            node.style.fill = 'black';
            node.style.fontWeight = 'bold';
            node.style.strokeWidth = '5px';
        });
     }
});
```

Listing 3: Exemple de code CSS utilisé dans le fichier JavaScript

— Construction d'un index en parallèle
— Recherche de tous les ARN guides possibles pour la séquence donnée
— Filtrage et classement des ARN guides en fonction de scores
— Génération d'un rapport au format HTML de prédiction de guide ARN
— Conversion du rapport aux formats HTML au format PDF

Dans l'interface web, nous constatons que l'enchaînement des opérations obtenu à partir du métamodèle élaboré est conforme à celui extrait du workflow. Ainsi, nous pouvons valider notre métamodèle pour ce workflow.

L'annexe G affiche des captures d'écran de l'interface web. La figure 11 de l'annexe G montre un tableau avec la liste des workflows présents dans la base de données. Une barre de recherche permet de trouver plus facilement les workflows. Il est possible de cliquer sur le nom du workflow pour le visualiser. Une fois le workflow choisi, une page s'affiche (figure 12) montrant le dataflow du workflow. À gauche, une légende des nœuds et des arêtes du graphe est affichée, tandis qu'à droite, les métadonnées du workflow sont présentées. Si l'on clique sur l'un des nœuds du workflow, les métadonnées de ce nœud remplacent celles du workflow. Par exemple, dans la figure 14, le nœud 'get_genome' est sélectionné et ses métadonnées sont affichées ; dans la figure 15, le nœud 'sequence_genome' est sélectionné et ses métadonnées sont également affichées. Pour afficher le controlflow du graphe, il suffit de cliquer sur le bouton 'Controlflow', ce qui fera apparaître une nouvelle visualisation (figure 13).

En ce qui concerne la structuration du code, le projet comprend un code Javascript, HTML et CSS. Le nombre de lignes écrites pour chaque langage est montré dans le tableau 4. C'est surtout le code JavaScript (plus de 600 lignes), qui permet d'interroger la base de données et de reconstruire un workflow à partir des triplets RDF, qui est l'élément central de l'application.

En conclusion, notre application comporte deux pages principales : une page d'ac-

Tableau 4 – Nombre de lignes de code par fichier

Fichier	Nombre de lignes
CSS	238
HTML (index)	38
HTML (workflow_visualization)	81
JavaScript	645

cueil avec la liste des workflows et une page de visualisation des workflows. Dans cette page de visualisation, deux vues du workflow sont possibles : le *dataflow* et le *controlflow*.

5 Conclusion

Dans cette dernière partie, je vais vous rappeler ce que j'ai accompli lors de mon stage à l'Université Claude Bernard Lyon 1, en mettant en avant les objectifs atteints ainsi que les perspectives d'évolution pour les travaux réalisés. Ensuite, je partagerai mon expérience personnelle en détaillant les compétences que j'ai développées en modélisation de données, gestion de bases de données et développement web.

5.1 Récapitulatif des réalisations

Lors de mon stage à l'Université Claude Bernard Lyon 1 j'ai eu l'opportunité de travailler sur un projet qui est la visualisation de workflows et de leurs métadonnées. Ce projet, en lien avec le projet national ShareFAIR, vise à améliorer la reproductibilité et la réutilisation des workflows scientifiques, en particulier dans le domaine de la bio-informatique.

Durant ce stage, j'ai travaillé sur deux grands objectifs. Le premier consistait en la modélisation et la mise en place d'une base de données capable de stocker et de gérer efficacement des workflows et leurs métadonnées. Cela a impliqué la conception d'un métamodèle adapté, l'implémentation de ce modèle en utilisant le langage RDF, ainsi que l'importation et la gestion des données à l'aide de SGBD adaptés comme Jena. Le second objectif concernait le développement d'une interface web permettant de rechercher et de visualiser les workflows stockés dans la base de données. Cette interface vise à fournir une représentation claire et accessible des workflows, facilitant ainsi leur compréhension et leur réutilisation par les chercheurs.

5.2 Perspectives

Les perspectives de mon travail incluent plusieurs axes de développement futurs. Tout d'abord, il sera essentiel d'évaluer et de tester les limites du métamodèle actuel. Cela nécessitera de disposer d'autres workflows traduits en RDF, car jusqu'à présent, j'ai implémenté un seul workflow complet et un autre partiellement. Étant donné que j'ai principalement traduit des workflows issus du système Snakemake, je n'ai pas pu réellement vérifier la visualisation du *controlflow* avec des opérateurs de contrôle tels que les conditionnels.

À l'avenir, il est prévu que l'extraction des données se fasse directement à partir du code, ce qui permettra de remplacer la traduction manuelle des données en RDF, qui est une tâche plutôt fastidieuse. Cette évolution simplifiera et automatisera le processus, rendant la récupération et la manipulation des données plus efficaces et moins sujettes aux erreurs humaines.

Pour compléter cette extraction automatique, un autre point d'amélioration concerne la définition d'un schéma sur le métamodèle. Actuellement, certains triplets du métamodèle permettent d'indiquer quels type de ressources sont attendus en sujet ou objet d'un prédicat donné (en RDFS notamment). Cependant, ce ne sont pas des contraintes (pas d'obligation de les respecter). Récemment des langages comme Shex ou ShaCL ont été développés pour imposer des contraintes sur les données RDF. Écrire un schéma permettrait de s'assurer que les données respectent bien le métamodèle, ce qui est d'autant plus important quand le processus est automatisé.

Au niveau de l'interface, de nouvelles fonctionnalités pourront être intégrées pour enrichir encore davantage les capacités de l'outil : la recherche actuelle se limite à trouver les workflows contenant un mot-clé donné. Mais le langage d'interrogation développé par Mouna, la stagiaire de M2, devrait permettre des recherches plus avancées, par exemple de lister les workflows utilisant tel outil puis tel processus sur des données génomiques. D'autres fonctionnalités pourraient inclure la possibilité de sélectionner plusieurs workflows et de les comparer côte à côte, visualisant ainsi les différences et les similitudes entre eux. Cela aiderait les chercheurs à identifier rapidement les variations et les points communs entre différents workflows, les aidant ainsi à choisir le workflow le plus adapté pour leurs analyses.

5.3 Retour d'expériences

En conclusion, ce stage m'a permis de développer des compétences techniques et méthodologiques en modélisation de données, en gestion de bases de données et en développement web. Au début, j'ai trouvé complexe le fonctionnement des promesses en JavaScript, mais cette difficulté initiale m'a permis de maîtriser progressivement

ce concept fondamental. En particulier, ce projet m'a initié au langage JavaScript, un langage qui m'était totalement inconnu auparavant. Cette expérience m'a non seulement permis de mettre en pratique mes connaissances en langage RDF, mais aussi de contribuer à petite échelle à un projet national de gestion de workflows scientifiques. En effet, ce stage m'a sensibilisé à l'importance de la reproductibilité des workflows scientifiques. À travers mes tâches, j'ai mieux compris comment des processus rigoureux et bien documentés peuvent garantir la validité des résultats scientifiques et faciliter la collaboration entre chercheurs.

Références bibliographiques

[1] Projet ShareFAIR. https ://projet.liris.cnrs.fr/sharefair/.

[2] Sarah Cohen-Boulakia, Khalid Belhajjame, Olivier Collin, Jérôme Chopard, Christine Froidevaux, Alban Gaignard, Konrad Hinsen, Pierre Larmande, Yvan Le Bras, Frédéric Lemoine, Fabien Mareuil, Hervé Ménager, Christophe Pradal, and Christophe Blanchet. Scientific workflows for computational reproducibility in the life sciences : Status, challenges and opportunities. *Future Generation Computer Systems*, 75 :284–298, 2017.

[3] Sarah Cohen-Boulakia and Ulf Leser. Search, adapt, and reuse : The future of scientific workflows. *SIGMOD Record*, 40 :6–16, 06 2011.

[4] Casper Visser, Lennart Johansson, Purva Kulkarni, Hailiang Mei, Pieter Neerincx, K. van der Velde, Peter Horvatovich, Alain Gool, Morris Swertz, Peter Hoen, and Anna Niehues. Ten quick tips for building fair workflows. *PLOS Computational Biology*, 19 :e1011369, 09 2023.

[5] Carole Goble, Sarah Cohen-Boulakia, Stian Soiland-Reyes, Daniel Garijo, Yolanda Gil, Michael R. Crusoe, Kristian Peters, and Daniel Schober. FAIR Computational Workflows. *Data Intelligence*, 2(1-2) :108–121, 01 2020.

[6] Johannes Starlinger, Bryan Brancotte, Sarah Cohen-Boulakia, and Ulf Leser. Similarity search for scientific workflows. *Proc. VLDB Endow.*, 7(12) :1143–1154, aug 2014.

[7] Youngmahn Han. Bioworks : A workflow system for automation of bioinformatics analysis processes. *International Conference on Ubiquitous Computing and Multimedia Applications*, pages 76–81, 2011.

[8] Caspar Schmitt and Thomas Kuhr. A workflow management system guide. *arXiv*, 12 2022.

[9] Juan José Cadavid, Benoit Combemale, and Benoit Baudry. An analysis of metamodeling practices for mof and ocl. *Computer Languages, Systems & Structures*, 41 :42–65, 2015.

[10] Luma Lombello, Rita Catini, Rodrigo Bonacin, and Julio dos Reis. A metamodel for bridging heterogeneous ontologies. *SN Computer Science*, 2, 02 2021.

[11] nextflow. https ://www.nextflow.io/docs/latest/index.html.

[12] Snakemake. `https://snakemake.readthedocs.io/en/stable/`.

[13] S Pettifer, Jon Ison, Matúš Kalaš, Dave Thorne, P McDermott, Inge Jonassen, A Liaquat, José Fernández, Jose Manuel Rodríguez, David Valcárcel, C Blanchet, Mahmut Uludag, Peter Rice, Edita Karosiene, K Rapacki, M Hekkelman, O Sand, Heinz Stockinger, AB Clegg, and INB-Partners. The embrace web service collection. *NUCLEIC ACIDS RES*, 38, 07 2010.

[14] Bartosz Kiepuszewski, Arthur Ter, and Wil Aalst. Fundamentals of control flow in workflows. *Acta Informatica*, 39 :143–209, 03 2003.

[15] Ontologie CWL. https ://www.commonwl.org/v1.1/Workflow.html.

[16] Kate Voss, Geraldine A. Van der Auwera, and Jeff Gentry. Full-stack genomics pipelining with gatk4 + wdl + cromwell. *F1000Research*, 6, 2017.

[17] Ontologie WDL. https://docs.openwdl.org/en/latest/WDL/Base_structure/.

[18] Ontologie EDAM. https ://edamontology.org/page.

[19] W3. https ://www.w3.org/TR/rdf11-concepts/.

[20] rdf graphe. https ://www.ldf.fi/service/rdf-grapher.

[21] L. Hoberecht, P. Perampalam, A. Lun, and J-P. Fortin. A comprehensive bioconductor ecosystem for the design of crispr guide rnas across nucleases and technologies. *Nature Communications*, 13 :6568, 2022.

[22] Y. Voichek and D. Weigel. Identifying genetic variants underlying phenotypic variation in plants without complete genomes. *Nature Communications*, 52 :534–540, 2020.

[23] Adnan Kivanc Corut and Jason G Wallace. kGWASflow : a modular, flexible, and reproducible Snakemake workflow for k-mers-based GWAS. *G3 Genes|Genomes|Genetics*, 14(1), 11 2023.

[24] Tran Luong and Le Canh. Javascript asynchronous programming. *Hue University Journal of Science : Techniques and Technology*, 128, 07 2019.

[25] snakemake-crispr guides. https ://github.com/MPUSP/snakemake-crispr-guides/tree/main#workflow-overview.

Table des annexes

Annexe A

Tableau 1 – Diagramme de Gantt du projet

Tâches	**Semaines**								
Phase 1 : Familiarisation du projet	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Lecture bibliographique	X								
Sélection des workflows pour le prototype	X								
Représentation schématique des workflows et des métadonnées		X							
Représentation schématique du méta-modèle			X						
Phase 2 : Conception de la base de données	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Transformation du méta-modèle en triplet RDF				X					
Transformation des workflows en triplet RDF					X	X			
Teste de requêtes SPARQL sur Blazegraph						X	X	X	
Phase 3 : Conception de l'interface web	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Entraînement/compréhension du langage JavaScript						X	X		
Importation des données							X		
Développement de l'interface en JavaScript, HTML et CSS							X	X	
Ajout de fonctionnalités dynamiques en JavaScript								X	X

Annexe B

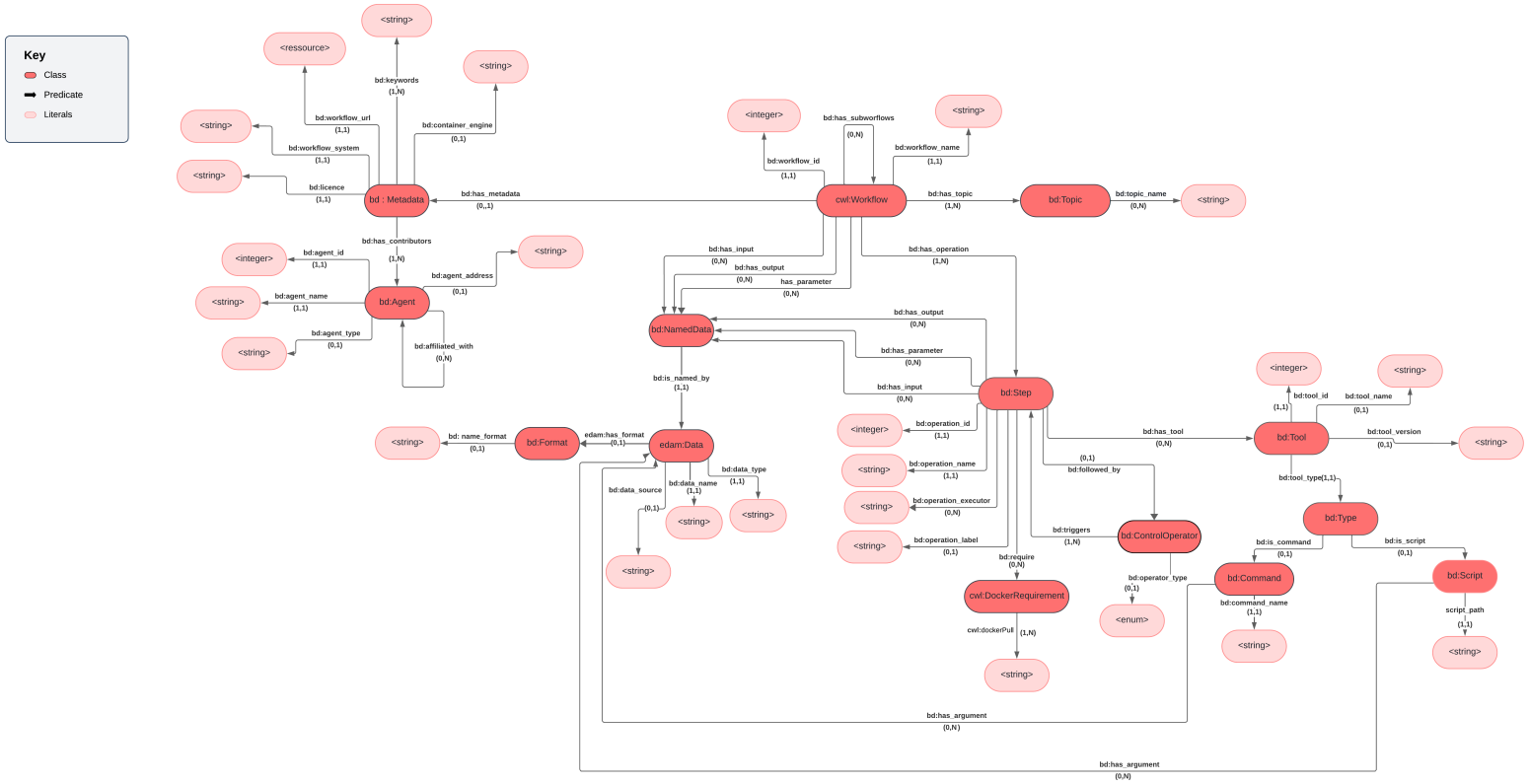

FIGURE 5 – Graphique RDF du métamodèle

La Figure 5 montre le schéma de notre métamodèle. Les classes en rouge définissent les types d'entités. Les flèches représentent les prédicats, et les littéraux en rose représentent divers types de données (chaînes de caractères, nombres, URI, etc.).

Annexe C

FIGURE 6 – Résultat de la fonction 'run_multiple_queries'

Annexe D

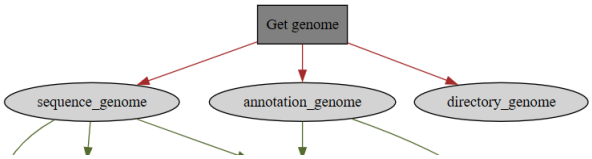

FIGURE 7 – Capture d'écran d'une partie du graphique représentant le *dataflow* du workflow

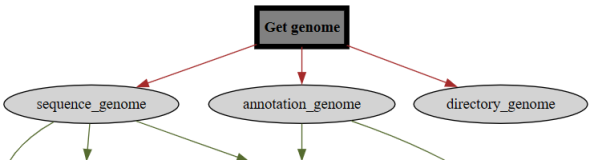

FIGURE 8 – Capture d'écran d'une partie du graphique représentant le *dataflow* du workflow avec un noeud sélectionné

Annexe E

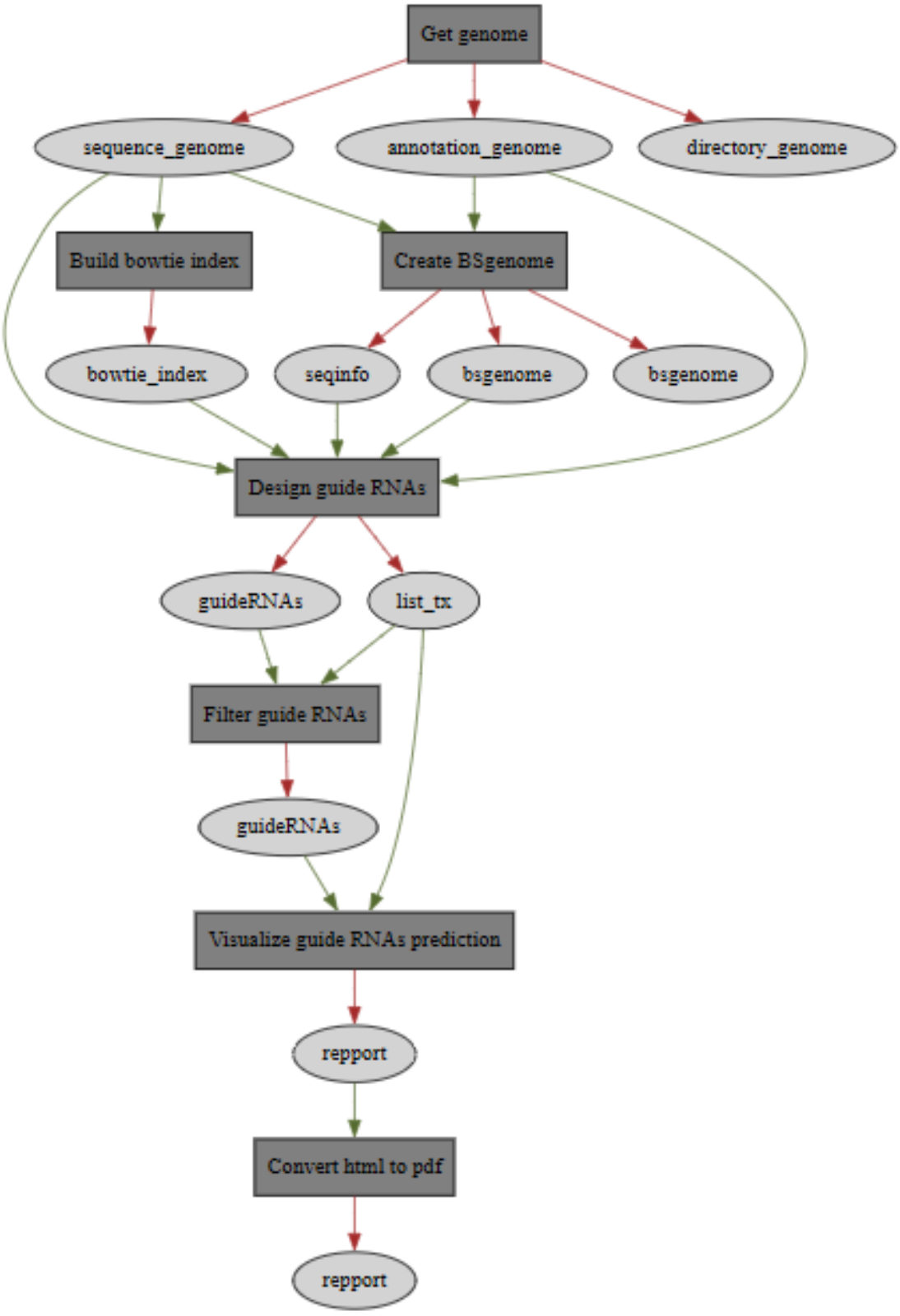

FIGURE 9 – Capture d'écran du *dataflow* crispr généré

Annexe F

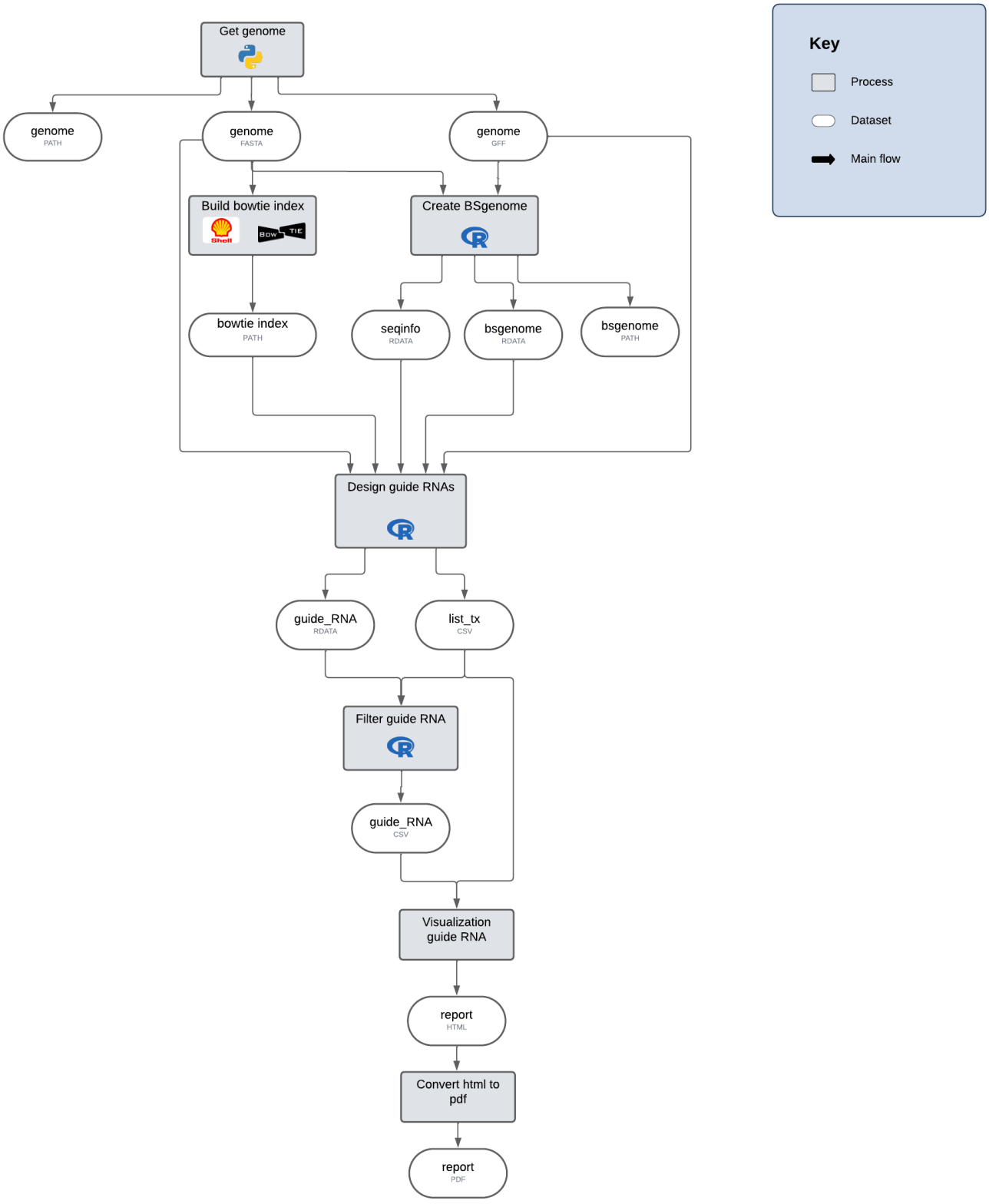

FIGURE 10 – Schéma du *dataflow* crispr généré

Annexe G

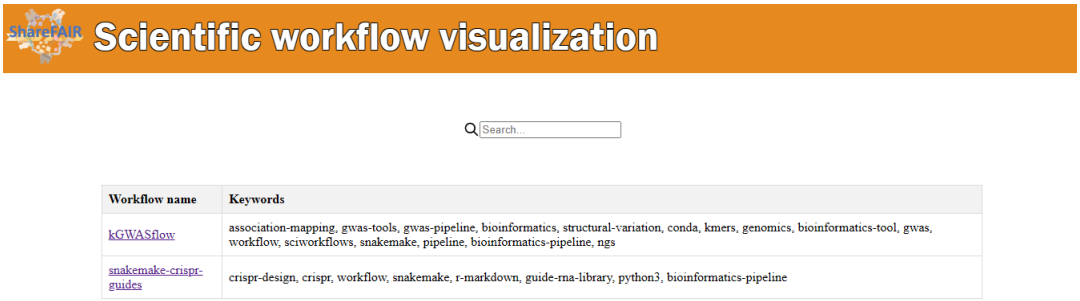

FIGURE 11 – Capture d'écran de la liste des workflows

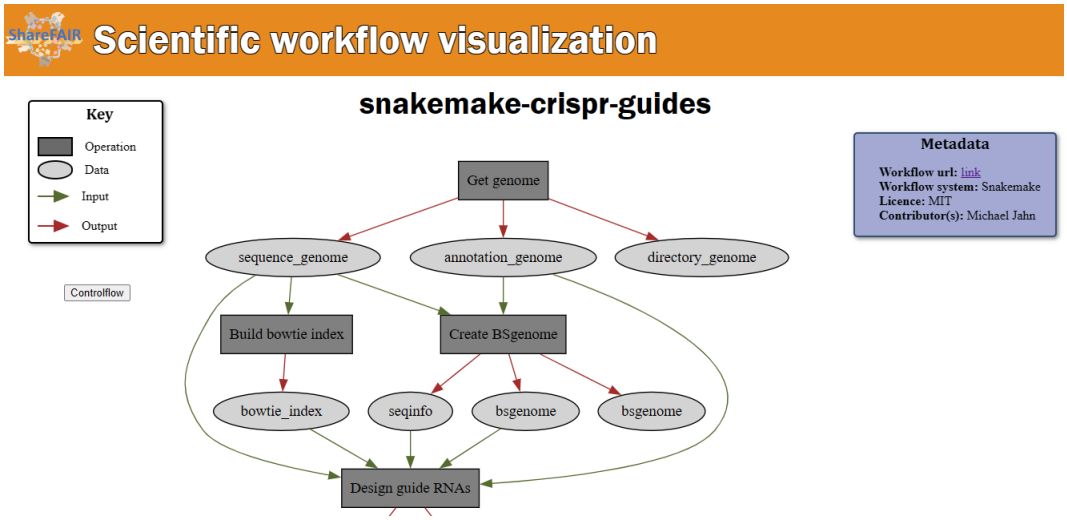

FIGURE 12 – Capture d'écran de la visualisation du *dataflow*

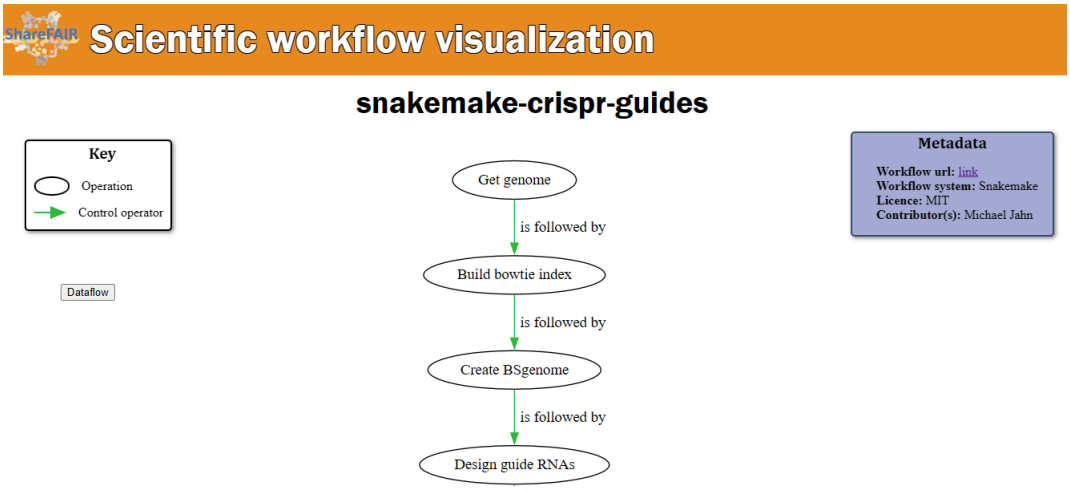

FIGURE 13 – Capture d'écran de la visualisation du *controlflow*

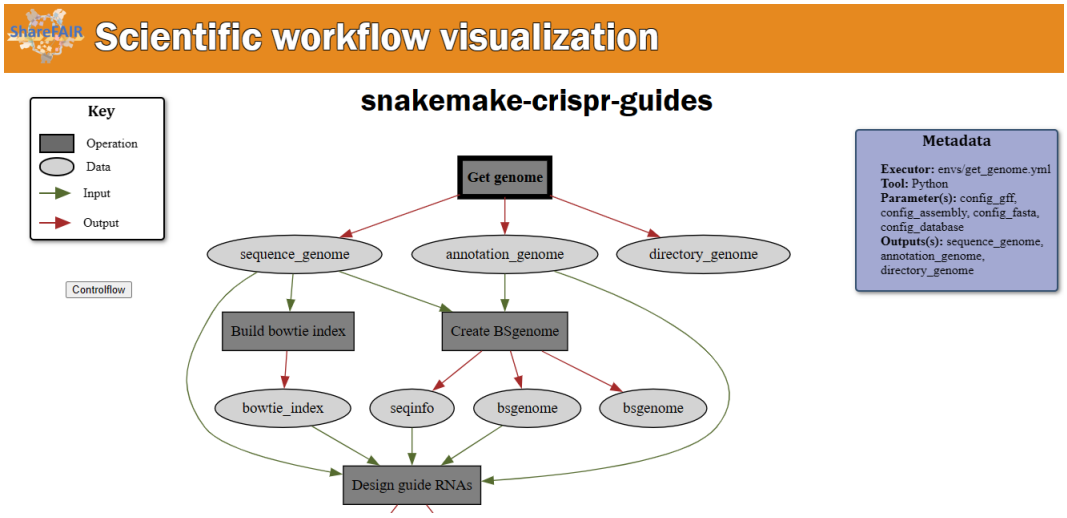

FIGURE 14 – Capture d'écran du workflow avec la sélection d'un noeud opération

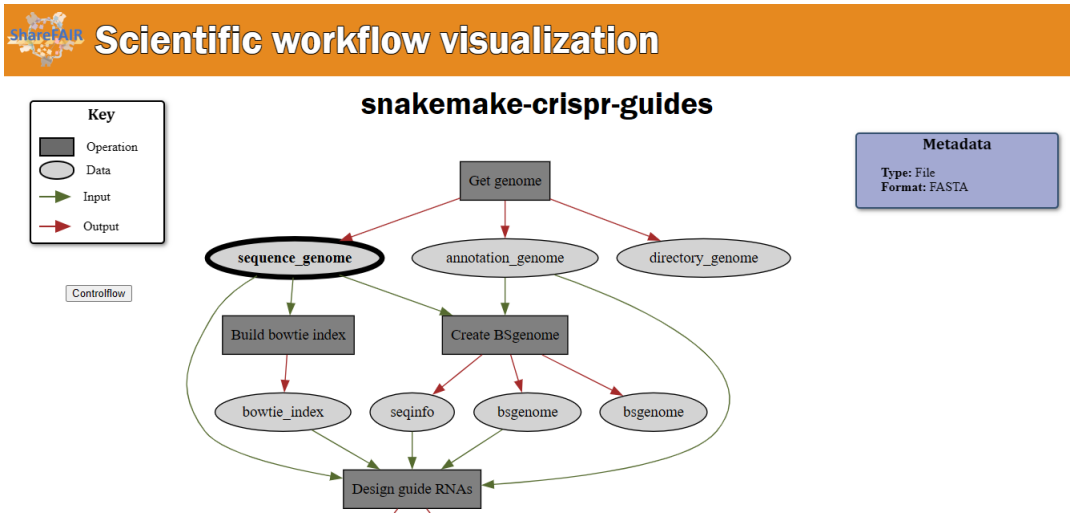

FIGURE 15 – Capture d'écran du workflow avec la sélection d'un noeud de données

Les figures 14 et 15 montrent que lorsqu'un nœud est sélectionné, il est mis en surbrillance. La table de métadonnées est automatiquement mise à jour avec les informations spécifiques au nœud sélectionné.