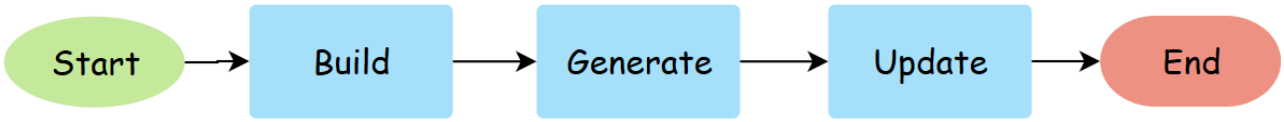
Rapport Projet Individuel

Configuration de workflows de Machine Learning



Responsable PI : Mireille Blay-Fornarino

Étudiant : Nicolas Lacroix

IUT Nice Côte d'Azur

Session 2019-2020

Remerciements

En premier lieu, je souhaite remercier Madame Mireille Blay-Fornarino qui a proposé et encadré ce projet passionnant. Je la remercie d'avoir partagée sa connaissance sur l'aspect relationnel avec le client ainsi que sur l'analyse de ses besoins. Je la remercie également pour l'opportunité qu'elle m'a offerte en prolongeant ce projet individuel en stage d'un mois rémunéré.

Je tiens également à remercier Monsieur Yassine El-Amraoui pour le temps qu'il nous a accordé pour exprimer ses besoins de client et pour ses retours sur le produit. Je souhaite également remercier les dirigeants de la société EZAKO Monsieur Bora Kizil et Monsieur Julien Muller pour leur confiance en me prenant en alternance pendant 3 ans dans le cadre de mes études d'ingénierie en alternance à Polytech Nice Sophia.

Je souhaite remercier toutes les personnes impliquées dans ce projet que ce soit pour son organisation ou pour son suivi ayant ainsi contribué à sa réussite et plus particulièrement monsieur Erold Acundeger chargé des stages au département informatique de l'IUT de Nice.

Enfin, je remercie toutes les personnes, professeurs et personnels administratifs, qui, durant ces deux années d'études, ont contribué à ma réussite par leur engagement et leurs efforts.

Résumé

Dans le cadre de mes études en IUT Informatique à Nice et pour valider mon obtention du DUT Informatique, j'ai réalisé ce projet individuel en remplacement du stage de fin d'année rendu impossible du fait de la pandémie actuelle. Organisé par Madame Blay-Fornarino, il a pour but de valider mes connaissances acquises au long de ces 2 années d'études.

Ce projet intitulé « Configuration de workflows de Machine Learning » traite de la génération de workflows de Machine Learning à partir de données d'un problème de Machine Learning. Ma mission est de poser les bases pour faire le lien entre une configuration et la génération de Workflows. Plus précisément, mon travail consiste à enregistrer les workflows réalisés par les Data Scientists, en extraire les différentes connaissances pour faire le lien avec les éléments de configuration et poser les bases pour générer les workflows en fonction de cette configuration. Au niveau technique, mon travail a consisté à lire et écrire des fichiers XML correspondant à des diagrammes BPMN et des FeatureModels, construire un micro-langage permettant l'association entre les éléments de configuration et les éléments d'un Workflow donné et générer en conséquence un Workflow que pourra utiliser un Data Scientist pour répondre à ses besoins.

La difficulté de ce projet est donc de donner les outils à l'expert pour construire cette base de connaissances sans accroître sa charge de travail, vérifier la cohérence des informations acquises au fil du temps tout en lui permettant de l'utiliser pour analyser de nouveaux problèmes.

Au terme de ce projet, un Data Scientist utilisant cette solution informatique peut en fonction des données d'un problème, configurer cette dernière et générer un workflow adapté. L'utilisation de ce logiciel est donc un réel gain de temps pour le client qui peut se focaliser sur l'application concrète des tâches constituante de ce workflow.

Abstract

As part of my studies at the IUT of Nice and in order to validate my IT graduation, I worked on an individual project as a replacement for my initial internship cancelled because of the current pandemic. Organized by Ms Blay-Fornarino, it aims at my knowledge validation gained throughout these 2 years of studies.

Entitled « Machine Learning workflows configuration », this project concerns the Machine Learning workflows generation on the basis of a Machine Learning problem's data. My assignment consists of laying the foundations for linking a configuration with a workflows generation. More precisely, it consists of saving Data Scientists' workflows, extracting their knowledge to link with the configuration elements and lay the foundations for the workflows generation based on the configuration. Technically, I worked on BPMN Diagram and FeatureModel XML files, elaborated a micro language that enables the configuration-workflow association and as a result, on a workflow generation process that a DataScientist will be able to use to respond to his needs.

The difficulty is that this project must give the expert some tools to build a knowledge tree without increasing its workload, verify its content consistency gained over time while allowing him to analyze new problems.

At the end of this project, a Data Scientist using this software will be able to configure it and to generate an appropriate workflow considering a problem data. Hence, the use of this software is a real timesaver for the client who can focus on the workflow's tasks concrete application.

Table des matières

Introduction

La construction de workflows de Machine Learning peut être d'une grande complexité. Au quotidien, les Data Scientists travaillent sur des problèmes liés à des jeux de données et à leur analyse. La construction de workflows représente une part importante dans leur travail. Véritable plan d'action, il décrit l'ensemble des tâches à exécuter pour s'assurer du bon déroulement de l'analyse. Cependant, ce dernier peut être constitué de plusieurs dizaines de tâches rendant cette conception complexe. En effet, pour chaque tâche, il peut y avoir plus de 100 algorithmes possibles et ce nombre évolue chaque jour. De surcroît, les tâches sont liées entre elles mais ces liaisons restent encore vagues. C'est ce qui explique la présence de travaux sur le Meta Learning qui ont pour objectif de décider automatiquement des tâches à accomplir en fonction d'un problème donné. Cette complexité se traduit donc par un accroissement du temps de réalisation et par conséquent par une diminution de l'efficacité de leur travail.

C'est dans ce contexte que s'inscrit mon projet individuel. Le client, Monsieur Yassine El-Amraoui, est un Data Scientist employé par la société EZAKO. La société EZAKO basée à Sophia Antipolis est une startup spécialisée dans la détection d'anomalies dans des jeux de données de séries temporelles. Elle accompagne le client dans la préparation de sa thèse dans le cadre d'une Convention Industrielle de Formation par la Recherche (CIFRE) effectuée avec le laboratoire I3S. L'objectif de son travail est de capitaliser sur la connaissance des Data Scientists acquise lors de l'analyse des données et la construction de workflows.

Ce projet exploratoire consiste à poser les bases d'une preuve de concepts et de lever les points difficiles conceptuellement. Entre recherche et application industrielle, ce projet individuel sera poursuivi par un mois de stage rémunéré afin d'y développer les dernières fonctionnalités.

Ce rapport présente dans un premier temps l'analyse et la conception menées sur ce projet afin de répondre au mieux aux besoins du client et d'assurer sa capacité d'évolution. Dans un second temps, la phase de développement est décrite et enfin les résultats sont exposés.

I. Analyse et Conception

1) Contexte

Figure 1 - Logo société EZAKO

La société EZAKO est une startup spécialisée dans la détection d'anomalies dans des séries temporelles. Plusieurs Data Scientists comme Monsieur El Amraoui travaillent à l'application des bons algorithmes afin de détecter au mieux ces anomalies. Cependant, cette sélection prend du temps et ils n'ont aucun outil pour automatiser celle-ci. C'est dans ce contexte que le client prépare sa thèse en collaboration avec Madame Blay et que mon projet individuel a lieu.

2) Les besoins du client

Le client et son entreprise souhaitent donc un outil qui serait une base pour son travail et qui limiterait le supplément d'effort. Cet outil doit être :

- Capable de construire une base de connaissance viable à partir de workflows préexistants en les enregistrant et si possible en y ajoutant les contraintes ayant conduit au choix de ces workflows.
- Capable d'évoluer avec l'ajout ou le retrait de workflows afin de garder une base de connaissance à jour.
- Capable d'ajouter des informations supplémentaires à la base de connaissance comme des informations sur les jeux de données utilisés, les références ayant élaboré ou contribué à l'élaboration de ces workflows et d'autres contraintes définies durant le développement.
- Facile à prendre en main et efficace afin qu'il ne soit pas un frein à son travail. Il s'agit surtout d'un outil d'automatisation de tâches qui lui permettra de se concentrer sur les relations entre les jeux de données et les algorithmes à employer plutôt que sur la construction de la base de connaissance.

L'une des caractéristiques de ses besoins est qu'ils sont évolutifs. En effet, ils s'inscrivent dans une démarche de recherche et ses besoins peuvent varier selon l'avancement de son travail. Il est donc primordial d'avoir une approche souple et d'être en constante relation avec le client afin de se tenir au courant de ses besoins.

3) Conception

a) Approche itérative et incrémentale

Ce projet adopte une approche itérative et incrémentale (voir Figure 2) afin de répondre au mieux aux besoins évolutifs du client. Cette approche provient des méthodes AGILE et a pour premier objectif d'impliquer au maximum le client dans l'élaboration de la solution informatique. Il suit ainsi le développement et est au courant de l'avancement de son produit. Tout d'abord, à chaque début d'itération, le client exprime ses besoins qui sont ensuite transcrit en langage technique par l'équipe de développement (phase de spécification). Il prend ensuite part à la validation du produit en le testant et en faisant un retour sur son expérience d'utilisation. L'équipe de développement peut ainsi déceler certains points à améliorer pour la prochaine itération.

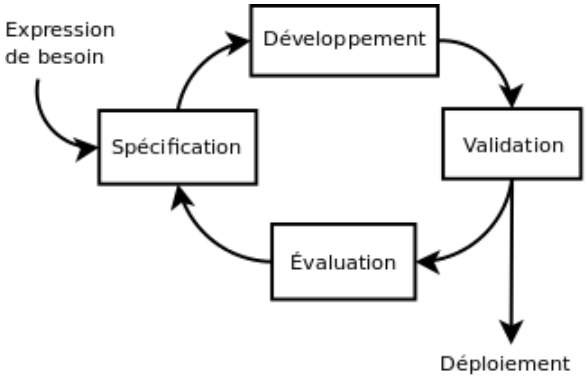


Figure 2 - Cycle approche itérative et incrémentale

Le deuxième objectif de cette approche est de rester flexible et de réagir rapidement aux variations des besoins du client. En effet, cette approche permet de se tenir régulièrement au courant des besoins du client et de ses attentes. Chaque itération apporte un nouvel « incrément » fonctionnel répondant à un besoin du client mais de précédents incréments peuvent-être supprimés s'ils sont devenus obsolètes.

Cette approche fut privilégiée car ce projet étant exploratoire, une grande variation des besoins de l'utilisateur était attendue. Cette variation se répercuta également sur la conception globale du projet qui par conséquent n'est pas fixe. L'objectif étant de rendre chaque incrément indépendant afin qu'il puisse être supprimé ou modifié sans impacter les autres fonctionnalités.

b) Indépendance vis-à-vis des représentations

La représentation BPMN est l'une des représentations les plus utilisées à ce jour pour la modélisation de processus d'affaires et la réalisation de workflows. Cela explique le choix de cette représentation pour la réalisation de ce projet. Cependant elle n'est pas la seule représentation possible. En effet, d'autres Data Scientists utilisent les diagrammes d'activité UML ou bien encore les diagrammes de flux de données et ces diagrammes ne sont pas forcément compatibles entre eux.

L'un des enjeux induit par ces différentes représentations est l'ouverture de l'application à tout type de représentation. Ainsi, si le logiciel a la capaciter de s'adapter à l'utilisateur, alors ce dernier n'aura pas à changer sa manière de travailler pour pouvoir l'utiliser. L'approche de conception utilisée a donc pour objectif de répondre à cet enjeu majeur. Elle comprend l'utilisation d'énumérations recensant leurs caractéristiques. Celles-ci sont utilisées pour la localisation et la récupération des informations et sont donc essentielles au bon fonctionnement du logiciel. C'est d'ailleurs sur ces points que les différentes représentations diffèrent et la spécification de ces caractéristiques est ainsi un moyen d'adaptation.

A terme, cette approche permettra d'ajouter un système de plug-in permettant de faire évoluer la compatibilité du logiciel sans avoir à modifier son code source. En effet, chaque plug-in contiendra les informations spécifiques d'une norme de représentation qui seront ainsi ajoutées.

Il est cependant à noter que la représentation de la base de connaissance en FeatureModel sur laquelle est basé le logiciel est très spécifique et le changement de cette norme de modélisation ne sera pas aussi facile que pour les workflows.

4) Structuration et normes

L'utilisation de fichiers XML pour la sauvegarde des données a induit la définition de normes afin d'avoir une structure précise et de pouvoir guider l'utilisateur dans son utilisation du logiciel. Au-delà des normes BPMN et FeatureModel, trois différents types de normes sont à différencier.

a) Normes structurelles

Ces normes s'appliquent sur la structure du XML et les éléments qui le composent. Le logiciel lors de la création et de l'analyse va respecter ces normes pour définir l'information qu'il est utile d'ajouter à la base de connaissance. Ces normes sont les suivantes :

- **Annotation globale** : cette annotation contient les informations générales sur le workflow et plus précisément son nom, la référence au meta-workflow s'il s'agit d'une instance, le nom du jeu de données impliqué ainsi que le nom de la référence (auteur ou article). Ces spécifications apportent des informations qualitatives à la base de connaissance. Cependant, elles ne sont pas nécessaires et en cas d'absence, le logiciel passera cette étape.
- **Contraintes et commentaires** : les contraintes doivent se trouver dans des commentaires afin d'être analysées. Cette norme permet également de garder un workflow propre et bien structuré.
- **Tâches** : les balises de tâche doivent être de type « Task » ou « User Task », dans le cas contraire, elles ne sont pas analysées.

b) Normes syntaxiques

Les normes syntaxiques permettent de localiser les informations contenues dans du texte. Elles servent notamment à différencier les simples commentaires des informations importantes et sont complémentaires des normes structurelles. En effet, l'analyse syntaxique intervient après l'analyse structurelle. Ces normes sont ci-dessous :

- **Informations générales** : Ces informations présentes dans la balise d'annotation globale se situent entre doubles accolades *{{information}}*.
- **Contraintes** : Les contraintes sont encadrées entre doubles crochets *[[contrainte]]*. Il est à noter que plusieurs contraintes peuvent être présentes dans un seul commentaire permettant ainsi de limiter le nombre de commentaires et de simplifier le workflow ainsi que sa clarté.
- **Tâches imbriquées** : Il est possible de représenter une hiérarchie de tâches en une seule tâche de workflow. Cette hiérarchie sera ensuite représentée au sein du FeatureModel. Cette fonctionnalité permet de construire une base de connaissance de manière plus efficace et concise. La syntaxe à respecter est la suivante : *Mère#Enfant#Sous-Enfant* … Chaque niveau de hiérarchie est donc séparé par un dièse *#*. À terme, il sera possible de créer plusieurs enfants par parent de cette manière.
- **Tâches génériques** : Il est recommandé (mais pas obligatoire) de nommer une tâche générique (meta-tâche) sous la forme : *NameTask_Step*. Le fait que le nom de cette tâche se termine par *_Step* permet à l'utilisateur de différencier rapidement les tâches génériques des tâches instanciées.

c) Normes personnalisables

Il s'agit de normes syntaxiques mais dont la syntaxe peut être personnalisée via un fichier de configuration (voir Figure 3). L'un des avantages est l'adaptation à l'utilisateur et l'objectif est à terme de transformer toutes les normes syntaxiques en normes personnalisables. L'absence de fichier de configuration n'est cependant pas bloquante et conduit à l'application d'une norme « par défaut » implémentée dans le logiciel. Pour l'instant, la syntaxe des contraintes est entièrement personnalisable. La syntaxe par défaut (extrait du fichier de configuration) est la suivante :

```
# this is the configuration file for the ml2wf app

# constraints syntax (name : arity : symbol)
# ------- NOTE: only change the last column -------

before : 2 : >>
after : 2 : <<

imp : 2 : =>
equ : 2 : <=>

conj : 2 : &
disj : 2 : |

not : 1 : !

# others
# not implemented yet
```

Figure 3 - Extrait du fichier de configuration

II. Développement

1) Environnement de travail

Ce projet est réalisé en Java sous l'IDE Eclipse. Le langage Java a été préféré pour sa portabilité et ses librairies de traitement de fichiers XML performantes. Il est hébergé sur le service web d'hébergement et de gestion de développement GitHub. Le projet ayant été réalisé à distance, la communication entre Madame Blay, le client et moi a été rendu possible grâce à Slack et Zoom pour les visioconférences. La qualité du code a été assurée tout au long du projet avec l'usage de l'outil SonarLint. La visualisation des workflows utilisant la norme BPMN et des (Extended)FeatureModels a été permise respectivement grâce à l'extension BPMN2 Modeler et à l'extension FeatureIDE.

2) Gestion du projet

a) Gestion des tâches

Les tâches à effectuer dont l'ajout de nouvelles fonctionnalités et la correction de bugs ont été définies en issues et rassemblées dans un « project board » GitHub (voir Figure 4).

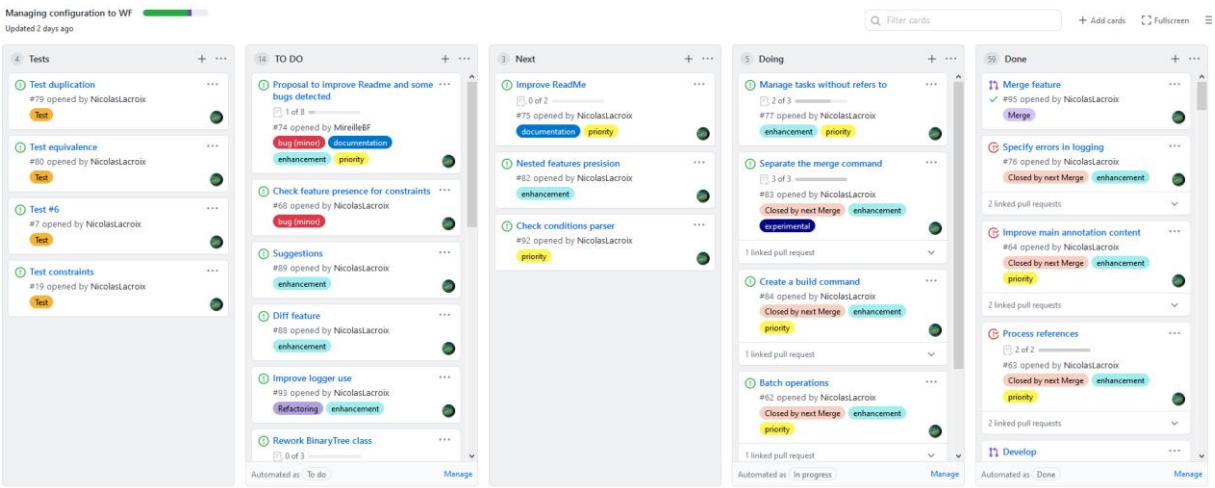


Figure 4 - Project Board

Ce dernier a facilité l'organisation du travail et a aidé à avoir un point de vu global du projet. Chaque issue est rédigée en anglais et catégorisée afin de qualifier les tâches du projet.

b) Branchage

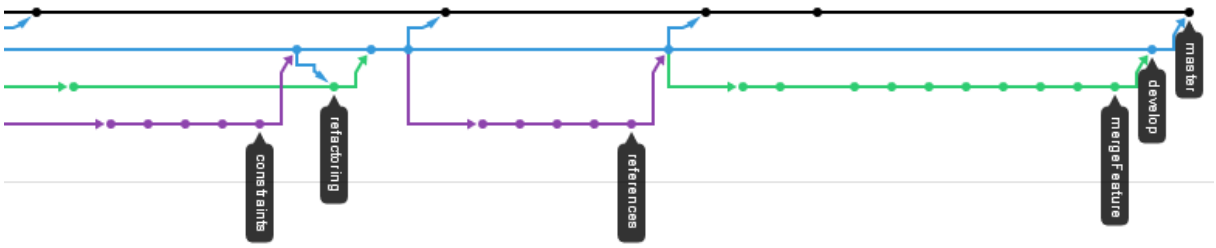


Figure 5 - Branches

L'utilisation de la fonctionnalité de branchage de GIT permit au projet de rester viable et structuré. En parallèle de la branche **master** s'ajoute une seconde branche dev qui sert de branche d'intégration pour les fonctionnalités. Autour de cette branche s'articulent les branches de fonctionnalités. Chaque fonctionnalité est développée sur une branche spécifique afin de pouvoir rendre les fonctionnalités indépendantes entre elles. La refactorisation du code étant un aspect important dans ce projet, une branche lui est donc dédiée (refactoring).

c) Cycle de développement

Le développement de la solution informatique a été organisée sous forme de cycle (voir Figure 6). Ce cycle prend place dans le contexte d'approche itérative et incrémentale. Dans un premier temps, la phase d'analyse des besoins du client est menée. Elle permet d'identifier ses besoins, ses priorités et de planifier le développement des fonctionnalités futures. Ces nouvelles fonctionnalités sont donc développées dans un second temps puis testées. Une fois testées, ces fonctionnalités sont publiées et la phase de refactorisation commence. Cette dernière permet de garder un code clair et logique avant de recommencer ce cycle de développement.

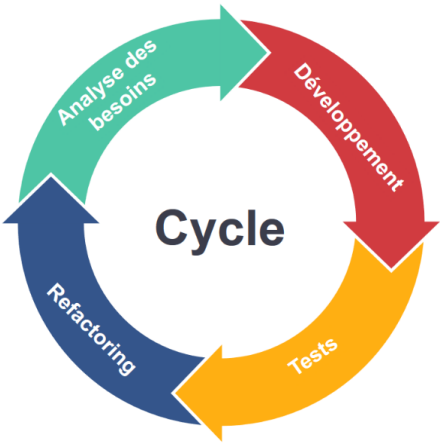


Figure 6 - Cycle de développement

L'une des spécificités de ce cycle est qu'il est nécessaire dans le cadre de ce projet exploratoire. Les besoins du client étant en constante évolution, il fallait adopter un cycle de développement permettant de réactualiser les besoins du client (en début de cycle) et d'ajouter les nouvelles fonctionnalités régulièrement tout en refactorisant le code pour le garder clair (en fin de cycle).

3) Principales difficultés et solutions apportées

a) Analyse des contraintes et conversion

Le FeatureModel sans contraintes n'a qu'un intérêt limité. En effet, c'est grâce à ces dernières que la validité et la consistance de la base de connaissances est assurée. L'un des objectifs de ce logiciel est de permettre au Data Scientist de saisir rapidement des contraintes sans avoir à interagir directement avec le FeatureModel.

Pour cela, la solution apportée analyse les commentaires laissés par l'utilisateur dans les workflows. Ces derniers servent à la description des tâches mais également à la définition de contraintes.

L'une des difficultés lors de l'analyse est d'identifier et d'isoler les contraintes. Pour cela, des **expressions régulières** ont été utilisées ce qui a permis de cibler les contraintes de manière performante.

La seconde difficulté consistait à convertir ces contraintes au format textuel vers un format XML compréhensible par le FeatureModel. Cette conversion se fait en plusieurs étapes.

- Tout d'abord, les éléments logiques sont séparés des parenthèses et les opérateurs sont regroupés avec leurs opérandes directs à l'aide d'un **parseur en tokens** (classe StringTokenizer en Java). L'objectif est de créer des groupes d'éléments logiques liés qui seront traités ensemble par la suite.
- Ensuite, les éléments logiques constitutifs de la contrainte sont triés selon leur profondeur d'imbrication dans les parenthèses, cela permet de définir un ordre de traitement (les plus imbriqués d'abord)
- Puis, la contrainte textuelle est convertie en arbre logique où chaque nœud correspond à un opérateur et chaque feuille correspond à un opérande.
- Enfin, cet arbre logique est converti en sous-arbre DOM qui sera ajouté au document principal. Cette conversion se fait par récursivité en utilisant l'algorithme suivant que j'ai développé :

```
01. function generateNode(tree: BinaryTree, base: Node)
02.     rootValue <- tree.root
03.     if rootValue == null then
04.         return
05.     end
06.     base <- base.appendChild(createNode(rootValue))
07.     foreach child in (tree.leftChild, tree.rightChild) do
08.         if child != null then
09.             generateNode(child, base)
10.         end
11.     end
12. end
```

Figure 7 - algorithme de conversion en DOM (pseudo code)

b) XML et JAVA

Ce projet s'oriente principalement sur l'analyse et la modification de fichiers XML en Java. En utilisant les normes BPMN et ExtendedFeatureModel, les workflows présents sous la forme de fichiers XML sont analysés ou modifiés en fonction des besoins de l'utilisateur. Divers enjeux sont induits par cette analyse.

Le premier est la localisation et la récupération des informations. Pour cela, l'API utilisée est **JAXP**. Cette dernière permet de convertir le fichier XML en un document modifiable, d'analyser les balises ainsi que leur contenu et de sauvegarder le résultat.

Il existe deux types de parseurs :

- **Type SAX** (Simple API for XML) : traitement séquentiel des données basé sur les évènements
- **Type DOM** (Document Object Model) : traitement libre du XML sous forme d'un document DOM

J'ai fait le choix d'utiliser le parseur de type DOM car ce dernier permet de se déplacer librement dans l'arbre DOM tout en autorisant la modification de sa structure ce qui est nécessaire pour l'insertion de tâches et la récupération d'informations rapide. Ainsi, le fichier XML analysé se présente sous la forme de DOM (voir Figure 8).

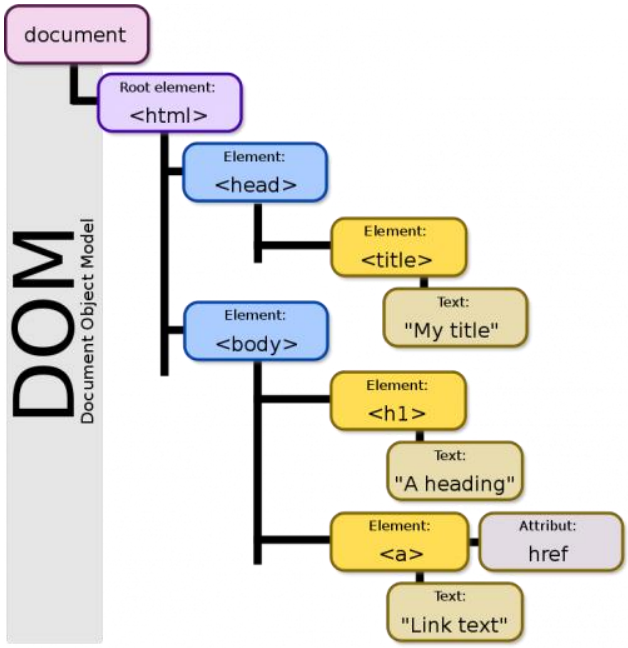


Figure 8 – Exemple de structure DOM

Cette interface de programmation normalisée consiste à traiter chaque balise en nœud ayant un parent s'il n'est pas la racine du document et un ou des enfants s'il n'est pas une feuille. Chaque nœud possède des attributs pouvant être

son nom, son identifiant… C'est grâce à ces attributs et à l'accès aux enfants que la récupération des informations est effectuée.

Le second enjeu est un enjeu de performances. En effet, le parcours de cet arbre pouvant être très grand peut prendre beaucoup de temps si cet aspect n'est pas pris en compte. Ainsi, plusieurs mesures ont été prises pour obtenir un logiciel performant. Tout d'abord, le FeatureModel n'est sauvegardé qu'une fois toutes les modifications effectuées. L'absence de sauvegarde intermédiaire représente un gain de temps précieux notamment pour les traitements par lots. Durant le traitement, la structure du FeatureModel est gardée en mémoire permettant ainsi d'accéder à son contenu sans avoir à l'ouvrir pour chaque importation. Une dernière fonctionnalité améliorant les performances est en cours de développement. Dès qu'un DOM sera parcouru, on gardera en mémoire les éléments qui seront recherchés ultérieurement. Cette méthode permettra donc de ne parcourir le DOM qu'une seule fois et de pouvoir accéder aux éléments plus rapidement.

c) Garder le code simple, clair et logique

Le développement dans le cadre d'un projet exploratoire est soumis à de nombreux changements au fur et à mesure des découvertes. L'un des dangers est le fait de perdre sa simplicité et sa logique. En effet, ces variations de choix se répercutent sur le code qui est en constante évolution. Certains éléments deviennent obsolètes tandis que d'autres voient leur utilisation étendue.

Un second frein au développement rencontré est le manque de clarté. En effet, l'accès aux informations des balises requiert pour l'API utilisée d'appeler plusieurs méthodes consécutivement. À cela s'ajoute la longueur de nom de certaines de ces méthodes (ex. getElementsByTagName(tagName)). La factorisation étant impossible au niveau des API utilisées, un gros effort de nommage et de conception a été réalisé dans le code développé lors de l'utilisation de celles-ci afin de le garder viable.

4) Autres fonctionnalités

a) Traitements par lots

Afin de faciliter et d'augmenter la productivité du logiciel, les fichiers peuvent-être traités par lots. Concrètement, au lieu de traiter un fichier par appel du logiciel, l'utilisateur peut passer en paramètre un répertoire et ce dernier sera analysé, chaque fichier correspondant à un workflow sera ensuite traité et la sauvegarde ne sera effectuée qu'à la fin du traitement du répertoire. Cette sauvegarde ne sera donc effectuée qu'une seule fois ce qui limite le nombre d'écritures dans le fichier procurant ainsi des performances accrues.

b) Personnalisation

Ce logiciel se base sur un fichier de configuration. Ce dernier contient les paramètres personnalisables de ce dernier. L'objectif est de s'adapter à l'utilisateur afin qu'il n'ait pas à s'adapter au logiciel. Cela passe par la personnalisation de la syntaxe des contraintes et à terme par la spécification de la norme employée. En cas d'absence de fichier de configuration, une configuration interne sera appliquée par défaut. L'objectif étant de rendre l'utilisation possible même en absence de ce fichier. Cela permet à l'application de rester portable.

5) Tests

La réalisation de tests est primordiale pour s'assurer que l'outil est valide et que la base de connaissance ne soit pas biaisée.

Figure 9 - Logo JUnit

a) Tests unitaires

La validité du code est assurée dans un premier temps par des tests unitaires via l'api JUnit dans sa 5ème version. Ces tests vérifient d'abord que les meta-workflows sont bien instanciés en vérifiant le contenu du workflow instance et en le comparant avec le contenu du meta-workflow associé. Ensuite, ils s'assurent que l'importation de l'instance soit bien effectuée en vérifiant le contenu du FeatureModel après importation.

b) Test d'intégration

Aux tests unitaires s'ajoutent des tests d'intégration continue sur GitHub. Ces tests s'inscrivant dans une démarche Agile permettent de garder une version fonctionnelle sur la branche principale afin qu'elle puisse être testée par le client quand il le souhaite. Cela permet au client de se tenir informé des dernières nouveautés sans s'exposer à des bugs non corrigés.

6) Documentation

Le code est entièrement documenté et représentant près de 52% du code, il est compatible avec l'outil javadoc. De plus, les algorithmes complexes ou importants sont écrits en pseudo code. L'objectif est de faciliter la réutilisation du logiciel. Etant un projet open source, sa maintenabilité ainsi que sa clarté sont deux aspects essentiels.

III. Résultats

1) Les commandes

L'utilisation se fait en ligne de commandes. Le logiciel se présente sous la forme d'un fichier ml2wf.jar pouvant être exécuté avec la commande : **`java -jar ml2wf.jar`**.

Le *synopsis général* est le suivant : **`java -jar ml2wf.jar [commande] [arguments]`**

À noter que toutes les commandes ont un argument de verbosité attendant un entier (4 par défaut) dont la table de correspondance est la suivante :

Tableau 1 - Niveaux de verbosité

0	OFF
1	FATAL
3	WARN
4	INFO
5	DEBUG
6	TRACE
7	ALL

a) Build

Synopsis : **`ml2wf build -f FeatureModel -m meta_directory -i instance_directory [-v]`**

La commande « Build » permet de créer une base de connaissance (FeatureModel) à partir d'un ensemble de Meta-Workflows et de Workflows instanciés. Elle est utile en début de projet si vous souhaitez construire votre base de connaissance à partir de workflows préexistants ou si vous voulez la reconstruire pour revenir à une version précédente.

b) Generate

Synopsis : **`ml2wf generate -i meta_workflow -o output_directory [-v]`**

La commande « Generate » permet d'instancier un Meta-Workflow précisé par **`-i`** (**`--input`**) pour que l'utilisateur puisse le personnaliser. Cette commande précède généralement les autres commandes puisqu'elle permet la création d'un nouveau workflow qui sera à terme ajouté à la base de connaissance.

c) Merge

Synopsis : **`ml2wf merge [--meta|--instance] [-fbv] -i workflow -o FeatureModel`**

La commande « Merge » permet d'ajouter un workflow (**`-i`**) dans le FeatureModel (**`-o`**).

L'utilisateur peut spécifier s'il souhaite ajouter ce workflow en tant que Meta-Workflow (**`--meta`**) ou Workflow Instance (**`--instance`**).

Il peut également spécifier s'il souhaite ajouter la relation entre le Meta-Workflow et son instance via l'argument **`-f`** (**`--full`**).

Enfin, il peut décider d'effectuer une sauvegarde de la base de connaissance avant toute modification par l'intermédiaire de la commande **`-b`** (**`--backup`**)

d) Save

Synopsis : **`ml2wf save [-bv] -i meta_workflow instance_workflow -o FeatureModel`**

La commande « Save » est l'équivalent de l'association des commandes **`merge --meta`** et **`merge --instance`** avec l'argument **`-f`**.

En effet, elle permet d'insérer dans une base de connaissance un Meta-Workflow avec son instance et les relations entre ces deux workflows.

2) Visualisation du résultat

Afin de visualiser le résultat de l'appel de ces commandes, il suffit d'ouvrir le FeatureModel dans un environnement supportant cette norme. Le résultat des tests ayant été visualisé avec FeatureIDE (voir Figure 10), il est recommandé d'utiliser cet outil afin de limiter les risques d'incompatibilité.

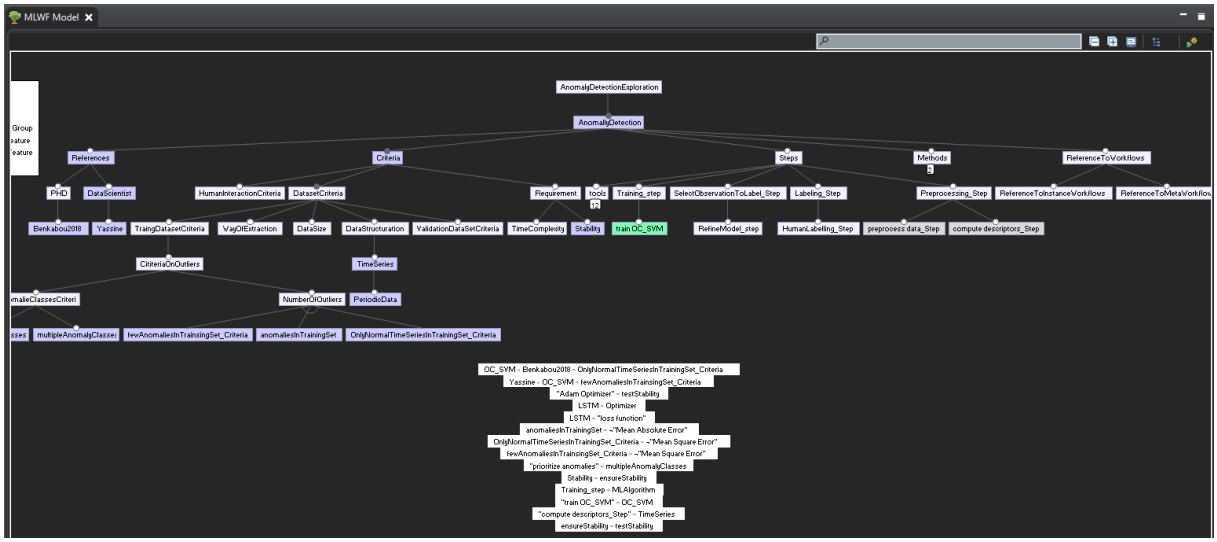


Figure 10 - Visualisation d'un FeatureModel avec FeatureIDE

À cela s'ajoute un système de logs. Ces logs configurables avec l'option de verbosité (-v) informent l'utilisateur de l'avancement du traitement et des potentielles erreurs. Ainsi, l'utilisateur peut facilement localiser et corriger les erreurs.

Conclusion

Pour conclure, ce projet pose les bases d'une preuve de concept sur la création d'une base de connaissance pouvant être utilisée dans le cadre de la création de workflows de Machine Learning et de l'application de Meta Learning. Comme l'illustre l'exemple en annexe, cette base de connaissance fournit à l'utilisateur la capacité de centraliser ses connaissances sur la création de workflows et de vérifier leur consistance. Cette consistance est une garantie pour l'expert de la validité de ses workflows et de sa connaissance.

Pour parvenir à un tel résultat, il a fallu fournir un travail important sur les points difficiles tels que la conversion de contraintes vers un format XML et identifier les conflits pouvant survenir lors de cette construction et de sa mise à jour. Cette identification des conflits permet à Monsieur El Amraoui de débuter son travail avec une vision globale du problème.

D'un point de vue personnel, il fut très enrichissant notamment grâce à son approche exploratoire. En effet, ce mélange des mondes du génie logiciel et de la recherche m'a apporté un regard nouveau sur le monde professionnel et mes méthodes de développement.

À cela s'ajoute l'approche avec le client. En effet, étant la première fois que je travaille pour un client avec des besoins concrets, j'ai pu prendre conscience de l'importance d'être à son écoute. Bien identifier ses besoins est primordial si l'on souhaite que le logiciel remplisse ses besoins et que ce travaille ne soit pas vain. De plus, j'y ai vu les difficultés pouvant apparaître lors de la phase d'expression des besoins du client. En effet, le client n'a pas toujours une idée précise de ses besoins et le développeur y joue alors un rôle d'accompagnement afin de le guider dans l'expression de ses besoins.

Ce projet concrétisant ces 2 années d'études au sein de l'IUT Nice Côte d'Azur m'a permis de mettre en pratique mes connaissances en Java, en conception et production. Il m'a appris à adopter une approche plus globale du problème, à identifier ses limites et à anticiper les besoins du client.

Enfin, ce projet m'a apporté bien plus car c'est grâce à celui-ci que je poursuivrai mes études dans le cursus d'alternance de Polytech Nice Sophia avec la société EZAKO. Il m'a véritablement fait entrer dans le monde professionnel et ses enjeux.

Glossaire

Bibliographie

Dépôt GitHub
https://github.com/MireilleBF/FromMLWFConfigurationToBPMN

https://github.com/MireilleBF/FromMLWFConfigurationToBPMN/blob/master/README.md

Workflow
https://www.iterop.com/quest-ce-quun-workflow/

https://towardsdatascience.com/workflow-of-a-machine-learning-project-ec1dba419b94

Norme BPMN
http://www.bpmn.org/

https://www.bpms.info/bpmn-la-norme-du-bpm/

FeatureModel
https://en.wikipedia.org/wiki/Feature_model

https://www.methodsandtools.com/archive/archive.php?id=49p2

https://featureide.github.io/

https://github.com/FeatureIDE/FeatureIDE/wiki

Détection d'anomalies
https://dataanalyticspost.com/Lexique/detection-danomalie/

https://fr.wikipedia.org/wiki/D%C3%A9tection_d%27anomalies

Approche itérative et incrémentale (AGILE)
https://www.les-traducteurs-agiles.org/2018/07/11/iteratif-et-incremental.html

https://fr.wikipedia.org/wiki/M%C3%A9thode_agile

https://www.geek-directeur-technique.com/2009/02/06/le-cycle-iteratif

EZAKO
https://ezako.com/fr/

Table des illustrations

Annexes

Exemple d'utilisation

Voici un exemple de séquence d'utilisation permettant de mieux comprendre l'utilisation et l'objectif de ce logiciel.

Etape 0 – Etat initial

Un Data Scientist possède un ensemble de workflows (1 générique et 2 « instanciés ») :

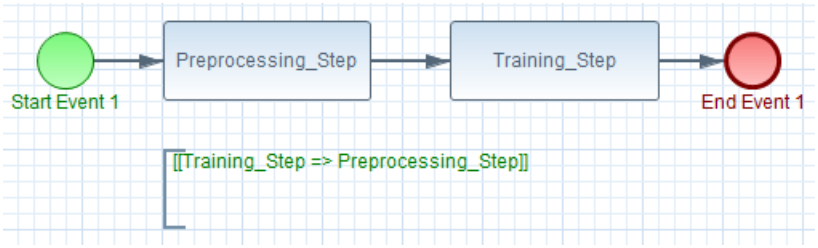


Figure 11 - Workflow générique

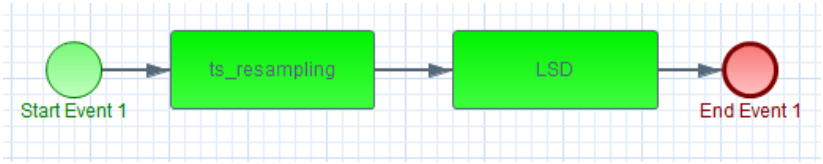


Figure 12 - Premier workflow instancié

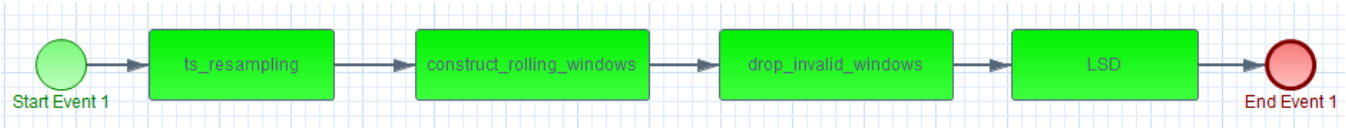


Figure 13 - Second workflow instancié

et le FeatureModel basique.

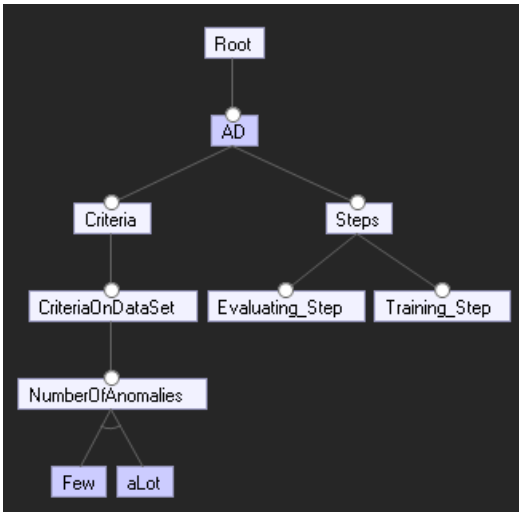


Figure 14 - FeatureModel basique

Etape 1 – Création de la base de connaissance

Il souhaite créer sa base de connaissance à partir de ces éléments.

Commande : **Build**

Résultat :

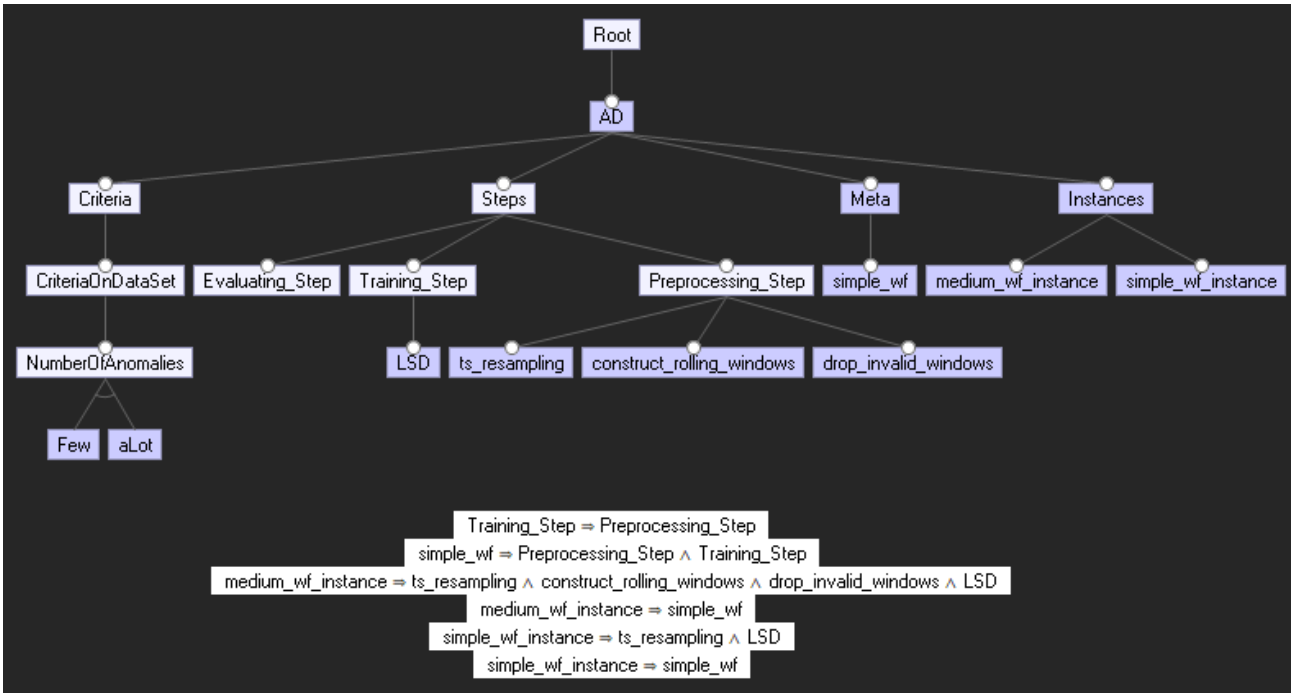


Figure 15 - FeatureModel complété

Etape 2 – Création d'une nouvelle instance de workflow

Le Data Scientist souhaite créer une nouvelle instance de l'un de ses workflows génériques.

Commande : **Generate**

Résultat :

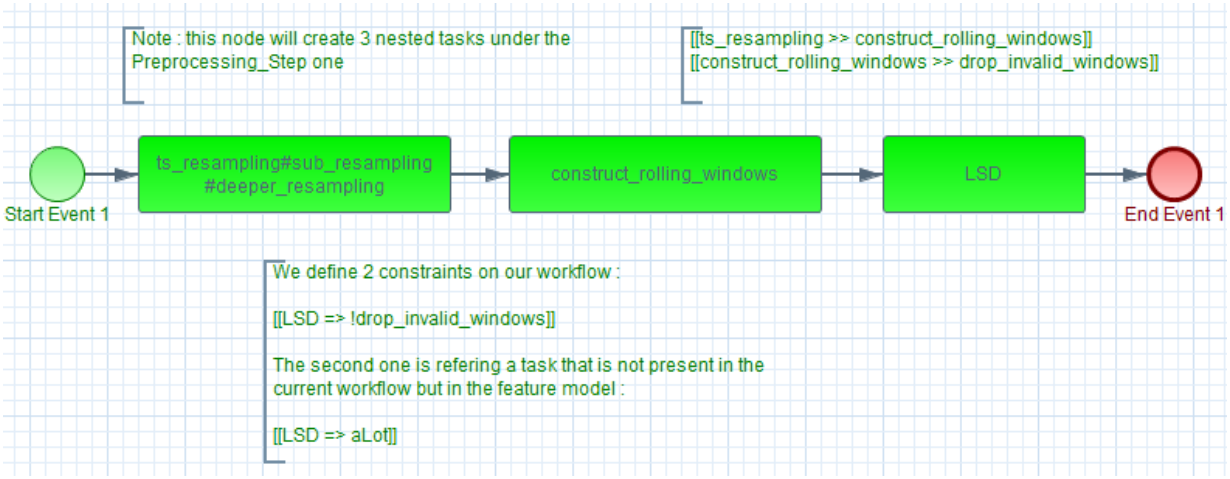


Figure 16 - Workflow instancié avec Generate

Etape 3 – Mise à jour de la base de connaissance

Il met à jour sa base de connaissance pour s'assurer se sa consistance.

Commande : **Merge**

Résultat :

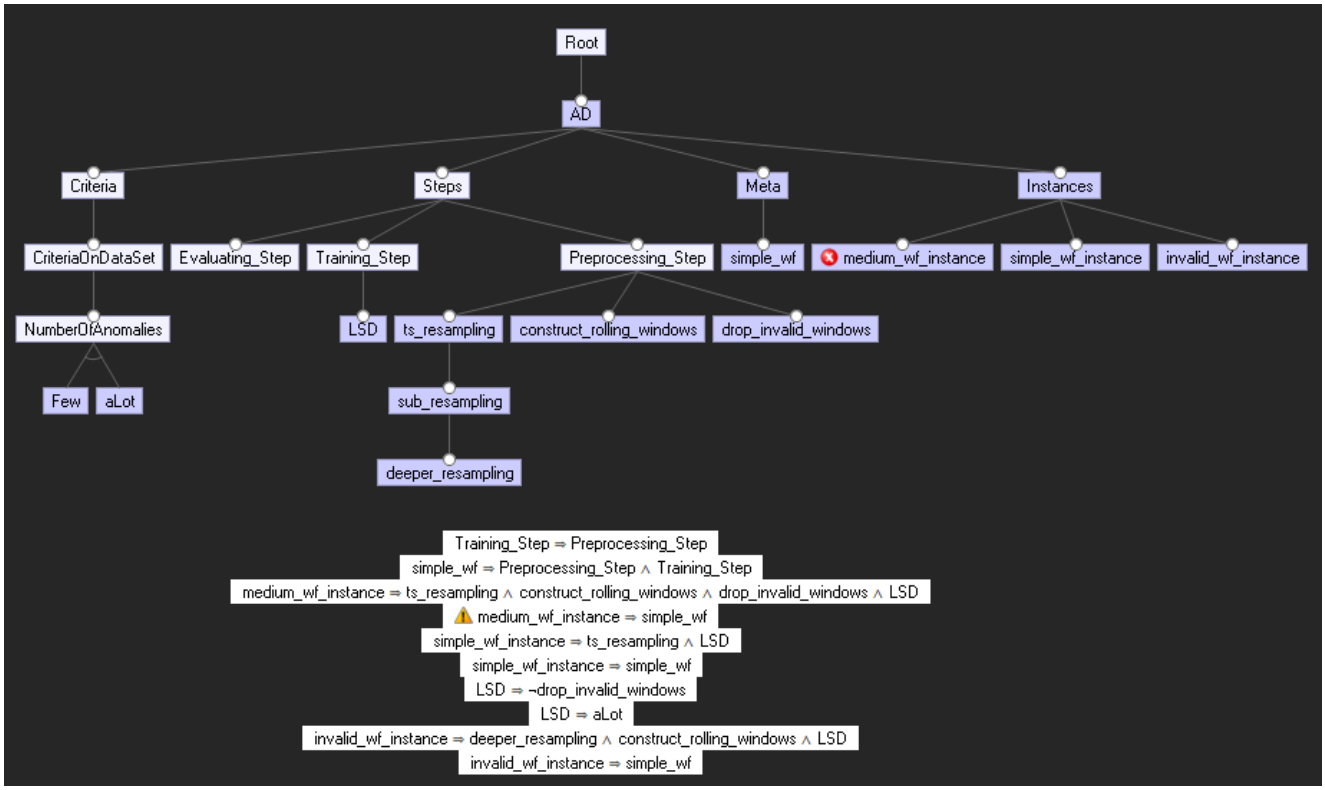


Figure 17 - FeatureModel complété

Etape 4 – Reconstruction de la base de connaissance

Il se rend compte que sa base de connaissance n'est plus consistante :

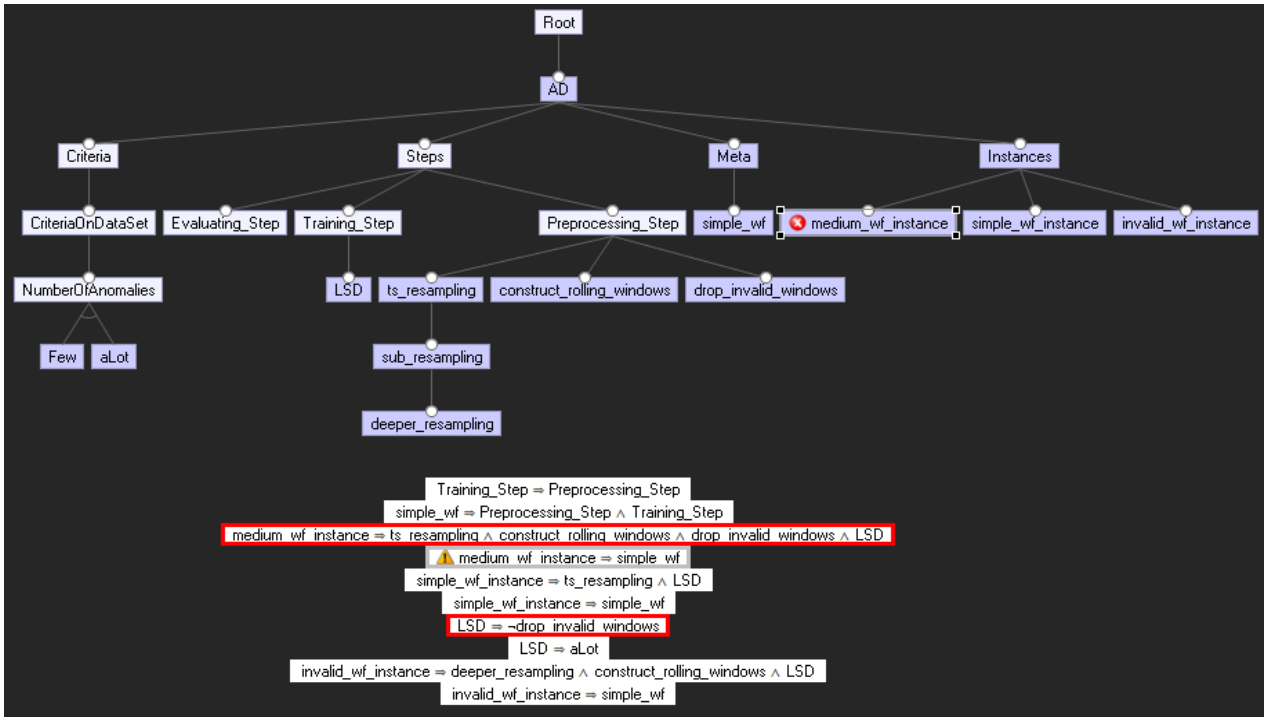


Figure 18 - FeatureModel invalide/inconsistant

et souhaite revenir à la version précédente :

Commande : **Build**

Résultat :

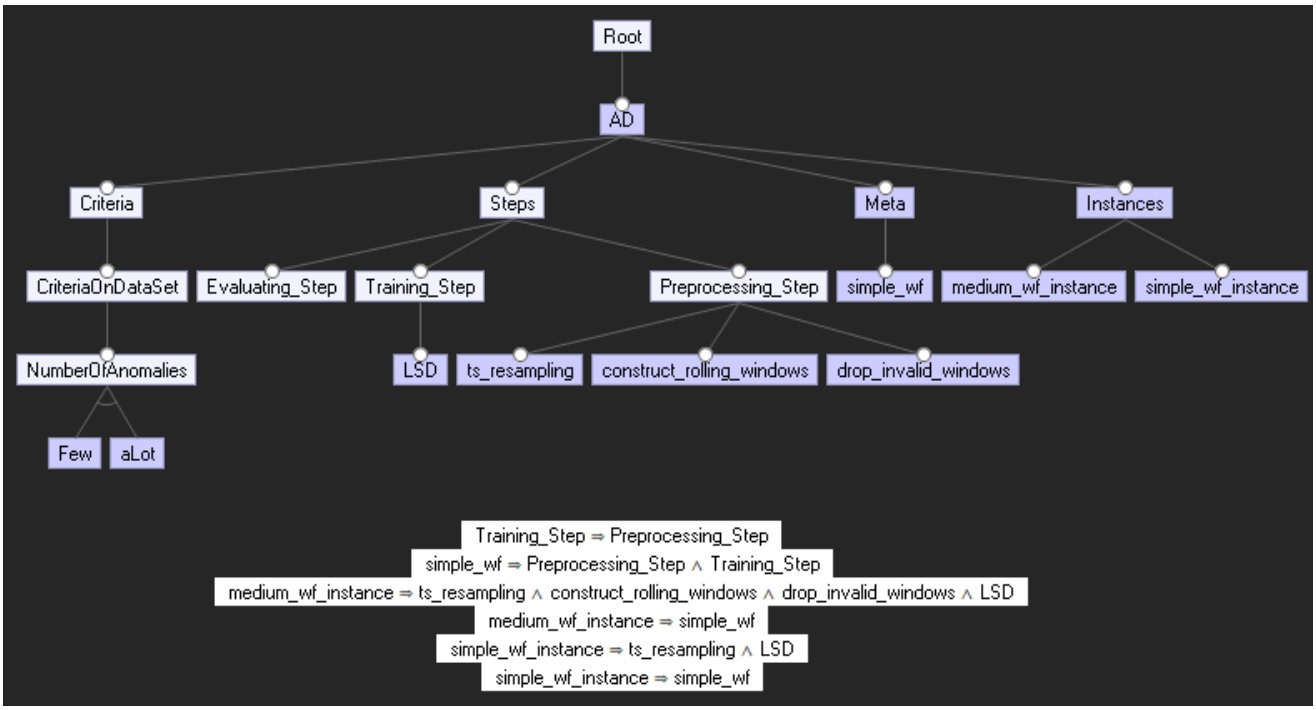


Figure 19 - FeatureModel consistant