**Université de Strasbourg**
IUT Robert Schuman

Maître de stage : Nicolas Lachiche
Tuteur universitaire : Marie-Paule Muller

# Cahier des charges

Jonathan HAEHNEL

Strasbourg, le 6 mai 2011

# Table des matières

# Introduction

### Sujet

Le stage consiste à réaliser une application Java permettant automatiser le test de plusieurs algorithmes de fouille de données. En effet, on va essayer d'étudier leur comportement respectif en faisant varier un paramètre initial. Cette application sera ensuite utilisée pour effectuer de nombreux tests toujours durant le stage. Effectuer des tests à la main est faisable, mais, ce processus est plutôt long et laborieux, il est donc important d'automatiser cette tache.

### Demandeur

M. Lachiche est le responsable du projet, mais également le maître d'ouvrage.

### Contexte

Le projet s'insère dans le contexte de la fouille de données (recherche d'un modèle à partir d'un échantillon de données). Il faut également constater que le projet est riche en ressources existantes. En effet, on doit utiliser quatre logiciels (Weka explorer, Proper, DataGenerator) et trois différents algorithmes (Cardinalisation, RELAGGS, Discrétisation).

### Limite du sujet

Les algorithmes utilisés durant le stage sont déjà implementés dans certains logiciels, il n'est pas demandé de les réécrire, ni de les corriger. Il n'est pas nécessaire de gérer des bases de données de profondeur supérieure à 1/footnoteImpliquant plus d'une association un-à-plusieurs.

### Type de sujet

Ce projet demande une grosse analyse de l'existant avant de se lancer dans le développement. En effet, notre application devra utiliser des fonctionnalités implémentées dans quatre logiciels différentsexistants déjà. Il faudra également de la rigueur pour réaliser une interface graphique simple et surtout ergonomique.

# Chapitre 1

# Etude de l'existant

## 1.1 DataGenerator

Le **DataGenerator** est un petit logiciel codé en Java avec une interface graphique permettant de générer aléatoirement des données dans une base de données en définisant les valeurs maximales des champs des différentes tables.
*Par exemple : génerer 100 instances d'îlots qui ont au maximum 10 instances de bâtiments chacun avec une aire d'un bâtiment variant de 0 à 100.*

Après avoir généré les données, le logiciel intègre une seconde fonctionnalité : **l'étiquetage des données**. Il permet de donner une valeur (par défaut, vrai ou faux) à une ligne de la table, si celle-ci respecte une condition de filtrage.
*Par exemple : pour la table îlot, la colonne «classe» est vraie, s'il y a au moins un bâtiment ayant une aire supérieure à 50, sinon c'est faux.*

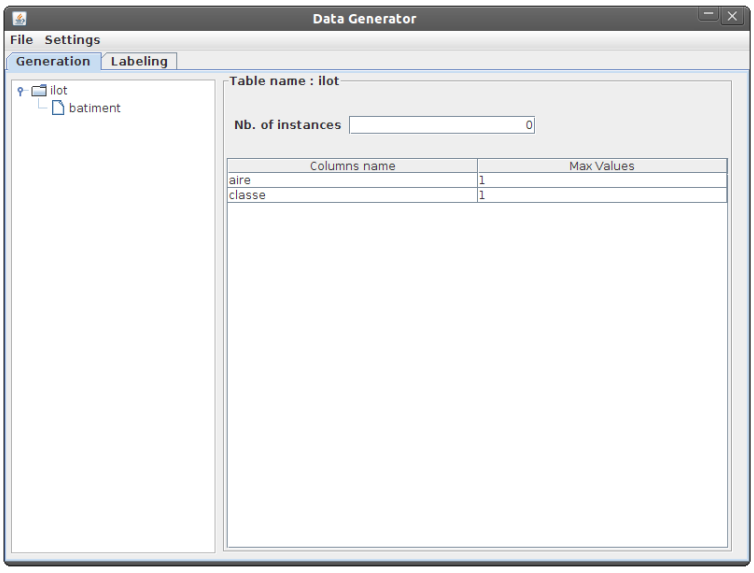



FIGURE 1.1 – Page d'accueil du DataGenerator

**Possibilités d'intégration :** Ce logiciel n'est pas exécutable de l'extérieur (en ligne de commande). Il compte 5 packages de classes dont un package moteur [1]. Ce package de petite taille sera directement intégré dans notre application.

1. Package avec les classes permettant de réaliser les fonctionnalités principales et algorithmiques de l'application

## 1.2 Proper

Proper est un vaste logiciel codé en Java. Je n'ai utilisé que les deux fonctionnalités suivantes :

1. Le **Builder** qui permet de choisir et de configurer les différents algorithmes, à ce stade le logiciel génère un fichier XML avec toutes les informations et paramètres choisis.
2. Le **Runner** permet à partir du fichier XML généré à l'étape précedente, d'exécuter les algorithmes choisis dans le Builder. Les tables résultats s'ajoutent au serveur et un fichier de type «ARFF» est généré par le système.

**Possibilités d'intégration :** Le comportement du Builder est plutôt facile à reproduire, en effet, notre logiciel pourra générer facilement le même fichier XML que le Builder. Par contre le Runner est indispensable, il faudra donc réussir à l'appeler depuis notre programme. Le logiciel «Proper» compte 31 packages de classes, de ce fait, il ne sera pas intégré directement intégré dans notre application.

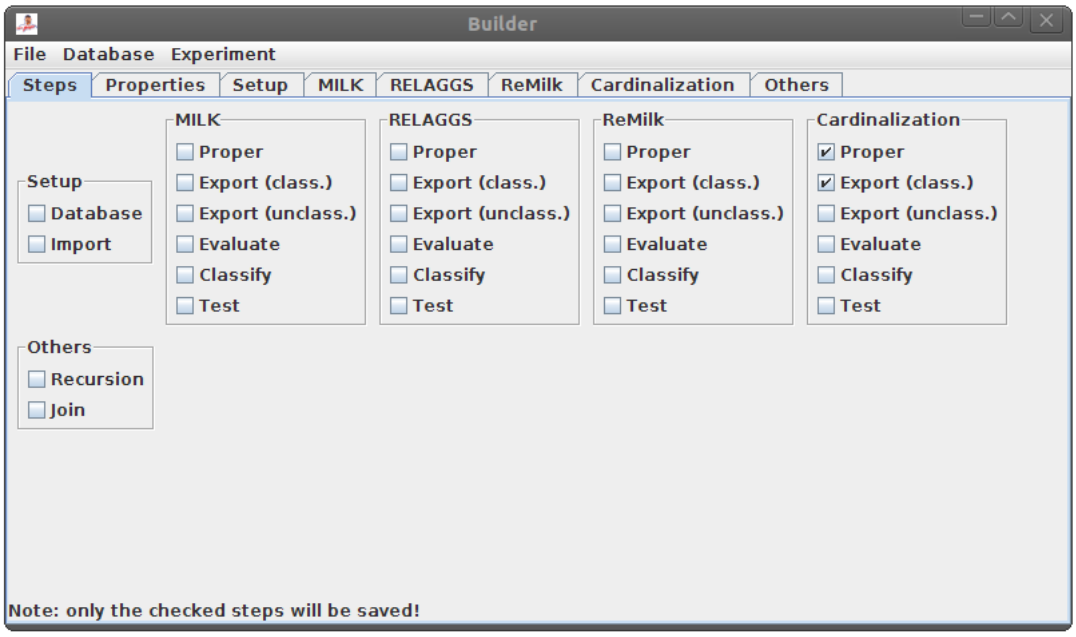



FIGURE 1.2 – Page d'accueil du logiciel Builder

## 1.3 Weka Explorer

Ce logiciel codé en Java permet d'interpréter les résultats des différents algorithmes. En effet, il prend en entrée un fichier ARFF et permet de configurer et d'exécuter l'un des classeurs WEKA sur l'ensemble de données actuel. On peut choisir d'effectuer une validation croisée ou de tester sur un ensemble de données distinct.
Le résultat de la validation est un pourcentage de réussite.

**Possibilités d'intégration :** Weka est normalement bien concu et il est possible de l'appeler directement en ligne de commande avec le classeur de son choix (J48 ou d'autres).

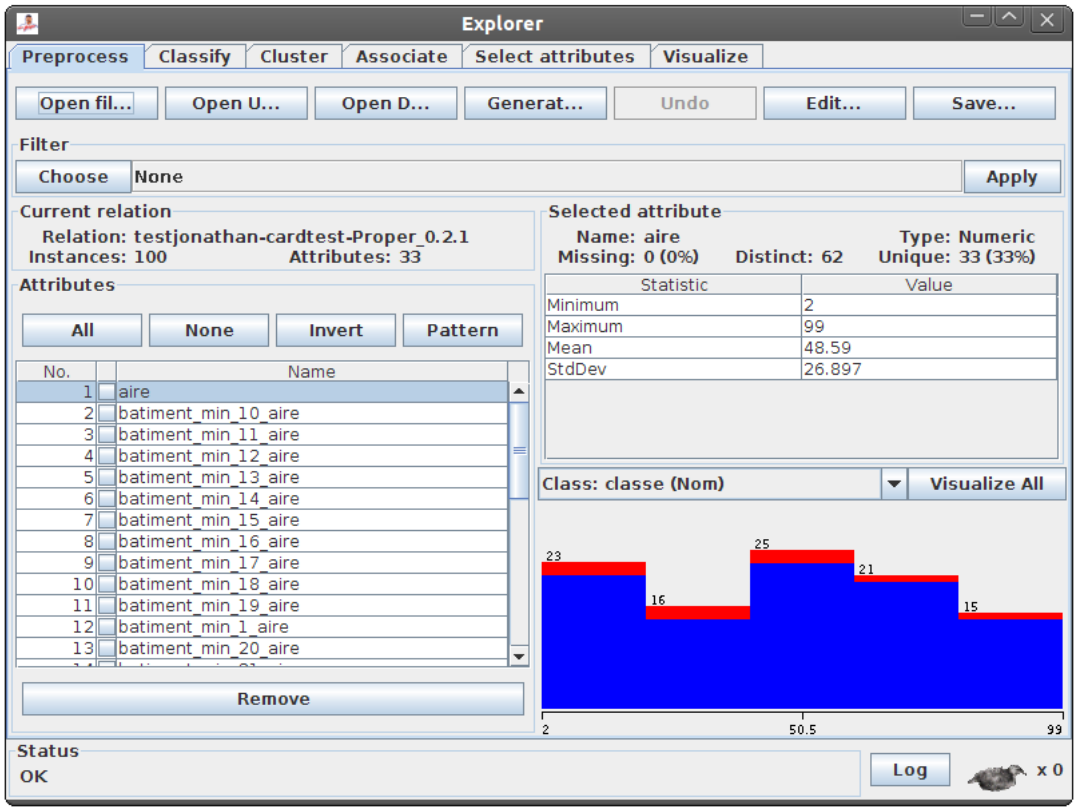



FIGURE 1.3 – Aperçu de l'explorer WEKA

# Chapitre 2

# Etude des fonctionnalités

Voici la liste des principales fonctionnalités de notre application. Celles-ci ne sont pas définitives, des modifications pourront être apportées pendant le développement.
Chaque fonctionnalité est classée par ordre d'importance :

**P1** Fonctionnalité primaire

**P2** Fonctionnalité secondaire

**P3** Fonctionnalité de confort

## 2.1 Fonctionnalités de base

| **Liste des fonctionnalités :** | | |
|---|---|---|
| Num. | Description | Type |
| 1 | Choisir un algorithme à tester :<br>*Test de l'algorigthme de Cardinalisation*<br>*Test de l'algorithme "RELAGGS"*<br>*Test de la discrétisation des données avec "RELAGGS"*<br>*Test de l'algorithme de Cardinalisation avec des quantiles (%)* | **P1** |
| 2 | Choisir le paramètre à faire varier durant le test (avec un pas, des valeurs minimale et maximale) | **P1** |
| 3 | Nettoyer récursivement la base de données de travail | **P1** |
| 4 | Générer aléatoirement et labeliser les données | **P1** |
| 5 | Générer automatiquement un fichier XML de configuration de l'algorithme à tester | **P1** |
| 6 | Exécuter ce fichier XML dans le Runner de Proper de façon automatique | **P1** |
| 7 | Ouvrir le fichier ARFF dans WEKA explorer et lire le résultat | **P1** |
| 8 | Enregistrer ce résultat dans un fichier CSV [1] | **P1** |
| 9 | Laisser la possibilité d'enregistrer également le temps d'exécution dans ce fichier CSV | **P2** |

1. Comma-separated values (CSV) est un format informatique ouvert représentant des données tabulaires sous forme de « valeurs séparées par des virgules ».

## 2.2 Fonctionnalités liées à l'interface

| **Liste des fonctionnalités :** | | |
|---|---|---|
| Num. | Description | Type |
| 1 | Module de connexion à une base de travail | **P1** |
| 2 | Module de choix de l'algorithme et des paramètres principaux du test | **P1** |
| 3 | Paramétrage du fichier XML | **P1** |
| 4 | Paramétrage des processus de génération des données et d'étiquetage des données. | **P1** |
| 5 | Paramétrage du classeur WEKA à utiliser durant le test | **P2** |
| 6 | Interface de visualisation des évenements/résultats durant le test avec une barre de progression | **P1** |
| 7 | Possibilité de stopper un test en pleine exécution | **P3** |
| 8 | Possibilité de lancer plusieurs tests en même temps (multi-threading) | **P3** |
| 9 | Ouverture du fichier CSV et visualisation de la courbe (JGraph API) | **P2** |

## 2.3 Fonctionnalités du support

| **Liste des fonctionnalités :** | | |
|---|---|---|
| Num. | Description | Type |
| 1 | Code commenté et génération de la JavaDoc | **P1** |
| 2 | Manuel du développeur pour la suite | **P1** |
| 3 | Petit manuel en francais et/ou en anglais d'aide à l'utilisation du programme | **P2** |

# Chapitre 3

# Contraintes et planification

## 3.1 Contraintes générales

### 3.1.1 Contraintes de temps

Le logiciel devra respecter un certain calendrier, dont les principales échéances sont :
- 18 avril 2011 : Lancement de la phase 2 (développement de datatester)
- 29 avril 2011 : Rendu final du cahier des charges
- 10 mai 2011 : Rendu du prototype de l'application
- 20 mai 2011 : Livraison de l'application et démarrage de la phase 3 (tests)
- 13 juin 2011 : Rendu du rapport de stage
- 25 juin 2011 : Soutenance de stage

### 3.1.2 Contraintes techniques

- Réalisation du logiciel en Java
- Interface graphique simple mais ergonomique (Swing/AWT)
- L'application utilise beaucoup d'outils existants déja, l'application sera externe à Proper.
- L'application devra gérer plusieurs types de classeurs Weka et déterminer (si possible) les meilleurs paramètres à utiliser.

### 3.1.3 Contraintes de portabilité

Étant développé en Java, notre logiciel sera portable sur la majorité des systèmes d'exploitation ayant une machine virtuelle Java active. Notre logiciel devra néanmoins être compatible avec toute base de données MySQL et PostgreSQL.

### 3.1.4 Environnement de développement

- Type de base de données à utiliser : MySQL et PostgreSQL
- Langages utilisés : Java
- Librairie de gestion des bases de données : JDBC
- Librairie Graphique utilisée : Swing , AWT, JGraph
- Logiciel de développement : Eclipse, Netbeans , Éditeur de texte, Kile
- Existant : Proper, Weka, DataGenerator
- SGBD utilisés : pgAdmin III, phpMyAdmin

## 3.2 Planning prévisionnel

| **ANALYSE DE L'EXISTANT ET PRÉ-TRAITEMENTS :** | | |
|---|---|---|
| Échéance | Description | Difficulté |
| **18/04/11** | Trouver comment lancer le Runner de Proper en ligne de commande et étudier le code du DataGenerator | Moyen |
| **20/04/11** | Afin de comprendre la structure de l'application finale, il faut commencer à réaliser un script qui permet de repondre au sujet du stage (mais, sans interface utilisateur) | Difficile |
| **28/04/11** | Comprendre et trouver le fonctionnement des classeurs WEKA (explorer) et intégrer le choix du classeur dans le script (avec paramètre variable) | Moyen |
| **28/04/11** | Première maquette graphique de l'application sous Netbeans | Facile |
| **29/04/11** | Rendu du cahier des charges de l'application | Moyen |
| **29/04/11** | Finalisation du script Java permettant l'automatisation des tests | Difficile |
| **01/05/11** | Intégration dans le script des autres algorithmes comme RELAGGS et Discretizor | Moyen |
| **DÉ VELOPPEMENT ET INTÉGRATION :** | | |
| Échéance | Description | Difficulté |
| **03/05/11** | Développement de l'interface graphique (sans événement/action) sous Eclipse | Facile |
| **05/05/11** | Intégration de l'algorithme de Cardinalisation | Moyen |
| **09/05/11** | Intégration de l'algorithme de RELAGGS | Moyen |
| **10/05/11** | Présentation du prototype | Facile |
| **13/05/11** | Intégration de l'algorithme de Discretisation | Moyen |
| **FINALISATION** | | |
| Échéance | Description | Difficulté |
| **16/05/11** | Tests et débogage | Facile |
| **17/05/11** | Commenter le code et génération de la JavaDoc | Moyen |
| **20/05/11** | Rendu du projet | Facile |
| **25/05/11** | Article d'aide à l'utilisation (ang/fr) | Moyen |

# Chapitre 4

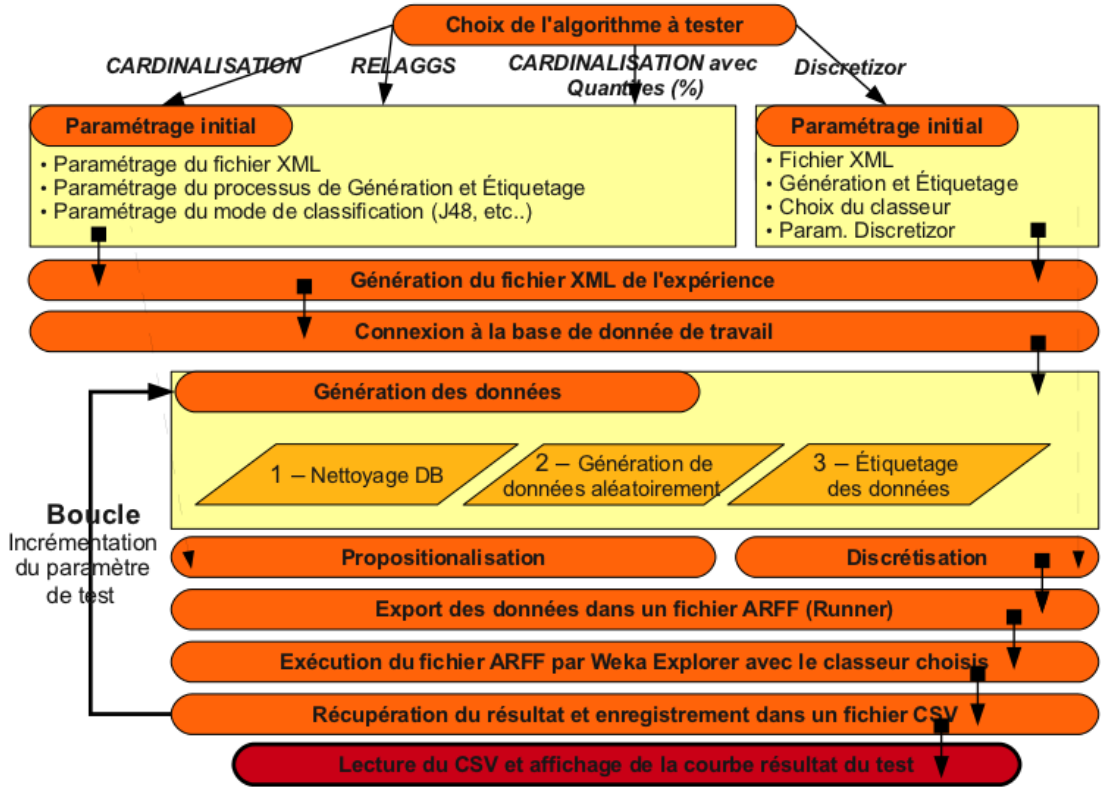
# Analyse et modélisation

## 4.1 Schématisation de l'application (processus)

On ne peut pas se lancer directement dans le développement de l'application. En effet, il faut d'abord étudier l'existant, car notre application doit utiliser simultanement trois logiciels et plusieurs algorithmes déjà existants.

Pour commencer, je dois créer un script Java permettant de réaliser les grandes actions de l'application finale automatiquement et sans interface utilisateur. Le script et ses classes peuvent donc être considérés comme le **moteur de l'application**. L'interface graphique n'est qu'un supplément que l'on viendra fixer sur ce script. Commencer par la création du script permet de ne pas partir dans toutes les directions et de bien structurer sa réflexion *(les choses de second plan comme la GUI sont écartées pour le moment)*.

*Ci-dessous, un schéma permettant d'expliquer simplement les différentes étapes de l'execution du script :*



## 4.2 Proposition de maquette

L'interface graphique aura trois grandes orientations :
- Une partie pour le paramètrage du test.
- Une partie pour executer le test et suivre son déroulement.
- Une partie pour afficher les courbes et les résultats.

Les maquettes arriveront dès que le script sera fonctionnel à 100% ! Avant, cela n'est pas la peine, car celle-ci changeront régulièrement..