ENSIIE
1 Rue de la Résistance
91000 Évry

Polytechnique Montréal
2500, chemin de Polytechnique
H3T 1J4 - Montréal (Québec)

INTERNSHIP REPORT

PROMOTION 2016

Services extraction from Ptidej to the Cloud

Student
Stéphane Di Miceli
stephane.dimiceli@ensiie.fr

Student
Ludovic DAVID
ludovic.david@ensiie.fr

Student
Charles Da Pont
charles.dapont@ensiie.fr

Internship Supervisor
Yann-Gaël GUÉHÉNEUC
yann-gael.gueheneuc@polymtl.ca

Internship Tutor
Marie SZAFRANSKI
marie.szafranski@ensiie.fr

From June 22 to 04 September
2015

Acknowledgement

First, I wish to express my sincere gratitude to Mr. Yann-Gaël Guéhéneuc and Mr. Foutse Khomh for providing me an opportunity to do my internship in the department of computer engineering and software engineering of Polytechnique Montreal and for their guidance and encouragement in carrying out this project work.

I sincerely thank Mr. Berthelot for providing me the opportunity to contact Polytechnique Montreal and to make all of this possible.

I also thank Mrs. Eva Mary Bures for providing me a good and comfortable apartment during the internship.

In addition, I would thank the others I have not mentioned who helped me to make this project become a reality.

Contents

Executive Summary

For my second year in the engineering school named ENSIIE, I have done a (approximatively) 3 months internship at Polytechnique Montreal.

I have worked in the department of computer engineering and software engineering and more precisely, in the Ptidej team. The mission was to extract some services of the Ptidej tool suite – a tool to enhance and compute the quality of object-oriented programs – to the Cloud.

This subject was interesting for multiple reasons. The first one was to better understand how describe the quality of a program and understand the existing work. The second one was to imagine and evaluate different solutions to resolve the mission. The third was to work in a famous research environment with expert peoples.

The first part of this internship report describe the context, the second one talk about what is the Ptidej tool suite and how it works in outlines. The last part describe how the mission was carried out, the technologies we have used and the perspectives of evolution.

Introduction

2.1 Polytechnique Montreal

Founded in 1878, the University of Montreal, with its two affiliated schools, the Polytechnique School and the HEC Montreal, is now the largest university in Quebec and the second largest in Canada. With over 60,000 students from around the world, the University awards more than 11,000 degrees every year. Its revenue from research is over a half-billion dollars, which makes it the third most dynamic Canadian university in terms of research. Since 2005 it has been the most dynamic in Quebec.

Figure 2.1: Polytechnique Montreal

Founded in 1873, Polytechnique Montreal is one of Canada's top institutions for the teaching and research of engineering, and the first in Quebec for the size of its student body and the scope of its research activities.

Polytechnique offers courses and programs in several engineering specialties and accounts for nearly one-quarter of the university research in these fields in Quebec. The university also conducts some of Canada's most intensive research activities through its approximately 60 research units and a faculty comprising world-renowned experts.

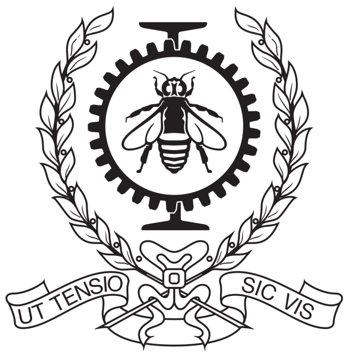


Figure 2.2: Emblem

Its emblem is a representation of a bee encircled by a gear, crossed by a steel beam. In general, bees represent the well-organized and planned work of engineers. The steel beam likely illustrates the first discipline of engineering instruction at the school: civil engineering. The gear suggests the industrial rise at the end of the 19th century, in which engineers played a major role. Finally, the laurel wreath symbolises excellence. The emblem is accompanied by the slogan "*Ut tensio sic vis*" which is based on a law of material resistance called: "Hooke's law". Its literal translation is "Elongation is proportional to force.". In a figurative sense, it means "results are proportional to the effort made": an edifying and inspiring slogan for all polytechnicians.

2.2 Department

The department of computer engineering and software engineering have approximatively 10 rooms for practicing infography, team software processing, networking, numeric systems, among other things.

2.2.1 PolyMORSE

The PolyMORSE (POLYtechnique MOntreal Researchers in Software Engineering) section regroup some laboratories :

- **Soccer Lab.**: SOftware Cost-effective Change and Evolution Research Lab
- **SWAT**: SoftWare Analytics and Technologies Lab.
- **MCIS**: Maintenance, Construction and Intelligence of Software
- **Ptidej Team**: Pattern Trace Identification, Detection, and Enhancement in Java

2.2.2 Ptidej team

The Ptidej Team (Pattern Trace Identification, Detection, and Enhancement in Java) aims to develop theories, methods and tools for understanding, evaluating and improving the quality of software systems by promoting the use of idioms, design patterns, and architectural patterns. They want to formalise patterns, to identify occurrences of patterns, and to improve the identified occurrences. They also want to examine and evaluate the impact of patterns on the quality of software systems. Various tools have been developed, most notably the Ptidej tool suite and Taupe, to evaluate and to enhance the quality of software systems, which promotes the use of patterns, either at the language-, design- or architectural-levels.

Figure 2.3: Ptidej Team

Situation Review

3.1 Ptidej

3.1.1 In some words

Ptidej is a tool suite written in Java to evaluate and enhance the quality of object-oriented programs. It is composed of several modules which contains, overall, more than 300 packages, 2.000 classes and 300 interfaces. It try to provide an architecture to model programs and to apply analyses, data conversion algorithms, and third-party programs on the models. This tool suite is actively used across the world to perform program analyses and to implement various tools.

3.1.2 Architecture

Ptidej is a large project. It is very difficult to explain it as a whole. We can discern 2 main parts : The *Meta-model* and the *User interface*.

Meta-model

- A meta-model, PADL (Pattern and Abstractlevel Description Language), to describe the structure of motifs —the “Solution” parts in pattern definitions— and of object-oriented programs,
- A library of design motifs from design patterns, including Abstract Factory, Chain of Responsibility, Composite, Observer, Visitor...
- Several parsers to build models of programs from different source code representations, including AOL, C++ files and Java class files.
- A library of software metrics, POM (Primitives, Operators, Metrics), to compute well-known metrics on models of programs, such as Chidamber and Kemerer’s metrics[1].
- A library of generators and analyses to be applied on models of programs and of motifs.

[1]http://www.aivosto.com/project/help/pm-oo-ck.html

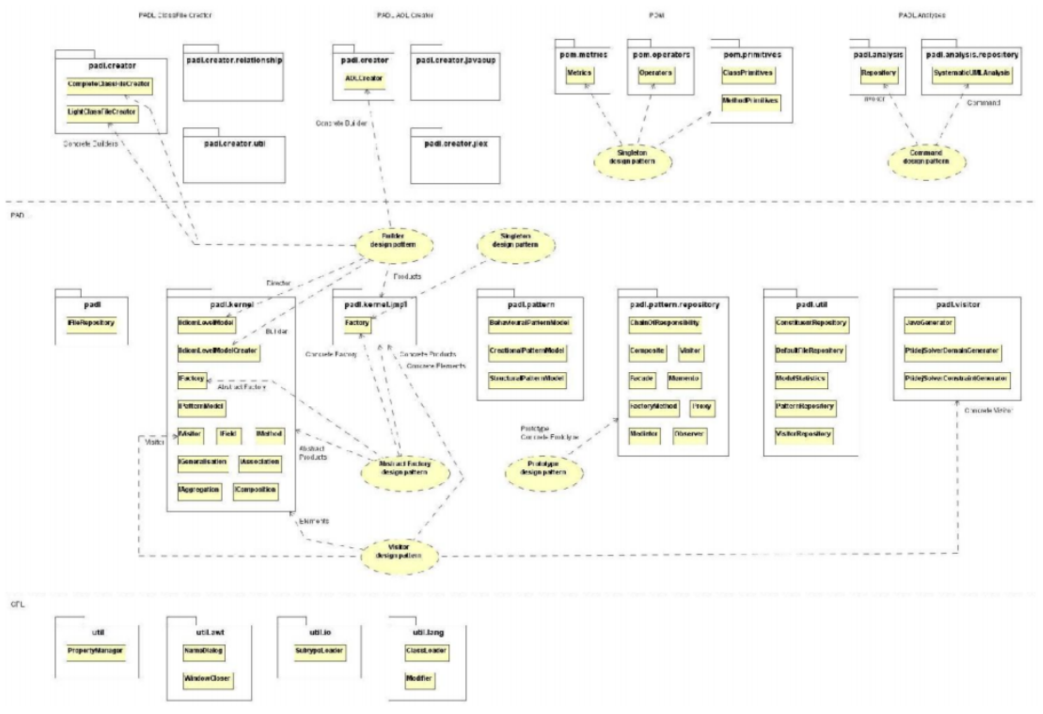


Figure 3.1: Meta-model architecture

- An explanation-based constraint solver, PTIDEJ SOLVER, to identify micro-architectures similar to motif models in program models.
- A dynamic analyser for Java, CAFFEINE, based on a Prolog engine and the Java debug interface to define relationships among classes precisely.

User interface

- A library of graphic widgets, PTIDEJ UI, to display models of motifs, of programs, and dynamic data from CAFFEINE.
- Several user-interfaces to access the functionalities provided by the Ptidej tool suite:
  - Parse and create models of programs.
  - Enhance models of programs with dynamic data from program executions.
  - Visualise created models.
  - Identify micro-architectures similar to a design motif model in a program model.
  - Visualise the identified micro-architectures.
  - Call generators, analyses, and external tools on models.

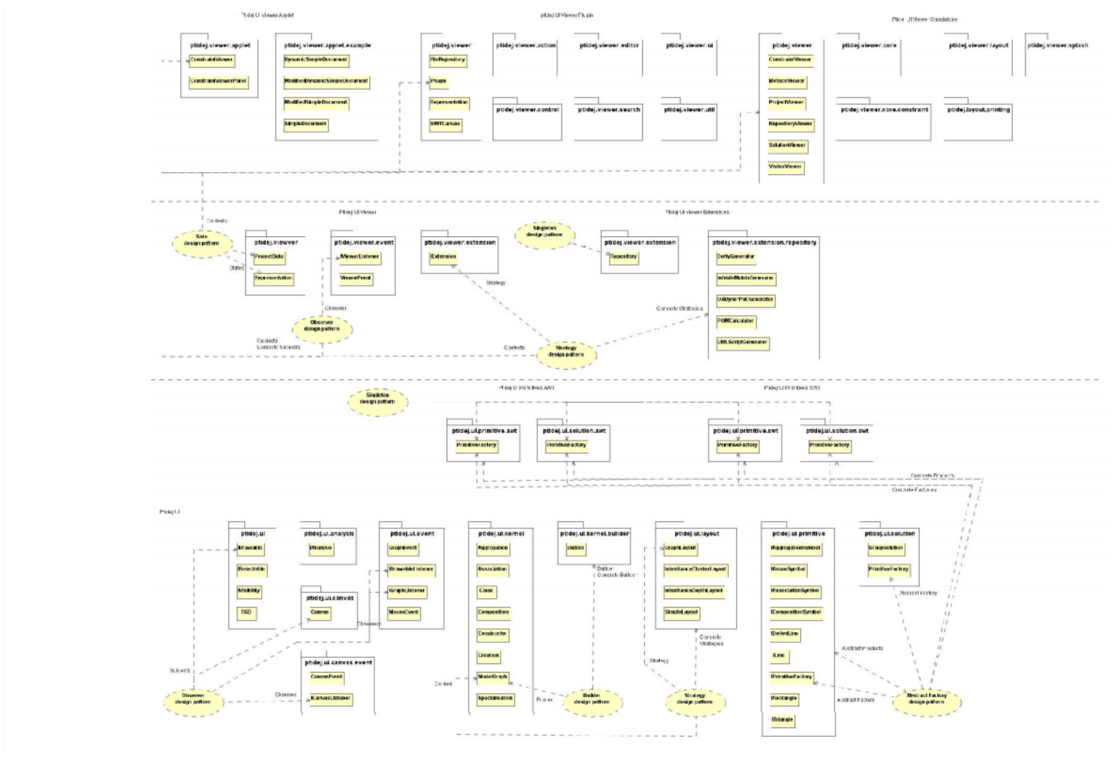
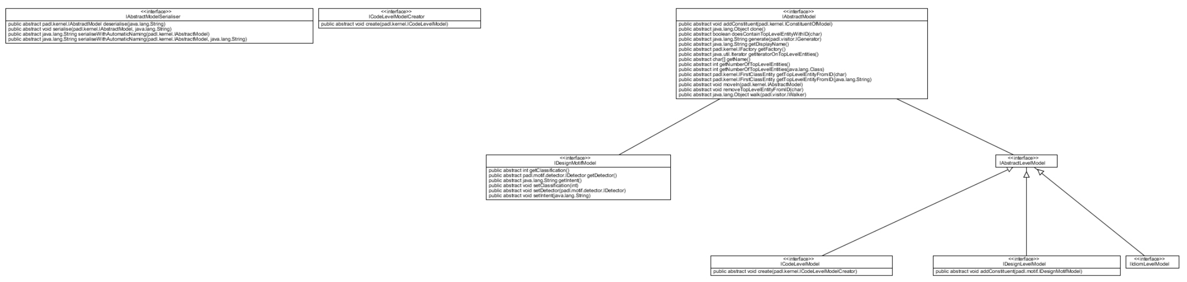


Figure 3.2: User interface architecture

3.1.3 Focus on some services

As we have seen, Ptidej Tool suite is composed of several modules. The mission consists in the extraction of some of them which here:

PADL



```
<<interface>>
IAbstractModelSerialiser
public abstract padl.kernel.IAbstractModel deserialise(java.lang.String)
public abstract void serialise(padl.kernel.IAbstractModel, java.lang.String)
public abstract java.lang.String serialiseWithAutomaticNaming(padl.kernel.IAbstractModel)
public abstract java.lang.String serialiseWithAutomaticNaming(padl.kernel.IAbstractModel, java.lang.String)
```

```
<<interface>>
ICodeLevelModelCreator
public abstract void create(padl.kernel.ICodeLevelModel)
```

```
<<interface>>
IAbstractModel
public abstract void addConstituent(padl.kernel.IConstituentOfModel)
public abstract java.lang.Object clone()
public abstract boolean doesContainTopLevelEntityWithID(char)
public abstract java.lang.String generate(padl.visitor.IGenerator)
public abstract java.lang.String getDisplayName()
public abstract padl.kernel.IFactory getFactory()
public abstract java.util.Iterator getIteratorOnTopLevelEntities()
public abstract char[] getName()
public abstract int getNumberOfTopLevelEntities()
public abstract int getNumberOfTopLevelEntities(java.lang.Class)
public abstract padl.kernel.IFirstClassEntity getTopLevelEntityFromID(char)
public abstract padl.kernel.IFirstClassEntity getTopLevelEntityFromID(java.lang.String)
public abstract void moveIn(padl.kernel.IAbstractModel)
public abstract void removeTopLevelEntityFromID(char)
public abstract java.lang.Object walk(padl.visitor.IWalker)
```

```
<<interface>>
IDesignMotifModel
public abstract int getClassification()
public abstract padl.motif.detector.IDetector getDetector()
public abstract java.lang.String getIntent()
public abstract void setClassification(int)
public abstract void setDetector(padl.motif.detector.IDetector)
public abstract void setIntent(java.lang.String)
```

```
<<interface>>
IAbstractLevelModel
```

```
<<interface>>
ICodeLevelModel
public abstract void create(padl.kernel.ICodeLevelModelCreator)
```

```
<<interface>>
IDesignLevelModel
public abstract void addConstituent(padl.motif.IDesignMotifModel)
```

```
<<interface>>
IIdiomLevelModel
```

Figure 3.3: PADL classes hierarchy

PADL stands for *Pattern and Abstract-level Description Language*. It can describe object-oriented programs at different levels of abstraction. There is four different levels :

- **ICodeLevelModel**: Represents the much lower model of a program as Java byte-codes, C/C++ source code, ...

- **IIdiomLevelModel**: Represents a level where idioms have been reified, in general, binary-class relationships
- **IDesignLevelModel**: Represents the level at which design information is available. By design, we understand for example occurrences of design motifs or of code smells
- **IDesignMotif**: Represents a design motif, in other words, the solution to a design pattern model in PADL Design Motifs

Ptidej Solver

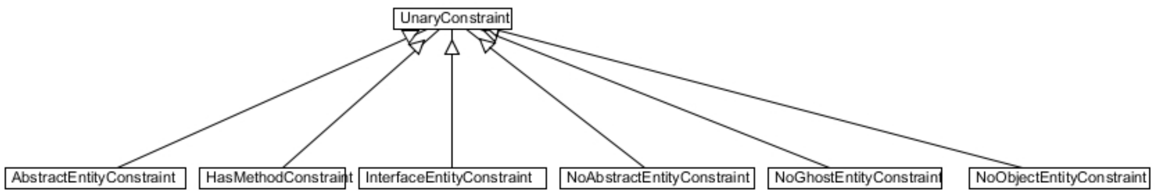


Figure 3.4: Ptidej Solver classes hierarchy

Ptidej Solver is used for automate the identification of design patterns performed using an explanation-based constraint solver (built on top of PaLM and JChoco). It support around twenty patterns like:

- **Abstract Factory**: provides a way to encapsulate a group of individual factories that have a common theme without specifying their concrete classes
- **Chain Of Responsibility**: consisting of a source of command objects and a series of processing objects
- **Facade**: provides a simplified interface to a larger body of code, such as a class library

SAD

SAD stands for *Software Architectural Defects*. It is based on rule cards for automatic code generation of anti-patterns detectors. SAD requires POM to compute various metrics involved in the definitions of the code and design smells. These metrics are referenced by name in the rule cards. It is used to detect code smells and anti-patterns like:

- **AntiSingleton**: A class that provides mutable class variables, which consequently could be used as global variables
- **LazyClass**: A class that has few fields and methods.
- **Blob**: A large controller class that depends on data stored in surrounding data classes. A large class declares many fields and methods with a low cohesion. A controller class monopolises most of the processing done by a system, takes most of the decisions, and closely directs the processing of other classes. Controller classes

can be identified using suspicious names such as Process, Control, Manage, System, and so on. A data class contains only data and performs no processing on these data. It is composed of highly cohesive fields and accessors.

POM

POM stands for *Primitives, Operators, Metrics*. It can compute a lot of different metrics. There is two fundamentals kinds of metrics: the unary and the binary. For the mission, our interest is only focused on the unary one. There is around seventy supported metrics like:

- **AID**: Average Inheritance of Depth of an Entity
- **DCMEC**: Descendants Class-Method Export Coupling of one entity
- **EIP**: Number of inheritance relationships where the super-entity is in the package containing the entity and the sub-entites is in another package

SQUAD

SQUAD stands for *Software Quality Understanding through the Analysis of Design*. It can compute two different models: *QMOOD* and *PQMOD*.

- **QMOOD**: the hierarchical model that defines relation between qualities attributes (like reusability, functionality, effectiveness, understandability, extendibility, flexibility) and design properties with the help of equations.
- **PQMOD**: based on *QMOOD*, but it is more powerful because it take into account design patterns. The list of qualities become: reusability, expandability, scalability, understandability, generality, modularity at runtime, and modularity.

Project

The mission was to extract some services of PTIDEJ to the Cloud. Previously, we have describe what exactly is the PTIDEJ TOOL SUITE and how it works in outlines. Before describe the methodology and the steps of execution, it is interesting to see what is exactly the Cloud, because it is a recent notion and this is not simple as we may think.

4.1 Cloud Computing

The Cloud computing is based on the Internet network and some observations extracted from previous methodologies. Before explaining it more in details, it is necessary to tell that the high majority of people see this technology just as a distant server to stock files, with some useful applications for manipulating us. But in reality, this is more than that.

In a few words, we can say that the Cloud is based on two main things : scalability and the expectation of failure.

Before it's apparition, when we needed to deploy a web service, we use a unique web server. The more people use the website, the bigger need to be the server. There is a lot of disadvantages with this technique. When nobody navigate on the web application, it is impossible to reduce the size of the server or use it for something else. All the power of the server is unused and wasted. With the Cloud, instead of buy a bigger computer, you buy an another one. With this technique, costs are reduced and the power is distributed. You can scale at convenience in function of the number of users. This is the explanation of the scalability.

For the expectation of failure, the Cloud use an efficient system of replication. All data is duplicated three times at least and stocked in a various ways: object, block or file-level storage[1]. So, when a virtual or real machine is down, the integrity of data are preserved and the machine is replaced by another one automatically redeployed for a virtual machine or added to the current cluster for a real machine.

Nowadays, there is a lot of software to manage all computers and virtual machines as a whole. But the technology is still young and in consequence not totally stable for the moment.

[1]http://docs.openstack.org/openstack-ops/content/storage_decision.html

4.2 Why on the Cloud?

Before talking about the employed methodology, we can discuss a little about the reason for which the Cloud is truly valuable and advantageous. Ptidej compute a lot of things, from metrics to design patterns and smells. Those computations take a precious time, and the Cloud is a perfect solution to reduce the CPU usage on the local machine, be sure of never lose computed data and be robust against a big community of users. Moreover, computed meta-models take a lot of space, with the Cloud, it's no longer a problem.

4.3 Methodology

For carrying out the mission, in a first time, we have discuss with our supervisor to extract the list of things to produce and to do:

- An API to provide an access to services of Ptidej,
- A documentation to explain how the API work and how install and configure it,
- A panel of unit test to prove the good functioning of the application,
- Configure a Cloud environment to run the API on.

Before all, we had evaluate and compare some cloud softwares : OpenStack, CloudStack, ... By the time and for it's simplicity, we had choice OpenStack ; a cluster was already available in the laboratory.

For the API, we choose the REST (REpresentational State Transfer) architecture because of it's characteristics described here:

- Client/Server: responsibilities are separated between the client and the server. The user-interface is separated from the storage of data. In this way, both can evolve independently,
- Without state: every request from a client to a server need to contain all the necessary information to allow the server to understand the request without a specific context. This free a lot of interactions between the client and the server,
- Cache: the server send a response who give some informations about the propensity of this response for be cached, like the freshness, it's creation date, if it need to be retained. In this way, some proxy servers are unloaded from constraints and clients can do not useless requests. This extend the extensibility of servers,
- Uniform interface: this constraint is based on four essentials rules:
  - Resources identification,
  - Manipulate resources across representations,
  - Auto-descriptive message,
  - Hypermedia as a state engine of the application
- Layered system: all states of the application are identified by an individual resources. All the information is not send in a single resource. Requests and responses

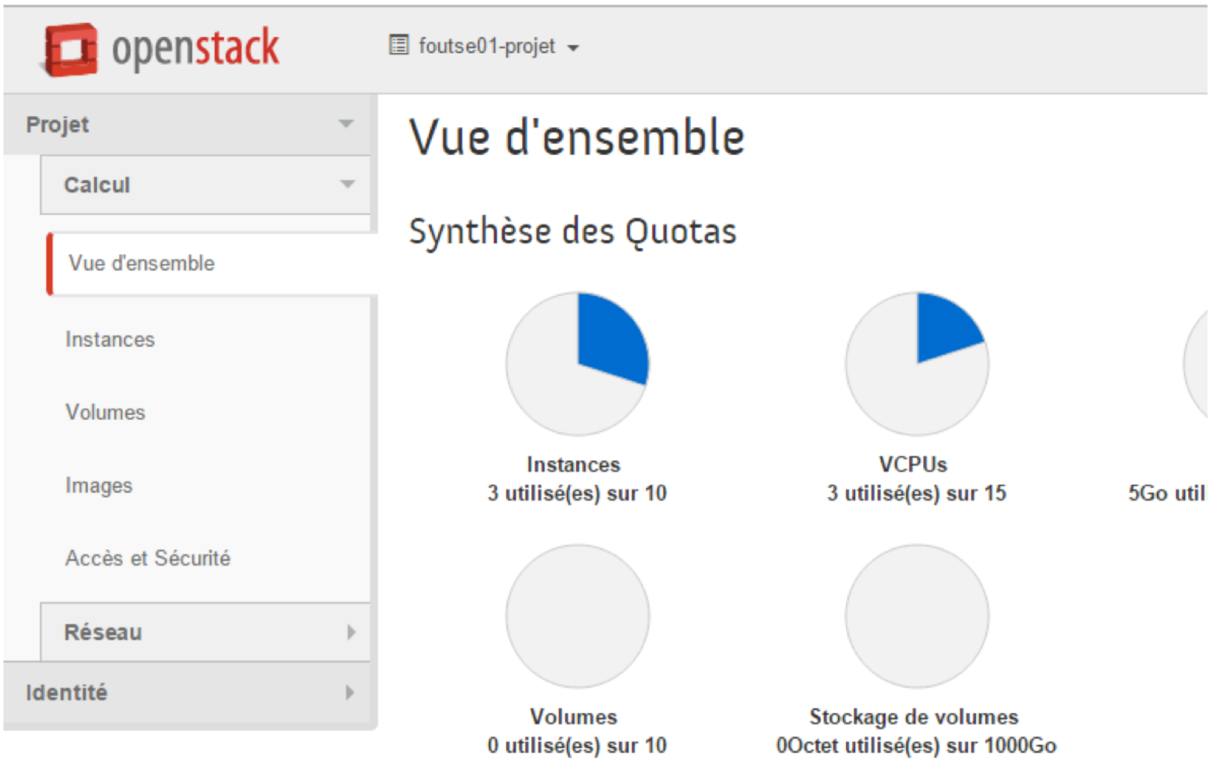
between the server and the client raise and can decrease the performance (that's why the cache is so important).

4.4 Steps of Execution

4.4.1 Ubuntu and Glassfish

We have used an Ubuntu server for the OS purpose. Before using it, some little things was needed. In first, it was necessary to install it on a VM. We have configured it for OpenStack. For example, because OpenStack need to launch it automatically, SSH keys are sent by the software itself, and the recovering need to be configured manually. A server to host our application was needed too. We have choice Glassfish because it's more than just a servlet container like Tomcat. We required it for more flexibility and power. For Glassfish, we have setting up the OS environment, in other word, set up users and ip-tables for security and network purpose. After, we have setting up Java with last version (jdk) and configure the ssl security. Then, we have configure the auto-start of the Glassfish server and complete the security configuration with certificates, headers, etc...After this process, we have convert the image in an understandable format for OpenStack: qcow2. And then, upload it on our OpenStack cluster as an image.

4.4.2 OpenStack



Vue d'ensemble

Synthèse des Quotas

Instances
3 utilisé(es) sur 10

VCPUs
3 utilisé(es) sur 15

5Go util

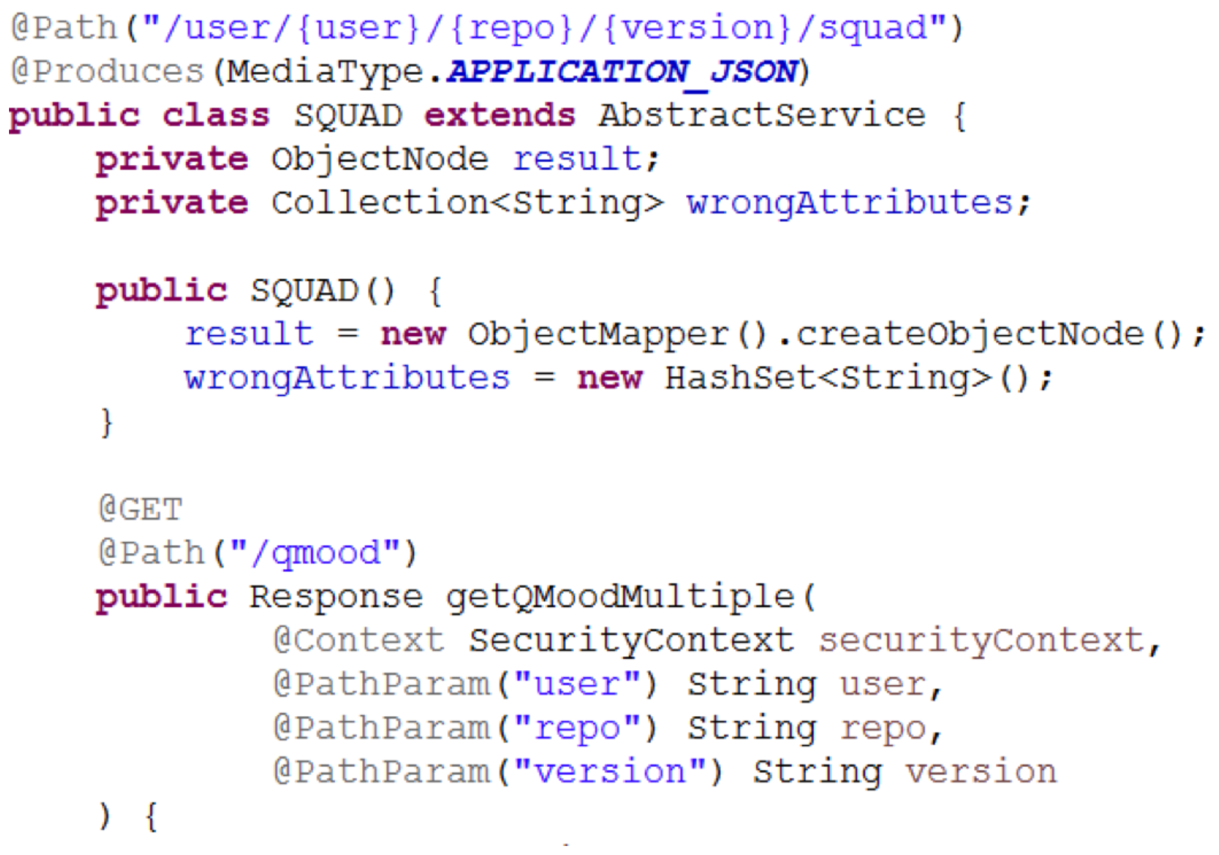
Volumes
0 utilisé(es) sur 10

Stockage de volumes
0Octet utilisé(es) sur 1000Go

Figure 4.1: OpenStack web office

For OpenStack, we have follow the instruction provided by the network administrator. In first, we have create our virtual network with gateways, routers, ... Then, we have configure groups of accessibility (like in a standard version of an OS). Then the access who correspond to a sort of ip-table. And to finish, launch an instance of our server on the cluster.

4.4.3 REST API



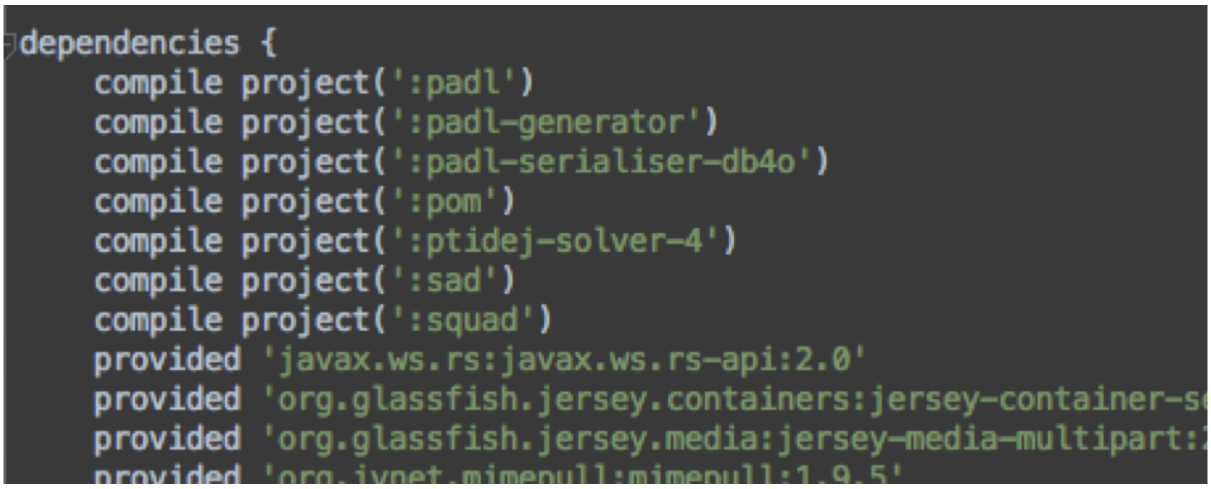
```
@Path("/user/{user}/{repo}/{version}/squad")
@Produces(MediaType.APPLICATION_JSON)
public class SQUAD extends AbstractService {
    private ObjectNode result;
    private Collection<String> wrongAttributes;

    public SQUAD() {
        result = new ObjectMapper().createObjectNode();
        wrongAttributes = new HashSet<String>();
    }

    @GET
    @Path("/qmood")
    public Response getQMoodMultiple(
            @Context SecurityContext securityContext,
            @PathParam("user") String user,
            @PathParam("repo") String repo,
            @PathParam("version") String version
    ) {
```

Figure 4.2: API source code based on Jersey

About the REST API, we have integrate Gradle as a dependency manager. Because all dependencies was previously not really managed but stocked in the configuration files of the Eclipse project. It was difficult because Ptidej is a complex application containing a lot of dependencies between modules. Moreover, afterward, it was really easy to add new dependencies for the REST API itself. We have use Jersey and Jackson who are respectively two frameworks. The first, Jersey, based on JAX-RS (Java API for RESTful Web Services) was essential and powerful in the development of the API. It allow us to use the power of annotations to code more easily a REST API. Concerning the second, Jackson, it allow us to use JSON format instead of XML who is, nowadays, too verbose and heavy for this kind of things.

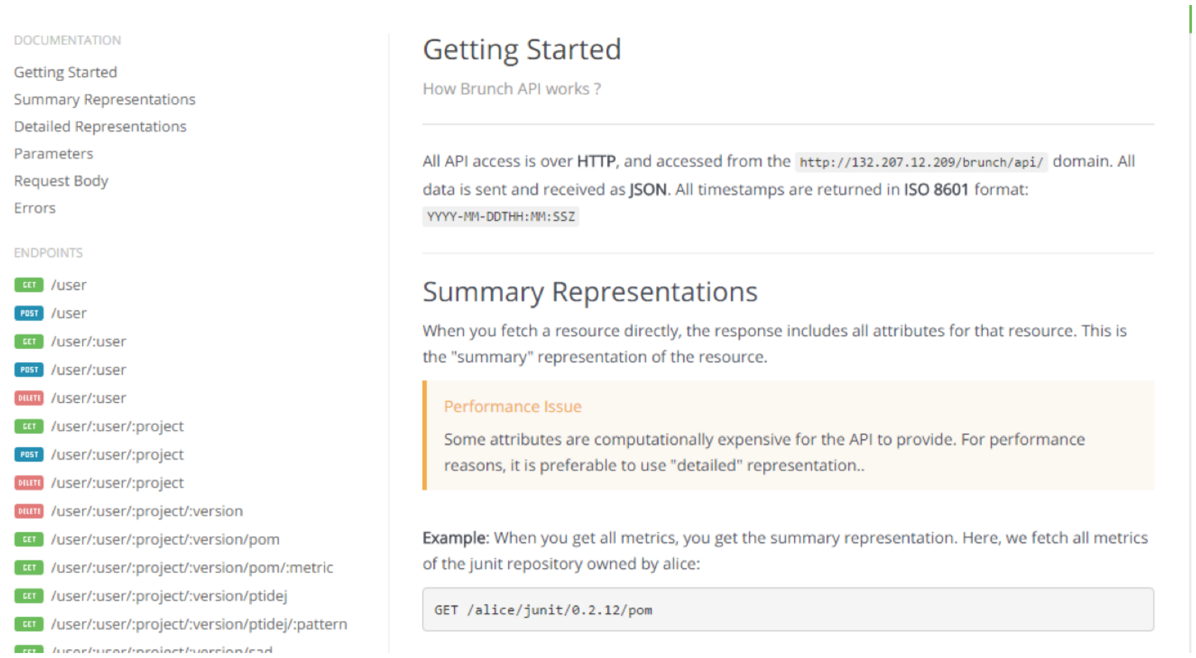


```
dependencies {
    compile project(':padl')
    compile project(':padl-generator')
    compile project(':padl-serialiser-db4o')
    compile project(':pom')
    compile project(':ptidej-solver-4')
    compile project(':sad')
    compile project(':squad')
    provided 'javax.ws.rs:javax.ws.rs-api:2.0'
    provided 'org.glassfish.jersey.containers:jersey-container-se
    provided 'org.glassfish.jersey.media:jersey-media-multipart:
    provided 'org.jvnet.mimepull:mimepull:1.9.5'
```

Figure 4.3: Gradle configuration

4.4.4 Documentation

Regarding the documentation, we have used AngularJS. A popular framework based on a fast flow due to AJAX communication who allow the system to don't refresh the page and update just a precise part. It was extremely rapid to develop, but the interesting part is the content. We have maintain it all along the project with all the explanation about how the API works. Which calls we can do with which parameters etc...



DOCUMENTATION

Getting Started
Summary Representations
Detailed Representations
Parameters
Request Body
Errors

ENDPOINTS

GET /user
POST /user
GET /user/:user
POST /user/:user
DELETE /user/:user
GET /user/:user/:project
POST /user/:user/:project
DELETE /user/:user/:project
DELETE /user/:user/:project/:version
GET /user/:user/:project/:version/pom
GET /user/:user/:project/:version/pom/:metric
GET /user/:user/:project/:version/ptidej
GET /user/:user/:project/:version/ptidej/:pattern
GET /user/:user/:project/:version/sad

Getting Started

How Brunch API works ?

All API access is over **HTTP**, and accessed from the `http://132.207.12.209/brunch/api/` domain. All data is sent and received as **JSON**. All timestamps are returned in **ISO 8601** format: `YYYY-MM-DDTHH:MM:SSZ`

Summary Representations

When you fetch a resource directly, the response includes all attributes for that resource. This is the "summary" representation of the resource.

Performance Issue

Some attributes are computationally expensive for the API to provide. For performance reasons, it is preferable to use "detailed" representation..

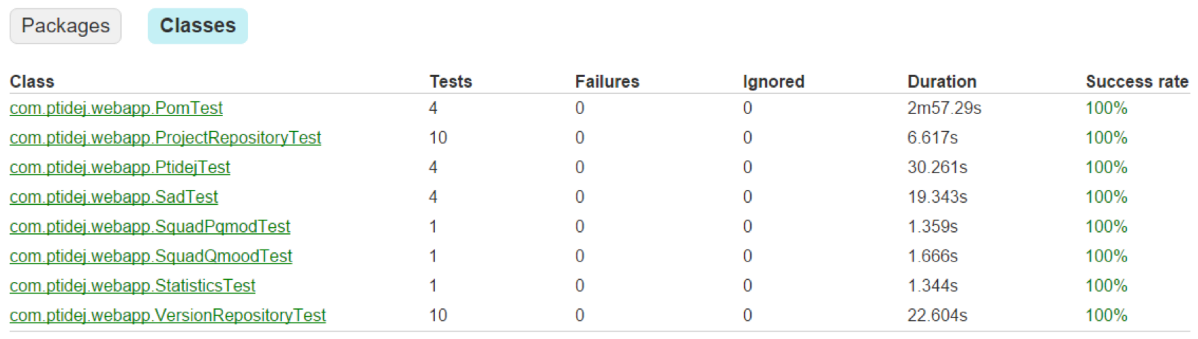
Example: When you get all metrics, you get the summary representation. Here, we fetch all metrics of the junit repository owned by alice:

```
GET /alice/junit/0.2.12/pom
```

Figure 4.4: Documentation

4.4.5 Unit tests

Involving the Unit testing part, we have used the Rest-assured framework. In this way, we can test our API like it was deployed. In this process, we use a secondary server, Jetty, who is auto-deployed during the compilation process just before the test execution.



Class	Tests	Failures	Ignored	Duration	Success rate
com.ptidej.webapp.PomTest	4	0	0	2m57.29s	100%
com.ptidej.webapp.ProjectRepositoryTest	10	0	0	6.617s	100%
com.ptidej.webapp.PtidejTest	4	0	0	30.261s	100%
com.ptidej.webapp.SadTest	4	0	0	19.343s	100%
com.ptidej.webapp.SquadPqmodTest	1	0	0	1.359s	100%
com.ptidej.webapp.SquadQmoodTest	1	0	0	1.666s	100%
com.ptidej.webapp.StatisticsTest	1	0	0	1.344s	100%
com.ptidej.webapp.VersionRepositoryTest	10	0	0	22.604s	100%

Figure 4.5: Unit testing results

4.5 Evolution perspectives

- **Degradation threshold:** When we compute design patterns, by default, there is a threshold of degradation who correspond of the number of link between classes who can be considered or ignored. It can be interesting to add this parameter in requests for design patterns.
- **Exhaustive save of meta-model:** When someone compute something on a meta-model, the meta-model is not updated, and when the resource is requested again, it is re-computed. It can be interesting to add a system of memoization to never re-compute things who have been already computed.
- **Incremental build of meta-model:** Our system support multiples versions of the same project. Between the versions, the project is not totally modified. It can be interesting to use the previous computed model as a base to construct the next one.
- **Improved Security:** There is a lot of things who can be implemented to improve the security like a white list of IPs, support of HTTPS, ...
- **Load balancing configuration**
- **PADL fully implemented:** Support of C++, C# and other format of project
- **User interface linked with API:** A user interface already exist, it can be useful to link all computations with the API
- **Automatic service addition:** For the moment, the process of addition of a new service from Ptidej need to be carried out manually. It can be truly useful to convert it in a automatic process.
- **Integration of/in other web services:** Create some plugins for Github, Gitlab, Mercurial, etc... can be interesting to share the project with more people
- **API documentation generated from sources:** The maintaining of the documentation is expensive in terms of time. If the API was automatically generated, a precious time will be saved.

Conclusion

In review this internship has been a great and rewarding experience. I have been able to meet and network with so many people that I am sure will be able to help me with opportunities in the future.

It allows me to understand the world of research and enhance my skills in computer science and my social competences. One main thing that I have learned through this internship is autonomy as well as self-motivation. I learned that I needed to be organized and have questions ready for when it was the correct time to get feedback.

The mainly mission was done, not without ambushes and problems, but more with conception, reflexions and discussions with supervisors. The team working was properly handled and was really advantageous due to the time of the mission.

In addition, before this internship, I wasn't planning to do a PhD. But after some discussions, today, the question then arises.

Webography

1. http://www.polymtl.ca/
2. http://ptidej.net/
3. http://wiki.ptidej.net/doku.php