

Développement d'une plate-forme d'évaluation pour le schema matching

Rapport de stage effectué au LIRMM



Réalisé par FIALIN Rémi

Encadré par BELLAHSENE Zohra et DUCHATEAU Fabien

Tuteuré par PALAYSI Jérôme

Année universitaire 2011-2012



Développement d'une plate-forme d'évaluation pour le schema matching

Rapport de stage effectué au LIRMM



Réalisé par FIALIN Rémi

Encadré par BELLAHSENE Zohra et DUCHATEAU Fabien

Tuteuré par PALAYSI Jérôme

Année universitaire 2011-2012

Remerciements

C'est avec plaisir que j'adresse mes remerciements aux personnes qui ont contribué à faire de ce stage une expérience enrichissante, et en particulier, à :

M.MOLNAR : Pour son aide dans ma démarche de stage, et sans qui je n'aurai jamais découvert le LIRMM.

M. DUCHATEAU et Mme BELLAHSENE : Pour m'avoir fait confiance dans la réalisation du projet et m'avoir accordé de leurs temps libre pour me fournir différents conseils et explications.

M. PALAYSI : Pour être venu me voir, s'être intéressé et avoir essayé de comprendre mon projet de stage.

SREY Roger : Ancien camarade de classe et de projet, qui m'a conseillé quand j'en avais besoin.

REALINI Audrey : Ancienne camarade de classe, qui m'a aidé et fourni des astuces pour arriver à certains buts.

Le LIRMM : Pour avoir pu profiter de son infrastructure et m'avoir fourni le matériel.

Merci aussi à toutes les personnes qui ont pu m'aider de près ou de loin dans ce stage, que ce soit pour mon projet ou mon contexte de travail :

LAVERNHE Laurie, SERRURIER Nicolas, TISON Aurélien, HORNBY Martine, GEORGES Stéphane, CARMONA Anthony, NOUREDDINE Yacine, CHARRON John, TAHRAT Sabiha, BOURIAL Maroua, Hinéa, Sébastien, Niihau et Valentin.

Et enfin, un remerciement spécial à ma famille, pour m'avoir soutenu durant tout ce stage et qui n'a de cesse de le faire dans tout ce que j'entreprends.

Présentation de l'entreprise

J'ai effectué mon stage au LIRMM (Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier) qui est une unité mixte de recherche, dépendant conjointement de l'Université Montpellier 2 et du Centre National de la Recherche Scientifique (CNRS). Il est situé sur le Campus Saint-Priest de l'UM2.

Le LIRMM est situé au Nord-Ouest de Montpellier, quartier « Hôpitaux Facultés », près du parc Euromédecine, dont l'adresse est :

Université Montpellier 2

LIRMM UMR 5506 - CC 477

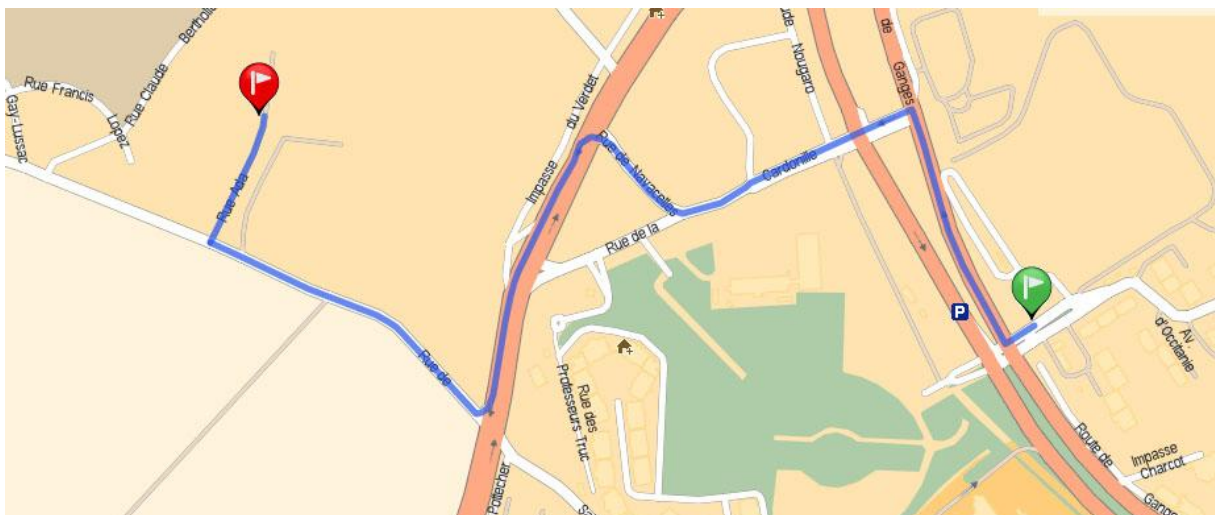
161 rue Ada

34095 Montpellier Cedex 5 France

GPS : 43° 38'.22 Nord, 3° 50'.42 Est

Point rouge : LIRMM

Point vert : IUT



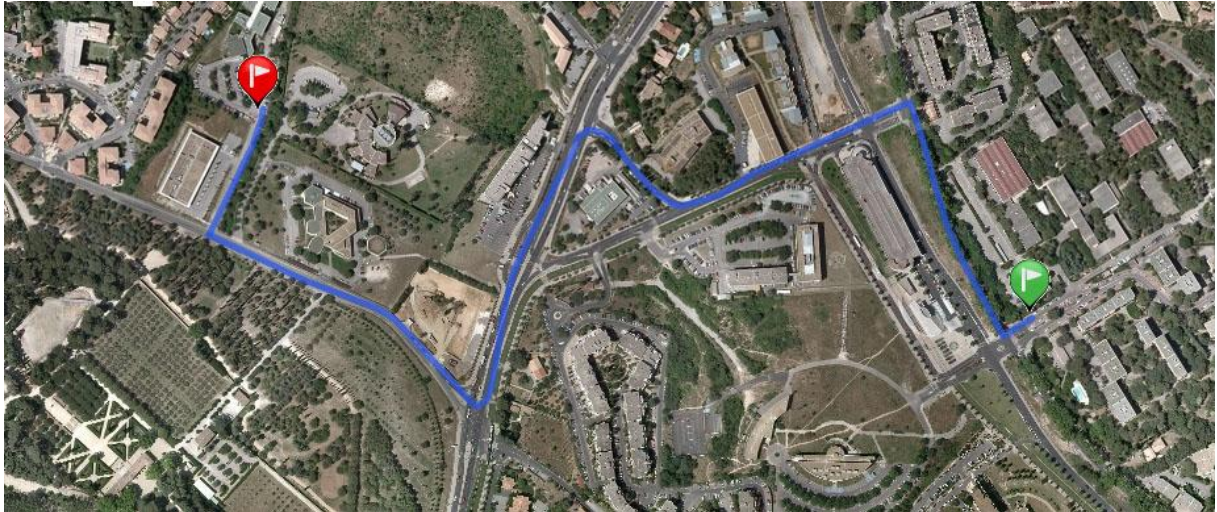


Figure 1 - Plan d'accès au LIRMM

Le personnel est composé de 405 personnes.

En octobre 2009, le LIRMM regroupait plus de 350 personnes, dont 160 enseignant-chercheurs et chercheurs, 150 doctorants et 40 personnels ingénieurs et techniciens.

Le LIRMM recrute dans toute la France mais aussi dans le monde entier (Japon, Chine, Singapour, Italie, Roumanie, Thaïlande, Brésil), que ce soit des doctorants ou des permanents.

Au sommet de la hiérarchie nous pouvons trouver :

Le Directeur :

Jean-Claude König (directeur@lirmm.fr)

Les Directeurs adjoints :

Marianne Huchard (huchard@lirmm.fr)

André Crosnier (crosnier@lirmm.fr)

Michel Renovell (renovell@lirmm.fr)

Le Responsable du département Informatique :

Christophe Paul (paul@lirmm.fr)

Le Responsable du département Robotique :

Philippe Poignet (poignet@lirmm.fr)

Le Responsable du département Microélectronique :

Patrick Girard (girard@lirmm.fr)

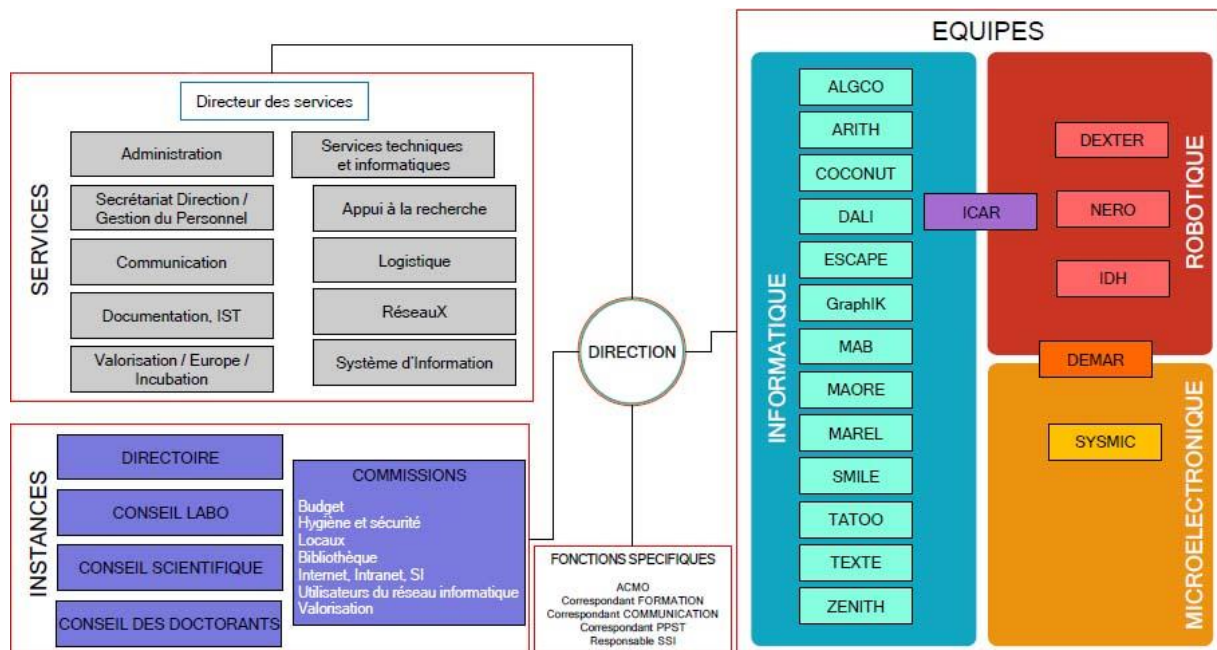


Figure 2 - Organigramme du LIRMM

Les travaux sont menés dans trois départements scientifiques de recherche (Informatique, Robotique, Microélectronique), eux-mêmes organisés en « équipes-projet », dix-neuf au total :

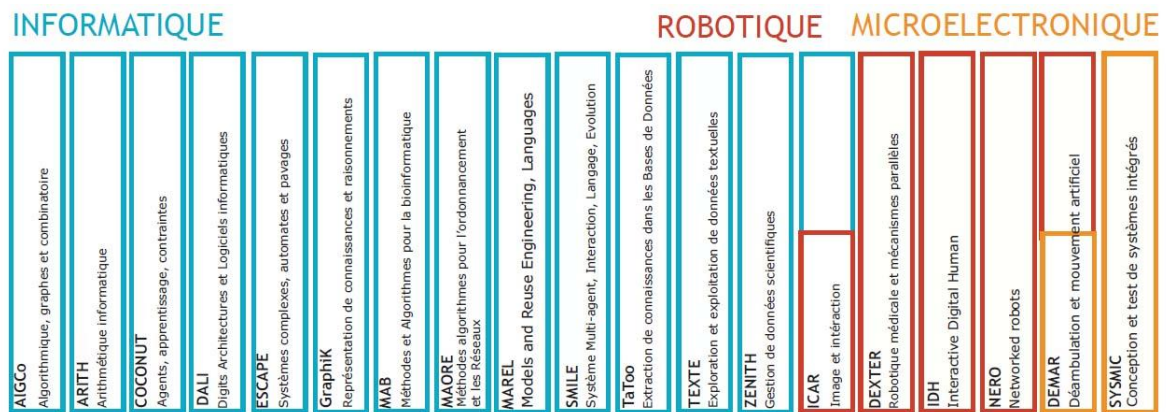


Figure 3 - Equipes-projet

Le Département Informatique du LIRMM est organisé en quatorze équipes-projets, dont l'une est commune avec le département Robotique. Il regroupe une centaine de chercheurs permanents et 78 doctorants. Les équipes du département couvrent un très large spectre de la recherche en informatique allant de l'informatique théorique aux applications et les interfaces avec de nombreuses autres disciplines.

L'animation scientifique du département s'organise autour de 4 pôles de recherche:

- Algorithmes et calculs
- Données et connaissances
- Intelligence artificielle
- Méthodes informatiques et modélisation pour le vivant et l'environnement

Le pôle Données et connaissances s'intéresse à la gestion et l'analyse de masses de données de nature complexe (textes, images, données scientifiques, etc.), à la gestion des connaissances qui peuvent être extraites de ces données, ainsi qu'aux aspects ingénierie des données et des connaissances. Il regroupe les équipes GRAPHIK, MAREL, TATOO, TEXTE, SMILE et ZENITH.

J'ai donc pu intégrer le pôle Données et connaissances du LIRMM au sein de l'équipe ZENITH en tant que développeur d'applications.

L'équipe ZENITH : C'est la gestion de données scientifiques grâce à l'intégration, l'analyse et le traitement des données scientifiques.

Sommaire

I.	Introduction-----	17
II.	Description du stage -----	18
A.	Présentation du sujet -----	18
B.	Cahier des charges -----	21
C.	Planning prévisionnel-----	22
III.	Démarche-----	23
A.	Analyse du projet-----	23
B.	UML -----	25
1.	Use case (cas d'utilisations)-----	25
2.	Diagramme de séquence – scénario (cas nominal) -----	26
3.	Diagramme de classes -----	27
C.	Technologies utilisées -----	28
D.	Planning -----	29
E.	Développement d'une plate-forme d'évaluation-----	30
1.	Présentation -----	30
2.	La classe Graph -----	30
3.	Première conclusion-----	31
F.	Couche Modèles-----	32
1.	Lectures -----	32
2.	Calculs -----	34
3.	Affichages -----	36
4.	Conclusion sur la couche modèles -----	39
G.	Interface Graphique -----	40
1.	Sélections -----	40
2.	Paramétrages -----	44
3.	Affichages -----	56
4.	Conclusion sur l'interface graphique -----	63
IV.	Résultat et discussion-----	64
A.	Logiciel obtenu -----	64
B.	Discussion -----	78
1.	Critique des erreurs/faillles -----	78
2.	Améliorations possibles -----	79

V.	Conclusion -----	81
A.	Organisation du travail -----	81
B.	Bilan du projet -----	81
VI.	Annexes -----	82
A.	Classe ParserXML -----	82
B.	Classe Filter -----	86
C.	Classe DownloadSchema -----	89
D.	Classe DownloadGraph -----	91

Table des figures

Figure 1 - Plan d'accès au LIRMM-----	6
Figure 2 - Organigramme du LIRMM-----	7
Figure 3 - Equipes-projet -----	7
Figure 4 - Exemple de deux schémas xml dit « mappé » -----	18
Figure 5 - Arborescence d'un scénario-----	19
Figure 6 - Liste des scénarios disponible en fonction des propriétés-----	19
Figure 7 - Planning prévisionnel -----	22
Figure 8 - Use case (cas d'utilisations)-----	25
Figure 9 - Diagramme de séquence -----	26
Figure 10 - Diagramme de classes -----	27
Figure 11 - Planning-----	29
Figure 12 - Les paramètres de la classe Graph -----	30
Figure 13 - Appels à de multiples contrôleurs-----	31
Figure 14 - Structure du format interne d'une correspondance dans l'application -----	32
Figure 15 - Exemple de parser simple -----	33
Figure 16 - Comparaison entre éléments de correspondances -----	34
Figure 17 - Les différentes mesures définies -----	35
Figure 18 - Construction d'un script suivant le système d'exploitation-----	36
Figure 19 - Appel vers le terminal suivant le système d'exploitation -----	36
Figure 20 – Ouverture d'une console avec un paramètre de script de console -----	37
Figure 21 - Exemple de script batch pour windows -----	37
Figure 22 - Exemple de script Gnuplot-----	38
Figure 23 - Exemple de fichier de données pouvant aller de pair avec un script Gnuplot-----	38
Figure 24 - Fonction retournant une image -----	38
Figure 25 - Ecran de démarrage de l'application -----	40
Figure 26 - Sélection d'un scénario -----	41
Figure 27 - Tri des scénarios en fonction des propriétés cochées -----	42
Figure 28 - Sélection d'une propriété et d'un scénario-----	43
Figure 29 - Vérification de la sélection d'un scénario -----	44
Figure 30 - Vérification de l'obtention d'une et une seule propriété-----	45

Figure 31 - Vérification pour garder intacte la relation entre le premier et le deuxième onglet -----	45
Figure 32 - Les différentes sections du deuxième onglet -----	46
Figure 33 - JFileChooser permettant le téléchargement des fichiers de schémas-----	47
Figure 34 - Deuxième section composée de trois outils de matching -----	48
Figure 35 - Mise en place des boutons "Plus" et "Moins" -----	48
Figure 36 - Compteur pris en charge par les boutons "Plus" et "Moins" -----	49
Figure 37 - Construction des "rangements" en fonction du compteur -----	49
Figure 38 - Fichiers "mappé" associés aux datasets -----	49
Figure 39 - Boutons "Browse" associés aux datasets d'un scénario -----	50
Figure 40 - Passage, par l'utilisateur, d'un fichier mappé (fichier contenant des correspondances) -----	51
Figure 41 - Les mesures définies de la troisième section -----	52
Figure 42 - Appels des calculs à effectuer pour chaque mesure sélectionnée -----	52
Figure 43 - Particularité de la troisième section-----	53
Figure 44 - Exemple de tableau contenant l'ordre de passage pour chaque mesure. --	53
Figure 45 - Déclaration d'un tableau d'entiers de taille 5 -----	53
Figure 46 - Décrémentation de toutes les cases d'un tableau d'entiers -----	53
Figure 47 - Disposition à la verticale par défaut des graphiques générés -----	54
Figure 48 - Passage entre verticale et horizontale de l'alignement des graphiques générés -----	54
Figure 49 - Dernier composant du deuxième onglet-----	54
Figure 50 - Composant permettant la génération de graphiques-----	55
Figure 51 - Les deux volets du troisième onglet -----	56
Figure 52 - Création d'un objet de type TextAreaWriter-----	56
Figure 53 - Classe TextAreaWriter et son héritage sur la classe Writer -----	57
Figure 54 - Classe Writer -----	58
Figure 55 - Fermeture de fichier d'un objet de type TextAreaWriter et appel de la classe TextAreaReader-----	58
Figure 56 - Classe TextAreaReader-----	59
Figure 57 – Composants de la partie droite du troisième onglet-----	60
Figure 58 - Tableaux dynamiques de JButton et tableau dynamique de String -----	61
Figure 59 - Création du « filename » et son ajout dans un tableau dynamique-----	61

Figure 60 - Ajout, dans un tableau dynamique, d'un JButton comportant l'image d'un graphique-----	61
Figure 61 - Lecture du fichier texte correspondant au graphique cliqué-----	61
Figure 62 - Libellé d'un graphe -----	62
Figure 63 - JLabel d'un graphique-----	62
Figure 64 - Les deux boutons de téléchargement d'un graphe -----	62
Figure 65 - Au clic sur un bouton "Download script and plot" -----	63
Figure 66 - Au clic sur un bouton "Download plot as .eps" -----	63
Figure 67 - Contenu des trois onglets au démarrage. -----	64
Figure 68 - Contenu du deuxième onglet en fonction du premier. -----	66
Figure 69 - Rappel du scénario et de la propriété sélectionnés -----	66
Figure 70 - Chronologie du téléchargement des datasets d'un scénario -----	68
Figure 71 - Différentes apparences du second encadré -----	70
Figure 72 - "Choose the evaluation metrics" -----	71
Figure 73 - Alignement des graphiques du troisième onglet -----	71
Figure 74 - Bouton de validation du contenu du second onglet-----	71
Figure 75 – Une des multiples apparences du troisième onglet-----	72
Figure 76 - Chronologie de téléchargement de graphe-----	76
Figure 77 - Bonus: Evaluation avec comparaison d'outils -----	77
Figure 78 - Réaction en chaine d'une erreur dans le champ d'un dataset -----	79
Figure 79 - Barre de menu-----	79

Glossaire

Implémentation : terme employé pour désigner le code en arrière-plan, que l'utilisateur ne voit pas.

Matching tool, matcher tool, outil de matching : traduit littéralement par outil de correspondances. Outil qu'utilise l'utilisateur pour créer des fichiers « mappé ».

Algorithme : suite finie et non-ambiguë d'opérations permettant de donner la réponse à un problème.

Schema matching : se traduit par correspondance de schémas. Un fichier de correspondances de schémas résulte des similitudes entre deux schémas.

UML : en anglais ***Unified Modeling Language*** ou « langage de modélisation unifié » est un langage de modélisation graphique à base de pictogrammes.

Plate-forme : Logiciel utilisé entre membres d'une communauté.

Correspondances : Similitudes entre deux schémas.

Scénario : Il est composé d'un ensemble de datasets.

Dataset : Il contient deux fichiers schéma et un fichier matché expert.

Fichier expert : fichier comportant les correspondances trouvées par des chercheurs.

Fichier « mappé », fichier de résultats : fichier comportant les correspondances trouvées par l'outil de matching.

Java : langage de programmation moderne. Ce langage est un langage orienté objet.

Serveur : dispositif informatique matériel ou logiciel qui offre des services, à différents clients. Dans notre cas, le plus souvent, ce sera pour le partage de fichiers.

Script : fichier contenant des lignes de commandes adaptées à son extension.

Gnuplot : logiciel permettant la constitution de graphiques à partir de fonctions ou de jeux de données.

En local : on parle d'une application en local lorsque celle-ci est directement accessible sans ressources extérieures.

En ligne : on parle, en général, d'une application en ligne lorsque celle-ci est accessible via un navigateur.

CRUD : de l'anglais, **Create – Read – Update – Delete**, ou en français, Créer – Lire – Mettre à jour – Supprimer.

Cas nominal : Cas normal d'utilisation sans possibilité d'erreurs.

Diagramme de classe en mode esquisse : Diagramme de classe dont les classes sont représentées sans leurs attributs ni méthodes.

Plug-in : extension d'un programme ou logiciel qui ajoute de nouvelles fonctionnalités.

MVC : Modèles, Vues, Contrôleurs, est un modèle de conception de logiciels.

Système d'exploitation : Exemple: Windows, Linux.

Classe : Une classe possède des attributs (exemple: prénom) et des méthodes (exemple: donner_prénom()).

Package : un package est composé d'un ensemble de classes.

Format interne : qui n'a de valeur qu'au sein de l'application.

Parser : définit la lecture d'un document.

ArrayList : équivalent d'un tableau en programmation Java

Programmation événementielle ou Programmation objets : type de programmation fondé sur les événements. Le programme sera principalement défini par ses réactions aux différents événements qui peuvent se produire.

JFileChooser : appellation pour définir une fenêtre de parcours d'arborescence de fichiers.

JTextArea: appellation pour définir une large zone pouvant contenir du texte.

JLabel: genre de libellé.

JBouton: désigne un bouton.

Tableau dynamique: tableau dont la taille peut être réduite ou augmentée au cours d'une même exécution du programme.

I. Introduction

Mon projet fut de réaliser une application nommée XBenchMatch, celle-ci devant contenir une interface graphique, une implémentation* de calculs ainsi qu'une représentation des résultats.

XBenchMatch est une application, pour les utilisateurs voulant tester, évaluer différentes caractéristiques de leur matcher tool* (ou outil de correspondances). Ceci n'étant pas facile au vu des différents fichiers à manipuler.

Afin de réaliser cette application, j'ai dû suivre diverses étapes ainsi que procéder à de diverses recherches qui seront précisées dans ce rapport.

Le programme possède une interface graphique permettant de guider l'utilisateur étape après étape dans son utilisation. Il comporte des algorithmes* de calculs basés sur les recherches de chercheurs concernant le domaine du schema matching*. Et il permet la représentation des résultats en termes de graphes.

Dans ce rapport sera présenté le sujet du stage et ce qu'il comporte.

Puis les détails des différentes démarches réalisées tout au long de ce projet avec : son analyse, ses diagrammes UML*, le planning, la conception et programmation de chacune de ses parties.

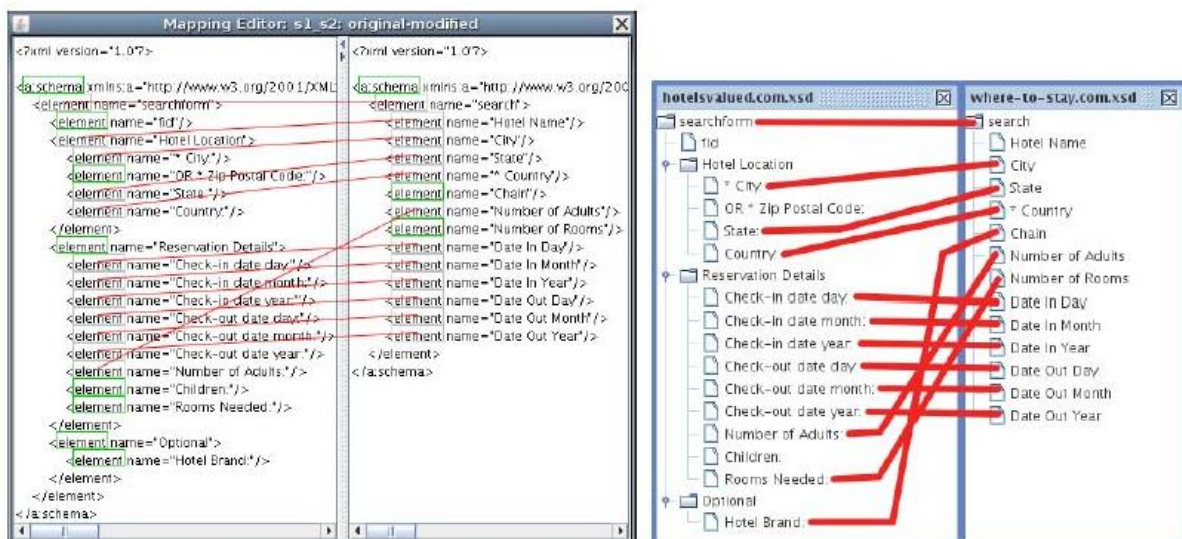
S'en suivra, une discussion sur la réalisation finale, ses failles et ses améliorations possibles.

Pour enfin finir, sur le bilan du stage.

II. Description du stage

A. PRESENTATION DU SUJET

Le sujet fut de développer une plate-forme* d'évaluation pour le schema matching*, que nous nous appliquerons à nommée XBenchMatch. Cette plate-forme sera une des bases de travail d'autres utilisateurs/chercheurs pour avancer dans leurs recherches, leurs créations. Le schema matching (ou correspondance de schémas) tient pour dit que deux schémas différents peuvent présenter des similitudes, nous appellerons cela des correspondances*, et c'est en cela qu'intervient l'évaluation de la qualité de ce schema matching, c'est-à-dire à montrer si ces correspondances sont effectivement correctes ou pas, à l'aide de mesures définies, que sont la précision, le rappel, ou encore le F-mesure et bien d'autres. Les schémas peuvent être de tous types, les principaux étant : xml, txt, csv, rdf.



(a) Correspondences discovered by a schema matcher

(b) Expert correspondences

Figure 4 - Exemple de deux schémas xml dit « mappé »

Les schémas font parti d'un scénario. Un scénario*, nommé suivant le domaine auquel il fait référence, est composé d'un ensemble de datasets* qui sont en général deux fichiers schéma et un fichier matché expert*, c'est-à-dire contenant des correspondances, trouvées par des chercheurs, entre ces deux schémas. Un scénario possède aussi un fichier « propriétés » qui liste ses différentes caractéristiques.

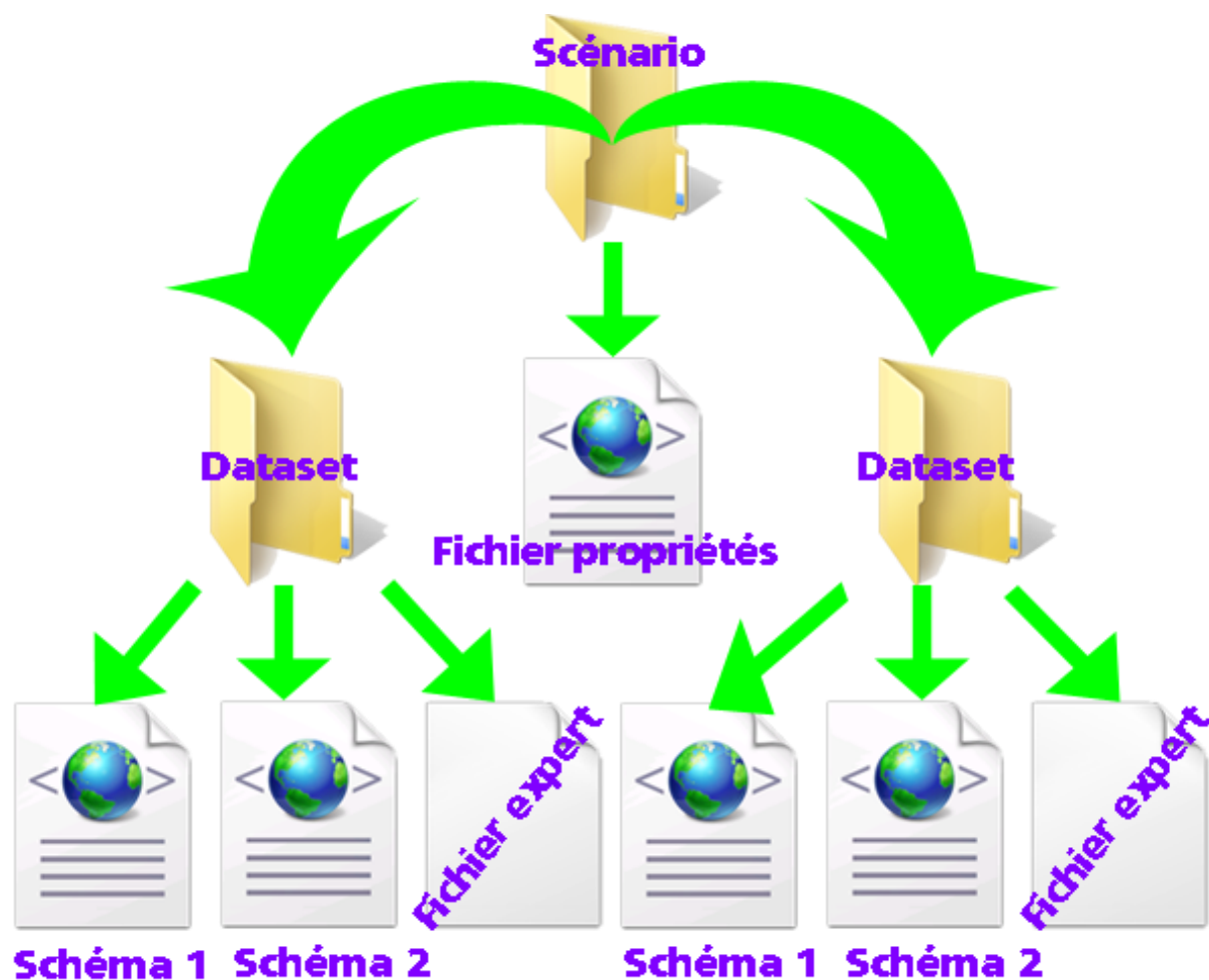


Figure 5 - Arborecence d'un scénario

	Label heterogeneity			Domain Specific	Average size			Structure		Number of schemas
	Low or normalized	Average	High		Small (<10)	Average (10-100)	Large (>100)	Flat	Nested (3<depth<7)	
Betting		x				x		x		12
Biology		x		x			x		x	2
Currency		x				x			x	2
Finance		x		x		x		x		8
Order	x						x		x	2
Person	x				x				x	2
Sms		x				x			x	2
Travel		x			x			x		20
Univ. courses		x				x		x		40
Univ. dept			x		x			x		2

Figure 6 - Liste des scénarios disponible en fonction des propriétés

Au stade où en sont les recherches au laboratoire, nous nous attacherons, à savoir, grâce à l'application développée, si un matcher tool* supporte le multilingue, s'il est toujours capable de découvrir des correspondances* quelque soit leur langue.

Plusieurs outils de schema matching* ont vu le jour, notamment COMA++, Similarity Flooding et YAM pour les plus curieux.

Programmer ce logiciel afin de fournir une assistance à l'utilisateur, jusqu'à l'obtention du résultat, et pouvoir tester tous les scénarios* de propriétés existant dans la base, sont les principales tâches à accomplir.

B. CAHIER DES CHARGES

Cette plate-forme* doit être utilisable par tous.

Ce que l'application doit respecter :

- Etre programmé sous Java*.
- Permettre à l'utilisateur de choisir/télécharger les schémas d'un scenario*
- Afficher les caractéristiques du scénario.
- Permettre à l'utilisateur d'envoyer, pour un scénario donné, un ou plusieurs fichiers de résultats* établis par son outil de correspondances*.
- Calculer des mesures sur la qualité du fichier résultat par rapport à un fichier expert*, le tout se faisant côté serveur*.
- Renvoyer l'évaluation à l'utilisateur sous différentes formes : textes, graphiques.
- Pouvoir télécharger tous les fichiers ayant servi à la représentation de l'évaluation : les données et script* gnuplot* notamment.
- Etre accessible en local*, via téléchargement préalable, et en ligne*, via une page web.
- Permettre un CRUD* pour les schémas de l'utilisateur.

C. PLANNING PREVISIONNEL

Il n'y avait pas vraiment de planning prévisionnel mais plus des objectifs à atteindre en vue de la prochaine réunion hebdomadaire. Ce tableau les représente en fonction de leurs durées dans le temps (en bleu) :

Semaines	semaine 1 05/03	semaine 2 12/03	semaine 3 19/03	semaine 4 26/03	semaine 5 02/04	semaine 6 09/04	semaine 7 16/04	semaine 8 23/04	semaine 9 30/04	semaine 10 07/05	semaine 11 14/05	semaine 12 21/05	semaine 13 28/05	semaine 14 04/06	semaine 15 11/06
Tâches															
S'approprier l'environnement de travail															
Concevoir une première interface de l'application															
Analyse, conception et programmation du premier onglet															
Filter les scénarios du premier onglet															
Conception du deuxième et troisième onglet															
Implémentation des mesures défini															
Mise en place d'un format interne															
Construction en dynamique du deuxième onglet															
Obtenir un graphique															
Tous les scénarios testables															
L'interface définitive															
Ajout de fonctionnalités supplémentaires															
Projet présentable pour une démo															
Mise en place de scénarios de comparaison d'outils															
Lancement d'une application prenant en compte le contenu d'un serveur															

Figure 7 - Planning prévisionnel

III. Démarche

A. ANALYSE DU PROJET

Tout d'abord, il a fallu analyser le projet et notamment donc examiner l'environnement de l'application.

Il y a donc des « scénarios » qui répondent à des propriétés spécifiques, et qui comportent chacun des fichiers de schémas pouvant être en .xml, .txt, .csv ou .rdf. Ces fichiers schémas comportant chacun des attributs.

Nous avons des *matcher tools** qui prennent en entrée deux de ces fichiers schémas et qui produisent en sortie un nouveau fichier, un fichier dit « mappé »* c'est-à-dire qui contient des correspondances* trouvé par l'outil, et dont l'extension est en fonction de l'outil de correspondances utilisé, autrement dit celles que l'on connaît déjà.

D'un autre côté, la disposition d'un serveur* qui, pour chacun des scénarios* possède un fichier dit « expert »* c'est-à-dire qui contient des correspondances trouvées par des Hommes. Leur extension peut être de plusieurs types.

De plus, il a fallu imaginer le résultat que je devais fournir à l'aide du cahier des charges.

Il faut une évaluation de ces deux fichiers, mappé (fichier venant donc de l'outil de correspondances de l'utilisateur) et expert (fichier venant des chercheurs), en s'appuyant sur des mesures déjà définies comme :

La précision : calcule la proportion de correspondances correctes découvertes par l'outil parmi toutes les correspondances découvertes.

Le rappel : calcule la proportion de correspondances correctes découvertes par l'outil parmi toutes les correspondances correctes.

Le F-mesure : F-mesure est un compromis entre précision et rappel.

L'Overall : évalue la quantité de travail que doit fournir un expert pour enlever les correspondances incorrectes découvertes par l'outil (faux positifs), et d'ajouter les correspondances manquées correctes par l'outil (faux négatifs). Il est basé sur la précision et le rappel.

Et le PostMatch Effort : évalue la quantité de travail que doit fournir un expert pour enlever les correspondances incorrectes découvertes par l'outil (faux positifs), et d'ajouter les correspondances manquées correctes par l'outil (faux négatifs). Il est basé sur un nombre d'interactions.

Le projet peut comporter une partie serveur* pour faire ces différents calculs, il faudra donc regarder les moyens de liaison qui peuvent être mis en œuvre entre l'application et le serveur.

Puis obtenir la représentation de ces différentes mesures sur des graphiques construits sous le logiciel Gnuplot*, programme en ligne de commande interactive de dessin de fonction.

Il faut donc créer une interface graphique facile à utiliser de sorte à pouvoir récupérer les bons fichiers de la part de l'utilisateur, et faire en sorte que ce dernier garde certaines marques et habitudes par rapport à d'autres logiciels qu'il a pu voir.

Il est donc important que l'interface graphique garde un certain standard même si par la suite nous pouvons lui donner une certaine originalité.

Tout ceci devant former une application complète et agréable.

B. UML

1. Use case (cas d'utilisations)

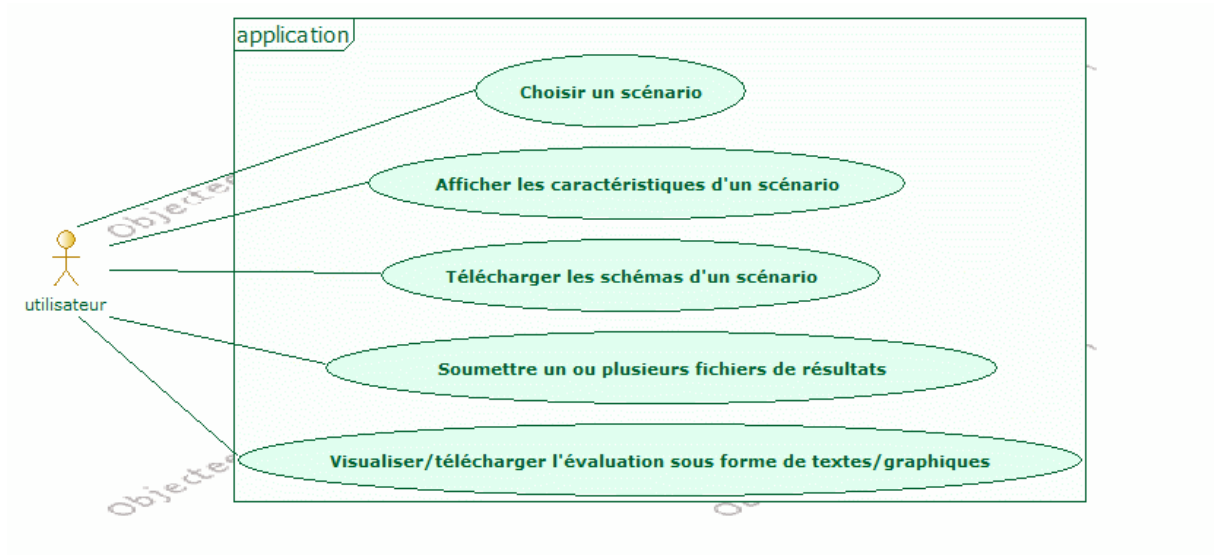


Figure 8 - Use case (cas d'utilisations)

2. Diagramme de séquence – scénario (cas nominal)

Ce diagramme prend en compte un scénario* ne contenant qu'un seul dataset* et l'utilisation d'un seul outil de correspondances*.

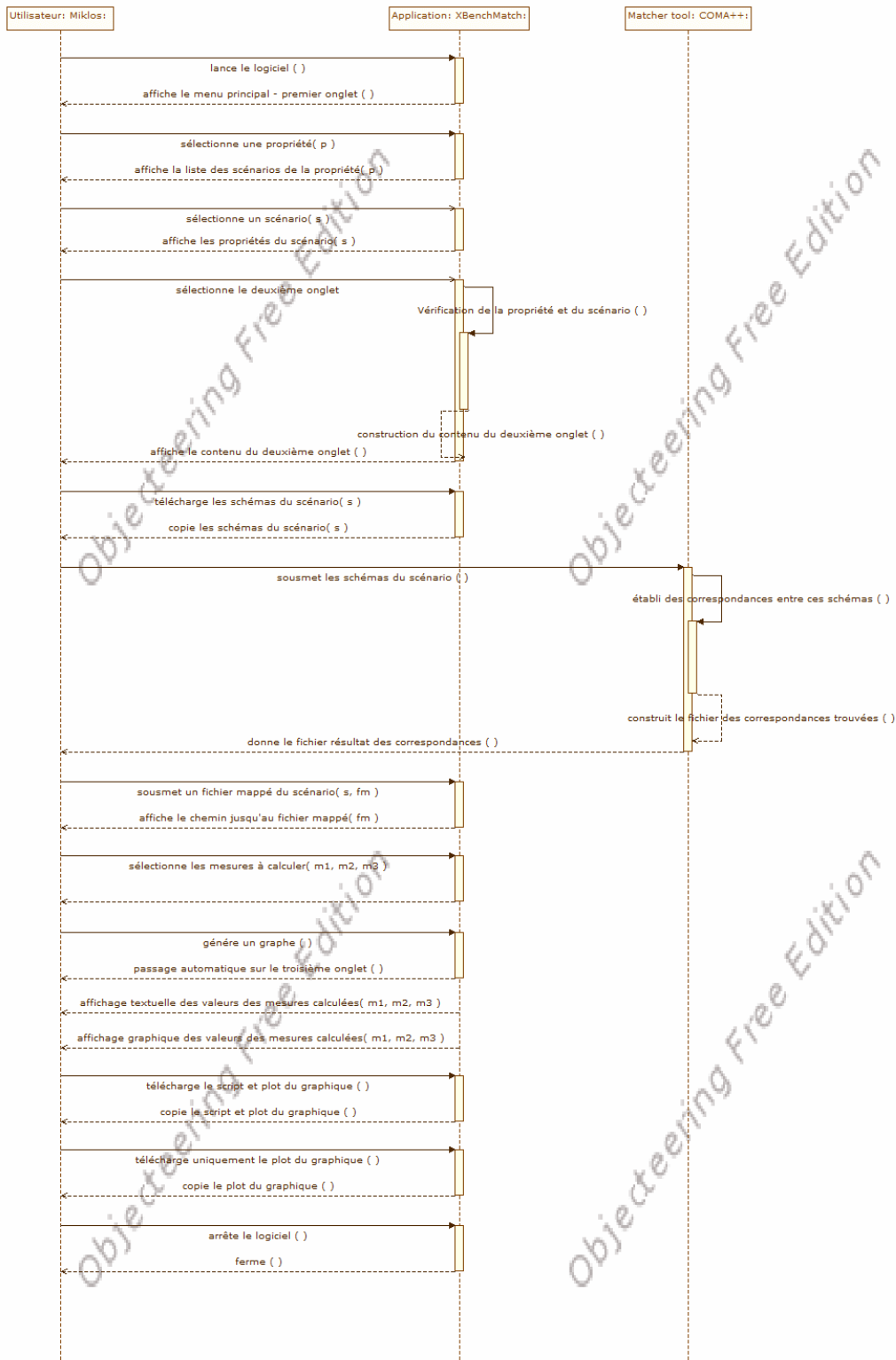


Figure 9 - Diagramme de séquence

3. Diagramme de classes

Pour des raisons évidente de lisibilité, le diagramme de classes est représenté en mode esquisse*.

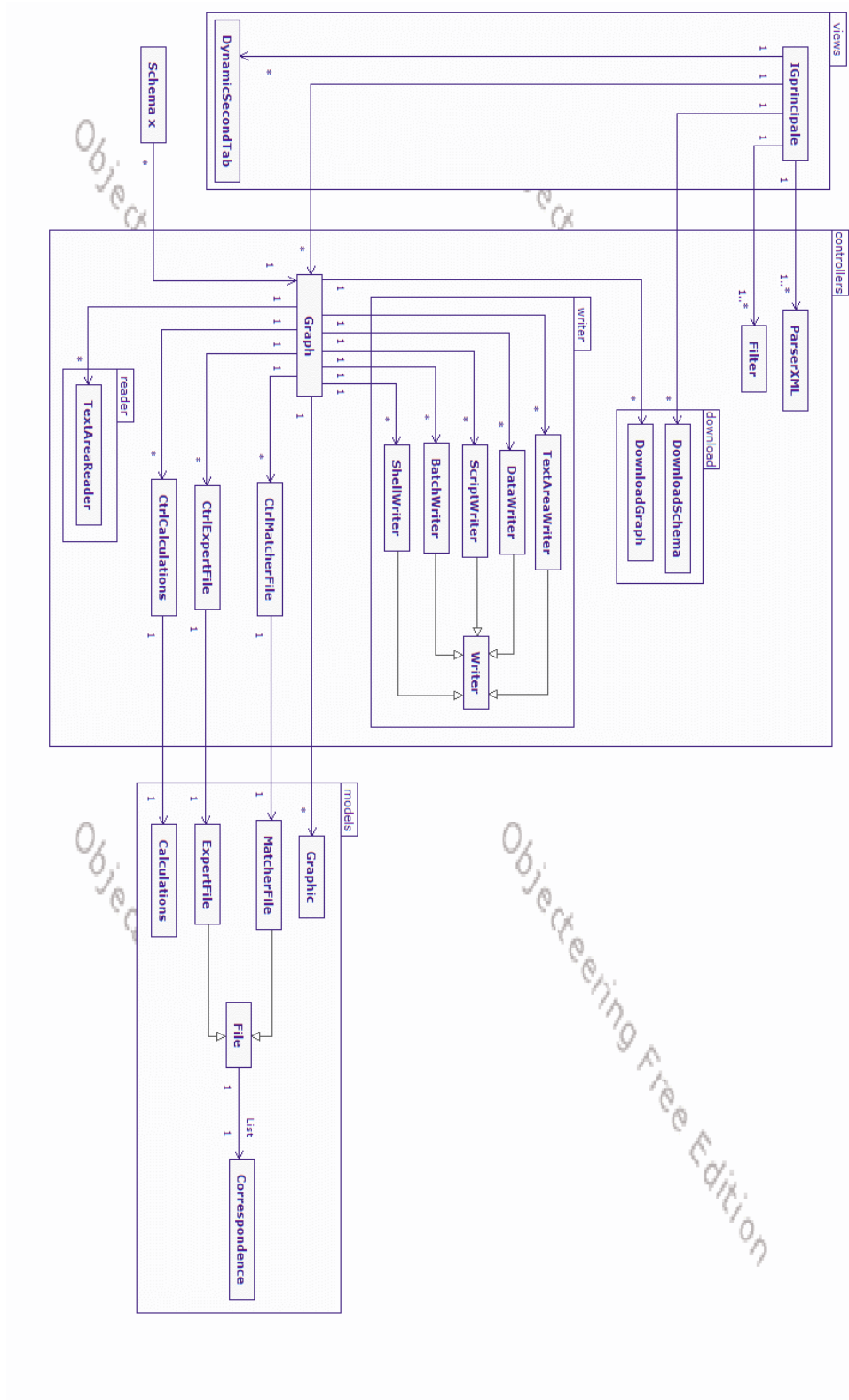


Figure 10 - Diagramme de classes

C. TECHNOLOGIES UTILISEES

Tout d'abord le langage Java, principal langage utilisé pour réaliser cette application. Java permet une programmation événementielle basée sur la réaction des différents composants (boutons, list).

FileZilla : logiciel qui permet une utilisation simplifiée d'un serveur* ftp grâce à une connexion rapide et des glisser-déposer pour les transferts de fichiers.

Ensuite, l'utilisation d'un serveur ftp, permettant le stockage de différents fichiers ainsi que la communication et l'échange d'informations, de fichiers avec l'utilisateur.

Il y a aussi le logiciel Gnuplot*, qui à l'aide d'un fichier script* et d'un fichier de données, permet d'éditer les graphiques que demande l'utilisateur.

Et enfin, les terminaux, des différents systèmes d'exploitations, liant ces deux derniers entre eux.

Ne pas oublier aussi les nombreux plug-ins*, qui à plus d'une fois, aident à faciliter les choses : Window builder pour n'en citer qu'un.

D. PLANNING

Voici le planning observé durant ce stage. Il est quelque peu différent du planning prévisionnel, ceci étant du au temps passé sur les différentes parties.

Semaines Tâches	semaine 1 05/03	semaine 2 12/03	semaine 3 19/03	semaine 4 26/03	semaine 5 02/04	semaine 6 09/04	semaine 7 16/04	semaine 8 23/04	semaine 9 30/04	semaine 10 07/05	semaine 11 14/05	semaine 12 21/05	semaine 13 28/05	semaine 14 04/06	semaine 15 11/06
S'approprier l'environnement de travail															
Première analyse et conception de l'interface du premier onglet															
Amélioration de la conception et programmation du premier onglet															
Deuxième analyse, conception et programmation de l'interface du deuxième onglet															
Amélioration de la conception et suite de la programmation du deuxième onglet															
Troisième analyse, conception et programmation de l'interface du troisième onglet															
Suite de la programmation du troisième onglet															
Remaniement d'une partie de la conception du troisième onglet et programmation de cette dernière															
Suite de la programmation du troisième onglet															
Amélioration de la programmation de l'application en générale															
Préparation de l'application a son passage vers le réseau.															
Implémentation de nouvelles fonctionnalités															
Dernière grosse programmation pour une application PC sans accès aux ressources serveur															
Transformation de parties statiques en parties dynamiques															
Connexion à un serveur et lecture de son contenu															

Figure 11 - Planning

E. DEVELOPPEMENT D'UNE PLATE-FORME D'EVALUATION

Notre application doit être en mesure d'interagir avec un utilisateur, pour cela, une interface simple et une bonne implémentation* du code en arrière-plan est de rigueur.

1. Présentation

Il a été décidé dès le départ que l'application devait être construite suivant le modèle MVC* (modèles, vues, contrôleurs).

La vue étant celle qui s'occupe principalement de l'interface graphique qui interagira avec l'utilisateur.

Et la couche modèles, celle s'occupant des multiples analyses de fichiers, des calculs des mesures définies et de leurs affichages respectifs.

Quand à la partie contrôleurs, elle est globalement définie par la classe* Graph qui va répartir les rôles sur presque toutes les autres classes de ce package*.

2. La classe Graph

Il peut être bon de s'attarder sur cette classe. En effet, celle-ci est ce que l'on pourrait définir comme « les piliers » qui maintiennent l'édifice, l'édifice étant l'application. Autrement dit, sans cette classe le logiciel ne tiendrait plus debout. Elle fait largement office de lien entre les couches vues et modèles.

La preuve étant son grand nombre de paramètres, reçu en entrée, par la classe IGprincipale situé dans le package vues :

```
public Graph (String filePath, String p_resultsdirectoryname,
List<JLabel> listJL, List<JTextField> listJTF, List<JCheckBox> listJCB,
JList list, JTextArea p_textArea_1, JPanel panel_3,
JCheckBox chckbxPrecision, JCheckBox chckbxRecall, JCheckBox chckbxFmeasure, JCheckBox chckbxOverall, JCheckBox chckbxPostmatch,
JRadioButton rdbYes)
```

Figure 12 - Les paramètres de la classe Graph

Et ses multiples appels aux différents contrôleurs ayant accès aux classes situées dans la partie modèles :

```
for (int j=0; j<listJL.size(); j++)
{
    CtrlMatcherFile listmatcher = new CtrlMatcherFile(listJTF.get(j).getText());

    File f = new File (filePath + "/" + listJCB.get(i).getText() + "/" + list.getSelectedValue() +
        "/" + listJL.get(j).getText());

    dw.append(j+1); // j+1 allow to get abscissa for the file .data

    taw.append("\nDataset \"" + listJL.get(j).getText() + "\" :");

    String[] mylist = f.list();
    int k=0;
    while ((k<mylist.length) && (!mylist[k].endsWith(".map")))
    {
        k++;
    }
    CtrlExpertFile listexpert = new CtrlExpertFile(f + "/" + mylist[k]);

    CtrlCalculations cc = new CtrlCalculations(listmatcher, listexpert);
}
```

Figure 13 - Appels à de multiples contrôleurs

3. Première conclusion

Chaque couche, modèles, vues, contrôleurs, a donc son importance, le tout étant de ne pas négliger les liens qui les unissent entre eux.

F. COUCHE MODELES

Cette couche va permettre le déroulement de toutes les tâches de fond, que ce soit pour la récupération des informations contenues dans les fichiers, ou pour les calculs précédant leurs représentations vectorielles.

1. Lectures

En effet, l'application doit partir à la recherche des fichiers dont elle a besoin pour la suite de ces opérations.

Elle cherche donc à récupérer un fichier mappé*, donné par l'utilisateur, et le fichier expert* lui correspondant, par rapport au scénario* sélectionné, auprès du serveur*.

Ceci étant, le programme ne peut pas effectuer de calculs à partir des lignes en bruts des fichiers. C'est à ce moment là, qu'intervient la notion de format interne* des correspondances*.

Chaque type de fichiers a sa propre structure de correspondances, il faut donc parser* ces fichiers de sorte à obtenir à chaque fois une structure de correspondances commune efficace pour l'application.

Il a donc été décidé qu'une correspondance est un ensemble de deux éléments, et qu'un fichier parsé contiendrait une ArrayList* de correspondances.

```
public class Correspondence {  
    private String element1;  
    private String element2;  
    private String relation;  
  
    public Correspondence(String P_element1, String P_element2, String P_relation)  
    {  
        element1 = P_element1;  
        element2 = P_element2;  
        relation = P_relation;  
    }  
}
```

Figure 14 - Structure du format interne d'une correspondance dans l'application

```

public class File {

    private List<Correspondence> listmatches;

    /**
     * Constructor who will read each line of the file passed as parameter
     * and put each one in the global array of Correspondence.
     */
    public File(String file)
    {
        listmatches = new ArrayList<Correspondence>();

        try{
            InputStream ips=new FileInputStream(file);
            InputStreamReader ipsr=new InputStreamReader(ips);
            BufferedReader br=new BufferedReader(ipsr);
            String line;
            while ((line=br.readLine())!=null)
            {
                //System.out.println(line);
                String[] listelement = line.split(";");//we cut the line in parts.
                if (listelement.length == 2)
                {
                    listmatches.add(new Correspondence(listelement[0].trim(), listelement[1].trim(), ""));
                }
                else
                {
                    listmatches.add(new Correspondence(listelement[0].trim(), listelement[1].trim(), listelement[2].trim()));
                }
            }
            br.close();
        }
        catch (Exception e){
            System.out.println(e.toString());
        }
    }
}

```

Figure 15 - Exemple de parser simple

Une fois les fichiers lus, la suite s'enchaîne.

2. Calculs

La première chose, est de savoir combien de correspondances* correctes ont été trouvées par l'outil de matching*.

Pour ce faire, l'utilisation des ArrayList* de correspondances, obtenues plutôt par le parcours des parser* sur les différents fichiers, va faciliter l'opération de par des chassés-croisés de comparaison entre les différents éléments des correspondances.

```
public class Calculations {  
    private int numberCorrectMatches = 0;  
    public Calculations (List<Correspondence> listmatches_MatcherFile, List<Correspondence> listmatches_ExpertFile)  
    {  
        for (int i=0; i<listmatches_MatcherFile.size(); i++)  
        {  
            for (int j=0; j<listmatches_ExpertFile.size(); j++)  
            {  
                if (((listmatches_MatcherFile.get(i).getelement1().equals(listmatches_ExpertFile.get(j).getelement1()))  
                    || (listmatches_MatcherFile.get(i).getelement1().equals(listmatches_ExpertFile.get(j).getelement2()))  
                    && ((listmatches_MatcherFile.get(i).getelement2().equals(listmatches_ExpertFile.get(j).getelement1()))  
                    || (listmatches_MatcherFile.get(i).getelement2().equals(listmatches_ExpertFile.get(j).getelement2())))))  
                {  
                    numberCorrectMatches++;  
                }  
            }  
        }  
    }  
}
```

Figure 16 - Comparaison entre éléments de correspondances

Par la suite, les différentes mesures définies sont vite implémentées* en donnant en paramètre une ou deux entrées en plus.

```
public float Precision(List<Correspondence> listmatches_MatcherFile)
{
    return ((float)numberCorrectMatches/(float)listmatches_MatcherFile.size());
}

public float Recall (List<Correspondence> listmatches_ExpertFile)
{
    return ((float)numberCorrectMatches/(float)listmatches_ExpertFile.size());
}

public float Fmeasure (float precision, float recall)
{
    if (precision + recall != 0.0)
        return ((2 * precision * recall)/(precision+recall));
    else
        return (0);
}

public float Overall (float precision, float recall)
{
    float overall;
    if (precision != 0.0)
    {
        overall = recall * (2 - (1/precision));
    }
    else
    {
        overall = -1;
    }

    if (overall<-1.0)
    {
        overall = -1;
    }

    return overall;
}

public double PME (int sizeMatcherFile, int sizeExpertFile)
{
    double mydouble = 0;
    try {
        mydouble = PostMatchEffort.extractValues(sizeMatcherFile,sizeExpertFile,sizeMatcherFile,
            numberCorrectMatches,(sizeExpertFile-numberCorrectMatches), "PME");
    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    return mydouble;
}
```

Figure 17 - Les différentes mesures définies

3. Affichages

Cette section se concrétise par une association entre la classe* Graph dans la couche contrôleurs et la classe Graphic dans la couche modèles.

Concrètement, la classe Graph va demander à la classe Graphic de lancer, exécuter un script*, qu'elle aura auparavant créé et correspondant au système d'exploitation* exécuteur. Graphic va exécuter les instructions (ou lignes de commandes) contenues dans le script via l'ouverture d'un terminal ou d'une console, qu'elle fermera une fois toutes les instructions lues ou appliquées.

```
if (System.getProperty("os.name").contains("Window"))
{
    new BatchWriter(resultsdirectoryname, filename);
}
else
{
    new ShellWriter(resultsdirectoryname, filename);
}
```

Figure 18 - Construction d'un script suivant le système d'exploitation

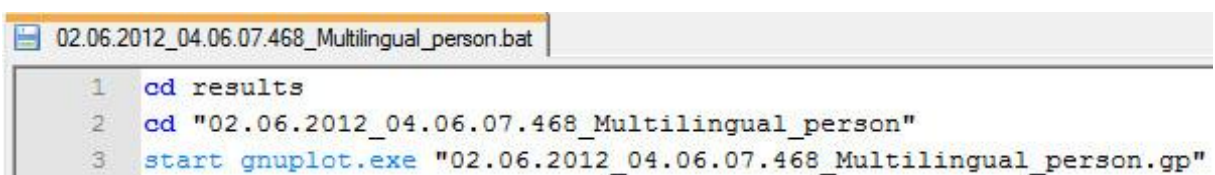
```
if (System.getProperty("os.name").contains("Window"))
{
    new Graphic().callTerminal("./" + resultsdirectoryname + "/" + filename + "/" + filename+ ".bat");
}
else
{
    new Graphic().callTerminal("sh " + "./" + resultsdirectoryname + "/" + filename + "/" + filename+ ".sh");
}
```

Figure 19 - Appel vers le terminal suivant le système d'exploitation

```
public class Graphic {  
    public void callTerminal(String filepath)  
    {  
        try {  
            Runtime.getRuntime().exec(filepath);  
        } catch (IOException e) {  
            // TODO Auto-generated catch block  
            e.printStackTrace();  
        }  
    }  
}
```

Figure 20 – Ouverture d’une console avec un paramètre de script de console

Pour un peu plus de détails sans rentrer dans le technique, le programme demande l’exécution d’un script* de console (ou terminal), celui-ci demande l’exécution d’un script Gnuplot*, pour rappel : le logiciel qui permet de dessiner des graphiques, qui fabrique une image des calculs effectués précédemment.



```
02.06.2012_04.06.07.468_Multilingual_person.bat  
1 cd results  
2 cd "02.06.2012_04.06.07.468_Multilingual_person"  
3 start gnuplot.exe "02.06.2012_04.06.07.468_Multilingual_person.gp"
```

Figure 21 - Exemple de script batch pour windows

```

02.06.2012_04.06.07.468_Multilingual_person.gp
1  # This is the script for build the plots through the software gnuplot.
2  set terminal postscript "Helvetica,22" color eps
3  set output "02.06.2012_04.06.07.468_Multilingual_person.eps"
4  set title "person"
5  set xlabel "Multilingual"
6  set ylabel "Value in percent"
7  set xrange [0:10]
8  set yrange [0:100]
9  #set xtics 0, 1, 10
10 set ytics 0, 10, 100
11 set xtics ("en" 1, "eo" 2, "fr" 3)
12 #set grid xtics ytics
13 #set key on right top box
14 #set size 0.8
15
16 # defining the style lines of each function : pt=pointstyle, ps=pointsize, lc=linecolor, lw=linewidth, lt=linestyle
17 set style line 1 lt 1 lw 2 pt 1 ps 2 lc rgb "green"
18 set style line 2 lt 1 lw 2 pt 1 ps 2 lc rgb "red"
19 set style line 3 lt 1 lw 2 pt 1 ps 2 lc rgb "blue"
20 set style line 4 lt 1 lw 2 pt 1 ps 2 lc rgb "dark-violet"
21 set style line 5 lt 1 lw 2 pt 1 ps 2 lc rgb "black"
22
23 plot '02.06.2012_04.06.07.468_Multilingual_person.data' using 1:2 title 'Precision' with linespoints ls 1, \
24 '02.06.2012_04.06.07.468_Multilingual_person.data' using 1:3 title 'Recall' with linespoints ls 2, \
25 '02.06.2012_04.06.07.468_Multilingual_person.data' using 1:4 title 'F-measure' with linespoints ls 3, \
26 '02.06.2012_04.06.07.468_Multilingual_person.data' using 1:5 title 'Overall' with linespoints ls 4, \
27 '02.06.2012_04.06.07.468_Multilingual_person.data' using 1:6 title 'Post Match' with linespoints ls 5

```

Figure 22 - Exemple de script Gnuplot

```

02.06.2012_04.06.07.468_Multilingual_person.data
1  # data file
2  # Abscissa Precision Recall F-measure Overall Post Match
3  #0 0 0 0 0 0
4  1 100 42 59 42 100
5  2 80 57 66 42 68
6  3 50 71 58 0 70

```

Figure 23 - Exemple de fichier de données pouvant aller de pair avec un script Gnuplot

A ce moment là, une méthode est faite pour que la classe* Graph puisse récupérer cette image et la rendre affichable.

```

public ImageIcon getImage (String resultsdirectoryname, String filename)
{
    ImageIcon icon = new ImageIcon("./" + resultsdirectoryname + "/" + filename + "/" + filename + ".png");
    return icon;
}

```

Figure 24 - Fonction retournant une image

4. Conclusion sur la couche modèles

Chacune des sections de cette couche dépend l'une de l'autre. Sans cette couche, le logiciel n'est plus qu'un programme avec une coquille vide. Encore faut-il pouvoir y accéder lorsqu'elle est pleine.

G. INTERFACE GRAPHIQUE

Le plus marquant sur cette application, au niveau visuel, est sa conception de l'interface graphique se présentant sous la forme de trois onglets. Chaque onglet se veut spécifique à un domaine d'interactions de la part de l'utilisateur, que ce soit pour la sélection, le paramétrage, ou encore l'affichage. Se faisant, chaque onglet devient alors dépendant du précédent.

1. Sélections

Dans cet onglet, qui fait office de page d'accueil du logiciel, l'utilisateur peut sélectionner un scénario* qui sera la base de la suite des opérations.

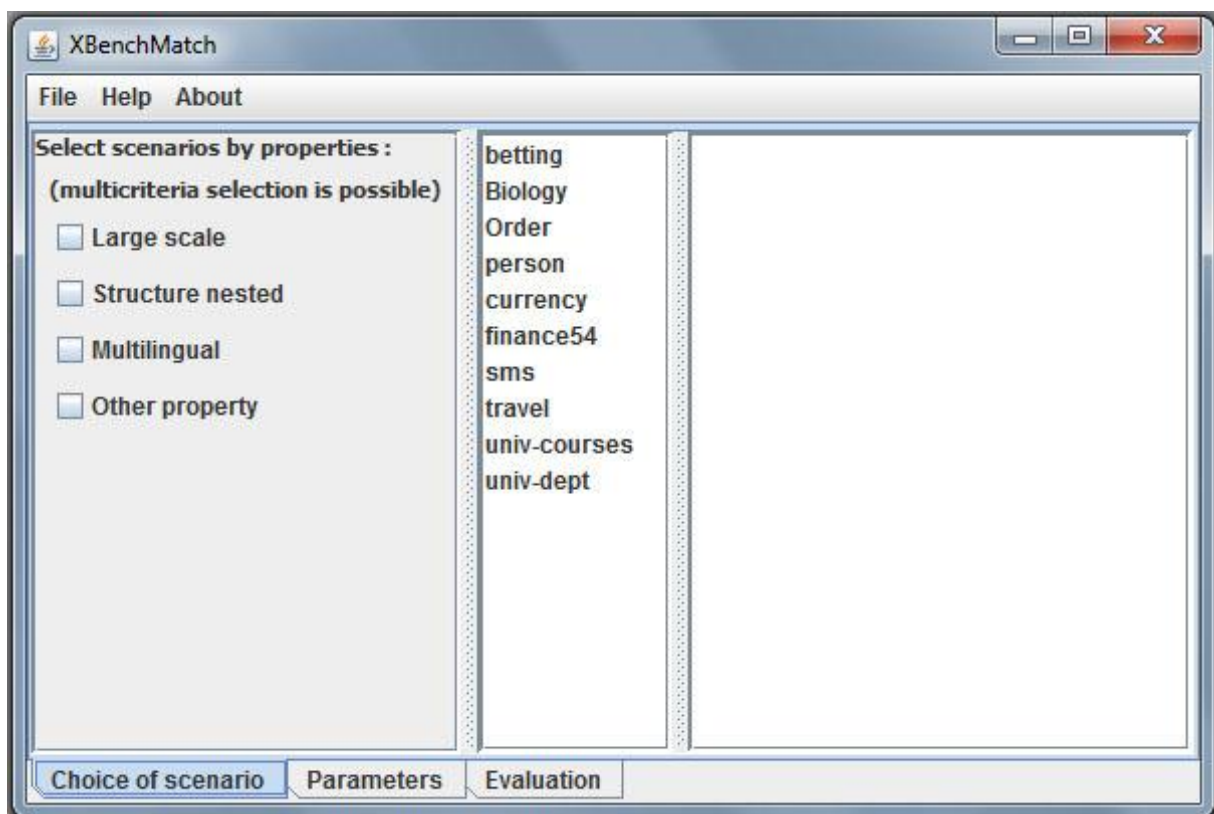


Figure 25 - Ecran de démarrage de l'application

La liste des cases à cocher est construite dynamiquement grâce à l'appel de la classe* ParserXML, effectué au lancement de l'application, et qui va récupérer, grâce à une lecture de fichier XML contenu dans chacun des scénarios, toutes les propriétés possédées par ces scénarios.

Lorsqu'un scénario* est sélectionné, ses propriétés sont affichées, ce qui permet de savoir dans combien de ces dernières il est possible de le rencontrer. Ces affichages d'informations sont des appels récurrents vers la classe* ParserXML.

Pour le code entier de la classe ParserXML, voir les annexes.

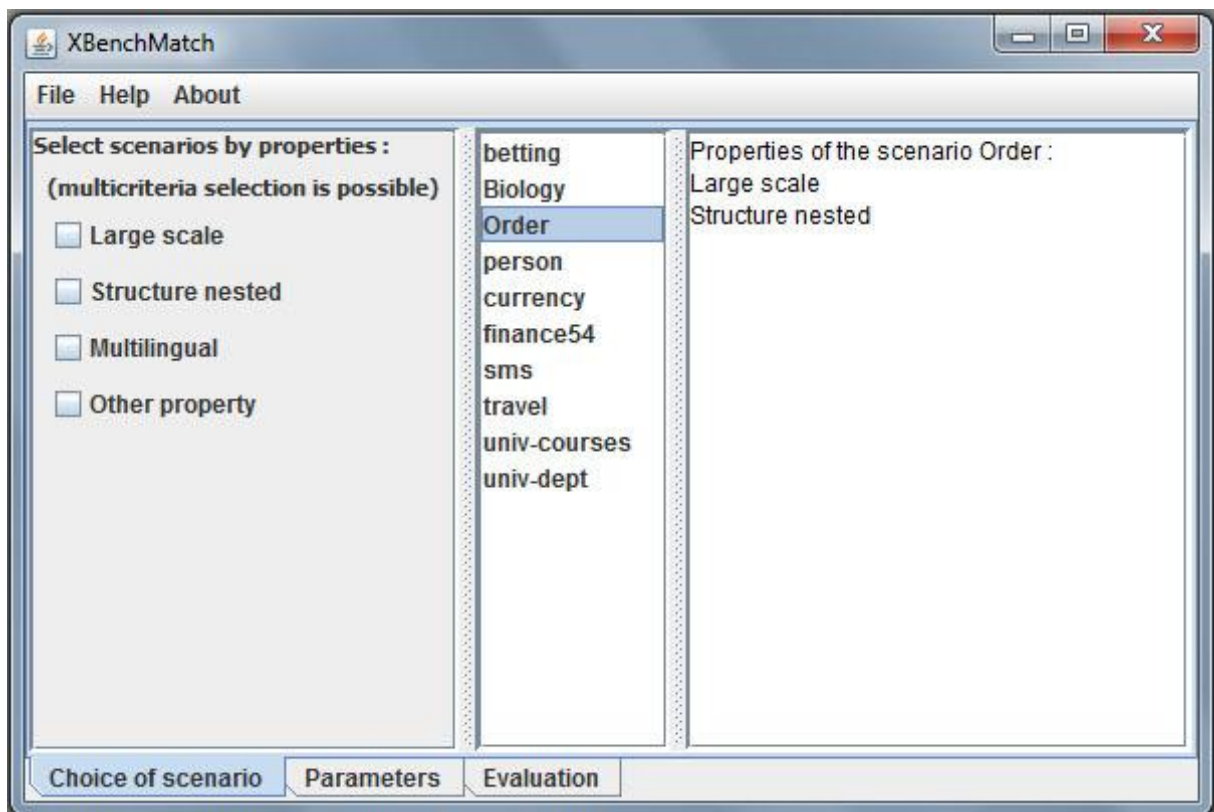


Figure 26 - Sélection d'un scénario

Une fonctionnalité supplémentaire est de pouvoir filtrer les scénarios par propriétés. Le résultat en ressortant alors, est la liste des scénarios ayant en commun le fait qu'ils partagent tous les propriétés cochées. Autrement dit, les scénarios en résultant sont disponibles dans chacune des propriétés cochées.

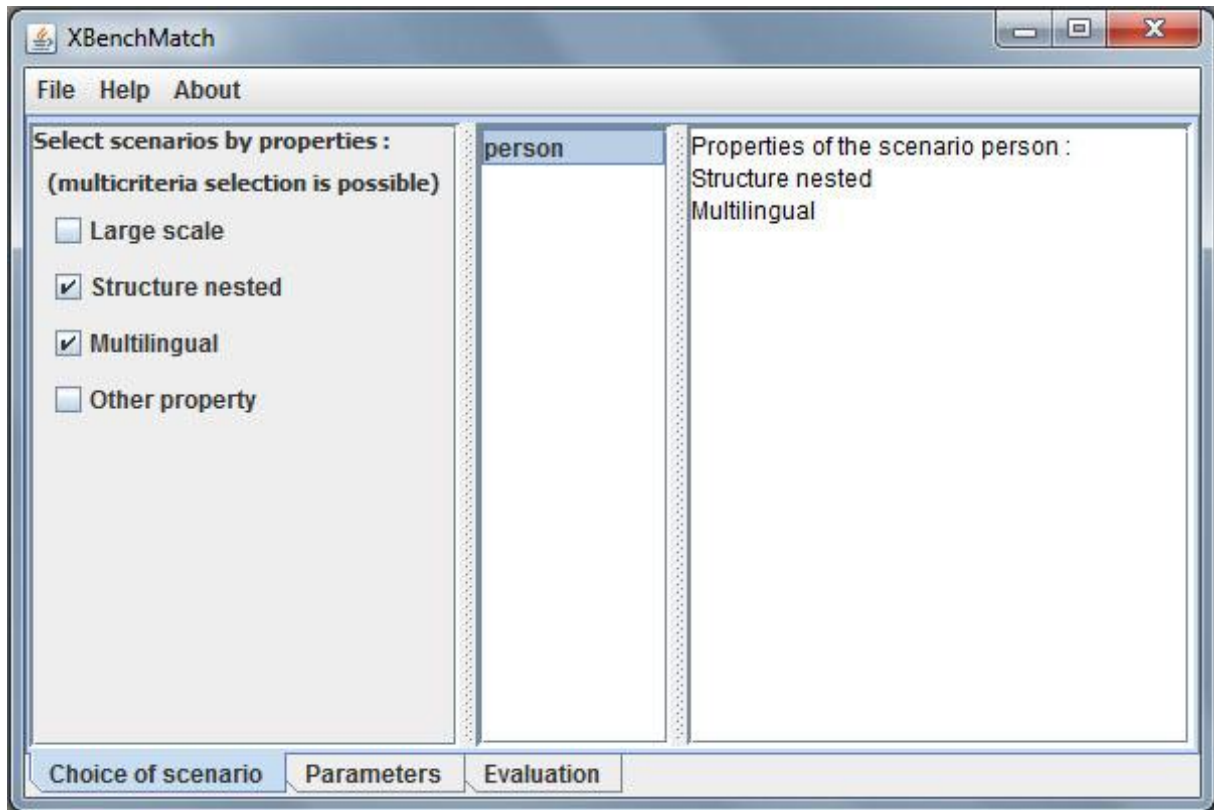


Figure 27 - Tri des scénarios en fonction des propriétés cochées

Cette fonctionnalité est prise en charge par la classe* Filter se trouvant dans la couche contrôleurs et dont le code est mis en annexe.

Pour passer correctement à l'étape suivante, le programme doit obligatoirement obtenir une propriété et un scénario* associé à cette dernière.

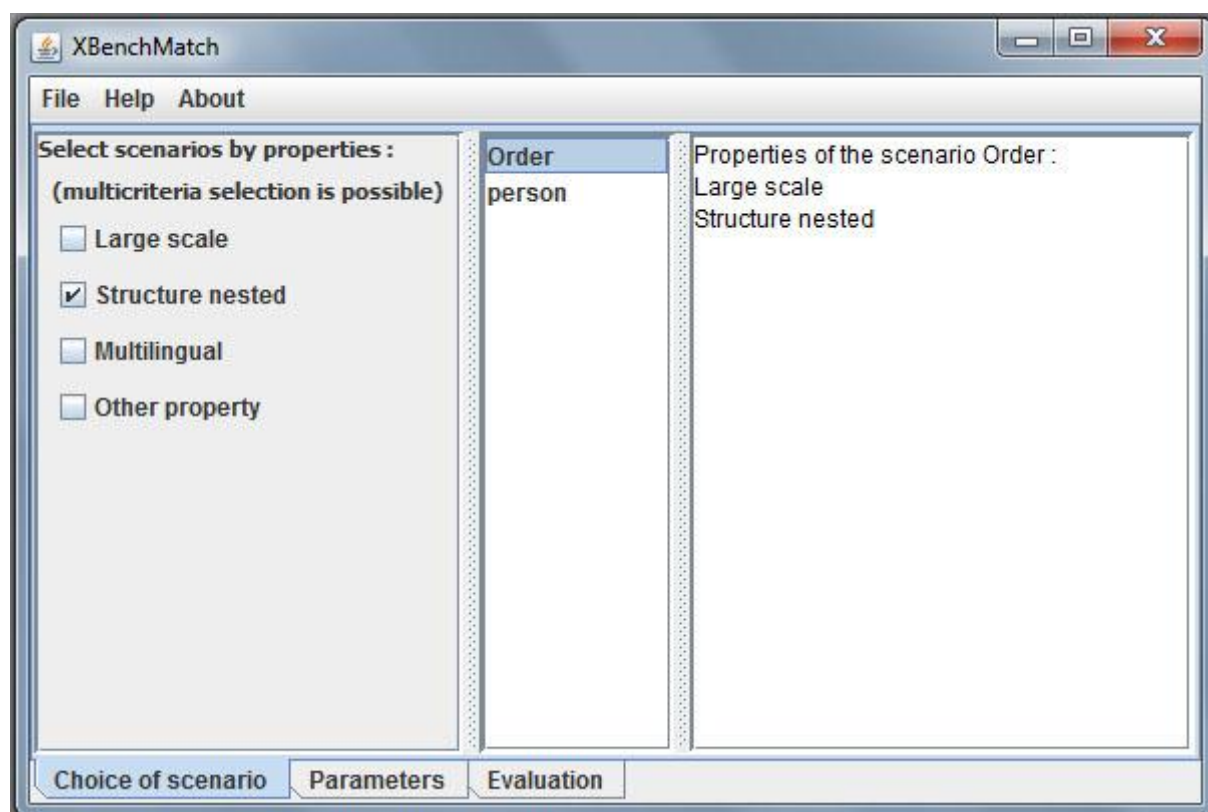


Figure 28 - Sélection d'une propriété et d'un scénario

2. Paramétrages

Si l'application était une maison, alors la pièce à vivre, son cœur, serait son deuxième onglet, car c'est celui qui accorde le plus d'interactions avec l'utilisateur.

Pour respecter la relation entre le premier onglet et celui-ci, le contenu de ce dernier ne sera présenté que si une et une seule propriété ainsi qu'un scénario* est sélectionné. Un peu comme à l'image d'une maison où s'il n'y a pas de clé pour ouvrir la porte d'entrée ou que celle-ci n'est pas correcte alors l'intérieur ne sera pas accessible. La classe* `DynamicSecondTab` est garante de cette vérification :

```
if (list.getSelectionEmpty())
{
    // a error message in RED.
    lblScenario = new JLabel("A scenario is required to run the BenchMark");
    lblScenario.setForeground(Color.RED);

    // Put the first tab (value 0) in color RED.
    tabbedPane.setBackgroundAt(0, Color.RED);
}
else //There are no errors
{
    lblScenario = new JLabel("Scenario : " + list.getSelectedValue());
    lblScenario.setForeground(Color.BLUE);
}
```

Figure 29 - Vérification de la sélection d'un scénario

```

int numberOfJCheckBoxchecked = 0;
for (i=0; i<listJCB.size(); i++)
{
    if (listJCB.get(i).isSelected())
    {
        numberOfJCheckBoxchecked++;
    }
}

if (numberOfJCheckBoxchecked == 0 || numberOfJCheckBoxchecked > 1)
{
    // a error message in RED.
    lblProperty = new JLabel("A property is required to run the BenchMark");
    lblProperty.setForeground(Color.RED);

    // Put the first tab (value 0) in color RED.
    tabbedPane.setBackgroundAt(0, Color.RED);
}
else //There are no errors
{
    i=0;
    while (!listJCB.get(i).isSelected())
    {
        i++;
    }

    lblProperty = new JLabel("Property : " + listJCB.get(i).getText());
    lblProperty.setForeground(Color.BLUE);
}

```

Figure 30 - Vérification de l'obtention d'une et une seule propriété

```

/*
 * If we have one CheckBox checked (property) and one element of the list selected (scenario),
 * then we build the suite of the tab
 * else we do nothing.
 */
if (numberOfJCheckBoxchecked == 1 && (!list.isSelectionEmpty()))
{

```

Figure 31 - Vérification pour garder intacte la relation entre le premier et le deuxième onglet

Ensuite, tout comme dans une cuisine où il y a le coin des casseroles, de la vaisselle, des instruments multiples et variés, dans cet onglet, il y a le coin téléchargement, passage de fichiers, sélection des mesures et autres valeurs associées.

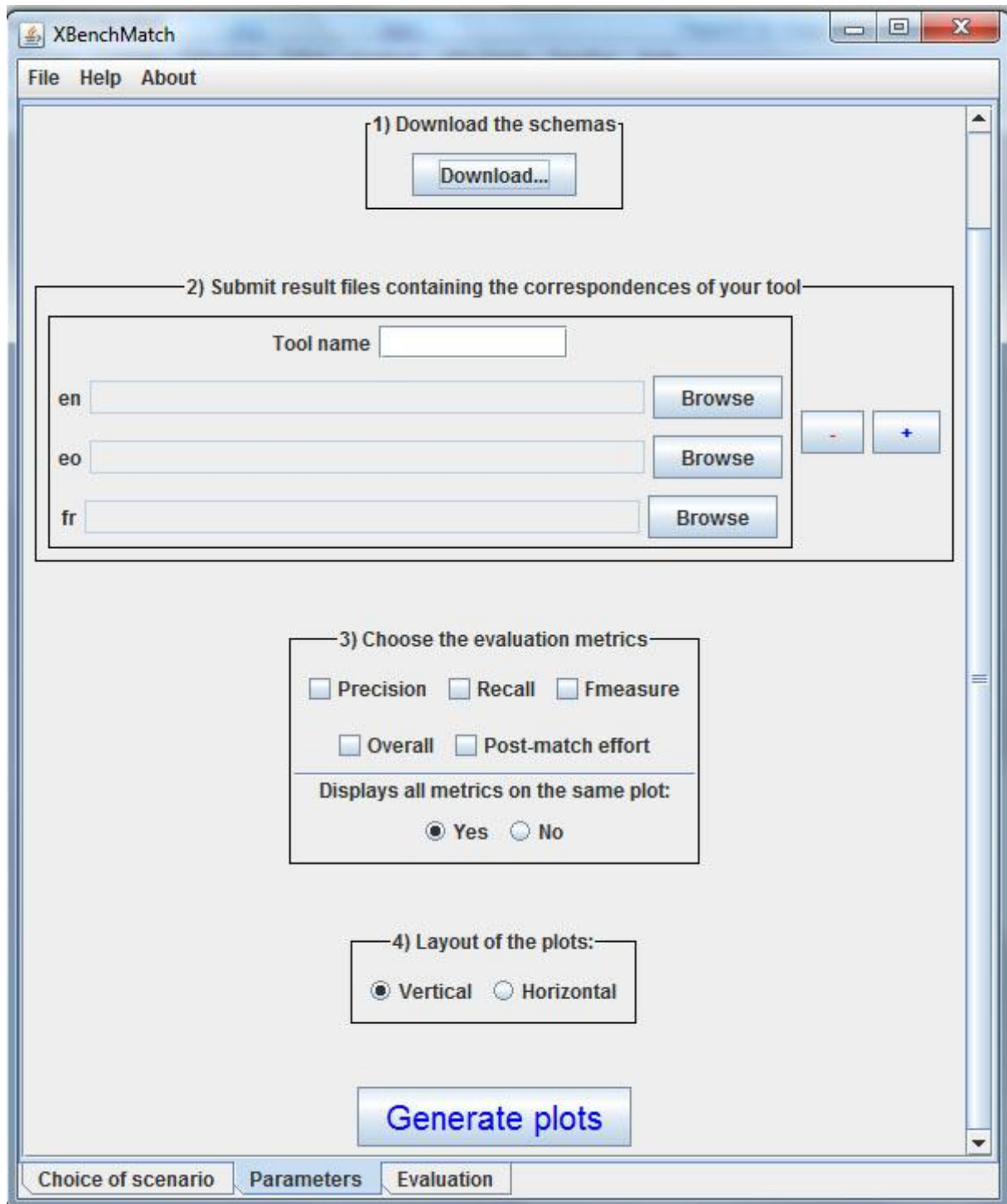


Figure 32 - Les différentes sections du deuxième onglet

La première section, ne contenant finalement qu'un seul bouton, permet d'obtenir les fichiers schémas du scénario* sélectionné de la propriété cochée. Un JFileChooser* permet tout de même de demander à l'utilisateur dans quel dossier il souhaite les mettre.

```

else if (e.getSource() == btnDownload)
{
    FileFilter ff_1 = new FileFilter(){
        public boolean accept(File f){
            if(f.isDirectory()) return true;
            else return false;
        }
        public String getDescription(){
            return "Folder";
        }
    };
    fc_1.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
    fc_1.setDialogTitle("Select a destination folder");
    fc_1.setApproveButtonText("Download");
    fc_1.setApproveButtonToolTipText("To put the scenario schemas in this folder.");
    fc_1.removeChoosableFileFilter(fc_1.getFileFilter());
    fc_1.setFileFilter(ff_1);
    if (fc_1.showOpenDialog(btnDownload) == JFileChooser.APPROVE_OPTION)
    {
        new DownloadSchema(filePath, listJCB, list, fc_1.getSelectedFile().getAbsolutePath());
    }
}

```

Figure 33 - JFileChooser permettant le téléchargement des fichiers de schémas

Où le code de la classe DownloadSchema est disponible en annexe.

La deuxième section, elle, est construite comme un tiroir, où seraient disposés des rangements, pour entreposer, par exemple, des cuillères à soupe, des cuillères à café, et des cuillères à thé. Notre section pourrait donc avoir un outil de matching* COMA++, un outil Similarity Flooding, et un outil Agreement maker. Mais certaines personnes peuvent ne pas avoir un tel équipement, et c'est en cela que des boutons sous forme de « plus » et de « moins » sont présents. Ils permettent de placer autant de parcelles que d'outils dont dispose l'utilisateur.

2) Submit result files containing the correspondences of your tool

Tool name COMA++		
en		Browse
eo		Browse
fr		Browse
Tool name Similarity Flooding		
en		Browse
eo		Browse
fr		Browse
Tool name Agreement maker		
en		Browse
eo		Browse
fr		Browse

Figure 34 - Deuxième section composée de trois outils de matching

```

btnMinus = new JButton("-");
btnMinus.setForeground(Color.RED);
btnMinus.addActionListener(IGp);
panel_22.add(btnMinus);

btnPlus = new JButton("+");
btnPlus.setForeground(Color.BLUE);
btnPlus.addActionListener(IGp);
panel_22.add(btnPlus);

```

Figure 35 - Mise en place des boutons "Plus" et "Moins"

```

else if (e.getSource() == btnMinus)
{
    if (countertool > 0)
    {
        countertool--;
    }

    DynamicSecondTab();
}
else if (e.getSource() == btnPlus)
{
    countertool++;

    DynamicSecondTab();
}

```

Figure 36 - Compteur pris en charge par les boutons "Plus" et "Moins"

```

for (int j=0; j<countertool; j++)
{

```

Figure 37 - Construction des "rangements" en fonction du compteur

Une fois, les rangements visualisés, le remplissage par les matériaux correspondants peut commencer.

A savoir que, s'il n'y a qu'un seul outil présent, alors il y aura une évaluation future des mesures sélectionnées, contrairement au cas où, s'il y a plus d'un outil, alors il y aura une évaluation future des mesures sélectionnées avec comparaison des outils, et c'est en cela que réside l'avantage du bouton « Plus ».

Dans chacun de ces rangements, l'utilisateur doit pouvoir donner un nombre de fichiers mappé*, par l'outil correspondant, égal au nombre de datasets* contenu dans le scénario* sélectionné. Afin de réaliser ces actions, des boutons nommés « Browse » sont associés à chacun des datasets de ce scénario.



Figure 38 - Fichiers "mappé" associés aux datasets

```

File f = new File (filePath + "/" + listJCB.get(i).getText() + "/" + list.getSelectedValue());
String[] mylist = f.list();

for (int k=0; k<mylist.length; k++)
{
    File f1 = new File (f + "/" + mylist[k]);
    if ((f1.isDirectory()) && (!f1.getName().equalsIgnoreCase(".svn")))
    // like that, the file "properties.xml" is not read.
    {
        kvirtual++;

        listJP1.add(new JPanel());
        counterpanel++;
        listJP1.get(referenceCounter).add(listJP1.get(counterpanel));

        listJL.add(new JLabel(mylist[k]));
        listJP1.get(counterpanel).add(listJL.get(kvirtual));

        listJTF.add(new JTextField());
        listJTF.get(kvirtual).setColumns(30);
        listJTF.get(kvirtual).setEditable(false);
        listJP1.get(counterpanel).add(listJTF.get(kvirtual));

        listJB.add(new JButton("Browse"));
        listJB.get(kvirtual).addActionListener(IGp);
        listJP1.get(counterpanel).add(listJB.get(kvirtual));
    }
}

```

Figure 39 - Boutons "Browse" associés aux datasets d'un scénario

```

/*
 * if one of our Button "Browse" on the second tab is activated
 */
for (int i=0; i<listJB.size(); i++)
{
    if (e.getSource() == listJB.get(i))
    {
        FileFilter ff = new FileFilter(){
            public boolean accept(File f){
                if (f.isDirectory())
                    return true;
                else if (f.getName().endsWith(".rdf"))
                    return true;
                else if (f.getName().endsWith(".xml"))
                    return true;
                else if (f.getName().endsWith(".txt"))
                    return true;
                else if (f.getName().endsWith(".csv"))
                    return true;
                else if (f.getName().endsWith(".map"))
                    return true;
                /*else if (f.isFile())
                    return true;*/
                else
                    return false;
            }
            public String getDescription(){
                return "File .rdf, .xml, .txt, .csv";
            }
        };

        fc.setFileSelectionMode(JFileChooser.FILES_ONLY);
        fc.setDialogTitle("Select a mappings file");
        fc.setApproveButtonText("Validate");
        fc.setApproveButtonToolTipText("Validate the selected file.");
        fc.removeChoosableFileFilter(fc.getFileFilter());
        fc.setFileFilter(ff);
        if (fc.showOpenDialog(listJB.get(i)) == JFileChooser.APPROVE_OPTION)
        {
            listJTF.get(i).setForeground(Color.BLACK);
            listJTF.get(i).setText(fc.getSelectedFile().getAbsolutePath());
        }
    }
}

```

Figure 40 - Passage, par l'utilisateur, d'un fichier mappé (fichier contenant des correspondances)

La troisième section, présente toutes les mesures définies, calculables par le programme. Ces dernières peuvent être cochées par l'utilisateur et seront calculées, plus tard, pour chacun des datasets* vus plus haut.

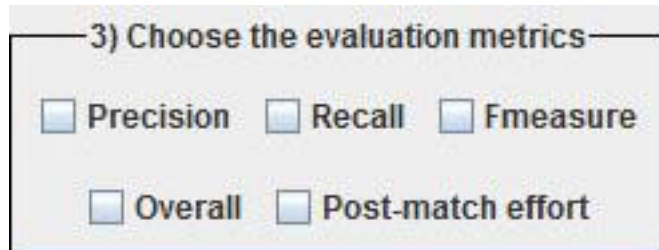


Figure 41 - Les mesures définies de la troisième section

```

if ((chckbxPrecision.isSelected()) && (JCBonlyI[0]==1))
{
    taw.append("Precision: " + cc.CalculatePercentagePrecision() + "%");
    dw.append(cc.CalculatePercentagePrecision());
}

if ((chckbxRecall.isSelected()) && (JCBonlyI[1]==1))
{
    taw.append("Recall: " + cc.CalculatePercentageRecall() + "%");
    dw.append(cc.CalculatePercentageRecall());
}

if ((chckbxFmeasure.isSelected()) && (JCBonlyI[2]==1))
{
    taw.append("F-measure: " + cc.CalculatePercentageFmeasure() + "%");
    dw.append(cc.CalculatePercentageFmeasure());
}

if ((chckbxOverall.isSelected()) && (JCBonlyI[3]==1))
{
    taw.append("Overall: " + cc.CalculatePercentageOverall() + "%");
    dw.append(cc.CalculatePercentageOverall());
}

if ((chckbxPostmatch.isSelected()) && (JCBonlyI[4]==1))
{
    taw.append("Post Match Effort: " + cc.CalculatePercentagePME() + "%");
    dw.append(cc.CalculatePercentagePME());
}

```

Figure 42 - Appels des calculs à effectuer pour chaque mesure sélectionnée

Cette section possède une petite particularité, celle de pouvoir préciser si les mesures cochées doivent être affichées sur le même graphique ou indépendamment les unes des autres.

Displays all metrics on the same plot:

☒ Yes
 ☐ No

Figure 43 - Particularité de la troisième section

Au niveau programmation cela se résume à un tableau contenant cinq cases d'entiers dont chaque case représente une mesure. Ces cases contiendront, à leur tour, un chiffre qui représente leur ordre de passage pour les calculs. C'est alors qu'à chaque calcul, ce tableau sera décrémenté de 1 pour ainsi prendre en compte les calculs restants.

Mesures	Precision	Recall	Fmeasure	Overall	Post-match
Ordre de passage	1	0	0	2	3

Figure 44 - Exemple de tableau contenant l'ordre de passage pour chaque mesure.

```
private int[] JCBonlyI = new int[5];
```

Figure 45 - Déclaration d'un tableau d'entiers de taille 5

```
for (int index=0; index<JCBonlyI.length; index++)
{
    JCBonlyI[index] = JCBonlyI[index] - 1;
}
```

Figure 46 - Décrémenter toutes les cases d'un tableau d'entiers

Enfin, la quatrième section, qui est plus de l'ordre du facultatif en tant qu'interaction avec l'utilisateur, permet, dans le cas de constructions de plusieurs graphiques, de les aligner dans une disposition soit verticale soit horizontale, sachant que par défaut, les graphiques seront disposés à la verticale.

```
/**
 * Construction of its Second flap
 */
scrollPane_5 = new JScrollPane();
splitPane_2.setRightComponent(scrollPane_5);

panel_3 = new JPanel();
panel_3.setLayout(new BoxLayout(panel_3, BoxLayout.Y_AXIS ));
scrollPane_5.setViewportViewView(panel_3);
```

Figure 47 - Disposition à la verticale par défaut des graphiques générés

```
else if (e.getSource() == rdbVertical)
{
    panel_3.setLayout(new BoxLayout(panel_3, BoxLayout.Y_AXIS ));
}
else if (e.getSource() == rdbHorizontal)
{
    panel_3.setLayout(new FlowLayout());
}
```

Figure 48 - Passage entre verticale et horizontale de l'alignement des graphiques générés

Une fois ce deuxième onglet rempli, l'utilisateur peut interagir avec son dernier composant qui est le bouton « Generate plots », permettant de générer le(s) graphique(s) en fonction des mesures demandées. Se faisant, l'application redirige l'utilisateur sur son troisième onglet.

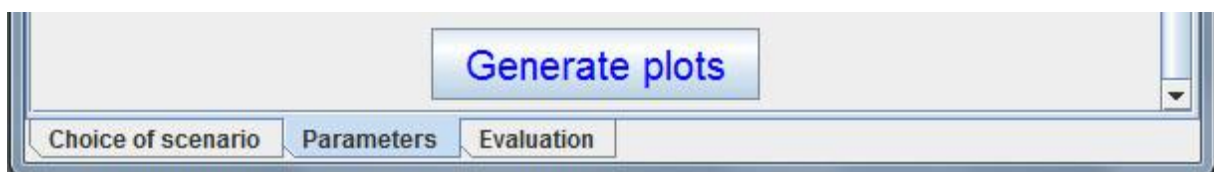


Figure 49 - Dernier composant du deuxième onglet

```
else if (e.getSource() == btnGeneratePlots)
{
    // if at least one JTextField is empty
    if (JTextFieldEmpty() != listJTF.size())
    {
        int i = JTextFieldEmpty();
        listJTF.get(i).setForeground(Color.RED);
        listJTF.get(i).setText("File path of matches is required.");
    }
    else // if no JTextField is empty
    {
        new Graph (filePath, resultsdirectoryname,
            countertool, listJTFtool,
            listJL, listJTF, listJCB,
            list, textArea_1, panel_3,
            chckbxPrecision, chckbxRecall, chckbxFmeasure, chckbxOverall, chckbxPostmatch,
            rdbYes);

        //Redirection on the third tab (value == 2)
        tabbedPane.setSelectedIndex(2);
    }
}
```

Figure 50 - Composant permettant la génération de graphiques

3. Affichages

Dans cet onglet, deux volets distincts : l'un contenant l'affichage des résultats en textuelle, l'autre celui des résultats en graphique.

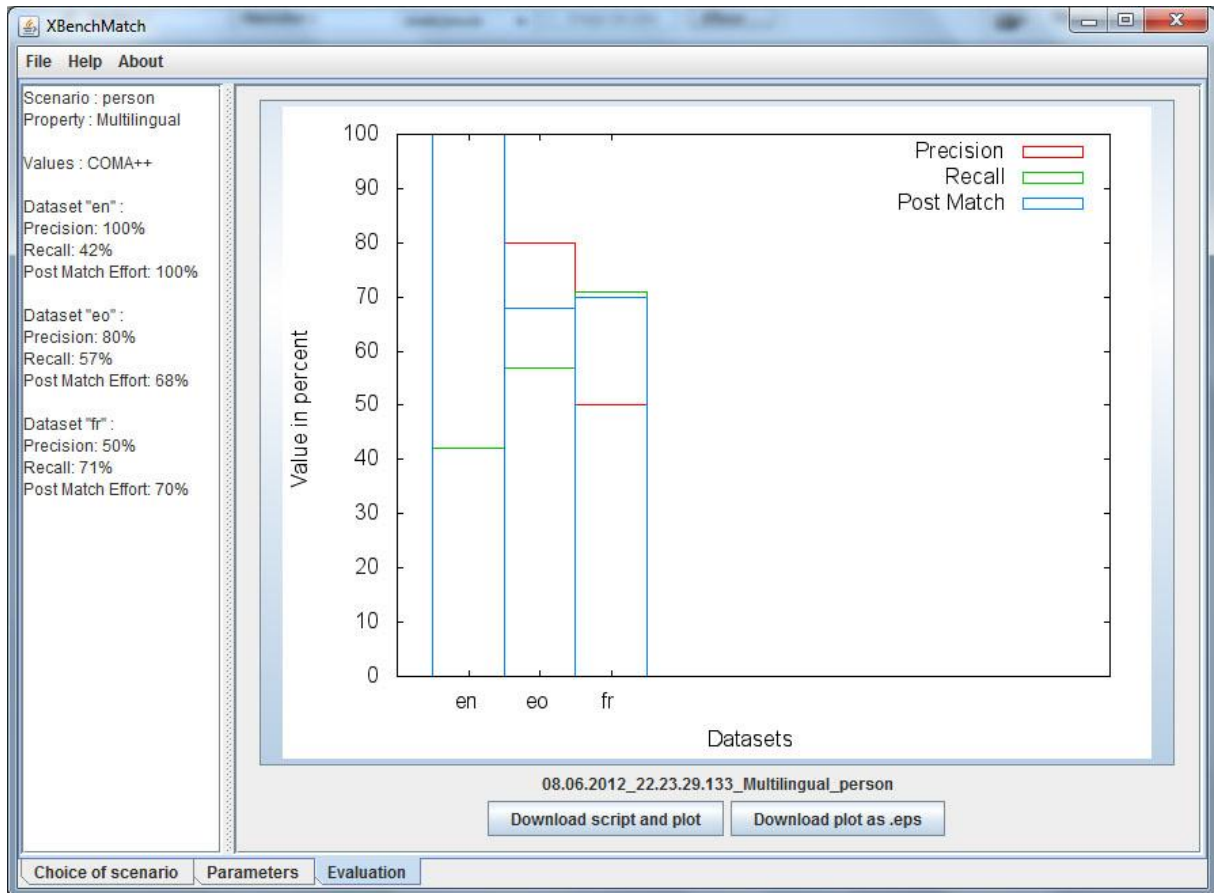


Figure 51 - Les deux volets du troisième onglet

La partie gauche se remplit grâce à la classe* `TextAreaReader`, qui va lire le contenu d'un fichier texte créé, suite aux calculs, et le transmettre au `JTextArea*`, contenu dans ce volet.

```
TextAreaWriter taw = new TextAreaWriter(resultsdirectoryname, filename);
```

Figure 52 - Création d'un objet de type `TextAreaWriter`

```
public class TextAreaWriter extends Writer {  
  
    private PrintWriter outfile;  
  
    public TextAreaWriter (String resultsdirectoryname, String filename)  
    {  
        super("./" + resultsdirectoryname + "/" + filename + "/" + filename + ".txt");  
        outfile = get_outfile();  
    }  
  
    public void append(String sentence)  
    {  
        outfile.println(sentence);  
    }  
  
    public void close()  
    {  
        outfile.close();//Method for finish the file construction.  
    }  
}
```

Figure 53 - Classe TextAreaWriter et son héritage sur la classe Writer

```
public class Writer {  
  
    private PrintWriter outfile;  
  
    public Writer (String file)  
    {  
        FileWriter fw;  
        try {  
            fw = new FileWriter (file);  
            BufferedWriter bw = new BufferedWriter (fw);  
            outfile = new PrintWriter (bw);  
        } catch (IOException e) {  
            // TODO Auto-generated catch block  
            e.printStackTrace();  
        }  
    }  
  
    public PrintWriter get_outfile ()  
    {  
        return outfile;  
    }  
}
```

Figure 54 - Classe Writer

```
taw.close();//Think to close else there is construction of an empty file.  
new TextAreaReader(resultsdirectoryname, filename, textArea_1);
```

Figure 55 - Fermeture de fichier d'un objet de type TextAreaWriter et appel de la classe TextAreaReader

```
public class TextAreaReader {  
    public TextAreaReader (String resultsdirectoryname, String filename, JTextArea textArea)  
    {  
        try {  
            textArea.setText(""); //for to empty the JTextArea  
  
            InputStream ips = new FileInputStream("./" + resultsdirectoryname + "/" + filename + "/" + filename + ".txt");  
            InputStreamReader ipsr=new InputStreamReader(ips);  
            BufferedReader br=new BufferedReader(ipsr);  
  
            String line;  
            while ((line = br.readLine())!=null)  
            { //We read line by line.  
                textArea.append(line + "\n");  
            }  
  
            br.close();  
        } catch (FileNotFoundException e) {  
            // TODO Auto-generated catch block  
            e.printStackTrace();  
        } catch (IOException e) {  
            // TODO Auto-generated catch block  
            e.printStackTrace();  
        }  
    }  
}
```

Figure 56 - Classe TextAreaReader

La partie droite, elle, contient un peu plus d'informations, avec notamment au minimum, un graphique comportant : un JLabel* donnant quelques précisions et deux boutons de téléchargements.

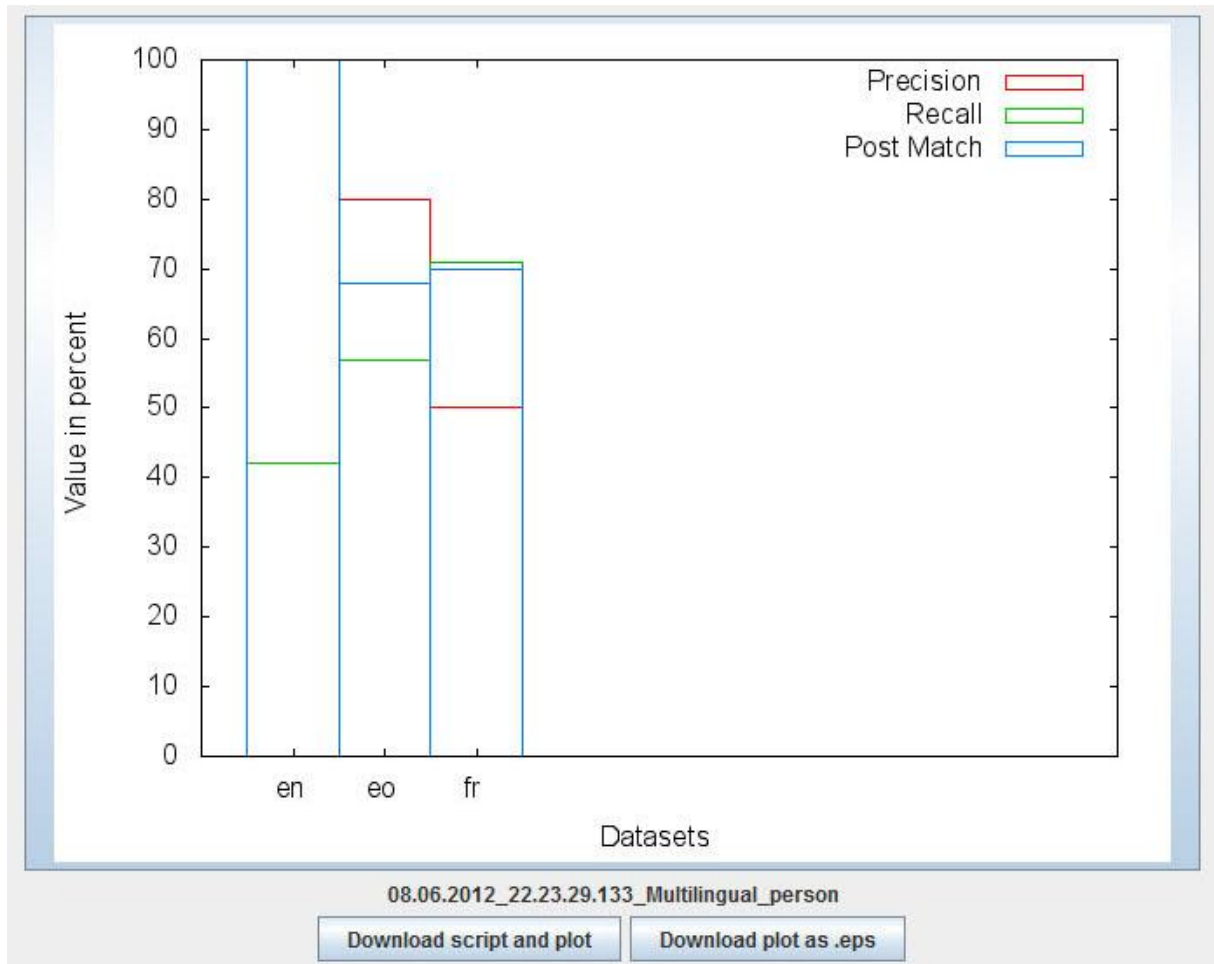


Figure 57 – Composants de la partie droite du troisième onglet

Plus haut dans ce document, il a été dit qu'une fonction établie permettait de récupérer l'image générée par le logiciel Gnuplot*. Et bien, c'est en l'utilisant qu'un gros bouton contenant l'image du graphique fait son apparition. Mais pourquoi, au niveau conception, avoir mis l'image dans un JButton* et non pas un JLabel ? La réponse est juste quelques paragraphes au-dessus. La lecture du fichier texte ! En effet, à chaque graphique correspond son propre fichier texte, et donc par le biais de manipulations de tableaux dynamiques*, le fichier correspondant peut être retrouvé et lu une nouvelle fois au clic de la souris. Ceci permettant à l'utilisateur de jeter un coup d'œil sur les valeurs ayant été calculées et apparaissant comme douteuses sur le graphique.

```
private List<String> ls = new ArrayList<String>();

private List<JButton> lb1 = new ArrayList<JButton>();
private List<JButton> lb2 = new ArrayList<JButton>();
private List<JButton> lb3 = new ArrayList<JButton>();
```

Figure 58 - Tableaux dynamiques de JButton et tableau dynamique de String

```
/*
 * d : day
 * M : month
 * y : year
 * H : hour
 * m : minute
 * s : second
 * S : millisecond
 * more informations: http://docs.oracle.com/javase/6/docs/api/java/text/SimpleDateFormat.html
 */
String date = new SimpleDateFormat ("dd.MM.yyyy_HH.mm.ss.SSS").format(new Date());
int i=0;
while (!listJCB.get(i).isSelected())
{
    i++;
}
filename = date + "_" +
    listJCB.get(i).getText().replace(" ", "-") + "_" +
    list.getSelectedValue().toString().replace(" ", "-");

ls.add(filename);
```

Figure 59 - Création du « filename » et son ajout dans un tableau dynamique

```
nbGraph++;
lb1.add(new JButton(gr.getImage(resultsdirectoryname, filename)));
lb1.get(nbGraph).addActionListener(this);
panel_16.add(lb1.get(nbGraph));
```

Figure 60 - Ajout, dans un tableau dynamique, d'un JButton comportant l'image d'un graphique

Où « gr » est un objet de type « Graphic » présent dans la couche modèles.

```
if (e.getSource() == lb1.get(i))
{
    new TextAreaReader(resultsdirectoryname, ls.get(i), textArea_1);
}
```

Figure 61 - Lecture du fichier texte correspondant au graphique cliqué

Ensuite, le JLabel*, se compose de la date et l'heure à laquelle le graphe a été généré, suivi par le nom de la propriété et le scénario* concernés. Ceci ayant été établi pour rappeler à l'utilisateur quel graphique a été créé en premier et quel graphique a été créé en dernier, formant ainsi une chronologie dans le cas de constructions de plusieurs graphiques.



Figure 62 - Libellé d'un graphe

```
JLabel lbl1 = new JLabel(filename);  
panel_17.add(lbl1);
```

Figure 63 - JLabel d'un graphique

Enfin, pour finir, les deux boutons de téléchargements du graphique permettent un appel à la classe* DownloadGraph dont la seule différence entre les deux, est de préciser, au moyen d'un booléen parmi les paramètres donnés, si l'utilisateur veut uniquement le fichier .eps (fichier contenant l'image du graphique en vectoriel) ou bien tous les fichiers nécessaires à la construction du graphe, à savoir :

- le .txt : les résultats textuels apparaissant à l'écran
- le .gp : le script* Gnuplot* utilisé
- le .data : le fichier de données allant de pair avec le script Gnuplot
- le .png : l'image standard du graphique
- le .eps : l'image vectorielle du graphique



Figure 64 - Les deux boutons de téléchargement d'un graphe

```

else if (e.getSource() == lb2.get(i))
{
    FileFilter ff = new FileFilter(){
        public boolean accept(File f){
            if(f.isDirectory()) return true;
            else return false;
        }
        public String getDescription(){
            return "Folder";
        }
    };
    fc = new JFileChooser();
    fc.setMultiSelectionEnabled(false);
    fc.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
    fc.setDialogTitle("Select a destination folder");
    fc.setApproveButtonText("Download");
    fc.setApproveButtonToolTipText("To put the files of graph in this folder.");
    fc.removeChoosableFileFilter(fc.getFileFilter());
    fc.setFileFilter(ff);
    if (fc.showOpenDialog(lb2.get(i)) == JFileChooser.APPROVE_OPTION)
    {
        new DownloadGraph(resultsdirectoryname, ls.get(i), fc.getSelectedFile().getAbsolutePath(), false,
            countertool, listJTFtool);
        //false because we want all.
    }
}

```

Figure 65 - Au clic sur un bouton "Download script and plot"

```

else if (e.getSource() == lb3.get(i))
{
    new DownloadGraph(resultsdirectoryname, ls.get(i), fc.getSelectedFile().getAbsolutePath(), true,
        countertool, listJTFtool);
    //true because we want to only the file .eps
}

```

Figure 66 - Au clic sur un bouton "Download plot as .eps"

La classe DownloadGraph est disponible en annexe.

4. Conclusion sur l'interface graphique

L'interface graphique est donc assez complète pour pouvoir exploiter au maximum la programmation en fond, et obtenir ainsi des résultats convaincants.

IV. Résultat et discussion

A. LOGICIEL OBTENU

Une fois lancé, l'utilisateur a accès à tous les onglets de l'application. Néanmoins, l'onglet sur lequel le logiciel démarre est le premier, nommé « Choice of scenario ».

L'utilisateur peut, bien évidemment, aller voir les autres onglets mais leur contenu sera soit vide, soit comportera des erreurs.

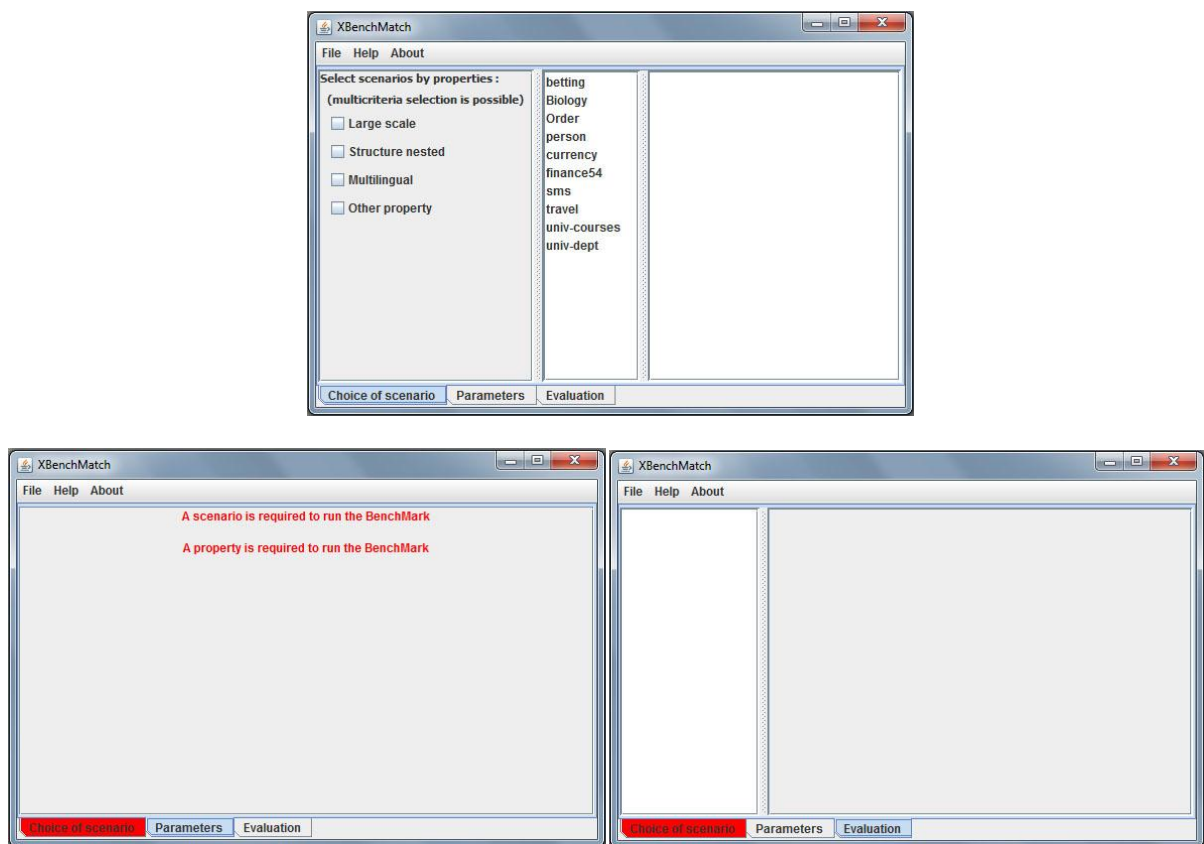
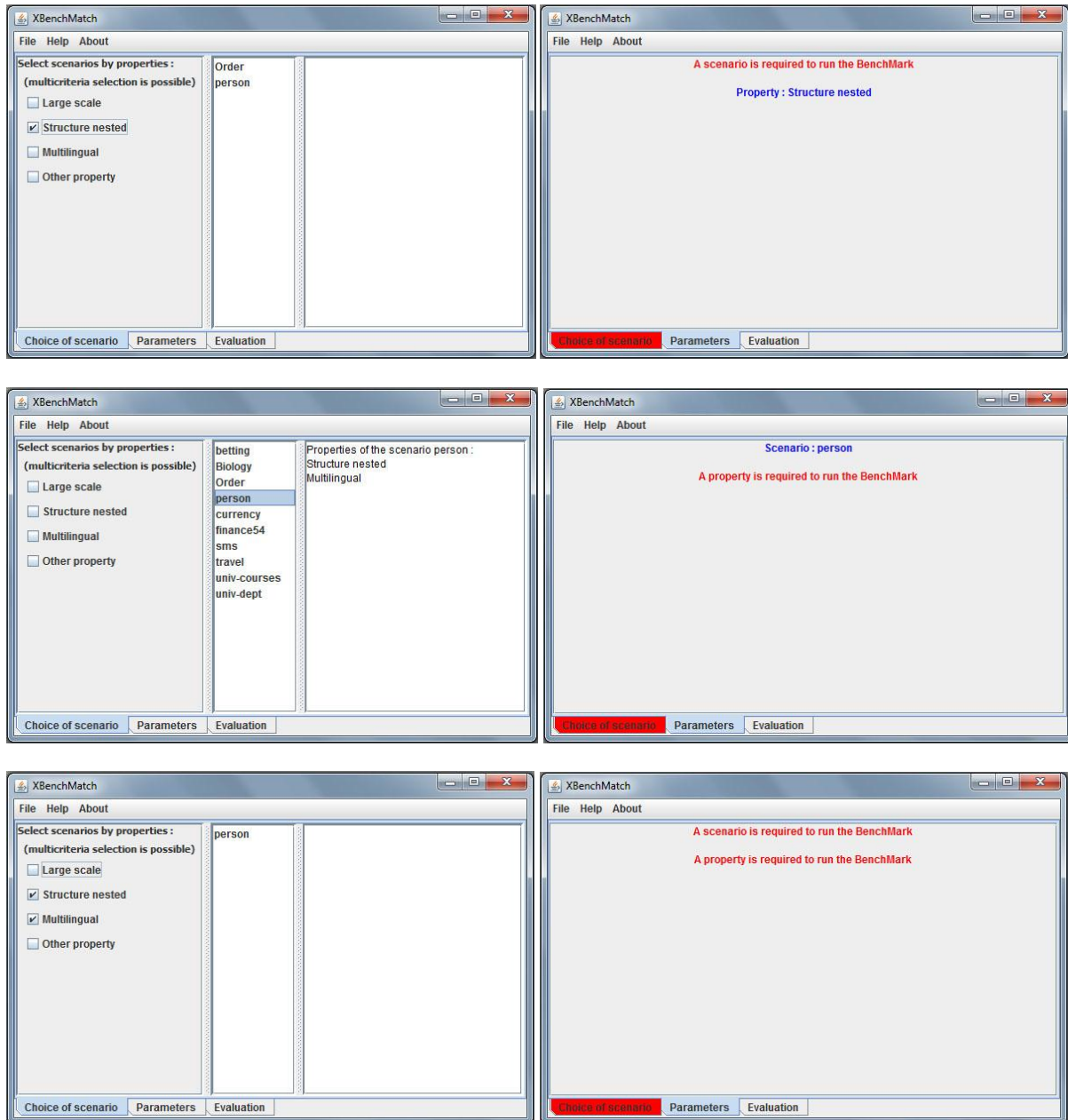


Figure 67 - Contenu des trois onglets au démarrage.

Pour avancer, l'utilisateur doit choisir une propriété et un scénario*, sachant que si un des deux est manquant, ou incorrect, le logiciel, lui, ne manquera pas de le lui signaler lors de son passage du premier onglet vers le second. Dès lors, l'utilisateur devra corriger son erreur en faisant marche arrière.

Il est aussi intéressant de noter que les informations, qui apparaissent lors de la sélection d'un scénario*, peuvent aider l'utilisateur à mieux appréhender tous les scénarios et propriétés que possède le logiciel.



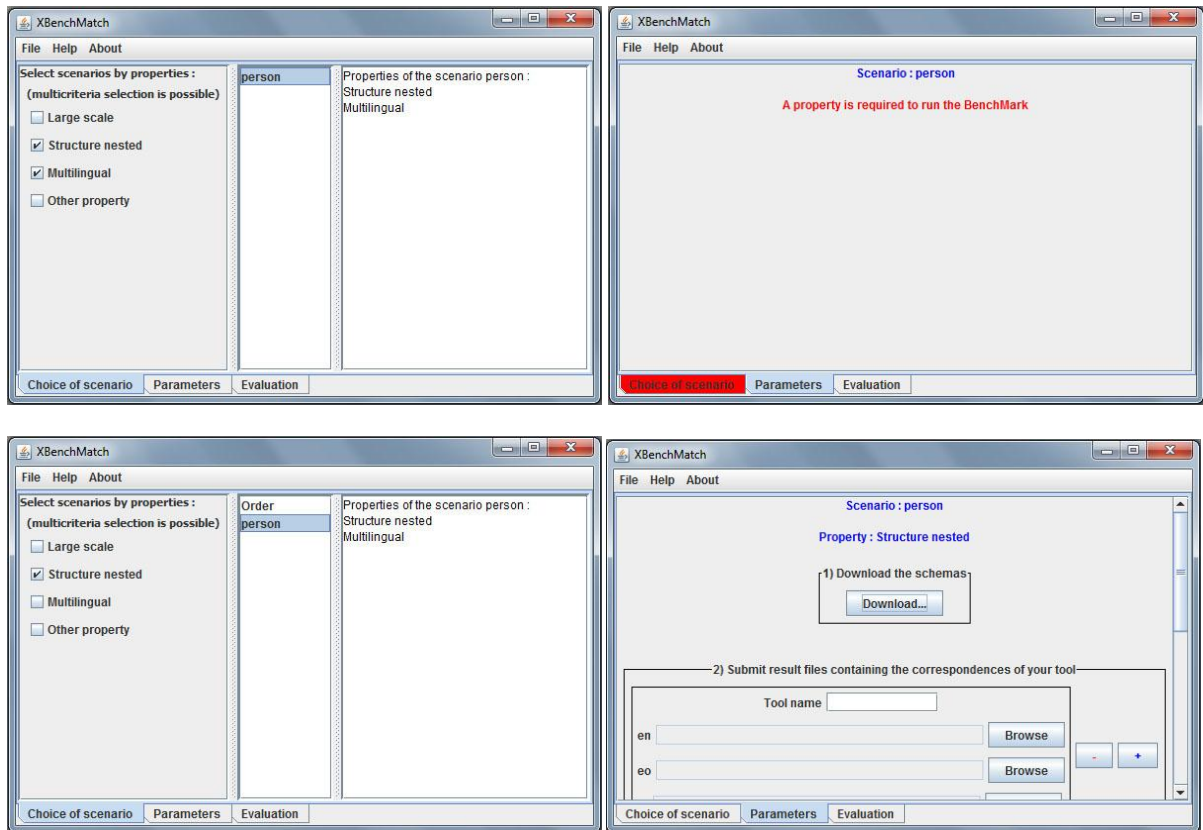


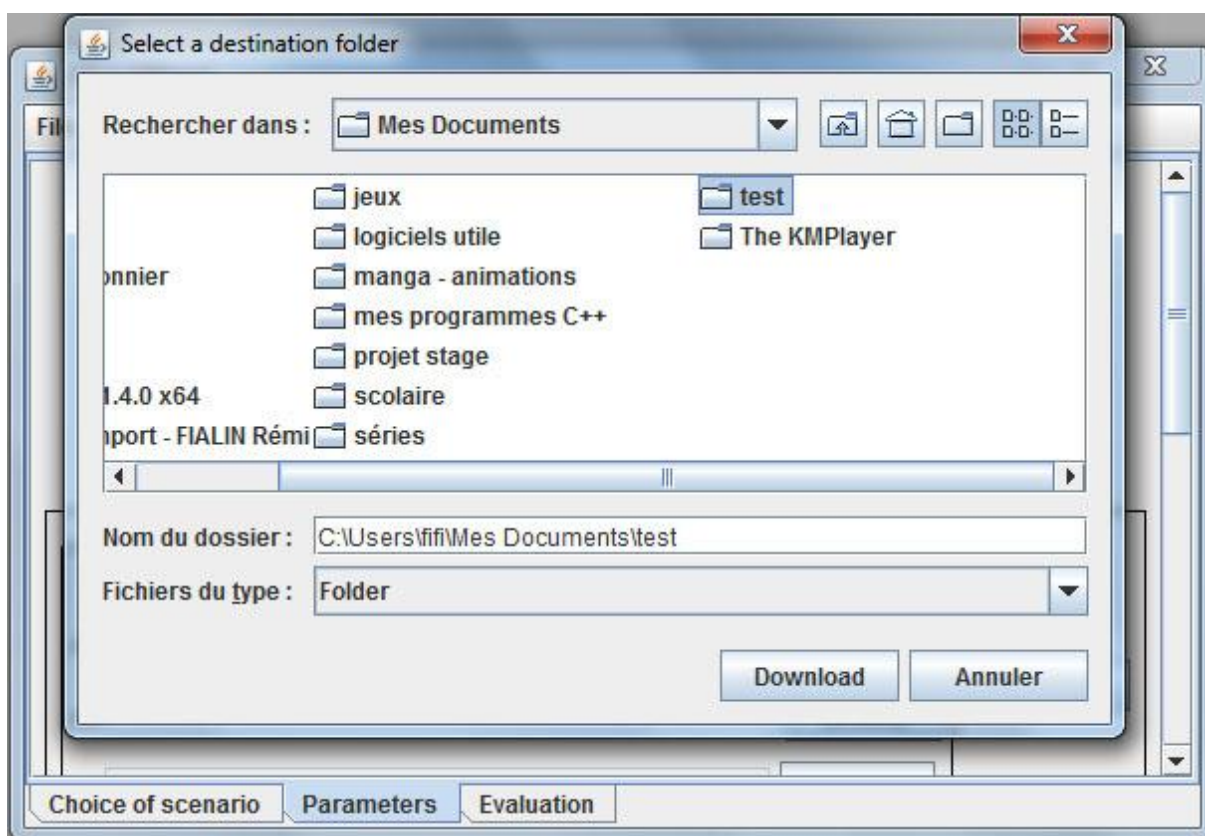
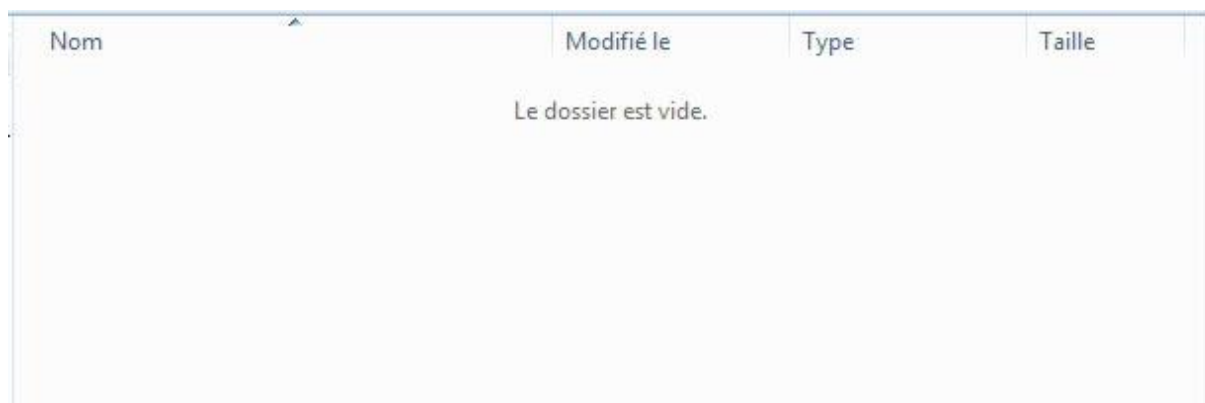
Figure 68 - Contenu du deuxième onglet en fonction du premier.

Une fois sur le deuxième onglet, un rappel du scénario* et de la propriété, finalement sélectionnés, peut être visualisé.



Figure 69 - Rappel du scénario et de la propriété sélectionnés

Avant de pouvoir effectuer la suite, l'utilisateur doit tout d'abord télécharger les schémas concernant le scénario pour cette propriété à l'aide du bouton « Download... » et indiquer le dossier dans lequel ils doivent être déposés. Ici, le dossier choisi s'appelle « test ».



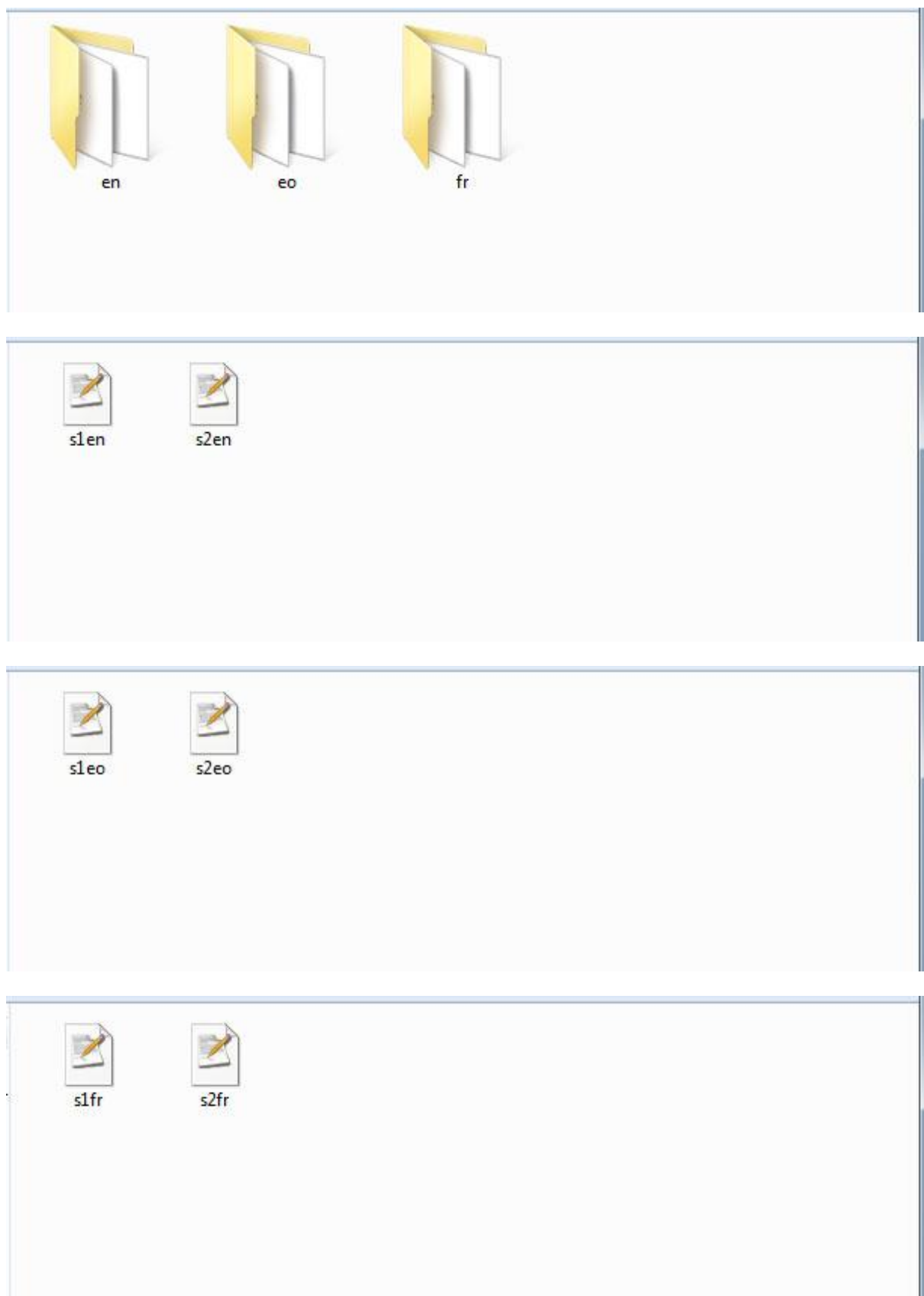


Figure 70 - Chronologie du téléchargement des datasets d'un scénario

Une fois les schémas acquis, il doit pouvoir les utiliser sous son/ses « outil(s) de correspondances »* (matcher tool(s)) pour récupérer à chaque fois un fichier contenant des correspondances (fichier mappé*) entre ces deux schémas.

Puis, retour à notre application. Le deuxième encadré contient une partie pour donner les fichiers mappé construits, pour chaque dataset*, par l'outil de matching*, dont le nom sera précisé dans un petit cadre prévu à cet effet. Il contient aussi un bouton « + » et un bouton « - » pour donner les fichiers mappé par d'autres outils. Ces boutons ne seront pas utilisés ici car un seul outil sera actif. Si jamais, l'utilisateur ne donne pas un fichier mappé et qu'un champ libre est ainsi apparent, s'il veut générer un graphique le logiciel l'avertira alors d'une erreur.

2) Submit result files containing the correspondences of your tool

Tool name	
en	
eo	
fr	

Browse Browse Browse

- +

Select a mappings file

Rechercher dans : rsc

- s1s2enCOMA++.map
- s1s2eoCOMA++.map
- s1s2frCOMA++.map

Nom de fichier :

Fichiers du type : File .rdf, .xml, .txt, .csv

Validate Annuler

2) Submit result files containing the correspondences of your tool

Tool name

en

eo

fr

-

Tool name

en

eo

fr

-

Tool name

en

eo

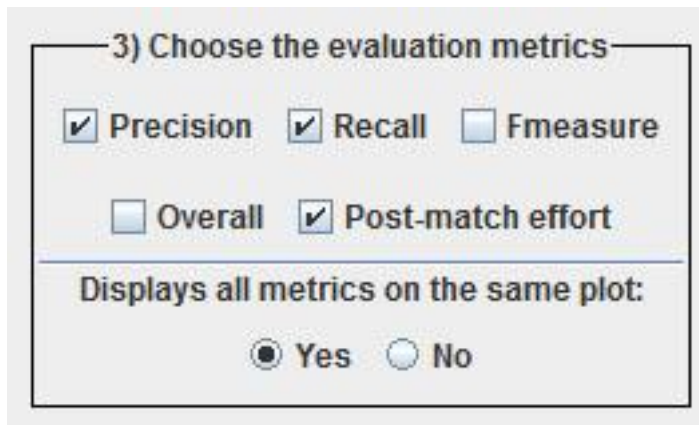
fr

-

Figure 71 - Différentes apparences du second encadré

Dans le troisième encadré, les mesures définies qui seront calculées doivent être sélectionnées, en précisant si oui ou non elles doivent être représentées sur le même graphique.

Si aucune case n'est cochée, alors le logiciel fait une redirection automatique sur le troisième onglet sans effectuer un seul calcul.



3) Choose the evaluation metrics

☒ Precision ☒ Recall ☐ Fmeasure

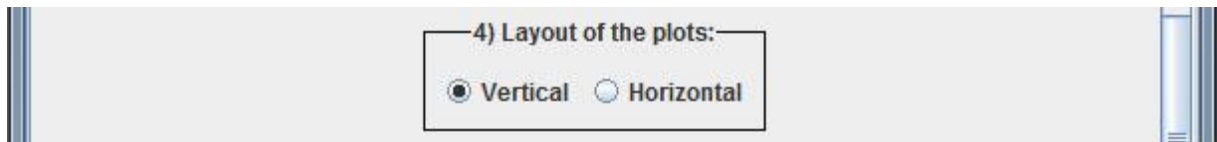
☐ Overall ☒ Post-match effort

Displays all metrics on the same plot:

☒ Yes ☐ No

Figure 72 - "Choose the evaluation metrics"

L'intérieur du quatrième encadré étant déjà sélectionné par défaut, il n'est pas très important de s'attarder dessus. Néanmoins, il est bon de savoir que chaque clic, sur l'un des deux boutons présents, rafraichit le contenu du troisième onglet en prenant en compte la nouvelle option. Ainsi, l'utilisateur peut changer l'orientation de la liste des graphiques créés sans pour autant relancer un « Generate plots ».

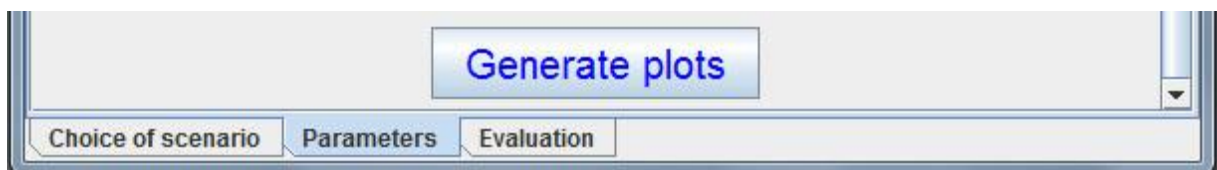


4) Layout of the plots:

☒ Vertical ☐ Horizontal

Figure 73 - Alignement des graphiques du troisième onglet

Une fois toutes ces étapes passées, et finalement sans erreur, un vrai clic sur le dernier bouton de cet onglet est accordé.



Generate plots

Choice of scenario Parameters Evaluation

Figure 74 - Bouton de validation du contenu du second onglet

Ainsi, les calculs sont lancés et l'utilisateur est redirigé automatiquement vers le troisième et dernier onglet de cette application.

L'onglet, alors atteint, permet l'exploitation des résultats autant en textuelle, sur une partie gauche, qu'en graphique, sur une partie droite.

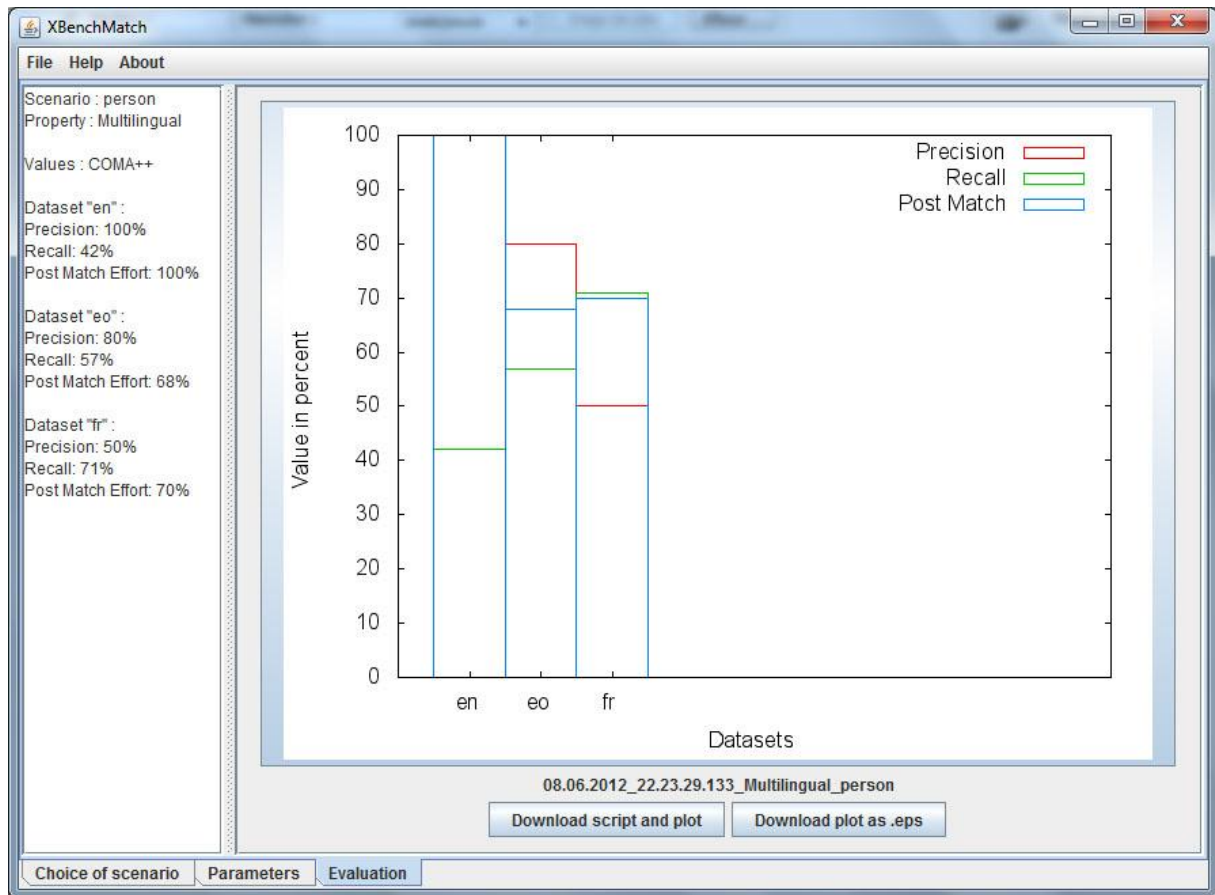
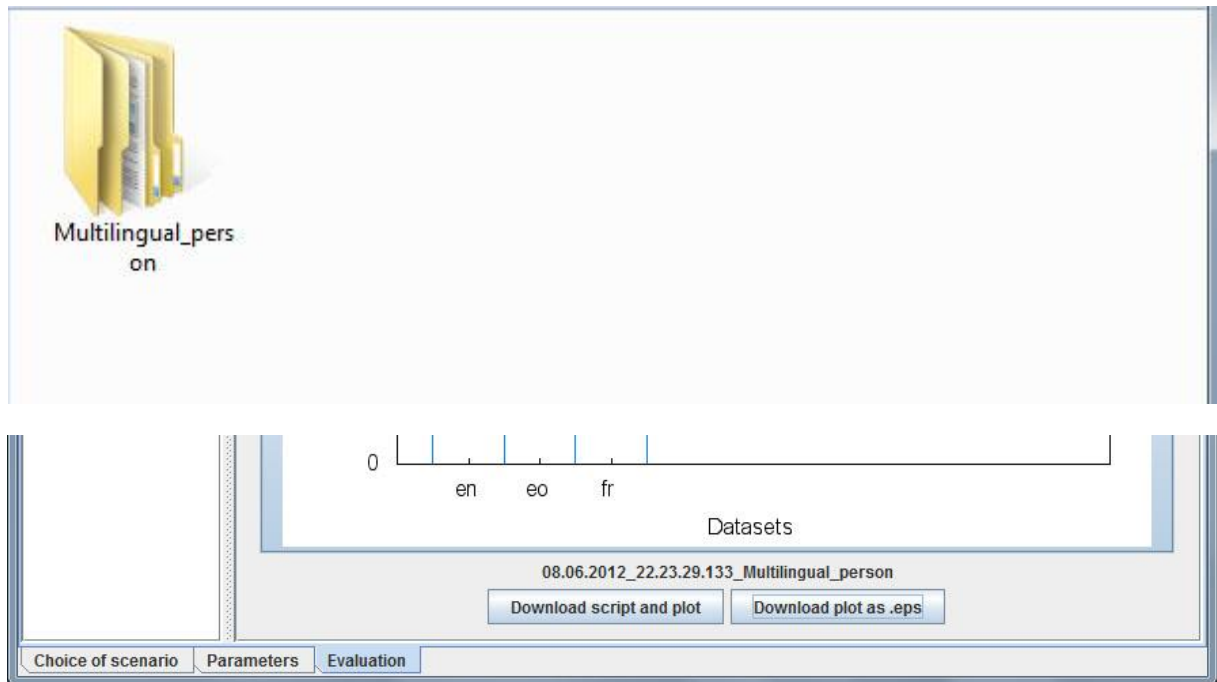


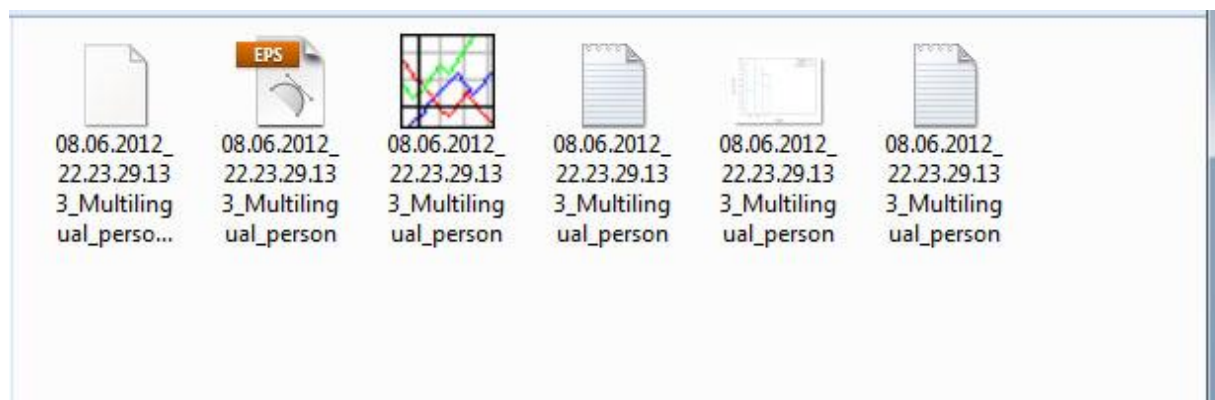
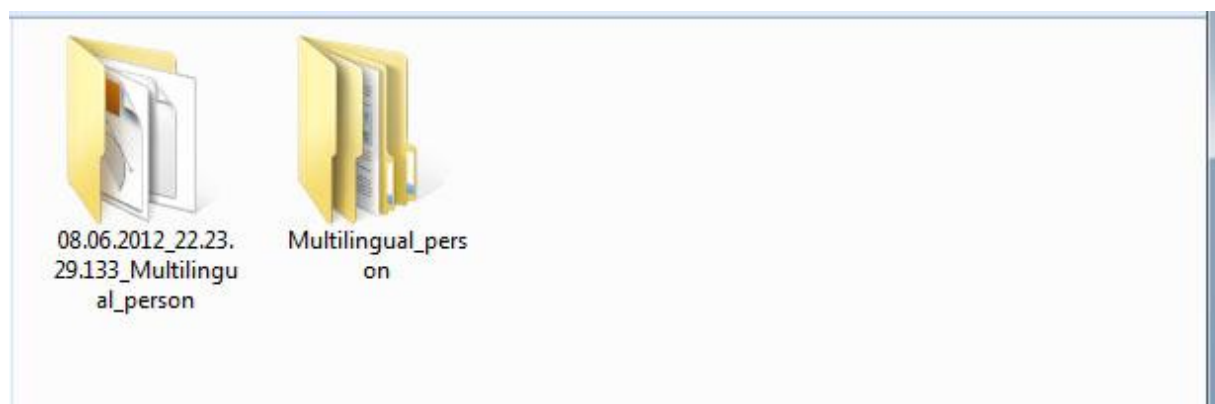
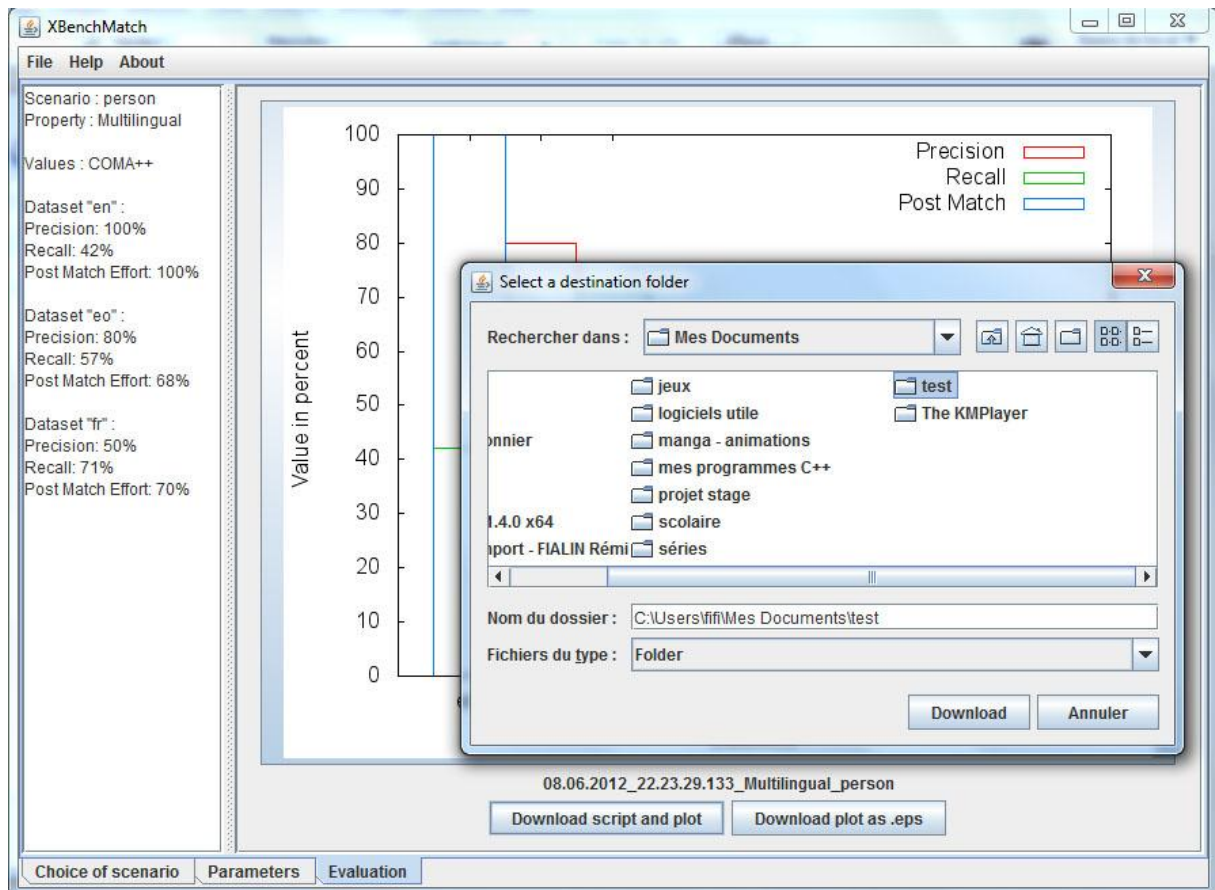
Figure 75 – Une des multiples apparences du troisième onglet

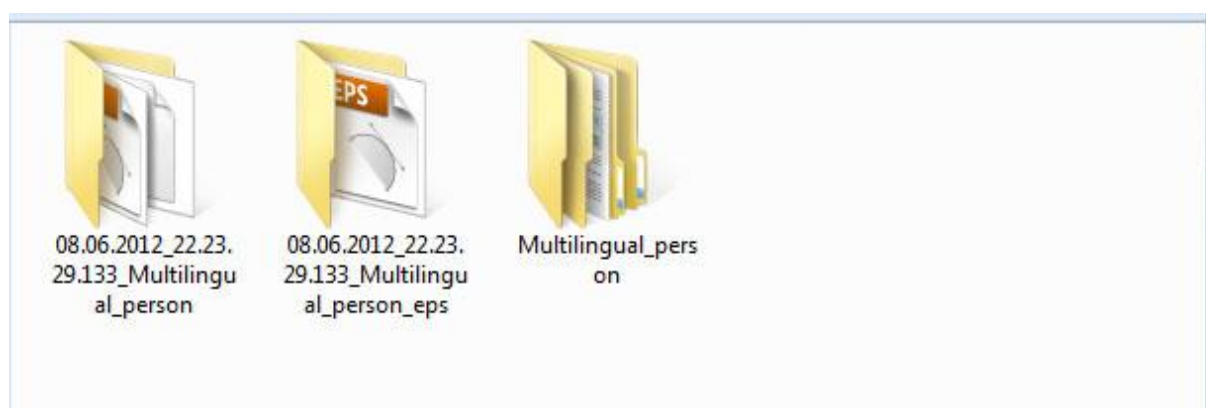
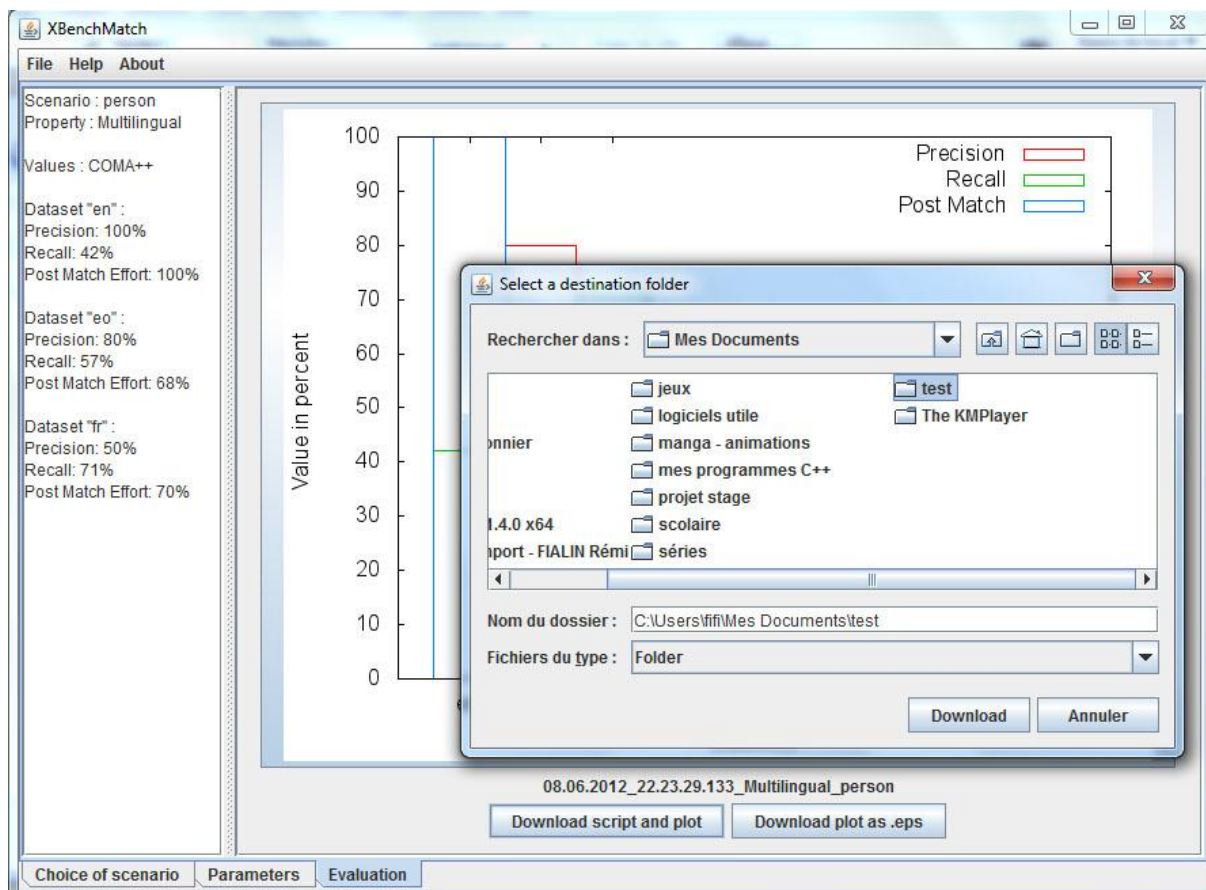
Ces deux parties contiennent donc, chacune sous différentes formes, le nom de la propriété et scénario* sélectionnés, le nom des différents datasets* évalués, les mesures calculées et leurs valeurs en pourcentage. Ce qui peut les différencier est le fait que l'affichage textuel indique le nom de l'outil utilisé et l'affichage graphique, la date et l'heure de la génération du graphe.

S'il y a plusieurs graphiques présents dans la partie de droite, alors l'utilisateur peut cliquer sur chacun de ces graphes pour voir apparaître ses caractéristiques propres dans la partie de gauche. Dans ce cas-ci, l'utilisateur a généré toutes les mesures sur le même graphique, donc il fera apparaître en permanence la même description.

Par la suite, l'utilisateur peut finir son utilisation du logiciel par le téléchargement de graphes. Cette dernière action peut être faite grâce aux deux boutons placés en dessous de chacun des graphiques créés. Il peut alors télécharger tous les fichiers qui ont été utilisés pour et par le graphe ou bien uniquement le dessin du graphique en vectoriel. Ou pourquoi pas, faire les deux.







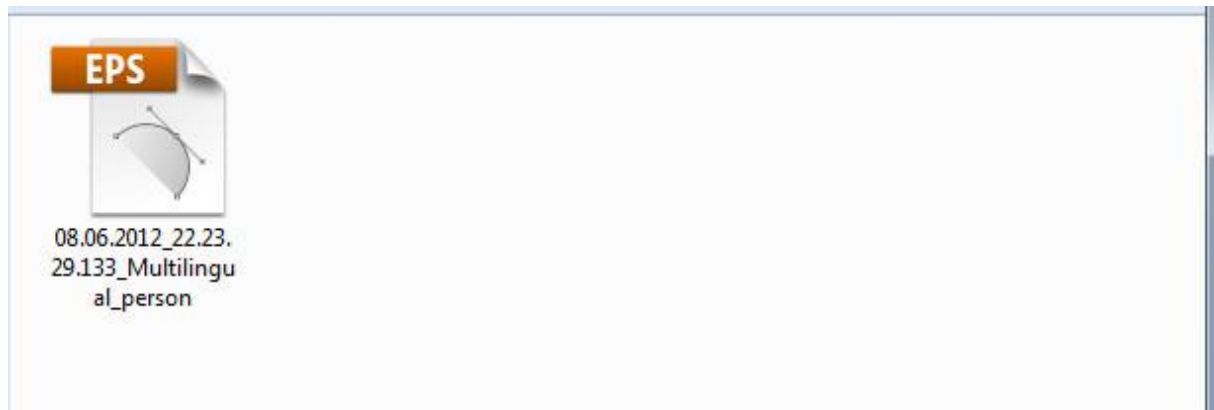


Figure 76 - Chronologie de téléchargement de graphe

A partir de là, l'utilisateur a tout ce dont il a besoin : il peut quitter l'application.

Rapide exemple d'une autre « histoire » :

2) Submit result files containing the correspondences of your tool

Tool name		COMA++
en	rs\workspace\XBenchMatch\resources\1s2enexpert.map	Browse
eo	rs\workspace\XBenchMatch\resources\1s2eomauvais.map	Browse
fr	rs\workspace\XBenchMatch\resources\1s2frparfait.map	Browse
Tool name		YAM
en	rs\workspace\XBenchMatch\resources\1s2enmauvais.map	Browse
eo	rs\workspace\XBenchMatch\resources\1s2eomauvais.map	Browse
fr	rs\workspace\XBenchMatch\resources\1s2frexpert.map	Browse

- +

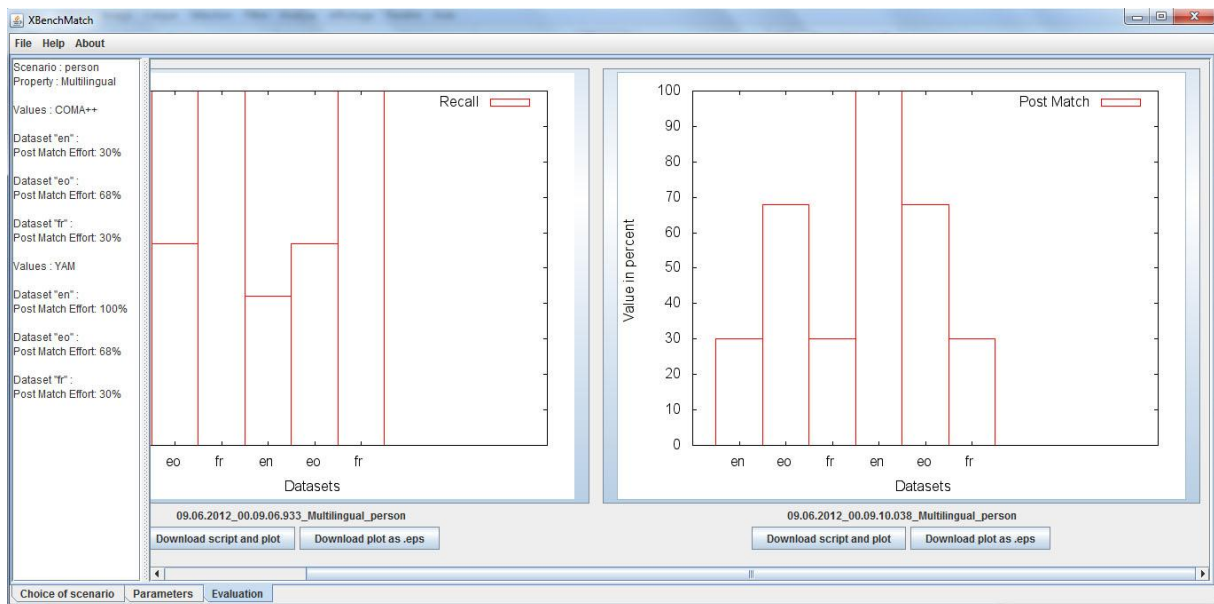
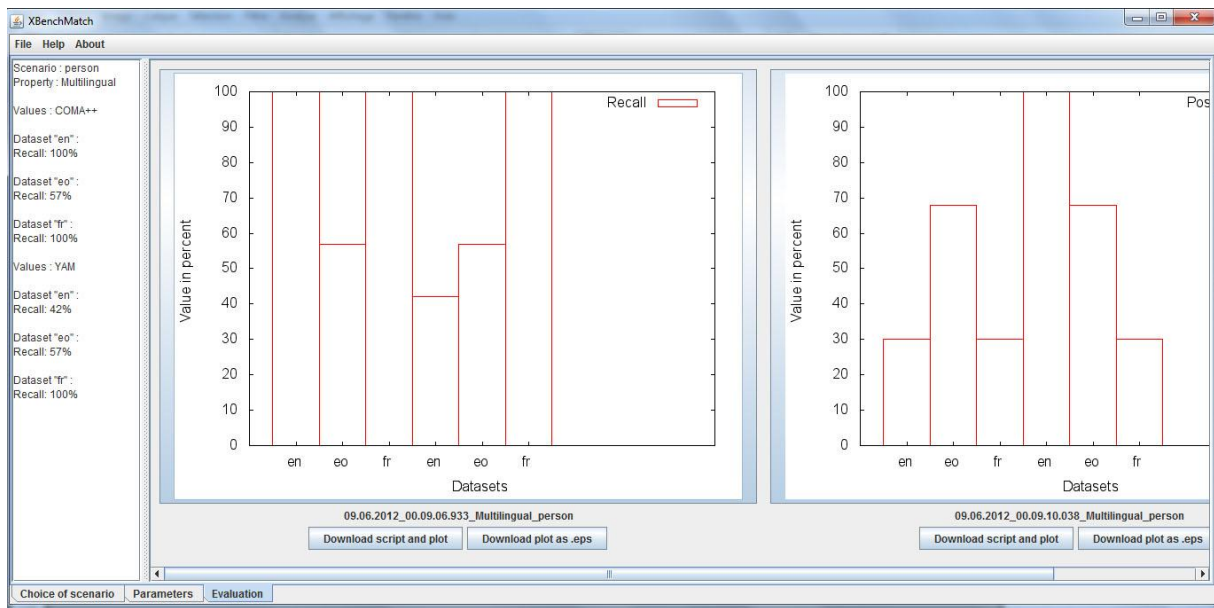


Figure 77 - Bonus: Evaluation avec comparaison d'outils

B. DISCUSSION

Suite à la production de cette application, quelques petites remarques peuvent être soulevées.

1. Critique des erreurs/faillles

Dans le deuxième onglet, le champ « tool name » de son second encadré n'est pas soumis à vérifications d'erreurs.

Toujours dans le même cadre, les erreurs sur les champs pour les différents datasets* pourraient être mieux gérées, car actuellement si tous les champs, ou au moins un, sont remplis par la phrase d'erreur, alors un graphique peut quand même être généré donnant pour valeur aux mesures du ou des datasets non présents, la valeur 0.

2) Submit result files containing the correspondences of your tool

	Tool name	
en	Users\ffil\workspace\XBenchMatch\rscls1s2enCOMA++.map	Browse
eo	File path of matches is required.	Browse
fr	Users\ffil\workspace\XBenchMatch\rscls1s2frCOMA++.map	Browse

- +

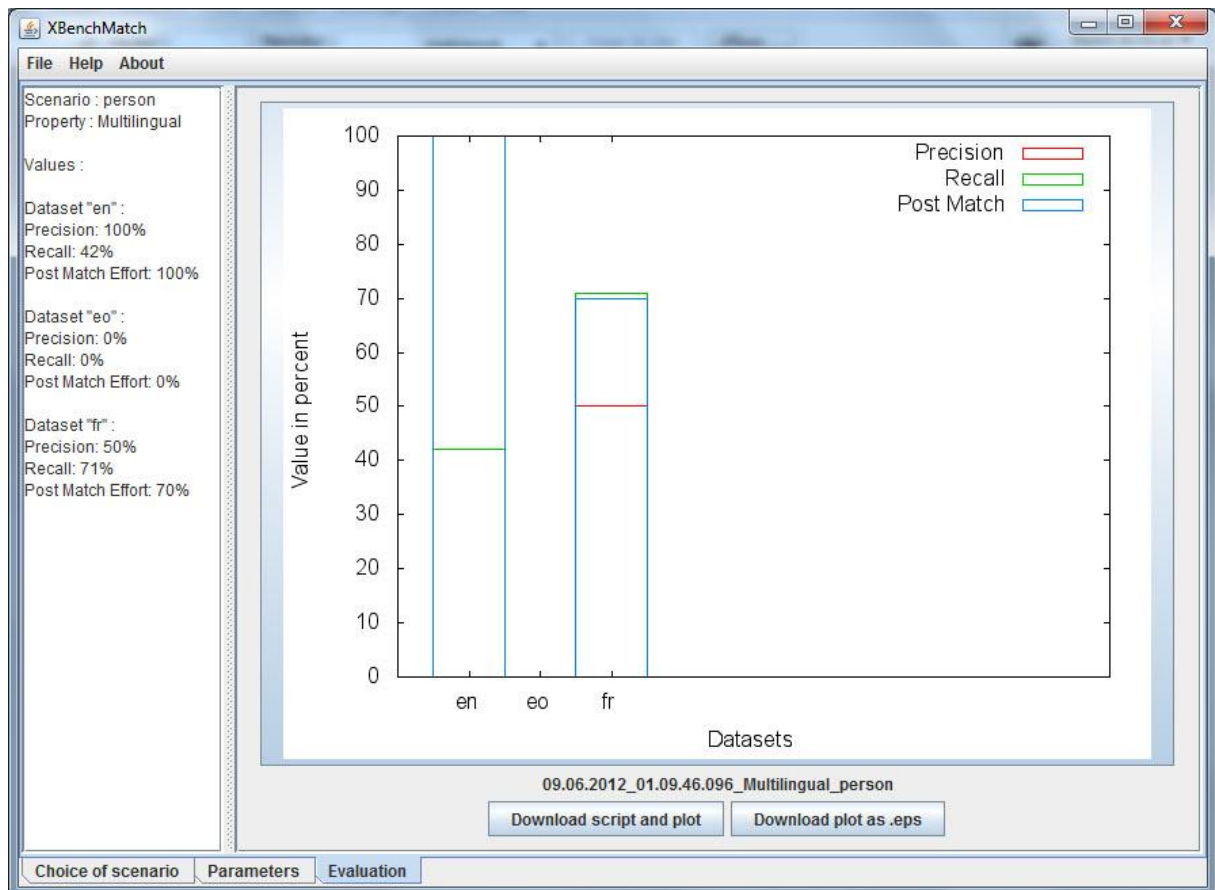


Figure 78 - Réaction en chaîne d'une erreur dans le champ d'un dataset

2. Améliorations possibles

Certaines personnes auront peut-être remarqué la présence d'une barre de menu comportant les boutons « File », « Help » et « About ». Ce document n'y fait en rien référence puisque ces « options » n'ont pas encore été implémentées.



Figure 79 - Barre de menu

Au point de vue esthétique, les différents encadrés du deuxième onglet pourraient tous avoir la même proportion.

Toujours dans le même onglet, un message d'erreur pour prévenir l'utilisateur qu'il n'a pas coché de mesures serait bien accueilli, car pour le moment, lorsque l'utilisateur clique pour générer un graphique alors qu'aucune mesure n'est cochée, la redirection est effectuée mais rien ne se passe, l'utilisateur peut alors se demander où il en est dans le déroulement.

V. Conclusion

En établissant les premières idées de conception et possédant ainsi un minimum de liberté d'action au départ, ce projet, apparaissant ardu mais intéressant, a suscité pleinement mon enthousiasme.

Ainsi le cahier des charges a été respecté et rempli à plus du tiers.

A. ORGANISATION DU TRAVAIL

L'organisation du travail a été assez simple au final puisque guidé par les réunions produites entre les tuteurs et moi.

Dans l'ordre établi, une réunion a lieu, des réflexions y sont faites, des améliorations données, et des objectifs construits.

Je devais donc mettre sous forme visuel les réflexions et faire les améliorations nécessaires durant un ou deux jours, voire au maximum trois. Puis employer le temps restant, jusqu'à la prochaine réunion, à mettre en place les objectifs donnés.

B. BILAN DU PROJET

Ce projet m'a permis de prendre confiance en moi dans le sens où celui-ci devait être programmé en Java* et que ce dernier n'était pas mon point fort. Cela étant dit, je suis très fier de ce que j'ai pu accomplir jusqu'ici, grâce aux excellentes bases que m'ont fourni les différents professeurs que j'ai pu avoir et sans qui je n'aurais su m'adapter.

Aussi, les réunions avec mes tuteurs, m'ont permis d'appréhender l'approche de dates buttoir qui m'ont aidé à faire la part des choses entre panique, stress, calme, et sérénité.

Cette période de stage m'a appris à me réfréner quant à la programmation, car pour mener à bien un projet, les phases d'analyse et de conception sont très importantes et doivent être proportionnellement étudiées au temps que l'on gagne par la suite. C'est par ses erreurs que l'on apprend, je sors donc plus fort de l'enseignement que j'ai pu en tirer.

Ensuite, je suis content d'avoir pu découvrir l'univers d'un laboratoire, au final assez différent de ce que je m'imaginais.

Enfin, ce projet, de par son interface graphique, m'a conforté dans l'idée que l'informatique « visuelle » est bien la voie que je devais suivre.

VI. Annexes

A. CLASSE PARSEXML

```

package controllers;

import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import javax.swing.JCheckBox;
import javax.swing.JList;

import org.jdom.Element;
import org.jdom.JDOMException;
import org.jdom.input.SAXBuilder;

public class ParserXML {

    private static String propertiesfile = "properties.xml";

    /**
     * Function which go to create our CheckBox of first tab.
     * @param filePath
     * @param listJCB
     * @param propertieslist
     */
    public void ParserPropertiesFile_getCheckBox(String filePath,
        List<JCheckBox> listJCB, List<ArrayList<String>> propertieslist)
    {
        File f = new File(filePath);
        String[] mylist = f.list();//we get the list of folders
        for (int i=0; i<mylist.length; i++)//for each folder
        {
            if (!mylist[i].equalsIgnoreCase(".svn"))
            {
                File f2 = new File(f + "/" + mylist[i]);
                String[] mylist2 = f2.list();//we get the list of
                folders

                for (int j=0; j<mylist2.length; j++)
                {
                    if (!mylist2[j].equalsIgnoreCase(".svn"))
                    {
                        SAXBuilder sxb = new SAXBuilder();
                        org.jdom.Document document;
                        try {
                            //we read its file of properties
                            document = sxb.build(new File(f2 +
                            "/" + mylist2[j] + "/" + propertiesfile));
                        } catch (JDOMException e) {
                            e.printStackTrace();
                        }
                        Element root =
                        document.getRootElement();
                        List criterionList =
                        root.getChildren("criterion");
                    }
                }
            }
        }
    }
}

```

```

criterionList.iterator();

        Iterator it =
        while(it.hasNext())
        {
            Element currentElement =

            if
            (currentElement.getChild("name") != null) //"name" is the regular expression of
            these files
            {
                /*
                * This block avoids the
                duplications in our CheckBox

                */
                int k=0;
                while
                ((k<listJCB.size()) &&
                (!listJCB.get(k).getText().equalsIgnoreCase(currentElement.getChild("name").getTex
                t()))
                {
                    k++;
                }
                if (k ==
                listJCB.size())//this property doesn't exist yet
                {
                    listJCB.add(new
                    JCheckBox(currentElement.getChild("name").getText()));//we create its CheckBox

                    propertieslist.add(new ArrayList<String>());//and its cell "property"

                    propertieslist.get(k).add(currentElement.getChild("name").getText());//we
                    add the property

                    propertieslist.get(k).add(mylist2[j]);//and we add also the name of folder
                    which have this property

                    k++;
                }
                else//this property
                exist already
                {
                    int m = 0;
                    while
                    (!propertieslist.get(m).get(0).equalsIgnoreCase(currentElement.getChild("name").ge
                    tText()))//we search
                    {
                        m++;
                    }
                    //we have the
                    property and we add another name of folder which have this property
                    if
                    (!propertieslist.get(m).contains(mylist2[j]))
                    {
                        propertieslist.get(m).add(mylist2[j]);
                    }
                }
            }
        }
    }

```

```

    } catch (JDOMException e1) {
        // TODO Auto-generated catch block
        e1.printStackTrace();
    } catch (IOException e1) {
        // TODO Auto-generated catch block
        e1.printStackTrace();
    }
}

}

}

}

/**
 * Function which helps to write the properties/criteria
 * of current scenario in the first tab.
 * @param filePath
 * @param propertieslist
 * @param list
 * @return
 */
public String ParserPropertiesFile_getcriteria(String filePath,
List<ArrayList<String>> propertieslist, JList list)
{
    SAXBuilder sxb = new SAXBuilder();
    org.jdom.Document document;
    String lines="";

    /*
     * We search the hut which contains the file name selected.
     */
    int i=0;
    while (!propertieslist.get(i).contains(list.getSelectedValue()))
    {
        i++;
        if (i==propertieslist.size())//if no hit contains the file name
selected
        {
            return ("No data");
        }
    }

    try {//the get(0) allows to get the name of the property.
        document = sxb.build(new File(filePath + "/" +
propertieslist.get(i).get(0) + "/" + list.getSelectedValue() + "/" +
propertiesfile));
        Element root = document.getRootElement();
        List criterionList = root.getChildren("criterion");
        Iterator j = criterionList.iterator();
        lines = "Properties of the scenario " + list.getSelectedValue()
+ " :\n";
        while(j.hasNext())
        {
            Element currentElement = (Element)j.next();
            if (currentElement.getChild("name") != null)//"name" is
the regular expression of these files
            {

```

```
        lines = lines +
currentElement.getChild("name").getText() + "\n";
        /*
allows to get the property which is between the two tags <name>
        */
    }
}
} catch (JDOMException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
} catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
return lines;
}
}
```

B. CLASSE FILTER

```

package controllers;

import java.io.File;
import java.util.ArrayList;
import java.util.List;

import javax.swing.DefaultListModel;
import javax.swing.JCheckBox;

public class Filter {

    /**
     * Function that gives the list of scenarios
     * from the result of crossing different properties checked.
     * @param filePath
     * @param model
     * @param listJCB
     * @param propertieslist
     */
    public Filter(String filePath, DefaultListModel model, List<JCheckBox>
listJCB, List<ArrayList<String>> propertieslist)
    {
        model.removeAllElements();

        int counter=0;
        List<String> mytableau = new ArrayList<String>();
        /*
         * we check each CheckBox
         */
        for (int i=0; i<listJCB.size(); i++)
        {
            if (listJCB.get(i).isSelected())
            {
                counter++;
                //we search the folders (scenarios) list which have this
property
                int k = 0;
                while
(!propertieslist.get(k).contains(listJCB.get(i).getText()))
                {
                    k++;
                }
                /*
                 * counter==1 because perhaps that mytableau can be
empty while counter==3,
                 * in this case we don't want the properties list.
                 */
                if ((mytableau.isEmpty()) && (counter==1))//we make a
copy of this list
                {
                    for (int m=1; m<propertieslist.get(k).size(); m++)
                    {
                        mytableau.add(propertieslist.get(k).get(m));
                    }
                }
            }
        }
    }
}

```

```

else
{
    /*
    * We compare our "mytableau", current, of folders
name
    * with the new folders list which possess the new
property.

    * We create a kind of filter.
    */
    int m=0;
    while (m<mytableau.size())
    {
        if
(propertieslist.get(k).contains(mytableau.get(m)))
        {
            m++;
        }
        else
        {
            mytableau.remove(m);
        }
    }
}

/*
* we can find three cases:
* ~ no CheckBox checked (counter==0)
* ~ the tableau is empty because there are no results
(mytableau.isEmpty()) :
* ~ no directory possess all the properties checked.
* ~ the tableau is full, so we can display the results :
* the directories which possess all the properties
checked.
*/
if (counter==0)
{
    File f = new File(filePath);
    String[] mylist = f.list();
    for (int i=0; i<mylist.length; i++)
    {
        if (!mylist[i].equalsIgnoreCase(".svn"))
        {
            File f2 = new File(f + "/" + mylist[i]);
            String[] mylist2 = f2.list();
            for (int j=0; j<mylist2.length; j++)
            {
                if ((!model.contains(mylist2[j])) &&
(!mylist2[j].equalsIgnoreCase(".svn")))
                {
                    model.addElement(mylist2[j]);
                }
            }
        }
    }
}
else if (mytableau.isEmpty())
{
    model.addElement("No scenario");
}

```

```
    }  
    else  
    {  
        for (int i=0; i<mytableau.size(); i++)  
        {  
            model.addElement(mytableau.get(i));  
        }  
    }  
}
```

C. CLASSE DOWNLOADSCHEMA

```

package controllers.download;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.List;

import javax.swing.JCheckBox;
import javax.swing.JList;

public class DownloadSchema {

    /**
     * Function which allows, for the user,
     * to get all the scenario schemas in one of his folder.
     * @param filePath
     * @param listJCB
     * @param list
     * @param userfolderpath
     */
    public DownloadSchema(String filePath, List<JCheckBox> listJCB, JList list,
String userfolderpath) {

        int i=0;
        //We know that there is only one JCheckBox selected because there are
no errors.
        while (!listJCB.get(i).isSelected())
        {
            i++;
        }

        /**
         * The fil# are for the user paths
         * and the f# are for the application paths (# being a number).
         */
        File fill = new File(userfolderpath + "/" + listJCB.get(i).getText()
+ "_" + list.getSelectedValue());
        fill.mkdir();//mkdir() creates the folder "listJCB.get(i).getText() +
"+" + list.getSelectedValue()"

        File f1 = new File(filePath + "/" + listJCB.get(i).getText() + "/" +
list.getSelectedValue());
        String[] mylist = f1.list();//We browse the files and folders in
"list.getSelectedValue()" (the scenario selected).
        for (int j=0; j<mylist.length; j++)
        {
            File f2 = new File(f1 + "/" + mylist[j]);//We convert the file
or folder in File()
            if ((f2.isDirectory()) &&
(!f2.getName().equalsIgnoreCase(".svn")))
            {
                /**
                 * To each new folder crossed in the application (
isDirectory() ),
                 * a folder is created in the user ( mkdir() ).

```


D. CLASSE DOWNLOADGRAPH

```

package controllers.download;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.List;

import javax.swing.JTextField;

public class DownloadGraph {

    /**
     * Function which allows to copy the graph files in a folder of the user.
     * @param resultsdirectoryname
     * @param filename
     * @param userfolderpath
     * @param eps
     */
    public DownloadGraph(String resultsdirectoryname, String filename, String
userfolderpath, boolean eps,
        int countertool, List<JTextField> listJTftool)
    {
        //We create a folder with the name "filename" in the folder indicated
by the user "userfolderpath".
        File fil1;
        if (eps == true)
        {
            fil1 = new File(userfolderpath + "/" + filename +
"_eps");
        }
        else
        {
            fil1 = new File(userfolderpath + "/" + filename);
        }

        fil1.mkdir();

        //We place ourselves on the last folder created by the program
("filename").
        File f1 = new File("./" + resultsdirectoryname + "/" + filename);
        System.out.println(f1.getName());
        String[] mylist = f1.list();//We retrieve the file list.

        for (int i=0; i<mylist.length; i++)
        {
            File f2 = new File(f1 + "/" + mylist[i]);

            if (eps == true)
            {
                //if the file is a .eps then we copy it.
                if (f2.getName().endsWith(".eps"))
                {
                    theDownloading(fil1, f2);
                }
            }
        }
    }
}

```

```
        else
        {
            //if the file isn't a .bat or a .sh then we copy it.
            if ((!f2.getName().endsWith(".bat")) &&
(!f2.getName().endsWith(".sh")))
            {
                theDownloading(fil1, f2);
            }
        }
    }

    public void theDownloading (File fil1, File f2)
    {
        //File creation:
        try {
            File fil2 = new File(fil1 + "/" + f2.getName());
            fil2.createNewFile();//The file "f2.getName()" will be created.

            //Statement and opening flows:
            //flow for the application:
            FileInputStream sourceFile = new FileInputStream(f2);
            //flow for the user:
            FileOutputStream destinationFile = new FileOutputStream(fil2);

            //Reading and writing by segment of 0.5MB:
            byte buffer[]=new byte[512*1024];
            int nbLecture;
            while( (nbLecture = sourceFile.read(buffer)) != -1 )
            {
                destinationFile.write(buffer, 0, nbLecture);
            }

            //We close the flows else we get a empty file.
            sourceFile.close();
            destinationFile.close();
        }
        catch (IOException e)
        {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

Résumé

XBenchMatch, plate-forme d'évaluation pour le schema matching.

Le projet est de développer cette application. Elle doit posséder une interface graphique simple alliant des calculs scientifiques complexes.

Ce programme se sert d'autres logiciels pour mener à bien l'obtention de résultats.

Il a été réalisé en Java, langage de programmation orienté-objet.

Summary

XBenchMatch, evaluation platform for the schema matching.

The project is to develop this application. She must possess a simple graphical interface that combines complex scientific calculations.

This program uses other softwares to get results.

It was realized in Java, programming language object-oriented.