Rapport de stage

Validation automatisée des contributions de la communauté Pharo

Par

Douaille Erwan

Tuteur entreprise : Christophe Demarey, Camillo Bruni

Tuteur universitaire : Samy Meftali

RMod

Inria
INVENTEURS DU MONDE NUMÉRIQUE

Université Lille1
Sciences et Technologies

Remerciements

RMOD est une superbe équipe. Je tiens a remercier l'ensemble des membres pour m'avoir à nouveau accueillis dans la bonne humeur et pour avoir partagé leurs savoirs.

J'aimerai tout particulièrement remercier Camillo Bruni et Christophe Demarey, pour m'avoir apporté les conseils et connaissances, qui m'ont aidé tout au long de ce stage.

Je remercie finalement, l'ensemble des enseignants de Licence Informatique, pour m'avoir apporté les compétences nécessaires pour accomplir ce stage.

Résumé

Dans le cadre de mon stage de fin d'étude, j'ai eu à réaliser une plateforme d'intégration continue, à Inria Lille - Nord Europe (Institut National de Recherche en Informatique et en Automatique) dans l'équipe RMoD. L'intégration continue est une méthode qui consiste à vérifier chaque modification effectuée sur le code source d'une application. Mon projet consiste à créer une application capable de charger, tester et rédiger des rapports sur les modifications apportées par la communauté sur le language Pharo.

Cet outil aura pour but de faciliter les corrections de bugs, de prévenir les incompatibilités, et d'indiquer si une version du logiciel est stable. Dans le cas d'un problème d'intégration, l'application aura pour objectif de fournir des rapports d'erreurs.

Abstract

For my internship at Inria (Institut National de Recherche en Informatique et en Automatique) in the RMoD team, I create a tool able to load and test commits made by the Pharo community. Continous integration is a practice where each modification to the source code is checked automatically. My project consists in creating a tool able to load, test and publish reports from each modification made by the Pharo community.

This tool will provide an easy way to maintain the Pharo platform, prevent some futures bugs and provide stable releases. In case of any bug, the tool has to publish reports which contain informations to help debugging.

Table des matières

1 Introduction

Actuellement étudiant en dernière année de Licence Informatique, j'ai effectué mon stage de fin d'étude à Inria (Institut National de Recherche en Informatique et Automatique) au sein de l'équipe RMod, du 2 avril jusqu'au 27 juin 2013.

Ce stage, effectué pour la deuxième fois au sein d'une équipe de recherche, a conforté l'ambition que j'ai d'intégrer ce domaine professionnel.

L'objectif de mon stage est de réaliser une un outil capable de validé de façon automatique les contributions soumises par la communauté de Pharo. Cet outil doit être capable de charger les données souhaitant être testé, des modifications de code source, d'effectuer les tests voulus, de générer et/ou poster des rapports de tests et finalement indiquer si la version de Pharo ayant chargé les données est stable.

La suite de ce rapport vous présentera comment j'ai concrétisé ce projet.

2 Contexte

Je vais maintenant vous présenter l'environnement dans lequel j'ai effectué mon stage. L'Inria (Institut National de Recherche en Informatique et en Automatique), institution pour laquelle j'ai travaillée. RMod, l'équipe que j'ai rejointe et Smalltalk, le langage avec lequel j'ai travaillé.

2.1 Inria

2.1.1 Présentation

Inria est un établissement français sous la double tutelle du ministère de la recherche et des industries dont le but est d'entreprendre des recherches dans les sciences appliquées aux Technologies de l'Information et de la Communication (TIC). L'institut fournit également un transfert de technologie fort et porte une attention particulière pour la formation par la recherche, la diffusion du développement scientifique et technique, l'expertise et la participation à des programmes internationaux.

2.1.2 Composition

Inria accueille 3800 personnes dans ses huit centres de recherche situés à Rocquencourt, Rennes, Sophia Antipolis, Grenoble, Nancy, Bordeaux, Lille et Saclay. 2800 membres du personnel sont des scientifiques de l'Inria et l'autre partie sont des partenaires d'organisations (CNRS, universités). Au total, ils travaillent dans plus de 160 équipes de recherche. Beaucoup de chercheurs de l'Inria sont également enseignants et leurs étudiants (environ 1000) effectuent leur thèse au sein des équipes de projet de recherche de l'Inria.

2.1.3 Inria Lille - Nord Europe

L'Inria Lille - Nord Europe, dirigé par David Simplot-Ryl, rassemble ses 10 équipes de recherches dans un bâtiment de 4000m2 acquis avec l'aide du

gouvernement et de fonds européen. Inria Lille accueille plus de 220 personnes, près de la moitié sont payés par l'institut. Le centre Inria est un atout pour la compétitivité du Nord - Pas-de-Calais dans la recherche et l'innovation.

2.2 Equipe RMod

2.2.1 Présentation

Le but de l'équipe RMod est d'aider à re-modulariser des applications orientées objet. Cet objectif fait suite à deux lignes complémentaires, la ré-ingénierie et la définition de nouvelles constructions pour les langages de programmation.

Pour aider à la ré-ingénierie, de nouvelles méthodes d'analyses sont proposées afin de comprendre de grosses applications (re-modulariser les métriques, visualisations adaptées, etc). Dans le contexte des langages de programmation, les constructeurs pour leur modularité, ainsi que des modules de vérifications sont étudiés.

L'équipe travaille également sur un nouveau noyau sécurisé pour Pharo, un environnement de développement intégré pour Smalltalk, utilisé et maintenu par l'équipe.

2.2.2 Re-modularisation

L'évolution d'une application est limitée par des dépendances fortes entre ses versions antérieures. C'est pourquoi il est crucial de répondre aux questions suivantes :

- Comment pouvons-nous remplacer une partie par une autre avec un impact minimal ?
- Comment faire pour identifier les éléments réutilisables ?
- Comment modulariser une application quand il y a des liens incorrects ?

Répondre à ces questions est le but de Moose, l'environnement d'analyse de l'équipe, qui fournit une panoplie d'outils d'analyses. Ce travail est divisé en 3 parties :

- des outils pour comprendre le fonctionnement des grosses applications (packages/modules)
- des analyses pour une re-modularisation

- des outils de qualité logiciel

2.2.3 Sémantique des éléments pour la modularité

Ce dernier paragraphe ce concentre sur la définition des éléments nouveaux pour la sémantique des langues dans le but de construire une application flexible et re-configurable. L'équipe travaille actuellement sur :

- la définition d'un langage uniquement basé sur Traits
- le rapprochement entre les langages réflexifs et la sécurité

2.3 Smalltalk

Le langage Smalltalk, autrement appelé Smalltalk-80, est un langage de programmation entièrement orienté objet, réflexif et dynamiquement typé, qui a vu le jour dans les années 80. Il influença de grands langages de programmation comme Python, Ruby ou encore Java.

Smalltalk est le langage de programmation utilisé par l'équipe, et est donc celui que j'ai moi même utilisé pendant mon stage.

2.3.1 Présentation de Smalltalk

Smalltalk est un langage de programmation orienté objet, réflexif et dynamiquement typé. Expliquons certains de ces mots :

- Orienté objet : en Smalltalk on manipule des objets par envoi de message, comme en Java ou C++.
- Réflexif : chaque objet peut inspecter et modifier sa propre structure pendant son exécution.
- Typage dynamique : les variables n'ont pas de type à l'initialisation, mais uniquement à l'exécution.
- Tout est objet : tout est objet, les classes, les méthodes, les messages ...

J'ai pour ma part pu apprendre ce langage avec Pharo By Example[**?**].

2.3.2 Quelques exemples

Déclaration de variable et initialisation :

```
| a |
a := 1.
```

Création et initialisation d'instance :

```
| a |
a:= Point new.
a x:1.
a y:2.
```

Tests d'égalités :

```
| a b |
a := 1@2.
b := 1@2.
a = b 'true'.
a == b 'false'.
```

Ouverture d'une fenêtre vide :

```
| aWindow |
aWindow := SystemWindow new.
aWindow openInWorld.
```

Ce que nous pouvons retenir de ces exemples ainsi que d'autres éléments de syntaxe :

- || déclare une variable
- := initialise une variable
- = tester l'égalité des valeurs de deux variables
- == tester si deux objets ont la même référence
- : façon de spécifier un paramètre aux méthodes
- self le receveur de la méthode, similaire à this en Java
- ĵpermet de retourner des valeurs, par défaut une méthode retourne self

3 Projet CI

L'intégration continue, dont notre projet porte le nom CI (l'abrégé anglais) à pour but de nous fournir une base solide nous permettant de définir si une version de Pharo est stable ou non, tout en intégrant les dernières modifications du système.

3.1 Présentation

Comme indiqué ci-dessus CI à pour but de fournir une suite logicielle permettant la validation d'un logiciel, stable ou non. L'intégration continue doit évaluer et tester chaque changement effectués sur le code source, afin de vérifier qu'il n'y ait pas de régression. L'intégration continue est une pratique de l'extrême programming.

La communauté Pharo se compose de 500 développeurs. Aujourd'hui, une moyenne de 40 intégrations des modifications de code source sont effectuées chaque semaine de facon manuelle.

Vérifier et tester les modifications de code est une tâche fastidieuse et qui se doit d'être faite par un automate fonctionnant 24h/24 et 7j/7. Mon stage à pour but de fournir à la communauté Pharo une suite logicielle réalisant des tests sur chaque nouvelle modification de code source effectuée par les développeurs Pharo, tout en générant des rapports d'erreurs. Mon projet à pour nom CI, et je vais vous présenter plus en détail les objectifs de ce projet ainsi que la méthode utilisée pour le développer.

3.2 Objectif

L'objectif de ce stage est tout simplement d'appliquer la méthodologie de l'intégration continue sur la plateforme que développe et maintient l'équipe RMoD : Pharo. Pharo, est un environnement vivant, qui peut devenir instable au moindre changement de code source. CI à donc pour objectif de fournir aux utilisateurs une version qui sera contrôlée et mise à jour en fonction des

modifications récentes tout en fournissant des rapports de validation/erreur, ce qui permettra à la communauté de comprendre les impacts sur le système.

Résumons ce que nous souhaitons obtenir :

- Une interface pour donner les instructions que l'on souhaite faire effectuer par CI
- Récupérer les changeset, des modifications sur du code existant ou des descriptions de projet qui contiennent des informations sur les dépendances nécessaires au projet
- Un orchestrateur qui va travailler avec les données reçues et gérer l'ensemble du système
- Une étape de validation ou l'on va exécuter les différentes validations (tests, analyse staique de code basée sur des règles, etc.)
- Et pour finir générer un rapport détaillé de l'état des validations

3.3 Problèmes rencontrés

Lors de ce stage, j'ai été confronté à quelques difficultés :

- S'approprier l'existant et comprendre les mécanismes en place. Ex : le fonctionnement du bug tracker, les interfaces pour la ligne de commandes ...
- Organiser les différentes briques logicielles, les assembler et les faire interagir.
- La communication entre machines virtuelles via différents mécanismes.

3.4 Mise en œuvre

CI se doit d'être un projet stable car de ce projet va dépendre de la stabilité de la plateforme Pharo.

Pour nous aider au maximum à identifier les différentes parties du projet, nous avons définis plusieurs briques (composants) visibles sur la figure 3.1. Sur ce schéma nous retrouvons l'ensemble des éléments nécessaires pour remplir les objectifs de CI.

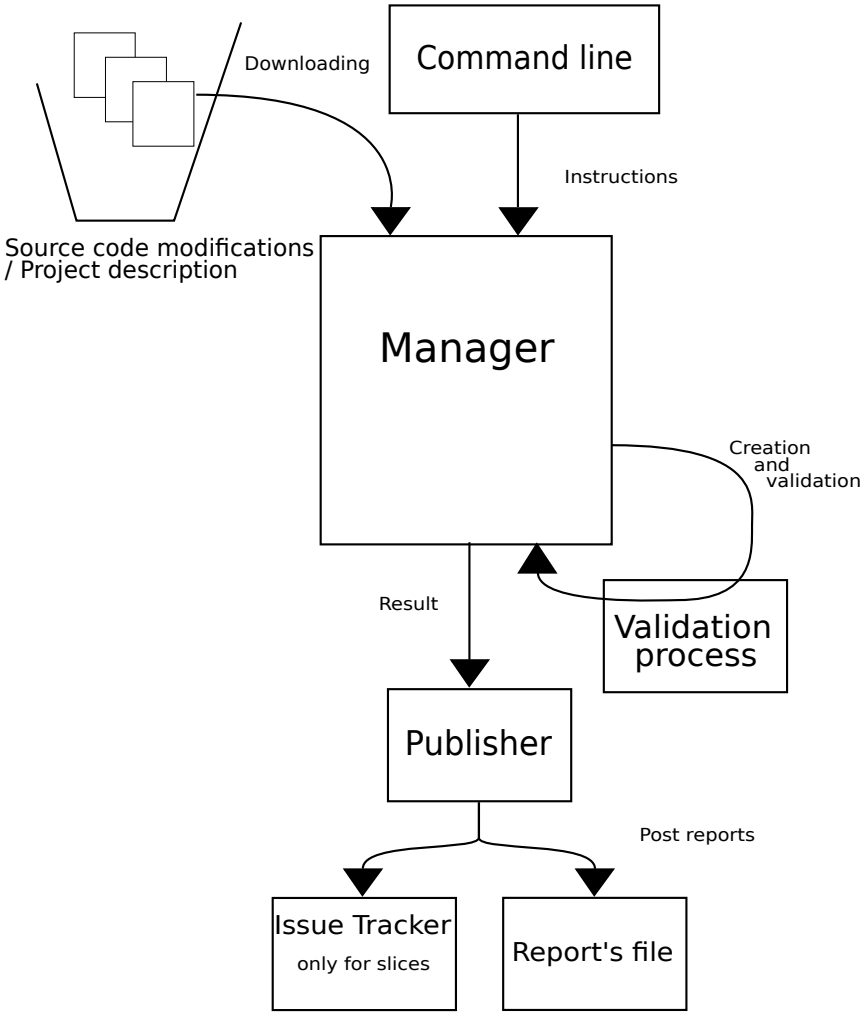

FIGURE 3.1 – Schéma de fonctionnement de CI

3.5 Méthodologie du projet

Pour concevoir une base logicielle solide ainsi que d'avoir des points de vues différents j'ai commencé par pair-programmé avec Camillo Bruni.

4 Manager

La partie centrale du projet CI se trouve dans le Manager visible dans la figure 3.1. Cette partie se compose de plusieurs éléments tel qu'un manager, des sources et des changes. Chacun de ces éléments seront détaillés dans leurs parties respectives.

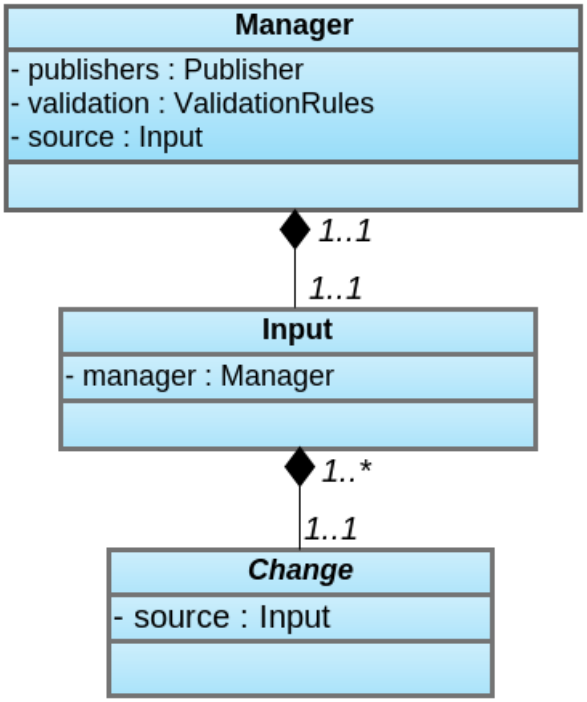

FIGURE 4.1 – UML du manager

4.1 CI Manager

Le manager est un petit élément de notre système par lequel toute les données et instructions vont transités. Le manager est créer et initialisé depuis la ligne de commande.

Il va contenir une source de données qui permettra de récupérer le changeset et savoir comment l'utiliser. La partie source de donné permet aussi au manager de connaître une liste des publishers à utiliser ainsi que le validateur qui exécutera les tests appropriés.

4.2 Source de données

Le source est la partie qui nous permet de communiquer et de fournir les éléments nécessaires au manager pour travailler. Exemple dans le cas d'un changeset qu'il faut aller récupérer sur le bug-tracker, le source va savoir comment et ou le télécharger tout en utilisant le processus d'authentification nécessaire. Le source contient l'élément avec lequel on va travailler par la suite, à savoir le changeset.

4.3 Changeset

Le change est le changeset. Il contient de nombreuses informations comme :

- comment charger dans le système l'élément, puisque plusieurs types de changeset existent
- la version et les packages à charger
- l'URL d'origine
- le nom, l'id ...

Le changeset sera utilisé par le manager pour lancer la validation.

```
CIManager >> validate: aChange
  "return a CIValidationResult which will be used by CIPublisher"
  ^ self validationRule validate: aChange
```

Voici un exemple de code que l'on peut retrouver qui utilise les changesets. Comme visible ci-dessus, la partie manager va lancer les tests et par la suite récupérer les résultats pour les publier.

5 Validation

5.1 Validation de règles

L'étape de validation pour savoir si le changeset est valide ou non est une partie cruciale pour le projet CI. Stabilité et gestion des erreurs possibles sont de mises dans cette partie.

5.1.1 Processus de validation

Le processus de validation est exécuté par le manager. CIValidator est la classe qui permet de lancer la validation de règle sur le changeset passé en paramètre. CIValidator possède une liste pré-définit de règles et les appliques une à une.

Il existe plusieurs type de règles que je vais vous présenter ci-dessous.

5.1.2 Règles basiques

Ce que j'appelle règles basiques, sont un ensemble de validations qui sont testées directement dans l'image. En effet ces règles ne sont pas dîtes sensibles et, peu importe le déroulement de leurs exécution, il n'y aura aucune impacte sur le système. Exemple, il existe un ensemble de règles pour tester si le changeset est prêt à être testé. Ces règles sont dites des règles basiques.

5.1.3 Les LintRules

Les LintRules sont un ensemble de règles permettant une analyse statique du code source. Elles sont présentes dans la plupart des langages de programmation et sont donc fournies par Pharo.

Nous nous servons des LintRules pour savoir si, par exemple, les classes sont bien commentées, les méthodes sont organisées ou encore si les variables sont correctement initialisées. Une cinquantaine de ces règles sont appliquées lors de la validation pour s'assurer de la qualité du code soumis.

5.1.4 Composition de règles

Souhaitant définir des règles de priorités sur l'exécution des règles, nous avons définit les règles composite. Voici une représentation de leurs implémentation avec le design pattern composite, figure 6.1, ainsi que leurs description.

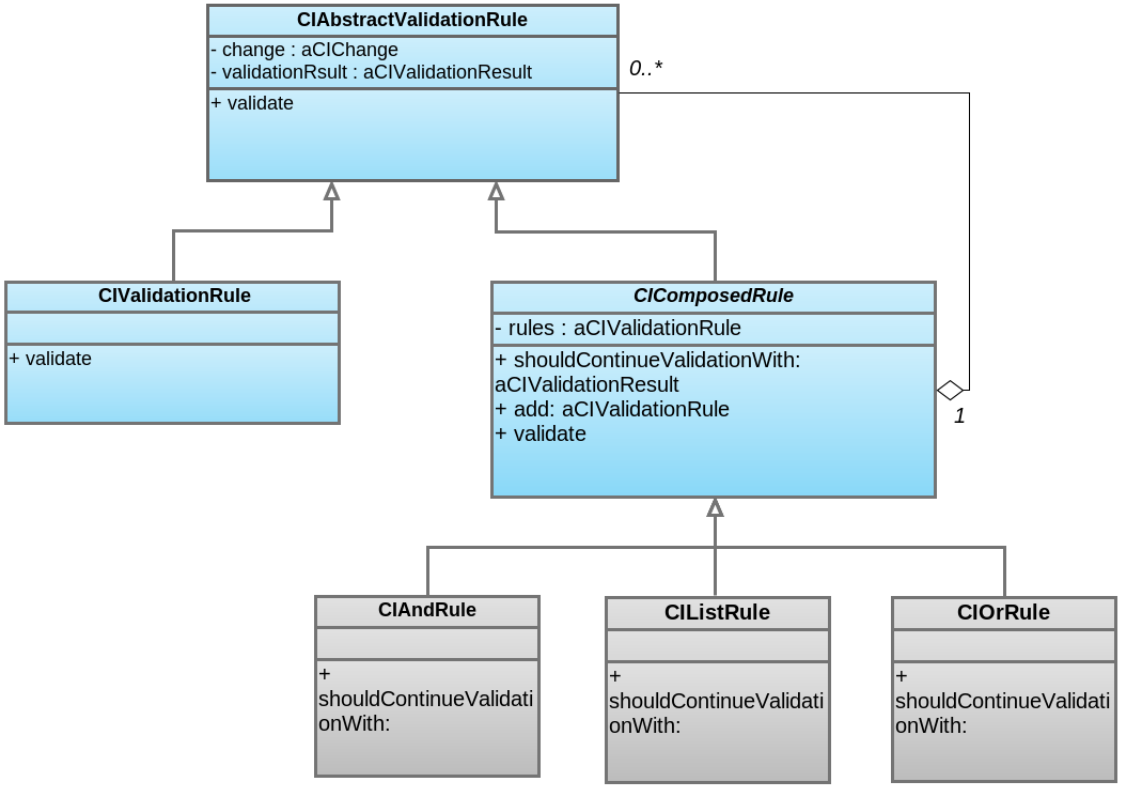

FIGURE 5.1 – Schéma d'implémentation des règles composites

CIAndRule : une classe qui permet d'assembler plusieurs règles et de définir le comportement suivant, l'ordre des règles est séquentiel et si une règle échoue on arrête le processus.

CIOrRule : une classe qui permet d'assembler plusieurs règles et de définir le comportement suivant, l'ordre des règles est séquentiel et continue tant qu'il n'y a pas un test réussi.

CIListRule : une classe qui permet d'assembler plusieurs règles et de les tester les unes à la suite des autres.

Exemple d'utilisation possible des règles composites :

```
Rule1 & Rule2 | Rule3 , Rule4

ChangeValidationRule & ListOfLintRules
```

Dans le dernier exemple, si la validation du changeset n'est pas un succès alors on n'exécute pas les Lint rules.

5.1.5 Règles clonées

Comment tester des modifications de code source, dans un environnement vivant, tout en s'assurant que le système courant ne sera pas impacté par du code erroné ?

En effet, si nous chargeons du code erroné il est possible que l'environnement entier se retrouve perturbé et se termine de façon inattendue. Cette fermeture nous empêcherait de tester la modification et de définir si le code est valide ou non.

La solution envisagée et retenue est la validation in-vitro en clonant l'image.

5.2 Mécanisme de clonage pour tester des modifications sensibles

Lorsque l'on clone une image Pharo, on va garder le même état. Nous aurons par la suite à différencier l'image originale et l'image clonée pour savoir qui donne les instructions et qui les reçoit.

La figure 6.2 montre le déroulement des instructions pour le mécanisme de clonage (fork).

5.2.1 Protocole de communication

Pour communiquer correctement avec notre image j'ai définit plusieurs prototypes qui se sont révélés peu fiable. J'ai trouvé une solution de communication inter-image créée par Nick Papoylias, appelée Seamless.

Seamless permet d'échanger des références sur les objets d'une image à une autre.

Pour faire fonctionner Seamless il faut initialiser 2 serveurs, client A, l'image originale et le client B, l'image clonée. A se connecte sur B et possède un accès, dans notre cas, à un dictionnaire de référence sur l'ensemble des

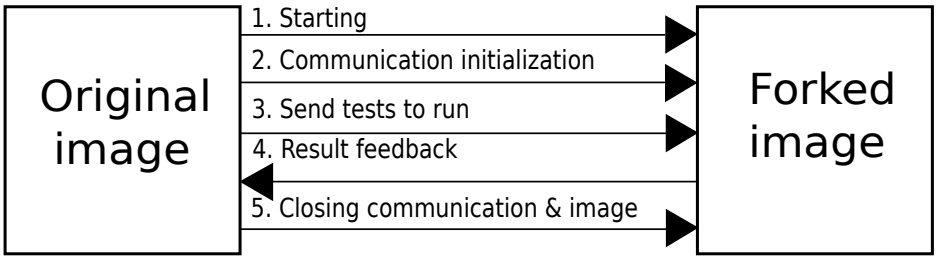

FIGURE 5.2 – Schéma de fonctionnement des tests via le clonage

classes de l'image B. Une fois cette étape accomplie il nous suffit d'indiquer à l'image distante quoi faire.

Self remote nous donne accès à un dictionnaire de classe existantes dans l'image clonée et ensuite nous pouvons lui envoyer des messages, ici foo : #bar.

```
(self remote at:#AClassName) foo:#bar
```

5.2.2 Problèmes rencontrés

Le problème rencontré avec cette technique est qu'il faut copier en local l'objet avec lequel on souhaite travailler. C'est possible via l'envoi de message asLocalObject.

```
(self remote at:#AClassName) asLocalObject
```

Maintenant nous pouvons travailler avec notre object. Autrement un problème de communication empêche Seamless de retourner notre résultat puisqu'il travaillait avec une référence de l'objet étant dans l'image originale.

Malgrès cela un autre problème apparaît. Avec notre procédure de test nous allons travailler avec des block. Un block est une fonction anonyme qui à accès au contexte d'exécution. Le problème qui apparaît lors de l'appel à asLocalObject est que cela nécessite la copie complète de la pile d'exécution nécessaire pour évaluer le block. Cela à comme impacte de rapidement augmenter la taille que Pharo occupe en mémoire. En pratique on passe de environ 40 Mo à plus des 120 Mo avec nos tests.

5.2.3 Ligne de commande pour le clonage

Pour utiliser le mécanisme de clonage il nous a fallu créer une nouvelle ligne de commande pour exécuter l'image clonée et initialiser les communications. Cette ligne de commande sera exécuté par notre image originale. Voici un exemple d'utilisation :

```
pharoVM pharo.image seamless 8080
```

6 Publisher

La dernière étape de ce processus , mais non moins importante, est de communiquer les résultats de la validation à l'utilisateur. Les publisher, autrement dit les publieurs, ont cet objectif.

6.1 Choix des formats

Le choix des formats est très important puisque c'est ce qui va permettre à l'utilisateur de pouvoir connaître l'impact des modifications qu'il a commité.

6.1.1 HTML

Dans l'état actuel du projet CI, le format HTML est le format qui est le plus rapidement utilisable. Il regroupe et affiche proprement les résultats de la validation.

Dans les améliorations que je souhaite réaliser, le format HTML disposera d'une ergonomie améliorée et sera le support final de visualisation des erreurs. On souhaiterai pouvoir sauvegarder la stack d'une erreur et la parcourir via notre page web. L'erreur est pour le moment stocké sous forme d'une chaîne de caractère .

6.1.2 XML

Le format XML à été créer dans l'optique d'avoir un format générique qui puisse être compréhensible aussi bien par un automate que par un utilisateur. L'idée de base est de pouvoir visualiser les résultats, via une implémentation style JUnit.

6.1.3 Affichage sur la console (sortie erreur)

Avoir un feedback en temps réel est très important pour suivre l'évolution d'une procédure de test. Cet outil peu notamment aider un utilisateur avancé

à débugger un problème qui aurait pu survenir pendant la phase de test.

L'affichage dans le shell nous a permis de comprendre notamment les erreurs que l'on obtenait avec le mécanisme de fork.

6.1.4 Fogbugz issue tracker

L'avantage d'avoir une plateforme comme fogbugz est de pouvoir controller et visualiser les bugs remontés par la communauté ainsi que l'avancement de leur résolution. Cet outil, sur lequel nous allons récupérer certains changeset se doit d'être alimenté par nos résultats de tests.

C'est ce que nous avons fait et voici un aperçu disponible dans la figure 7.1.

On peut y voir, que la première vérification via CI à détecté un problème pendant les tests et à été signalé via le message Failures. Ensuite CI à été relancé et comme aucun problème n'a été détecté, le changeset à été validé.

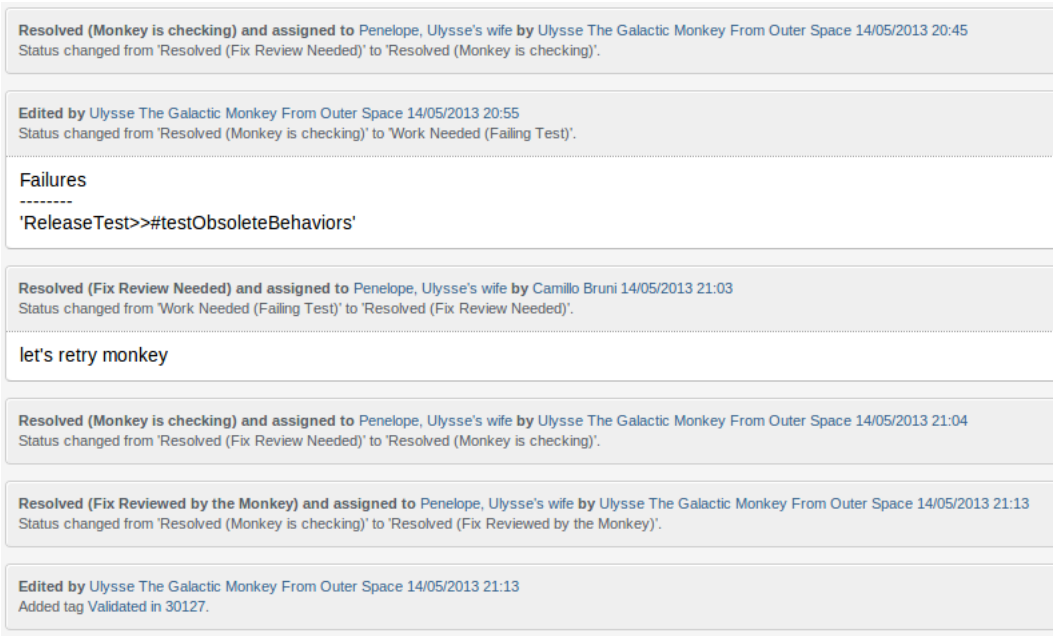

FIGURE 6.1 – Résultat de la publication sur FogBugz

À l'avenir nous aimerions avoir un simple lien (URL) redirigeant l'utilisateur sur une page HTML qui contiendra nos données.

7 Mise en application

L'étape finale du projet est la mise en application. Nous avons pu tester notre projet sur la plateforme Jenkins de l'équipe : https ://ci.inria.fr/pharo/. Cette mise en situation concerne le test des dernières modifications de code soumises par la communauté.

7.1 La ligne de Commande

Ce sur quoi j'ai commencé à travailler est le point d'entrée de l'application, à savoir la ligne de commande. La ligne de commande nous permettra par la suite de tester manuellement les nouvelles fonctionnalités.

Parmis les modifications que nous vérifions, nous faisons dans CI la différence entre les modifications de code (changeset) et les descripteurs de projet. Voici la syntaxe de la ligne de commande.

```
pharo.vm pharo.image ci [slice | configuration] [options−arguments]
```

La vm ainsi qu'une image Pharo contenant l'application CI sont à préciser au début de la ligne de commande. L'indicateur ci indique au système quel ligne de commande à choisir parmis celles disponibles dans le système. Suit le mot clé slice ou configuration pour spécifier ce que nous souhaitons traiter. Les options vont maintenant être détaillées dans leurs sections respectives.

7.1.1 Chargement et test de modifications de code

Pour les modifications de code, trois options sont disponibles :

- list
- load
- test

```
wawan@wawan-MS-16F1:~/Dropbox/Smalltalk/image$ pharo 28mai.image ci slice list
issue tracker authentification succeded
10775   https://pharo.fogbugz.com/f/cases/10775 Update Zn+Zdc May 2013
10840   https://pharo.fogbugz.com/f/cases/10840 WeakSet>>#size utterly broken
10837   https://pharo.fogbugz.com/f/cases/10837 CheckBox glitch fixed!
10839   https://pharo.fogbugz.com/f/cases/10839 VirtualMachine>>#maxExternalSema
phores: has bogus implementation (sort of)
```

FIGURE 7.1 – Exemple de listing via la ligne de commande

Voici un exemple de l'option list visible dans la figure 7.1.

L'option load installe par défaut le dernier commit ou celui que l'on lui indique en argument via : –issue=modification.

L'option test, quand à elle, charge et teste la modification de code grâce aux règles de validation qui seront présentées plus tard. Ensuite le résultat est posté sur le tracker, https ://pharo.fogbugz.com/ dans notre cas.

7.1.2 Descripteur de projet

Pour utiliser et tester les descripteurs de projet, il faut fournir 2 paramètres obligatoires :

- une URL indiquant le repository du descripteur
- le nom du descripteur

Voici un exemple complet d'une ligne de commande pour tester un descripteur de projet nommée Pastell, la version 1.3 et avec le paquet 'Tests' installé.

```
pharo.vm pharo.image ci configuration http://smalltalkhub.com/mc/PharoExtras/Pastell/
    main ConfigurationOfPastell --version=1.3 --group=Tests
```

L'option version permet de spécifier la version du projet. Cela peut-être un nombre '1.3' ou une version symbolique comme '#stable'

Group permet de spécifier quel groupe de paquets installer. Cette option est utile par exemple, dans le cadre d'un projet qui contient beaucoup de dépendance et que l'on souhaite tester seulement une partie de ce projet.

7.2 Application sur Jenkins

La ligne de commande suivante permet de lancer la vérification des derniers commit effectués de la communauté.

```
./pharo ci.image ci slice test
```

FIGURE 7.2 – Déploiement sur Jenkins

On peut s'apercevoir que l'application fonctionne et que les commits sont testées un à un. La figure 6.1 fournit un exemple d'un rapport de test effectué sur un changeset.

8 Test

Le projet CI se doit d'être stable pour assurer la fiabilité d'intégrations des modifications du source code. Pour cela j'ai créé un ensemble de tests.

Au final 93 tests nous permettent de connaître si notre projet est fonctionnel ou non. L'image ci-dessous montre le résultat des tests.

```
93 run, 93 passes, 0 skipped, 0 expected failures, 0 failures, 0
errors, 0 unexpected passes
```

FIGURE 8.1 – Résultat des test

Le principe du TDD, Test Driven Development, est d'écrire un ensemble de test et ensuite d'écrire les éléments qui vont être testés.

Pour la plupart des tests j'ai appliqué cette méthodologie.

9 Conclusion

Bilan techniques

Smalltalk

Ayant déjà fait un stage avec l'équipe RMoD je connaissais le language Pharo. Cette deuxième expérience avec Smalltalk m'a permis d'améliorer mes connaissances.

LaTeX

Comme vous avez pu le constater en lisant ce rapport, j'ai au cours de ce stage utilisé LaTeX. Ce rapport m'a donc permis d'apprendre Latex avec une mise en pratique immédiate.

Anglais

L'équipe RMod est connue au niveau international et comporte de nombreux étrangers. Par conséquent la langue par défaut est l'anglais. Cette expérience fut très bénéfique car j'ai pu enrichir mon vocabulaire non seulement par des termes techniques mais également par un langage plus courant.

Monticello

Monticello est le gestionnaire de version de code source dans Pharo. J'ai quotidiennement mis en ligne l'avancée de mon projet pour que n'importe qui puisse avoir accès et donc contrôler l'avancée de mon stage. J'ai donc appris à versionner mon code, diffusé sur le site www.smalltalkhub.com. Pour permettre une récupération aisée du projet, j'ai créé un descripteur du projet précisant les dépendances du projets, les différentes versions du projet (stable, développement). Ainsi, lors de l'installation toutes les dépendances externes sont installées.

Jenkins

Travailler avec la plateforme Jenkins est un plus pour comprendre l'intérêt de l'intégration continue. J'ai également via un autre projet de l'équipe, travailler avec l'API Rest Jenkins afin de récupéré des données depuis celui-ci. Grâce à ce stage je suis apte à utiliser Jenkins et de comprendre les mécanismes de ce type de plateforme.

Bilan humain

L'équipe RMod

Intégrer pour la seconde fois une équipe de recherche est un grand point positif pour ce stage car cela m'a permis de rencontrer des personnes de différentes cultures et de conforter mon ambition professionnelle, qui est pour le moment d'intégrer une équipe de recherche.

La communauté Pharo

Entré en contact avec la communauté Pharo l'année précédente, via une liste de diffusion, j'ai cette année contribué au sein de la communauté en résolvant certains problèmes que des utilisateurs rencontrés et j'ai également put partager l'avancement du projet CI. Cela nous a également permis de connaître le projet Seamless qui fut fort utile au mécanisme de clonage de l'image.

Pair programming

Comme indiqué dans ce rapport j'ai eu la chance de pair programmé avec Camillo Bruni. Ceci m'a permis d'approfondir mes connaissances sur Pharo mais surtout d'avoir une expérience de travail d'équipe.

Bilan de l'avancée du projet

Le projet CI que j'ai développé est aujourd'hui à un stade fonctionnel et stable. Nous allons prochainement ajouter de nouvelles fonctionnalités tel qu'un mécanisme de benchmark pour tester la fiabilité des modifications de code source ou encore embellir les rapports HTML.

10 Glossaire

UML (en anglais Unified Modeling Language ou "langage de modélisation unifié") est un langage de modélisation graphique à base de pictogrammes. Il est apparu dans le monde du génie logiciel, dans le cadre de la "conception orientée objet". Couramment utilisé dans les projets logiciels, il peut être appliqué à toutes sortes de systèmes ne se limitant pas au domaine informatique.

Pharo est une implémentation libre créée en 2009 sous licence MIT avec quelques parties sous licence Apache du langage de programmation Smalltalk. Pharo est de plus en plus un language à part entière qui s'inspire de Smalltalk

Licence MIT ou licence X11 est une licence de logiciel libre et open source, non copyleft, permettant donc d'inclure des modifications sous d'autres licences, y compris non libres.

Machines virtuelle sont utilisées pour exploiter les logiciels d'une machine qui n'existe plus dans le commerce (ordinateur, console de jeu, assistant personnel, ...), pour cacher la machine simulatrice et simuler une machine fictive, telle que la machine virtuelle Java. La VM Pharo est un interpréteur de code permettant de faire communiquer le byte code Pharo avec les instructions processeur

Monticello est l'outil de gestion de paquetages de Pharo. Malheureusement il ne gère pas les dépendances de chaque paquetages, ce que fait Metacello.

Metacello est le gestionnaire de paquetages de Monticello. Il permet de contrôler la dépendance des paquetages, les charger tout en contrôlant la plateforme de destination. Ces descripteurs sont couramment appelés ConfigurationOf.

Jenkins est un outil open source d'intégration continue.Il est écrit en Java. En fonction des jobs que l'on créer, on peut visualiser les rapports d'erreur.

Clonage/Fork est une fonction qui permet à un processus de donner naissance à un nouveau processus qui est sa copie conforme, par exemple en vue de réaliser un second traitement parallèlement au premier

Commit est l'action d'envoyer sur un repository le source code modifié.

Résumé

Dans le cadre de mon stage de fin d'étude, j'ai eu à réaliser une plateforme d'intégration continue, à Inria Lille - Nord Europe (Institut National de Recherche en Informatique et en Automatique) dans l'équipe RMoD. L'intégration continue est une méthode qui consiste à vérifier chaque modification effectuée sur le code source d'une application. Mon projet consiste à créer une application capable de charger, tester et rédiger des rapports sur les modifications apportées par la communauté sur le language Pharo.

Cet outil aura pour but de faciliter les corrections de bugs, de prévenir les incompatibilités, et d'indiquer si une version du logiciel est stable. Dans le cas d'un problème d'intégration, l'application aura pour objectif de fournir des rapports d'erreurs.

Abstract

For my internship at Inria (Institut National de Recherche en Informatique et en Automatique) in the RMoD team, I create a tool able to load and test commits made by the Pharo community. Continous integration is a practice where each modification to the source code is checked automatically. My project consists in creating a tool able to load, test and publish reports from each modification made by the Pharo community.

This tool will provide an easy way to maintain the Pharo platform, prevent some futures bugs and provide stable releases. In case of any bug, the tool has to publish reports which contain informations to help debugging.