TELECOM Nancy
193 av. Paul Muller
54602 Villers-Lès-Nancy

LORIA
Campus scientifique
BP 239
54506 Vandoeuvre-lès-Nancy Cedex

Projet Interdisciplinaire ou Découverte de la Recherche

Serveur d'application pour la Programmer's Learning Machine

Auteurs :
Mathieu
Morainville
& Cédric Huguenin

Responsables :
Martin Quinson
& Gérald Oster

TELECOM Nancy deuxième année (2013-2014)

Remerciements

Nous tenons à remercier Messieurs Martin Quinson et Gérald Oster pour leur soutien et leur disponibilité tout au long du projet ainsi que pour leur enthousiasme et les échanges d'idées intéressants que nous avons pu avoir lors des réunions.

Table des matières

1 Introduction

La Programmer's Learning Machine[Qui14] est une plate-forme d'enseignement de la programmation développée par Gérald Oster et Martin Quinson dans le cadre des cours à Telecom Nancy depuis 2007. Depuis sa création, l'outil a été utilisé dans le module de remise a niveau en informatique proposé aux élèves venant des CPGE en début de première année ainsi que dans le module « techniques et outils pour programmer ». Cette plate-forme présente plusieurs avantages pédagogiques. Tout d'abord, elle permet la mise en œuvre pratique des notions abordées en cours de façon théorique et parfois déroutante pour les débutants. Ensuite, la représentation graphique des problèmes semble plus motivante pour les étudiants. De plus, l'exécution interactive et son cycle de développement/tests court permet aux étudiants d'évaluer eux-mêmes leur travail et de résoudre les problèmes de façon incrémentale. Enfin, la base relativement importantes d'exercices permet à chacun de s'entraîner individuellement à son rythme jusqu'à maîtriser les notions abordées. Dans son état actuel, l'environnement est surtout centré sur l'élève : il reste difficile pour l'enseignant de suivre les avancées des élèves pendant la séance afin de détecter les élèves ayant besoin d'aide, c'est-à-dire ceux qui sont bloqués sur un exercice.

C'est dans ce contexte que se situe notre PIDR (Projet Interdisciplinaire ou Découverte de la Recherche). Ce projet encadré par un enseignant-chercheur ou un chercheur permet un premier contact concret avec la recherche menée dans les laboratoires de l'Université, en particulier le LORIA, le CRAN et l'IECN. Dans le cas interdisciplinaire, le projet ne se déroule pas nécessairement en laboratoire et concerne des aspects liés à différentes disciplines autour de l'informatique. Dans notre cas, il s'agit des aspects pédagogiques liés à l'apprentissage de la programmation. Durant les quatre mois pendant lesquels se déroule le PIDR, nous devrons donc trouver et mettre en œuvre des solutions pour qu'il soit plus facile pour les enseignants comme pour les élèves de suivre les progrès réalisés lors de la résolution des exercices.

2 Analyse du problème

Tout d'abord, nous allons dans cette première partie présenter l'état de la PLM au début du projet puis analyser les problèmes à traiter pour faire en sorte qu'elle réponde aux besoins du sujet. Nous énoncerons alors plus clairement le but à atteindre et décrirons les améliorations à apporter afin de remplir les objectifs poursuivis.

2.1 Situation de départ

La PLM existe depuis 2007 et se trouve donc dans un stade de développement très avancé, ayant également été amélioré au travers d'autres PIDR. Le nombre élevé de classes et de packages la composant nécessite donc de la modifier avec soin et de s'assurer que chaque modification n'entrave pas son fonctionnement. Pour cela des tests unitaires concernant les exercices existent (`plm.core`, fichier ExoTest.java) et nous devions nous assurer qu'après chaque modification importante nous pouvions encore passer chacun de ces tests.

Au départ, et comme nous l'avons stipulé dans l'introduction, la PLM est déjà un outil pédagogique très utile pour les élèves. Néanmoins, à ce moment, ces derniers ne pouvaient récupérer leur ancien code que s'ils se connectaient sur la même session que celle sur laquelle ils avaient ouvert la PLM pour la dernière fois. Ceci afin de pouvoir récupérer les données stockées dans le répertoire .plm, qui stocke sous forme de fichiers zip diverses informations sur l'avancement de l'utilisateur en question et notamment le code source de chaque exercice qu'il a tenté et/ou réussi. Le classe qui s'occupe d'enregistrer et de charger le code source de l'élève s'appelle ZipSessionKit, du package `plm.core.model.session`.

Cette classe n'est cependant pas très adaptée à une utilisation de la PLM en mode « connecté » pour ainsi dire. En effet, la compression opérée sur les fichiers ne permet pas un traitement aisée des données. Ainsi pour l'instant, l'utilisateur n'a aucun moyen de récupérer son code sur une autre machine autre que celui de copier son répertoire .plm sur une clé et de le coller au bon endroit sur un autre ordinateur.

Avec la volonté de déporter les données des utilisateurs vers un serveur pour y avoir accès plus facilement, d'autres questions se sont posées et, pour résumer également ce qui a été dit précédemment, les problèmes suivants ont été remarqués :

- l'utilisateur n'a pas de moyen simple de retrouver son avancement sur une autre machine ;
- l'utilisateur et les professeurs n'ont pas de moyen simple de consulter les résultats des exercices réalisés ;
- l'historique des modifications n'est pas sauvegardé ;
- il y a un seulement un utilisateur par machine.

2.2 But à atteindre

À terme, la PLM devrait pouvoir être utilisé efficacement non seulement par les élèves mais également les professeurs, qui pourront avoir accès aux données des élèves via une interface d'administration sur un serveur distant afin d'en tirer éventuellement des statistiques ou des conclusions quant au travail effectué par les élèves. Cela leur permettrait alors d'être plus efficace lors des séances de travail PLM et de pouvoir adapter leurs

objectifs d'enseignement.

Afin de déporter les données générées par la PLM et de permettre son utilisation sur différentes machines, les améliorations suivantes seront à traiter :

- les modifications d'un utilisateur sont stockés à la fois localement mais aussi sur un serveur distant ;
- dès lors que les données d'un utilisateur sont stockées « dans le nuage », les progrès de cet élève sont consultables à distance via une interface web par les professeurs ;
- l'utilisateur devrait être en mesure de pouvoir récupérer sa progression sur n'importe quel machine ;
- il pourrait alors être intéressant de gérer plusieurs utilisateurs sur une même machine et de permettre de passer de l'un à l'autre facilement.

2.3 Problèmes rencontrés

Lors de l'analyse du problème et de la recherche de solutions pour mener à bien notre projet, plusieurs problèmes se sont posés. Cette partie va nous servir à faire le point sur ces différentes difficultés et à les expliquer plus en détail.

2.3.1 ZipSessionKit ↓

La PLM actuelle propose une interface ISessionKit qui doit être implémentée par les objets dont la responsabilité est de stocker le code de l'utilisateur sur le disque mais aussi de le charger pour qu'il puisse continuer son travail. En plus, il doit renseigner la PLM sur les exercices déjà réussis en fonction des langages de programmation disponible.

L'objet qui est actuellement utilisé est ZipSessionKit. Comme son nom l'indique, il sauvegarde toutes les données utiles dans des archives zip : une par leçon. Chaque archive contient un dossier par langage et ces dossiers contiennent chacun autant de dossiers qu'il y a d'exercices dans la leçon.

Cette solution de stockage ne permet pas d'être facilement lue. Ainsi, elle ne convient pas à notre démarche d'extraction de données des travaux des élèves. De plus, elle ne donne que la dernière version du code de l'utilisateur. Nous aurions aimé profiter de la possibilité de changer cette façon de stocker les sessions des utilisateurs pour mettre en place un système qui conserverait toutes les étapes du code de l'utilisateur.

2.3.2 Serveur distant ↓

En imaginant avoir un objet implémentant l'interface ISessionKit d'une façon qui permette de remplir nos objectifs plus simplement, le code de l'élève resterait pour le moment toujours sur sa machine locale. Le but de notre projet étant de pouvoir récupérer le code de tous les utilisateurs de la PLM qui sont connectés à Internet, il faudrait donc que l'implémentation d'ISessionKit permette d'envoyer le code sur un serveur distant.

2.3.3 Anonymat des données ↓

L'anonymat des données est primordial. Les utilisateurs avertis se méfient de plus en plus du respect de la vie privée dans le monde numérique. Les données recueillies doivent être accessible au plus grand nombre sans permettre d'identifier les utilisateurs. En effet, les données pourront alors servir aux chercheurs et aux professeurs.

2.3.4 Identification des utilisateurs ↓

En apparente contradiction directe avec le problème précédent, il est à noter que comme les données doivent aussi permettre aux professeurs de suivre le travail de leurs élèves, il est impératif d'avoir un système capable de lier une identité sur un utilisateur anonyme. Ce système serait indépendant de l'implémentation du ISessionKit utilisé. Le système doit aussi être simple pour les utilisateurs : il devrait, dans l'idéal, ne pas nécessiter de mot de passe mais tout de même permettre une certaine sécurité. Ce système ne sera utilisé que dans le cadre éducatif a priori et donc seuls les professeurs pourront effectivement faire le lien entre une identité anonyme et un élève en particulier.

3 Organisation

Après avoir analysé l'état de la PLM au début du projet, nous devions nous organiser, aussi bien avec nos encadrants qu'entre nous, pour pouvoir mener à bien notre PIDR. Cette partie a donc pour but de décrire notre méthode de travail et la façon dont nous nous sommes réparti les tâches au cours des quatre mois du projet.

3.1 Méthode de travail

Au commencement du projet, nous avons convenu avec Martin Quinson et Gérald Oster de nous réunir chaque mercredi afin de faire le point sur les progrès réalisés mais également et surtout de définir les limites et le cadre du projet. En effet, beaucoup d'améliorations peuvent encore être apportées à la PLM et les premières réunions étaient surtout des brainstormings afin de trouver des idées et de voir ce qui pouvait être fait dans le cadre temporel limité du PIDR.

À la fin de chaque réunion, nous nous fixions une liste de choses à faire pour la prochaine réunion et entre chaque réunion nous nous efforcions donc de mener à bien ces tâches. Lorsque les limites du projet étaient mieux définies et que les tâches à effectuer commençaient à devenir plus complexes, les fréquences des réunions se sont espacées afin de toujours nous permettre de mener à bien nos tâches. Lorsque les réunions ne pouvaient vraiment pas avoir lieues pour cause d'indisponibilités ou d'incompatibilité des emplois de temps, nous envoyions tout de même un mail récapitulatif quand l'avancement nous semblait assez significatif.

Pour nous permettre de prendre facilement part au développement de la PLM, nous avons également forké le dépôt GitHub de la PLM[QO08] afin de pouvoir faire des pulls requests dès que nous avions des modifications stables à incorporer à la PLM.

3.2 Comptes-rendus des réunions

Afin de mieux se rendre de l'évolution des attentes et des tâches à réaliser au fur et à mesure de l'évolution du temps, nous avons décidé d'écrire dans cette section les résumés des principales réunions que nous avons eues avec nos encadrants.

15/01/2014 :

Ordre du jour : présentation du contexte, de la PLM et des attentes vis-à-vis du PIDR.

Décisions prises : utiliser Git pour stocker les données des utilisateurs. Cela permettra de ne pas dépendre d'une technologie en particulier pour accéder à ces données.

À faire pour la prochaine réunion : protocole sur papier, s'informer sur JGit, étudier le code de la PLM, notamment la partie sur les espions (`plm.core.model.tracking`).

24/01/2014 :

Ordre du jour : présentation des buts à long terme de la PLM. Cadrage des limites du projet et des objectifs à court terme attendus.

Décisions prises : ne pas essayer de tout faire, mais créer une base stable qui pourra être réutilisée ultérieurement.

À faire : créer un espion Git qui commit localement.

31/01/2014

Ordre du jour : aller dans les détails et voir techniquement ce qu'il est possible de faire avec les outils à disposition.

Décisions prises : écrire des Use Cases pour voir quelles implémentations et solutions sont envisageables en pratique.

À faire : écrire les Use Cases, améliorer l'espion Git, se familiariser avec le framework Play.

17/02/2014

Ordre du jour : étudier les Use Cases.

Décisions prises : demander le moins d'informations à l'utilisateur au démarrage de la PLM (éviter les mots de passe et les authentifications). Faire un lien entre une identité anonyme et un utilisateur.

À faire :

- - trouver un moyen de pusher anonymement avec JGit tout en permettant aux professeurs de trouver l'identité d'un élève derrière un commit anonyme ;
- simplifier les Use Cases ;
- faire les premiers essais de push vers un dépôt Git distant avec JGit.

26/03/2014

Ordre du jour : évaluation de l'état d'avancement et des solutions concrètes proposées.

Décisions prises : rédaction des Use Cases/scénarios définitifs. Les utilisateurs seront identifiés par un identifiant unique.

À faire :

- - implémenter GitSessionKit pour remplacer ZipSessionKit ;
- faire en sorte que l'espion Git enregistre les changements aussi lorsqu'un utilisateur change d'exercice à l'intérieur de la PLM ;
- implémenter les différents scénarios.

3.3 Tâches et répartition du travail

Notre cheminement pour résoudre les différents problèmes posés au cours des réunions a connu plusieurs étapes. Nous nous sommes réparti les tâches selon la nature et la complexité de ces étapes et avons souvent dû réfléchir et programmer ensemble. Dans cette partie, nous allons donc présenter ces étapes d'une manière claire et simplifiée.

Première étape (5h) : arriver à faire compiler correctement la PLM sur nos machines. Pour compiler sous Eclipse, nous ne devions pas oublier d'installer le plugin Scala (`http://scala-ide.org/download/current.html`) et le plugin Python (`http://pydev.org/download.html`) et de bien configurer le Build Path.

Deuxième étape (20 h) : prendre connaissance du code et de la structure de la PLM.

Lors de cette étape, nous avons découvert un certain nombre de petits bugs que nous avons pu corriger et nous avons alors soumis la correction par Pull Requests. Ces bugs incluaient :

- correction des numéros de ligne affichés sous Windows lors d'une exception générée par le code d'un utilisateur ;
- encryption SSL des messages automatiques de succès postés sur Twitter ;
- correction d'un problème de synchronisation entre la vue utilisateur dans la PLM et le code sur le disque du à une frappe au clavier trop rapide ;

Nous avons également incorporé quelques nouvelles fonctionnalités, parmis lesquelles :

- la possibilité de poster les rapports de bugs directement sur le projet GitHub ;
- la possibilité d'accéder à une leçon en double-cliquant sur son icône au lieu de cliquer sur le bouton « Aller » ;
- la sensibilité de l'interface aux changements effectués dans le code exécuté pour le monde des Buggles.

Troisième étape (25 h) : création et gestion d'un dépôt local pour un utilisateur. Cette étape a nécessité beaucoup de recherches et de tests étant donné que la documentation de JGit était assez limitée, bien qu'elle nous a semblé être la librairie Java permettant d'effectuer des commandes Git ayant la plus grande base d'utilisateurs. Durant cette étape, nous avons travaillé ensemble sur la création d'un espion Git, via la classe GitSpy que nous avons implémentée.

Quatrième étape (40 h) : push des données sur un dépôt Git distant dans une branche spécifique et possibilité de pull cette branche uniquement, prise en main du framework Play et remplacement de ZipSessionKit par GitSessionKit afin de charger le code du dépôt local dans la PLM lors de son démarrage.

C'est là le cœur du projet. Nous avons travaillé la plupart du temps ensemble sur chacun des points cités étant donné qu'ils nécessitaient tous beaucoup de recherches, de tests mais également de discussions afin de trouver les meilleures solutions à utiliser dans chacun des cas.

Cinquième étape (30 h) : gestion des utilisateurs et de la liaison d'identité.

Cette dernière étape n'a pas pu être menée à bien aussi loin que nous l'aurions voulu, notamment la phase de liaison d'identité, pour cause de manque de temps. Néanmoins, nous nous sommes efforcés de faire en sorte que la reprise du code de ces fonctionnalités par quelqu'un d'autre soit simple et déjà bien entamée.

Pour conclure cette partie, nous avons donc travaillé environ 120 heures sans compter la rédaction du rapport et la préparation de la soutenance. Nous nous sommes en général réparti le travail sur la résolution des petits bugs et l'ajout de nouvelles fonctionnalités, mais nous avons plutôt souvent réfléchi et programmé ensemble sur les points les plus complexes et importants.

4 Résolution

4.1 ZipSessionKit ↑

Pour pouvoir répondre aux objectifs notre PIDR, nous avons décidé de développer un nouveau SessionKit. Cette nouvelle implémentation nous permettra de stocker les progrès des élèves de manière locale d'une manière plus simple. Nous aurons également plus facilement accès au code de chaque exercice. Pour cela, en accord avec Martin Quinson et Gérald Oster, nous avons choisi d'utiliser comme solution technique Git car cela permettrait de s'affranchir au maximum d'une technologie qui aurait pu être trop spécifique. En effet, Git est utilisé par 40 %[Fou13] des programmeurs et est un outil qui est fait pour durer et dont les bases ne sont pas prêtes de changer dans les années à venir. Étant donné qu'il a été créé à l'origine pour pouvoir gérer le développement du noyau Linux, il est stable et performant. La gestion de version offerte par Git pourrait également permettre un retour en arrière facile pour permettre aux enseignants de visualiser les différentes étapes du code de l'élève lors de son raisonnement pour résoudre le problème posé. À terme, cela pourrait même être utilisé pour permettre à l'élève de revenir en arrière simplement dans son code.

Le code de l'utilisateur est alors sauvegardé à chaque fois qu'il teste ou change d'exercice. Il est en fait mis dans un commit [1] dont le message associé est formaté en JSON [2]. Il contient :

- le type d'évènement ;
- le langage utilisé ;
- l'exercice courant ;
- le nombre de tests de l'exercice courant ;
- le nombre de tests satisfaits par le code de l'utilisateur.

Listing 1 – Message de commit

```
{"os":"Windows 8 (version: 6.2; arch: amd64)","plm_version":"2.3beta
    (20140125)","evt_type":"started","java_version":"1.7.0_51 (VM: Java HotSpot(TM)
    64-Bit Server VM; version: 24.51-b03)","start_date":"2014\/05\/21 13:08:26"}

{"course":"","exolang":"Java","exoswitchto":"The small cousines of
    Buggles","evt_type":"switched","totaltests":"-1","passedtests":"0","exoname":"The
    small cousines of Buggles"}
```

Le commit contient d'autres fichiers que le code de l'élève. En effet, l'énoncé peut être amené à être modifié, c'est pourquoi nous le conservons aussi ainsi que la correction associée. Cela permettra également aux professeurs qui regardent un commit en particulier de connaître le contexte et l'énoncé à l'instant t où l'élève a fait tel exercice. Un fichier contenant les erreurs de compilation du code de l'utilisateur est aussi sauvegardé.

Les commits sont gérés par une classe GitSpy qui implémente `ProgressSpyListener`. Elle récupère des évènements de type :

- démarrage de la PLM ;

1. Sauvegarde des fichiers à un instant t.
2. Format de données textuelles, générique, qui permet de représenter de l'information structurée.

– exécution du code ;
– changement d'exercice.

Utiliser Git pourra également nous permettre de gérer facilement plusieurs utilisateurs sur une même machine, à partir du moment où nous utilisons une branche spécifique pour chaque utilisateur. Voir la figure 1.

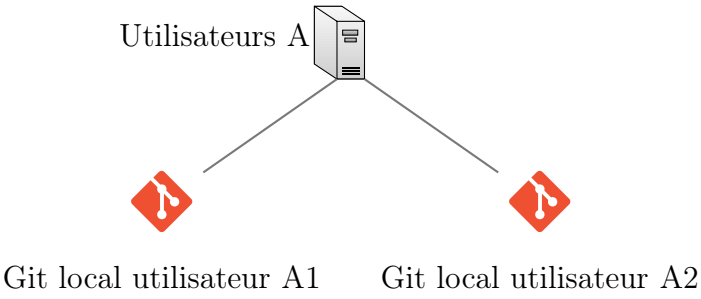

FIGURE 1 – Git locaux

4.2 Serveur distant ↑

Une fois les données des utilisateurs mis en forme dans un dépôt Git local, il faut que les professeurs puissent y accéder facilement. Nous avons donc décidé assez logiquement d'utiliser un serveur Git distant puisque nous n'aurions alors qu'à pusher les modifications locales. Ce dépôt central devra contenir le code de tous les utilisateurs de la PLM. Pour ne pas que le code des différents usagers se mélange, nous avons décider d'utiliser une branche par personne. Chaque fois que le code sera envoyé sur le dépôt central, il sera envoyé dans une branche spécifique. Comme il n'est pas forcément possible ni très sécurisé de permettre à n'importe qui de pusher anonynement dans un dépôt Git distant, nous avons préféré lors de nos tests créer un dépôt sur Bitbucket[PLM14] avec un compte utilisateur créé dans le seul but de permettre à la PLM d'y pusher du code. Les identifiants de ce compte utilisateur sont stockés dans la PLM. Voir la la figure 2.

4.3 Anonymat des données ↑

Le code écrit par tous les utilisateurs est disponible sur le dépôt Git central. Ceci nous permet de proposer aux utilisateurs de continuer une session entamée sur une autre machine. Pour cela, nous générons et donnons un identifiant unique à l'utilisateur. Celui-ci sert de mot de passe pour continuer sa session sur une autre machine. Cet identifiant est généré par Java grâce à la classe UUID[Ora10]. Il fait 128 bits de long et est créé à partir de la date et d'une partie générée aléatoirement. La probabilité d'une collision est très faible[IET05]. Cet identifiant peut donc être demandé au lancement de la PLM pour continuer une session existante sur n'importe quelle machine. L'identifiant unique est utilisé pour la gestion des branches du dépôt central.

Étant donné que tout le code de tous les utilisateurs sera disponible de manière public, le nom des branches est publique. Il faut toutefois que l'on protège les données de chaque utilisateur : on ne doit pas, en clonant le dépôt central, pouvoir continuer le travail d'autrui. C'est pourquoi, au lieu d'utiliser l'UUID en guise de nom de branche, nous préférons

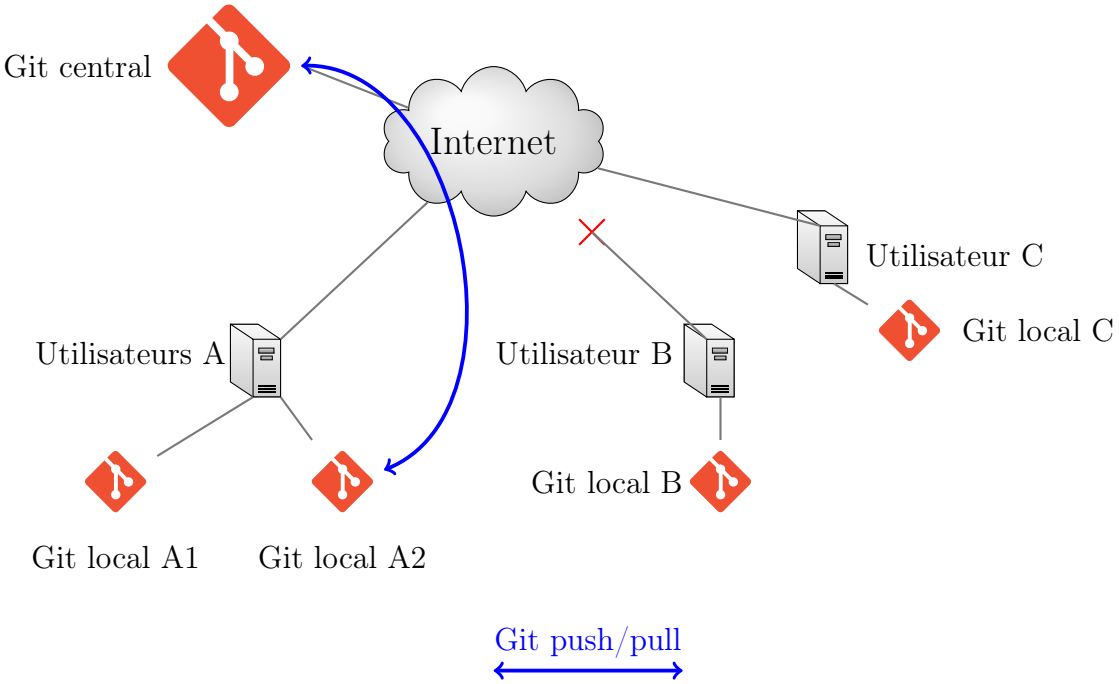

FIGURE 2 – Avec un dépôt Git central

se servir de son hashé. De la sorte, même si quelqu'un récupère le nom des branches sur le dépôt central, il ne pourra continuer le travail d'un autre que s'il arrive à inverser la fonction de hashage. Nous ne stockons aucune information quant à l'identité des utilisateurs dans les messages de commits ou dans les fichiers stockés dans le dépôt local de chaque élève. Voir la figure 3.

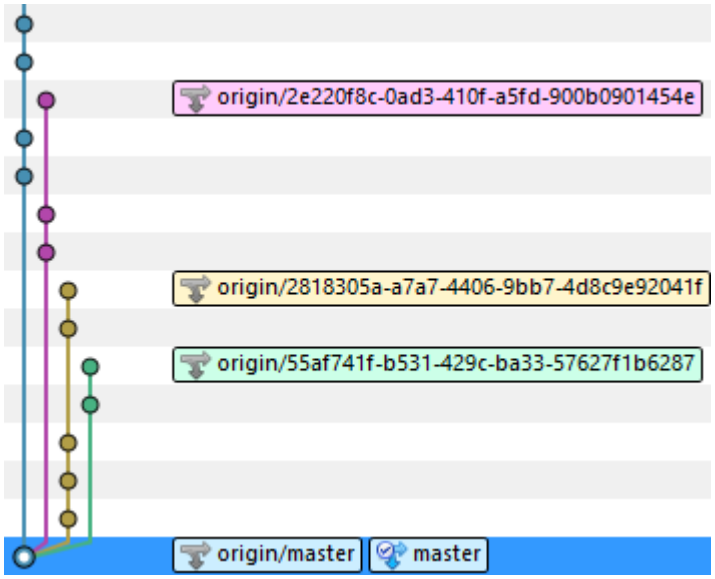

FIGURE 3 – Branches du dépôt Git central

4.4 Identification des utilisateurs ↑

Le but initial de notre PIDR est de permettre aux professeurs de suivre l'avancement de leurs élèves. Il faut donc pouvoir les identifier sans compromettre leur identité sur le dépôt central. Nous avons donc décider d'utiliser un serveur Play[Pla14] pour centraliser les données authentifiantes.

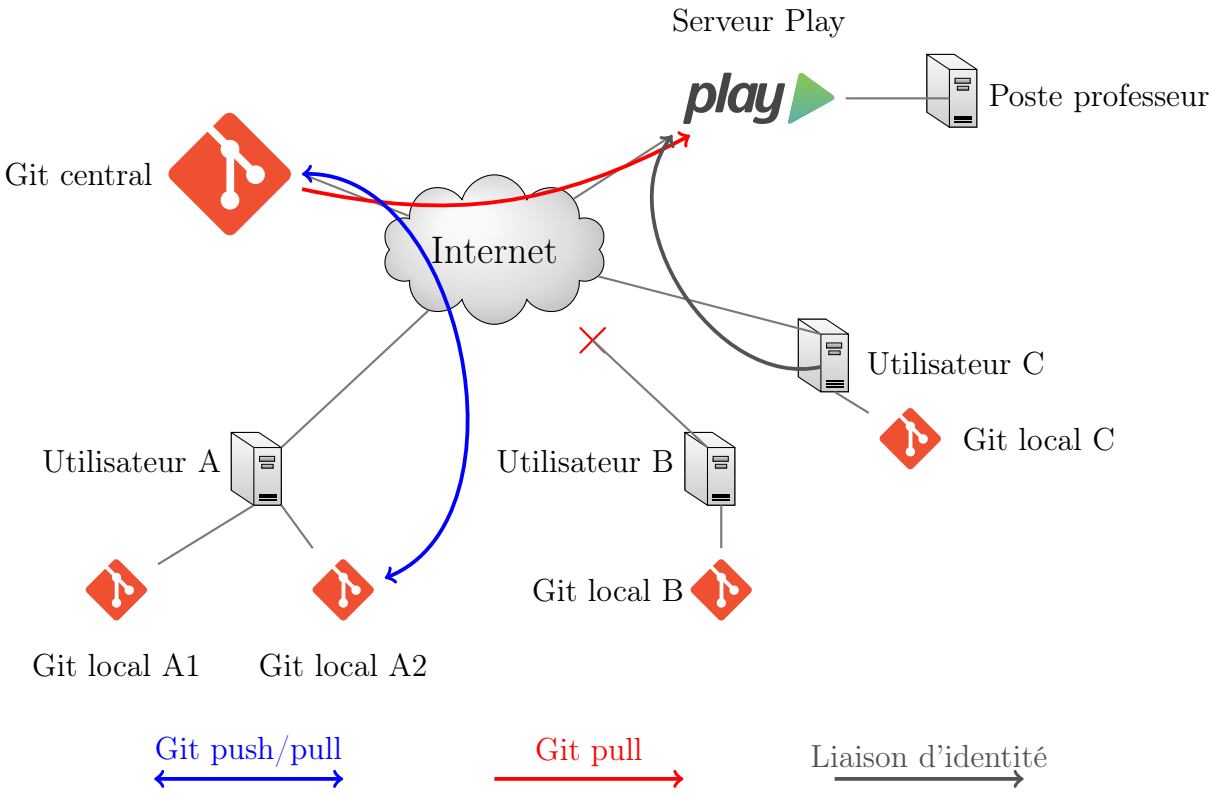

FIGURE 4 – Schéma global

Les élèves seront invités par leur professeur à se connecter sur un serveur Play. Ils devront alors renseigner leur nom, prénom, adresse mail et classe par exemple.

Cela permet de s'assurer que chaque utilisateur « anonyme » créé au lancement de la PLM le restera jusqu'au moment où l'utilisateur voudra lier son identité réelle à son identité anonyme. Ce lien se fera sous la forme d'une requête au serveur qui stockera dans une base de données le nom réel, l'adresse mail et le hashé de l'UUID fourni par l'utilisateur afin de savoir à quel utilisateur appartient telle branche vu qu'il n'y a pas d'informations personnelles stockées sur le dépôt distant.

Le reste du PIDR nous ayant déjà demandé beaucoup de temps, le serveur Play ne se trouve pas dans un état très avancé à la fin du projet et ne permet que de pull le dépôt Git central et de consulter la base de données des utilisateurs ayant lié leur identité. Cela pose néanmoins les bases de ce qu'il sera possible de réaliser à l'avenir, comme par exemple offrir aux professeurs une interface web permettant de gérer leurs groupes d'élèves par matière. Voir la figure 4.

5 Conclusion

Nous espérons que les modifications que nous avons apporté à la PLM permettront à terme aux professeurs de mieux cerner les besoins et les difficultés de leurs élèves et aux utilisateurs de pouvoir mieux visualiser leurs progrès grâce à la gestion de versions apportée par l'utilisation de Git comme moyen de stockage privilégié. Nous pensons avoir réussi à faire en sorte que la PLM soit bien préparée à se voir ajouter de nouvelles fonctionnalités destinées aux professeurs. Le chemin reste cependant encore long avant un usage efficace de la PLM par ces derniers. Les données collectées seront utilisables non seulement par les professeurs à partir du moment où elles sont authentifiables mais également par les chercheurs puisqu'elles seront anonymes et accessibles à tous. Une perspective directement envisageable qui serait la suite directe de notre travail serait de traiter toutes ces données pour en tirer des représentations graphiques plus facilement interprétables. Dans le cadre des cours à Telecom Nancy, le traitement pourrait être réalisé par le serveur Play, ce qui fournirait aux professeurs une interface graphique agréable à utiliser pour suivre les progrès des élèves. Nous aurions aimé pouvoir réaliser nous-même cette étape si nous n'avions pas été pris par le temps vers la fin du PIDR. Néanmoins, nous pensons avoir laissé une base stable à nos éventuels successeurs pour réaliser ces améliorations et porter encore plus loin la PLM dans son objectif d'enseignement pédagogique de la programmation.

Références

[Fou13] The Eclipse FOUNDATION. *Eclipse community survey report*. Page 14. Juin 2013. URL : http://fr.slideshare.net/IanSkerrett/eclipse-survey-2013-report-final (cf. p. 9).

[IET05] IETF. *A Universally Unique IDentifier (UUID) URN Namespace*. Juil. 2005. URL : http://www.ietf.org/rfc/rfc4122.txt (cf. p. 10).

[Pla14] PLAY. *Play*. Jan. 2014. URL : http://www.playframework.com/ (cf. p. 12).

[PLM14] PLM. *Dépôt central sur Bitbucket.org*. Mai 2014. URL : https://bitbucket.org/PLM-Test (cf. p. 10).

[QO08] Martin QUINSON et Gérald OSTER. *PLM repository*. Oct. 2008. URL : https://github.com/oster/PLM (cf. p. 6).

[Qui14] Martin QUINSON. *Programmer's Learning Machine*. Jan. 2014. URL : http://www.loria.fr/~quinson/Teaching/PLM/ (cf. p. 2).

[Ora10] ORACLE CORPORATION. *Javadoc Class UUID*. 2010. URL : http://docs.oracle.com/javase/1.5.0/docs/api/java/util/UUID.html (cf. p. 10).

Table des figures