Licence Informatique

La Rochelle Université

Création d'une catactéristique du type XPath pour le framework GALACTIC

Kevin Voisin

2022

Table des matières

Liste des figures

Remerciements

Tout d'abord, je souhaite remercier Monsieur Yacine GHAMRI-DOUDANE, directeur du laboratoire L3i pour m'avoir accepté au sein de son laboratoire pour mon stage.

Je souhaite remercier mon maître de stage Monsieur Christophe DEMKO, enseignant-chercheur à l'université de La Rochelle, pour sa proposition de sujet de stage qu'il m'a faite, ainsi que pour toute l'aide qu'il m'a prodiguée pendant le stage. Cela m'a permis de mieux comprendre la plateforme sur laquelle j'allais devoir travailler, mais aussi pour ses explications sur les incompréhensions que j'ai pu avoir pendant mon stage.

Je remercie aussi ma tutrice, Madame Karell BERTET, enseignante-chercheuse à l'université de La Rochelle, pour ses explications concernant le fonctionnement d'un projet de recherche, mais aussi pour ses explications concernant la plateforme et l'algorithme NextPriorityConcept utilisé pour le projet.

Je souhaite aussi remercier les enseignants-chercheurs et les doctorants travaillant dans l'équipe du projet Galactic, pour leurs explications concernant la plateforme pendant les réunions.

Enfin, je remercie Alexandre BUFFARD, Jules BRICOU, Martin EHLINGER et Eléonore THONNEAU qui ont effectué leur stage en même temps que moi, pour leur aide et leur soutien tout au long de celui-ci. Je souhaite aussi les remercier pour le travail commun effectué sur la présentation d'entreprise ainsi que la présentation de la plateforme.

Résumé - Abstract

Résumé en français

Mon stage s'est déroulé au Laboratoire Informatique, Image et Interaction (L3i) de l'université de La Rochelle. J'ai travaillé au sein de l'équipe Modèles et Connaissances et plus précisément dans l'équipe travaillant sur le projet GALACTIC, qui est un projet de recherche sur le développement d'une plateforme qui étudie la théorie des treillis, la théorie des concepts ainsi que les systèmes d'implication et de fermetures.

Mon travail au sein de ce projet était de créer un plugin permettant l'extraction d'une donnée à partir d'un chemin d'accès dont la syntaxe était à définir. Le but est de simplifier l'extraction de données contenues dans des structures de données comme des listes ou des dictionnaires.

Dans un premier temps, j'ai dû installer la plateforme GALACTIC afin de pouvoir y travailler par la suite. Puis, j'ai fait des recherches sur différentes syntaxes existantes afin de pouvoir définir ma propre syntaxe pour le plugin. Enfin, j'ai dû développer le plugin de caractéristique ainsi que les différents tests.

Abstract

My internship took place in the "Laboratoire Informatique, Image et Interaction"(L3i) at the university of La Rochelle. I worked inside the team "Modèles et Connaissances" and more precisely with the team working on the GALACTIC project. The GALACTIC project stands for GAlois LAttices, Concept Theory, Implicational systems and Closures, and the goal is to develop a platform that studies all these concepts.

My work inside this project was the development of a plugin allowing the extraction of data from a path whose syntax was not defined yet. It aims to simplify the data extraction inside lists as well as in data structures such as dictionaries.

First, I had to install the GALACTIC platform so that I could work on it later. Then, I did some research on different existing syntaxes, to define my own syntax for the plugin. Finally, I had to develop the characteristic plugin as well as the tests.

1 Contexte du stage

1.1 Présentation de l'entreprise

La présentation de l'entreprise est issue du site de l'entreprise (https://l3i.univ-larochelle.fr/Presentation-317) et il s'agit d'un travail fait en commun avec Jules BRICOU, Alexandre BUFFARD, Martin EHLINGER, Eléonore THONNEAU et moi-même.

Le Laboratoire Informatique, Image et Interaction (L3i) a été créé en 1993 et près de 100 personnes travaillent au sein du laboratoire. Il s'agit d'un laboratoire de recherche de Sciences du Numérique de l'Université de La Rochelle et il associe les chercheurs en informatique de l'IUT ainsi que les chercheurs du Pôle Sciences de l'Université de La Rochelle. Il compte 33 chercheurs et chercheuses, 35 doctorants et doctorantes, 4 personnes permanentes de soutien à la recherche et 25 personnes sur projets et est dirigé par Monsieur Yacine GHAMRI-DOUDANE.

Le laboratoire L3i fait partie de réseaux de recherches régionaux (Fédération CNRS MIRES, ERT " Interactivité Numérique "), nationaux (GDR I3 et GDR IRIS) et internationaux (IAPR) dans les secteurs de visibilité de son action scientifique, autour d'un projet stratégique lié à la gestion intelligente et interactive des contenus numériques. Cela est renforcé grâce à une politique de volontariste de participation ou de pilotage de projets de recherches labellisés (ANR, PCRD, …), au sein desquels le laboratoire occupe couramment une position de leadership.

Le Laboratoire est également membre de l'Alliance Big data lancée en 2013 pour favoriser le développement, en France, de nouveaux services et projets dans ce domaine.

Ses travaux sont menés en partenariat avec une dizaine de centres de recherches nationaux (dont l'INRIA, institut spécialisé). Le L3i entretient par ailleurs des liens privilégiés avec de nombreux centres de recherche à travers le monde. Il est également engagé dans près d'une vingtaine de partenariats industriels sur l'ensemble du territoire français.

Le laboratoire du L3i est divisé en 3 équipes:

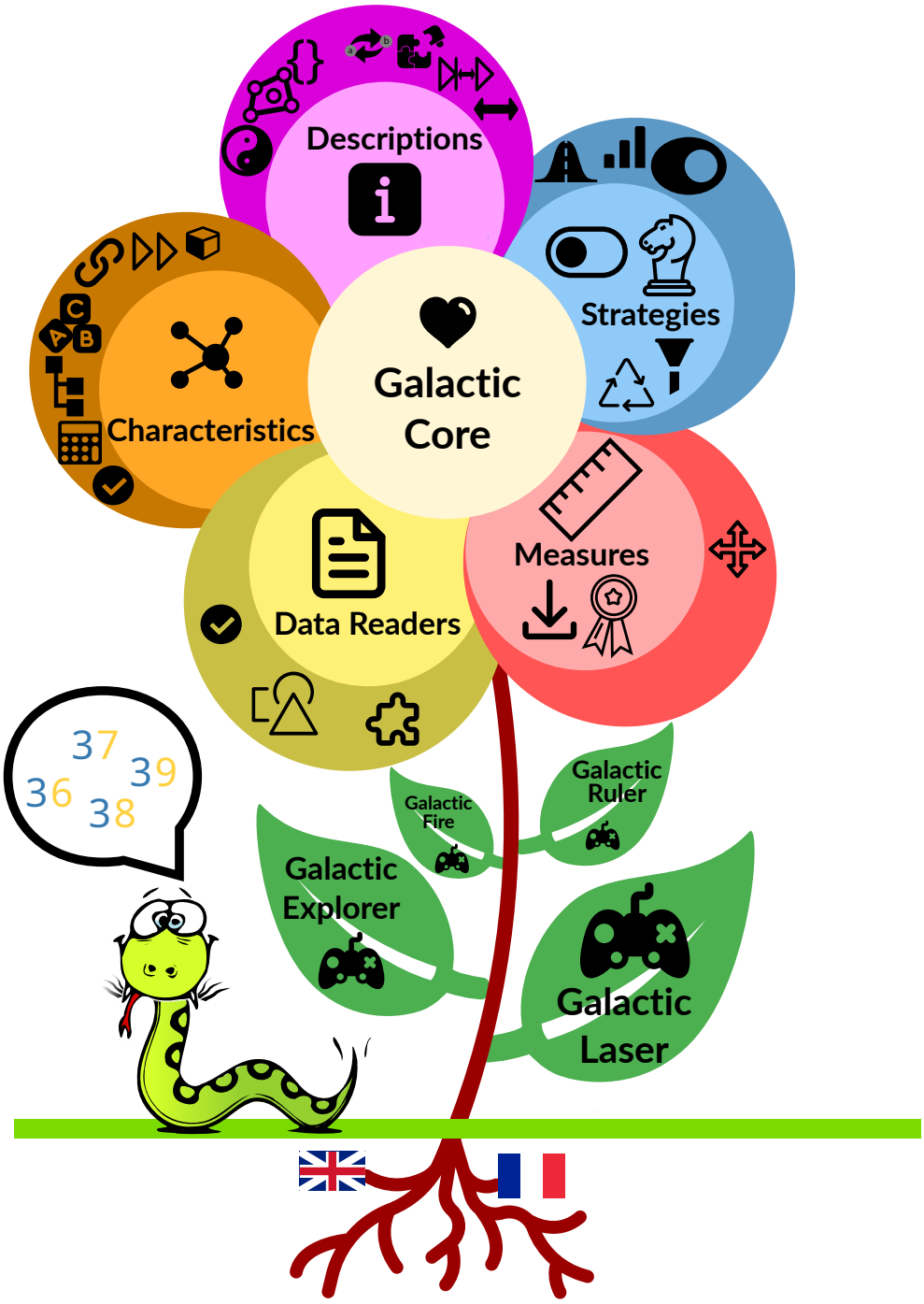
- Modèles et Connaissances
- Images et Contenus
- Dynamique des systèmes et Adaptativité

Mon stage se déroule au sein de l'équipe Modèles et Connaissances et plus précisément autour du projet GALACTIC de cette équipe.

1.2 La plateforme GALACTIC

La présentation de la plateforme GALACTIC est issue du du diaporama de présentation de l'architecture de la plateforme (https://galactic.univ-lr.fr/slides/architecture/Galactic-Architecture-slides-20min-complete.pdf) et il s'agit d'un travail fait en commun avec Jules BRICOU, Alexandre BUFFARD, Martin EHLINGER, Eléonore THONNEAU et moi-même.

GALACTIC est l'acronyme de **GA**lois **LA**ttices **C**oncept **T**heory **I**mplicational system and **C**losures. L'objectif de la plateforme est de pouvoir mettre en place un framework permettant de travailler sur l'Analyse Formelle des Concepts ainsi que la Théorie des treillis. La plateforme GALACTIC est structurée de la façon suivante:



- Un noyau, représenté par le coeur de la fleur, qui contient les opérations de base et structures de données et qui implémente un nouvel algorithme (NextPriorityConcept) inspiré des pattern structures.

- Des plugins, représentés par les pétales de la fleur:
  - Caractéristiques : Booléen, Numérique, Catégorique, String, Séquentielle, Chain, Triadic. Ils permettent de définir les caractéristiques.
  - Descriptions : Booléen, Logique, Catégorique, String (regex), String (distance), Chain, Séquentielle, Séquentielle (distance), Triadic. Ils définissent les prédicats et les espaces de description utilisés pour représenter et définir les données avec précision.
  - Stratégies : Booléen, Logique, Catégorique, Numérique, String, String (distance), Chain, Séquentielle, Séquentielle (distance), Triadic. Ils définissent la manière utilisée pour explorer les données et utilisent des descriptions pour générer des prédécesseurs pour chaque concept dans le treillis.
  - Mesures : predecessor Cardinality, sucessor Cardinality, Confidence. Ce sont des paramètres des stratégies de filtrage prédéfinis dans la librairie core.
  - Data Readers : Ils sont utilisés pour lire différents types de fichiers de données. Le moteur de base détecte le type de fichier en utilisant son extension. Les plugins Data Readers existants sont: YAML, JSON, CSV, TOML, INI, TXT, SLF, DAT, CXT.
  - Localization: Ils sont utilisés pour traduire les applications dans d'autres langues.

- Des applications, représentées par les feuilles de la fleur, qui sont développées afin d'utiliser la librairie de la plateforme. Ce sont des interfaces pour l'utilisateur:
  - GALACTIC Laser : pour construire les treillis et explorer les données.
  - GALACTIC Explorer : pour explorer de façon interactice les treillis contruits.
  - GALACTIC Ruler : pour extraire les règles d'implication.
  - GALACTIC Fire : pour exécuter un système de règles.

- La bulle au dessus du serpent indique les différentes versions de Python avec lesquelles la plateforme GALACTIC est compatible.

- Au niveau des racines on peut voir les différentes langues disponibles sur la plateforme.

1.3 Présentation du sujet

Mon stage prend place dans le laboratoire L3i dans le projet GALACTIC de l'équipe Modèles et Connaissances présenté plus tôt.

Mon sujet est la création d'un plugin de caractéristique qui devra permettre l'accès à une donnée se situant dans des structures de données telles que des listes ou des dictionnaires.

Lors de la présentation de la plateforme et de nos sujets par mon maître de stage, il m'a précisé que ce plugin prendra place dans le noyau de la plateforme et non pas en plugin externe comme je le pensais en ayant lu le sujet.

L'objectif de ce plugin est de pouvoir y accéder en donnant en paramètre une chaîne de caractères contenant le chemin d'accès jusqu'à la donnée souhaitée. La syntaxe du chemin n'étant pas définie, j'ai pour mission dans un premier temps de la définir. Pour cela, mon maître de stage m'a donné plusieurs pistes proposant des syntaxes de chemins pour accéder à des données, comme les sélecteurs du langage CSS ou bien les chemins Xpaths.

Les personnes travaillant sur le projet GALACTIC peuvent déjà accéder à ce type de données, cependant, il n'existe pas de méthode simple pour le faire, et elles doivent parcourir les structures une à une afin de trouver la donnée. Mon plugin devra donc automatiser le processus et parcourir toutes les structures précisées dans le chemin pour ensuite retourner la ou les valeurs souhaitées.

2 Déroulement du stage

2.1 Installation de la plateforme

Avant de pouvoir travailler sur le sujet en tant que tel, j'ai d'abord dû installer la plateforme GALACTIC sur mon ordinateur, pour cela, je me suis aidé du guide d'installation fourni par le site de GALACTIC (voir annexe). Dans un premier temps, j'ai tenté l'installation sous le système d'exploitation Windows 10, car étant celui que j'utilise principalement. Cette installation utilise le logiciel MSYS2 qui permet l'installation de paquets comme Python et GALACTIC dans notre cas. Cependant, j'ai eu de nombreux problèmes lors de l'installation, car le guide n'était plus à jour, comme m'avait prévenu mon maître de stage. Avec son aide, j'ai réussi à l'installer sur ma machine, cependant, il ne s'agissait que de la dernière version stable, ce qui n'était pas bon pour notre stage, car nous devions travailler sur la version développeur.

J'ai donc décidé d'installer un dual boot Linux avec la distribution Ubuntu afin de pouvoir l'installer correctement. De nombreux enseignants-chercheurs et doctorants travaillant sous Linux, j'ai donc pensé qu'il s'agissait de la meilleure solution. J'ai réussi l'installation sans soucis majeurs, mon maître de stage m'ayant guidé tout au long de l'installation. Les commandes permettant l'installation étaient similaires à celles du guide à une commande près.

2.2 Recherches

Après avoir réussi l'installation de mon environnement de travail, j'ai commencé mes recherches sur les différentes syntaxes qui existaient pour les chemins d'accès de données. Notre syntaxe doit pouvoir gérer les différentes structures de données. Il fallait aussi que je me renseigne sur leurs imbrications et donc chercher de potentiels séparateurs entre les éléments.

Je me suis principalement renseigné sur la syntaxe du langage Python, qui est le langage de programmation utilisé pour le développement de la plateforme. Puis j'ai exploré les pistes des sélecteurs en CSS ainsi que sur les chemins Xpath. Je n'ai pas regardé la syntaxe des autres langages de programmation, car ces syntaxes sur ses types de données sont presque toutes identiques à celle de Python.

2.2.1 Python

J'ai commencé par regarder la syntaxe du langage Python, car est celui utilisé pour programmer la plateforme GALACTIC. Le langage Python possède bien les deux types de structures de données, ce qui est un point positif.

Regardons de plus près un exemple de la syntaxe liée à une liste:

```python
# Définition d'une liste
liste = ["abricot","pêche","mangue", 3]
# Sélection d'un ou plusieurs éléments de la liste
liste[1]
liste[0:1]
# Sélection d'un élément dans des listes imbriquées
```

```
liste[2][1]
```

Python nous permet d'accéder de façon facile à des éléments de listes imbriquées en ayant juste à préciser les différents index. Cependant, il n'y a pas de caractère permettant la séparation des différents éléments, ce qui pourrait poser problème au niveau de la lisibilité.

La syntaxe liée aux dictionnaires est également relativement simple:

```python
# Définition d'un dictionnaire
dictionnaire = {"nom":"Dupont", "age":30, 10:"oui"}
# Sélection d'un élément
dictionnaire["nom"]
dictionnaire[10]
#Sélection d'un élément dans des dictionnaires imbriqués
dictionnaire["profils"]["age"]
```

On peut remarquer que l'on va avoir plusieurs problèmes pour notre syntaxe si on utilisait que celle du Python natif. Le problème étant davantage pour notre compréhension que pour la machine. En effet, le type de la clé d'un dictionnaire peut être un entier, ce qui va nous poser problème, car la syntaxe d'accès à un élément d'une liste ou d'un dictionnaire est identique. On ne pourrait donc pas savoir s'il s'agit d'une donnée dans un dictionnaire ou dans une liste, sans connaître l'arborescence. Après discussion avec mon maître de stage, nous pensons qu'il faudrait prévenir l'utilisateur de ne pas utiliser d'entier en tant que clé. Il serait tout de même toujours possible d'utiliser une clé uniquement numérique, mais sous forme de chaîne de caractères.

Enfin, comme pour la liste, il n'y a pas de séparateur entre les différents éléments que l'on parcourt, ce qui peut réduire la lisibilité.

2.2.2 Chemins de sélecteurs CSS

Une autre piste pour notre syntaxe pourrait être les sélecteurs CSS. En effet, ce langage devant parcourir l'arborescence d'un fichier HTML, cela se rapproche de notre situation. Il n'existe cependant pas de syntaxe concernant les listes (au sens défini par des langages comme Python) ou des dictionnaires. Néanmoins, le CSS permet de sélectionner de façon précise des balises. Il existe plusieurs combinateurs pour sélectionner par exemple des balises adjacentes, ou bien des balises descendantes. Il existe aussi des variantes à certains combinateurs afin de pouvoir optimiser les performances et nos sélections. Par exemple, le combinateur "+" permet de sélectionner uniquement la balise suivante au même niveau de l'arborescence tandis que le combinateur "~" permet de sélectionner toutes les balises qui suivent.

```
ul + p {} /*S'applique à la première balise p suivant une balise ul*/
ul ~ p{} /*S'applique à toutes les balises p suivantes*/
```

Il existe deux combinateurs, ">" et " " qui permettent les mêmes subtilités, mais pour les balises enfants. Il existe aussi le sélecteur universel "*" qui permet de sélectionner tous les éléments. Cependant, dans notre cas d'étude, je ne pense pas qu'il nous serve, car il nous suffirait de nous arrêter à un niveau de l'arborescence plus haut pour alléger l'écriture, tout en conservant le même résultat. Il existe aussi des sélecteurs nous permettant d'ajouter des contraintes, comme des expressions régulières à des balises.

Cela cible particulièrement les attributs dans une balise.

```
a[href*="galactic"]{} /*S'applique aux liens contenant "galactic"*/
a[hrefˆ="galactic"]{} /*S'applique aux liens commençant par "galactic"*/
a[href$="lr.fr"]{} /*S'applique aux liens finissant par "lr.fr"*/
```

Il existe des pseudo-classes permettant de sélectionner des éléments en particulier dans une liste, comme par exemple *nth-child(n)* qui permet de choisir le nième élément. La pseudo-classe *nth-last-child(n)* permet la même manipulation, mais en partant de la fin de la liste.

```
li:nth-child(2){} /*S'applique au 2e élément de la liste*/
li:nth-last-child(2){} /*S'applique à l'avant dernier élément */
```

Il existe aussi des pseudo-classes permettant de choisir non pas les balises enfants, mais les types de balises. Elles utilisent la même syntaxe que pour les enfants.

2.2.3 Xpath

Une autre solution pourrait être de s'inspirer de Xpath qui est un langage de requête permettant l'exploration des arborescences XML. Xpath a pour objectif de définir une syntaxe plus simple que celle proposée par le XML.

De plus, depuis la mise à jour 3.1, le langage permet l'utilisation des maps (dictionnaires) ainsi que des tableaux.

Syntaxe des chemins Xpath

Dans Xpath, pour naviguer entre les noeuds, on peut préciser plusieurs paramètres.

- Un **axe**: il permet de situer les autres noeuds dans l'arbre par rapport au noeud courant.
- Un **test de noeud**: il permet de limiter les noeuds présents sur l'axe en utilisant des filtres, sur des noms ou sur des types de données.
- Des **prédicats**: ils permettent de filtrer davantage le résultat en utilisant par exemple des expressions régulières, des fonctions ou bien des conditions.

```
axe::test[predicat1][predicat2]...
```

Pour séparer les différents noeuds du chemin, Xpath utilise le caractère "/". Il s'agit de la même séparation utilisée dans les liens de Windows ou bien dans des URL. Je pense que ça pourrait être une bonne solution pour notre syntaxe, afin d'ajouter de la lisibilité.

Voici ci-dessous deux exemples d'un chemin Xpath:

```
/order/items/book[title="Harry Potter"]/comment/text()

para[@type="warning"][5]
```

Dans le premier exemple, on parcourt les différents noeuds, en précisant pour le noeud "book" que le titre doit être "Harry Potter", puis on récupère le texte contenu dans le noeud "comment".

Dans le deuxième exemple, on choisit de récupérer le cinquième élément du résultat de la recherche, qui doit être un noeud "para" avec un attribut "type" contenant la valeur "warning".

On remarque que dans le deuxième exemple, je n'ai pas précisé le "/" au début du chemin. Si on précède le chemin d'un "/", on choisit un chemin absolu, c'est-à-dire à partir de la racine du document. Dans le deuxième cas, on précise un chemin relatif, c'est-à-dire à partir du noeud courant.

On peut aussi choisir de sélectionner tous les noeuds enfants en utilisant le symbole "*". Cette notation est aussi présente dans des langages comme le SQL par exemple.

Syntaxe des structures Xpath

Maintenant, que nous connaissons comment fonctionne les chemins, nous allons nous intéresser à différentes structures de données présentes dans Xpath.

Je détaillerai ici les deux types de structures: les listes et les maps.

Les listes possèdent deux syntaxes différentes en fonction des usages. Tout d'abord, il existe les listes "classiques", qui possèdent la même syntaxe pour les définir que le Python par exemple.

```
[1,2,3,$x]

/*Pour sélectionner un élément*/
[1,2,3,$x](2) /* Affiche 2 */
```

Il existe aussi une syntaxe s'apparentant plus à un tableau qui est moins permissive. Par exemple, il n'est pas possible de conserver deux séquences (tuples) dedans.

```
array { 1,2,"bonjour",7 $x ,(1,2,3)}
```

Il est possible d'imbriquer plusieurs listes entre elles, cependant, il n'a pas l'air possible de le faire avec la deuxième syntaxe.

```
["a",[1,2,3]](2)(3) /*Pour récupérer la valeur 3 du tableau*/
```

Les maps se définissent aussi presque comme les dictionnaires en python.

```
map{"nom":"Dupont", "age":30}

nomMap("nom") /*Pour récupérer la valeur*/
```

On peut aussi naturellement imbriquer des maps entre elles ainsi que différentes maps dans des listes.

2.2.4 Comparaison des solutions trouvées

	Python	**CSS**	**Xpath**
Listes:			
Créer une Liste	✓	×	✓
Sélectionner un élément	✓	×	✓
Sélectionner plusieurs éléments	✓	×	×
Gestion listes imbriquées	✓	×	✓
Maps:			
Créer une Map	✓	×	✓
Sélectionner un élément	✓	×	✓
Gestion maps imbriquées	✓	×	✓
Autres:			
Présence de séparateur(s)	×	✓	✓

	Python	CSS	Xpath
Sélectionner un élément	✓	✓	✓
Sélectionner plusieurs éléments	✓	✓	✓

2.3 Définition de la syntaxe

Maintenant que nous avons analysé plusieurs solutions possibles pour notre syntaxe, nous allons devoir la définir.

Après une concertation avec mon maître de stage, nous avons convenu de commencer les chemins avec le caractère "~", qui correspond à la base de notre arborescence. De plus, nous préciserons uniquement les clés quand il s'agit d'une map, et l'index lorsqu'il s'agit d'une liste.

Comme nous avons pu le voir, si on utilise uniquement la syntaxe de python pour sélectionner dans une liste ou une map, il pourrait y avoir une ambiguïté, car les deux utilisent les caractères "[]". C'est pour cela que l'on utilisera les crochets uniquement pour les listes, car tous les langages l'utilisent. Pour le cas d'une map, nous écrirons simplement le nom de la clé en le précédant d'un "/".

De plus, notre syntaxe devra nous permettre de sélectionner plusieurs éléments dans une liste, s'il s'agit du dernier élément de notre chemin. Pour cela, on utilisera la syntaxe native de python à ce sujet, à savoir l'utilisation du caractère ":" dans les crochets pour pouvoir préciser si l'on souhaite un index de début, et un index de fin.

2.3.1 Exemples

On souhaite accéder à la valeur de l'index 1 situé dans une liste contenue dans un dictionnaire avec la clé "galactic":

```
~/galactic[1]
```

On souhaite accéder à la valeur de la clé "10" d'une map située dans un dictionnaire contenu à l'élément 3 d'une liste contenue dans une valeur avec la clé "université":

```
~/université[3]/10
```

On souhaite accéder aux 5 premières valeurs d'une liste située à la racine:

```
~[0:4]
```

On souhaite accéder à toutes les valeurs sauf les 3 premières d'une liste située à l'index 2 d'une autre liste :

```
~[2][2:]
```

On souhaite accéder au nom d'une personne dans un dictionnaire, situé au dernier index d'une liste:

```
~[-1]/nom
```

2.3.2 Définition des expressions régulières

Note

Pour plus d'informations sur la syntaxe des expressions régulières en Python: https://docs.python.org/fr/3/howto/regex.html

Dans cette partie, nous allons nous intéresser à la définition de l'expression régulière qui nous permettra de valider ou non la chaîne de caractères donnée en paramètre. Nous décomposerons la définition en plusieurs parties, pour ensuite les assembler.

Liste

Pour les listes, notre expression devra uniquement accepter une chaîne de caractère commençant par un "[" , et se finissant par "]". À l'intérieur de celle-ci, il devra y avoir au minimum un chiffre, ou un ":". Il pourra y avoir évidemment plusieurs chiffres.

J'ai donc pensé à l'expression régulière suivante:

```
"^\[((\d*:\d*)|(\d+))\]$"
```

On commence donc notre chaîne par un crochet ouvrant, puis il peut soit y avoir 0 ou plusieurs chiffres allant de 0 à 9 , un caractère ":", suivi de 0 ou de plusieurs chiffres allant de 0 à 9, soit il y aura au moins un chiffre allant de 0 à 9. L'expression devant se terminer par un crochet fermant.

Exemple de cas valides:

```
[:] # sélection de tous les éléments
[0] # sélection d'un élément
[0:] # sélection de tous les éléments après celui précisé en index
[:3] # sélection de tous les éléments avant celui précisé en index
[0:3] # sélection des éléments se situant entre les deux index
```

Exemple de cas non valides:

```
[] # Cas où l'on donne une liste vide
[a] # Cas où l'on donne une chaîne de caractère en argument
[a0] # Cas où l'on donne une chaîne de caractère contenant un chiffre
[!a] # Cas où l'on donne une chaîne de caractère contenant un caractère
↪  spécial
[a:0] # Cas où le premier argument est une chaîne de caractère
[0:a] # Cas où le deuxième argument est une chaîne de caractère
```

Dictionnaire

Pour les clés des dictionnaires, mon maître de stage m'a demandé de commencer par les clés alphanumériques, car elles sont plus simples à définir. Pour définir une clé, nous avions choisi de les commencer par le caractère "/". Notre expression devra donc reconnaître au début de notre chaîne de caractère un "/", suivi d'une chaîne pouvant contenir des lettres et des chiffres.

J'ai donc pensé à l'expression régulière suivante:

```
"^\/\w+"
```

La macro "\w" désigne toutes les lettres, ainsi que les chiffres et le "_".

Exemple de cas valides:

```
/a # Cas où il y a une chaîne en minuscule
/A # Cas où il y a une chaîne en majuscule
/1 # Cas où il y a des chiffres
/a1Bé # Cas mixte
```

Exemple de cas non valides:

```
/ # Cas où il n'y a pas de clé
a # Cas où il n'y a pas de "/"
/a1! # Cas où il y a un caractère spécial
```

Nous allons devoir modifier l'expression afin qu'elle puisse contenir des caractères spéciaux. Pour éviter les confusions, nous précéderons tous les caractères spéciaux par un "\".

```
"^\/(\w|(\\([!&^.~#[\]$()*@°+/?|\\{}])))+"
```

On commence donc notre chaîne de caractère par un "/", puis un ou plusieurs caractères alphanumériques ou des caractères spéciaux précédés d'un "\".

Exemple de cas valides:

```
# Cas avec une chaîne purement alphanumérique
/a1Bcé
# Cas où il y a des caractères spéciaux utilisés dans notre syntaxe
/ea\[12\/
```

Exemple de cas non valides:

```
#Cas où il n'y a pas de "/" en début de chaîne
abc\[
#Cas où les caractères spéciaux ne sont pas précédé d'un "\"
/ab[]1
```

Définition de l'expression complète

Maintenant, que l'on a défini les différentes parties de notre expression, nous allons devoir en créer une qui nous permette de définir des chemins complets. Cependant, il y aura certaines restrictions, comme par exemple, si on souhaite accéder à plusieurs valeurs d'une liste, elle devra se situer en fin de chemin.

```
"^~((\[\d+\])|(/(\w|(\\[!&^.~#[\]$()*@°+/?|\\{}]))+))*(\[\d*:\d*\])?"
```

Cette expression nous demande donc de commencer par le caractère "~", comme me l'a conseillé mon maître de stage, puis peut contenir une suite d'index de liste ou bien de clé. A la fin de celle-ci, on pourra ajouter une liste contenant un index de début et un de fin.

Nous avons choisi de commencer tous nos chemins avec le caractère “~” afin d’ajouter en lisibilité, mais aussi, car cela représente l’origine du chemin et donc de notre structure à explorer.

Exemple de cas valides:

```
#Cas où l'on cherche un index dans une liste contenue dans un dictionnaire
~/galactic\\\[core1[1]
#Cas où l'on cherche plusieurs clés imbriquées
~/galactic/src/main
#Cas où l'on cherche plusieurs éléments dans plusieurs listes imbriquées
~[1][2][3:5]
```

On remarque que pour utiliser un caractère spécial, on l’a précédé du caractère “\”.

Exemple de cas non valides:

```
#Cas où on a un caractère spécial non précisé
~/galactic[abc
#Cas où l'expression ne commence pas par "~"
/galactic
```

Après avoir parlé avec mon maître de stage, nous avons convenu que les clés de dictionnaires contenant uniquement des chiffres devraient être de type chaîne de caractères, car il n’existe pas de moyen de vérifier si la clé est un entier ou une chaîne de caractères.

2.4 Développement du plugin

Note

Cette partie est encore en cours de développement et le code présenté ici n’est donc pas représentatif du plugin final.
Pour plus d’informations concernant les méthodes présentées, le lien vers la documentation officielle est présent en annexe.
Plus d’informations sur le module **re** de Python dans l’annexe.

Après avoir défini la syntaxe, il faut désormais réaliser le plugin. Comme dit précédemment, il devra se situer dans le coeur de l’application, avec la définition des autres plugins de caractéristiques présents nativement dans celui-ci. Ce programme contiendra la classe “Path” et devra implémenter différentes méthodes, comme les méthodes __init__ et __call__ qui sont des méthodes appelées automatiquement par la classe.

La méthode __init__ est appelée lors de la création de notre objet. Elle doit permettre l’initialisation des variables de la classe, dont notre chemin que l’on donnera en paramètre de fonction.

Ce code est en cours de réalisation:

```
def __init__(self,
             path: str = "~",
             characteristic: Optional[Characteristic] = None,
             **kwargs: Any,
             ) -> None:
```

```
    """
    Initialise a :class:`Path` instance.

    Keyword Arguments
    -----------------
        path: str, optional
            The  path to the value.
        characteristic: :class:`Characteristic`, optional
            The inner characteristic (if any).
        **kwargs:
            Unused but useful for multi-class inheritance.
    """
    if characteristic is None:
        super().__init__(**kwargs)
    else:
        super().__init__(characteristic, **kwargs)

    self.__regex =\
↪   re.compile(r'^~((\[\d+\])|(/(\w|(\\[!&^.~#[\]$()*@°+/?|\\{}]))+))*(\[\d*:\d*\])?')
    # Initialise and compile the regular expression for the verification
    ↪  of the path syntax
    if self.__regex.fullmatch(fr'{path}'):
        # If the path fully correspond to the syntax, we create a variable
        ↪  and store the value.
        self.__path = path
    else:
        # If the path doesn't respect the syntax, we raise an error
        raise ValueError("The given path doesn't suit the syntax")
```

Comme nous pouvons le constater, notre fonction utilise un chemin, cependant, j'ai choisi de mettre une valeur par défaut "~", qui sera utilisé dans le cas où on ne préciserait pas de chemin. Je pense que dans ce cas, il faudrait simplement retourner la ou les valeurs contenues dans la structure donnée. Cette structure de donnée sera donnée en paramètre de notre fonction __call__ qui n'est pas encore définie.

Le fonctionnement de notre méthode __init__ est simple, on commence par utiliser le constructeur de la classe mère "Characteristic", qui est déjà définie dans le coeur de la plateforme, et qui sert de base à toutes les différentes caractéristiques. Puis on crée notre variable contenant l'expression régulière que l'on a définie précédemment et qui est compilée. Pour les expressions régulières, j'ai utilisé le module **re** de Python.

Enfin, nous vérifions que le chemin donné en paramètre corresponde à notre syntaxe, dans ce cas, on stocke notre chemin dans une variable. Dans le cas contraire, on lève une erreur.

Je dois aussi définir une méthode __call__ qui servira à l'extraction de la ou les valeurs souhaitées. Elle prendra comme je l'ai déjà précisé une structure en paramètre, une liste ou un dictionnaire pour être

plus précis. Si la structure et le chemin donné ne correspondent pas, je pense qu'il faudra renvoyer une valeur "None", qui voudrait dire que la valeur souhaitée n'existe pas. Dans le cas contraire, on devra retourner la valeur.

De plus, après avoir regardé le fonctionnement des autres plugins de caractéristiques, il faudra aussi implémenter différentes méthodes, comme une méthode __str__ qui devrait retourner une chaîne de caractère contenant les valeurs. Il y aura aussi une fonction d'égalité à créer, et qui doit servir à comparer notre objet à une valeur.

Je devrais aussi implémenter des assesseurs qui devront retourner les valeurs de nos paramètres, comme par exemple une méthode retournant notre chemin.

Voici un exemple où la méthode retourne soit la caractéristique si elle existe, ou None:

```python
    @property
    def characteristic(self) -> Optional[Characteristic]:
        """
        Get the inner characteristic.

        Returns
        -------
            :class:`Characteristic`, optional
                The inner characteristic.
        """
        if len(self.components) == 0:
            return None
        return self.components[0]
```

Dans cette fonction, on regarde si la variable "components" de la classe "Characteristic" est vide. Dans ce cas, il n'y a pas de valeur à retourner, sinon on retourne la caractéristique, qui se situe dans l'index 0 de notre tableau. La précision de ce type de retour est possible grâce au type Optional, qui permet de préciser un type de retour, avec un retour de None par défaut.

Après avoir développé la classe, il faudra que je développe un fichier contenant les différents tests des méthodes de ma classe.

2.5 Avancement

Dans cette partie, je présenterai deux plannings concernant mon avancement dans mon stage. Le premier sera celui prévisionnel et le deuxième sera un planning effectif de mon avancée.

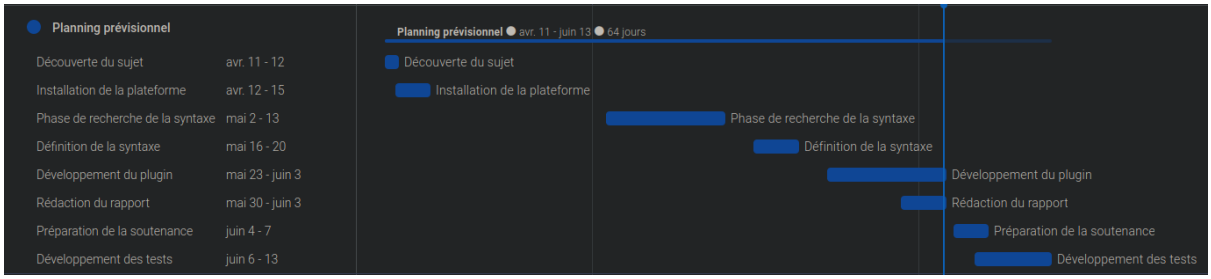


Figure 1: Planning prévisionnel

Dans ce premier planning, j'avais estimé qu'il faudrait que je passe la première semaine de mon stage à consacrer à la découverte de mon sujet, de la plateforme, mais aussi à l'installation de la plateforme. On remarque qu'il y a une absence de deux semaines dans le planning, cela est dû à l'absence de mon maître de stage durant cette période. Il nous a donc proposé de faire une pause à ce moment-là. Ensuite, la recherche de la syntaxe et sa définition étant une grande partie de mon sujet de stage, j'ai prévu deux semaines pour effectuer ces différentes tâches.

Il restait donc à ce moment 3 semaines pour développer le plugin ainsi que les tests, j'avais donc prévu de passer un peu plus d'une semaine au développement du plugin, comme ça, il me restait un peu plus d'une semaine pour réaliser les différents tests. Je devais aussi prévoir du temps pour la rédaction de mon rapport ainsi que pour la préparation de la soutenance.

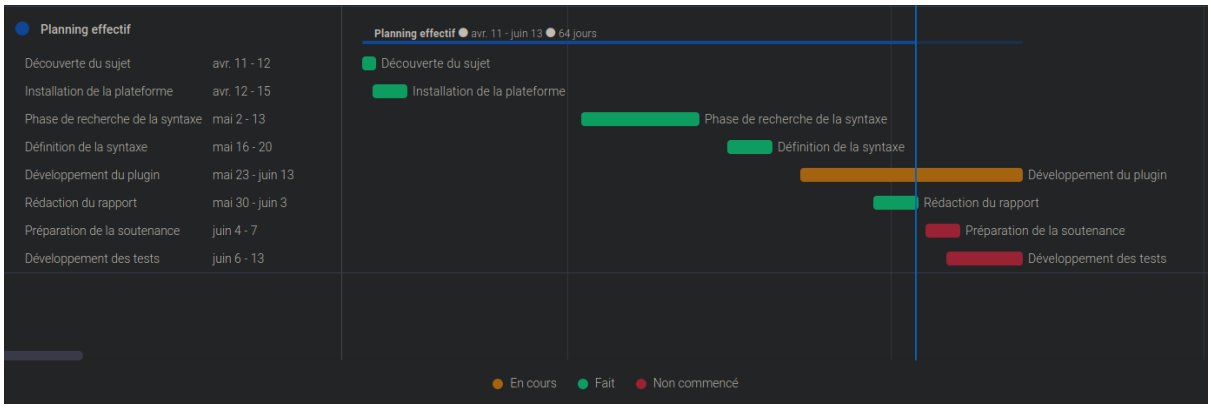


Figure 2: Planning effectif

Voici le planning effectif de comment se déroule mon stage. J'ai réussi à suivre le planning que je m'étais fixé dans la majorité des étapes. Cependant, j'ai pris du retard sur le plugin et il n'est pas fini à l'heure actuelle. Cela va donc aussi retarder le développement des différents tests. Cependant, je pense tout de même être en mesure de finir le développement de mon sujet dans les temps. En effet, j'ai déjà regardé la manière dont les tests sont réalisés et je pense connaître la majorité des tests dont mon plugin aura besoin.

3 Conclusion

Mon stage s'est donc réalisé au L3i où j'ai été très bien accueilli par les membres du laboratoire, et particulièrement par l'équipe du projet GALACTIC dans laquelle je l'effectuais.

Mon sujet était le développement d'un plugin de caractéristique écrit en Python qui devait simplifier le parcours dans des structures de données comme des listes ou des dictionnaires en précisant le chemin d'accès à parcourir en chaîne de caractères. La syntaxe de cette chaîne n'étant pas définie, j'ai dû dans un premier temps effectuer des recherches sur différentes syntaxes existantes afin de m'en inspirer pour définir celle que l'on utilisera pour la plateforme.

Puis, une fois les recherches et la syntaxe terminées, j'ai commencé à travailler sur le développement du plugin de caractéristique. Je n'ai pas encore pu finir son développement pour ce rapport, car j'ai eu des problèmes concernant le fonctionnement des méthodes à utiliser et la façon dont le code est construit. Il faut donc que je finisse de développer le plugin ainsi que ses tests pour réaliser à bien le sujet qui m'a été confié par mon maître de stage.

Mon plugin doit permettre aux utilisateurs de la plateforme de pouvoir accéder à des données stockées dans des structures de données (qui peuvent être imbriquées) de façon simple, sans avoir à parcourir les structures une à une.

Pour conclure, je souhaite tout d'abord remercier encore une fois le laboratoire pour son accueil et sa bienveillance. J'ai grandement apprécié ce stage, qui m'a permis d'en apprendre plus sur les projets de recherches en laboratoire, mais aussi sur la façon dont sont développées des applications. J'ai aussi pu améliorer mes connaissances et compétences en langage Python, et notamment sur la façon dont fonctionne les expressions régulières et les différentes structures de données.

Je souhaite faire un master en développement logiciel afin de travailler dans ce domaine après mes études, et ce stage m'aura permis d'acquérir de l'expérience, car nous n'avons pas réellement eu l'occasion de le faire pendant nos cours, tout en me motivant davantage dans ce choix d'orientation.

4 Sources - Annexe

4.1 Installation:

Guides d'installation du projet GALACTIC:

- https://galactic.univ-lr.fr/docs/main/latest/install/index.html
- https://galactic.univ-lr.fr/guides/guide-installation/Galactic-Installation-Guide-latest.pdf

4.2 Sources pour les recherches

Pour les sélecteurs CSS, je me suis renseigné sur le site développeur de Mozilla:

- https://developer.mozilla.org/fr/docs/Web/CSS/CSS_Selectors

Pour la syntaxe des chemins Xpath, je me suis principalement documenté sur le site développeur de Mozilla, ainsi que sur la documentation officielle de Xpath sur le site du W3C:

- https://developer.mozilla.org/fr/docs/Web/XPath
- https://www.w3.org/TR/2017/REC-xpath-31-20170321/
- https://www.ionos.fr/digitalguide/sites-internet/developpement-web/tutoriel-xpath/
- https://docs.microsoft.com/en-us/previous-versions/dotnet/netframework-4.0/ms256471(v=vs.100)

4.3 Développement du plugin

Liens vers la documentation Python pour les méthodes d'initialisation et d'appel:

- https://docs.python.org/3/reference/datamodel.html#object.__call__
- https://docs.python.org/3/reference/datamodel.html#object.__init__

Lien vers la documentation du module **re** :

- https://docs.python.org/fr/3/library/re.html#module-re